Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2019/2020

Appello del 20 gennaio 2020

Esercizio 1 (8 punti)

Si consideri il seguente programma Java:

```
class A {
	int n;

	public A(int n) {
		this.n=n;
	}

	public int getN() {
		return n;
	}

	public A metodo (A a) {
		A ris = new B (n+ a.n);
		return ris;
	}

	public B metodo (B b) {
		B ris = new B (n + b.n + 1);
		return ris;
	}
}
class B extends A {
	public B(int n) {
		super (2*n);
	}

	public A metodo (A a) {
		A ris = new A (n + a.n + 2);
		return ris;
	}
}
public class MainClass {
	public static void main(String[] args) {
		A a = new A(3);
		B b = new B(2);
		A c = new B(1);
		System.out.println (c.metodo(a).getN());
		System.out.println (c.metodo(a.metodo(b)).getN());
	}
}
```

Dire cosa stampa il programma, giustificando la risposta mostrando le firme associate a tempo di compilazione ad ogni chiamata di metodo e l'evoluzione della memoria (stack e heap).

Esercizio 2 (12 punti)

Si scrivano le classi Java per gestire il sistema delle contravvenzioni per infrazioni stradali in un comune.
Si scriva una classe astratta **Personale** con

- una variabile di istanza **nome** (tipo String, private);
- una variabile di istanza **cognome** (tipo String, private);
- una variabile di istanza **matricola** (tipo int, final, private).

Implementare i seguenti metodi di istanza:

- un costruttore che crea un Vigile dati **nome, cognome e matricola**;
- metodi pubblici accessori **`getNome()`**, **`getCognome(), getMatricola()`**;

La classe Personale è estesa dalle sottoclassi **Vigile** e **Ausiliario,** per le quali vanno scritti opportunamente i costruttori che richiamano il costruttore della classe Personale e il metodo pubblico **`boolean equals (Object o)`** che restituisce true se e solo se o è un oggetto della stessa classe avente matricola uguale a quella di this (si noti che possono esistere un vigile e un ausiliario con la stessa matricola, che chiaramente **non** sono da considerare uguali).

Si assuma, senza scrivere nulla, l'esistenza di una classe **Multa** con

- una variabile di istanza **descrizione** (tipo String, final, private)
- una variabile di istanza **puntiPatente** (tipo int, final, private)
- una variabile di istanza **importo** (tipo double, final, private)
- una variabile di istanza **targa** (tipo String, final, private)
- una variabile di istanza **autore** (tipo Personale, final, private);
- un costruttore che prende tutti e 5 i parametri sopra descritti;
- metodi pubblici accessori per tutti e 5 i parametri sopra descritti.

Si progetti infine una classe **ComandoMunicipale** con

- una variabile di istanza **persone** (tipo ArrayList<Personale>, final, private);
- una variabile di istanza **multe** (tipo ArrayList<Multa>, final, private);

Implementare i seguenti metodi di istanza:

- un costruttore senza parametri che crea arraylist vuoti per persone e multe.
- un metodo **public double getImporto (int i)** che restituisce il totale degli importi delle multe fatte dal vigile o ausiliario in posizione i dell'arraylist persone (si sfrutti il metodo equals scritto per le classi Vigile e Ausiliario);
- un metodo **public Personale getMigliore ()** che restituisce il Vigile o Ausiliario che ha fatto multe per il più alto importo;
- un metodo **public boolean vigiliVsAusiliari ()** che restituisce true se e solo se il totale degli importi delle multe fatte dai vigili è superiore a quello relativo a multe fatte dagli ausiliari.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Esercizio 3 (10 punti)

Dato un intero positivo x ed un array a di n interi positivi, progettare un algoritmo di programmazione dinamica che ritorni il valore **true** se e solo se esistono o meno indici $i_1, i_2, \ldots, i_k$ (tra loro distinti) nell'array a tali che:

$$\sum_{j=1}^{k} a_{i_j} = x$$

Esempio: Si consideri a= {4, 7, 9, 2). È possibile ottenere 13 con tre indici 0,1 e 3 (ovvero a[0]+a[1]+a[3]=13).

Suggerimento: indicare con F(i,y) la soluzione al problema che richiede di formare l'intero y utilizzando i primi i elementi dell'array a.

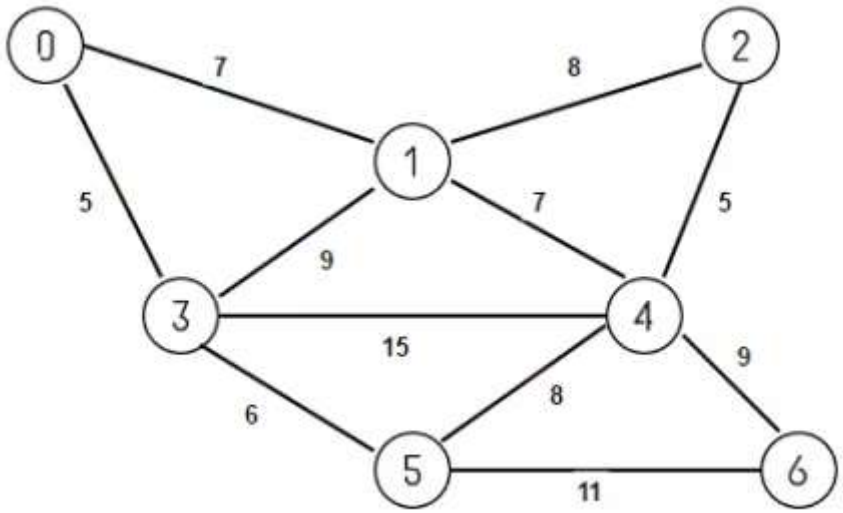
$$F(i,y) = \begin{cases} true & se\ y = 0 \\ false & se\ y > 0\ e\ i = 0 \\ ??\ or\ ?? & altrimenti \end{cases}$$

Nella terza riga (caso ricorsivo) bisogna considerare due casi: il caso in cui l'i-esimo elemento di a (cioè a[i-1]) concorre a formare y, e il caso in cui non concorre.

a) Si fornisca una implementazione ricorsiva in Java dell'algoritmo proposto.
b) Si fornisca una implementazione ricorsiva con **memoization** in Java dello stesso algoritmo.

Esercizio 4 (10 punti)

Si consideri il grafo non diretto in figura.



a. **[5 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Kruskal** per il minimo albero ricoprente.
b. **[5 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente a partire dal nodo sorgente **6**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2019/2020

Appello del 4 febbraio 2020

Esercizio 1 (8 punti)

Si consideri il seguente programma Java:

```
class A {
	protected int n;

	public A(int n) {
		this.n=n;
	}
	public int getN() {
		return n;
	}
	public A metodo (A a) {
		A ris = new B (n+ a.n);
		n = ris.n+1;
		return ris;
	}
	public B metodo (B b) {
		B ris = new B (n + b.n + 1);
		n = ris.n-1;
		return ris;
	}
}
class B extends A {
	public B(int n) {
		super (2 + n);
	}
	public A metodo (A a) {
		A ris = new B (n + a.n + 2);
		n = ris.n;
		return ris;
	}
}
public class MainClass {
	public static void main(String[] args) {
		A a = new A(1);
		B b = new B(2);
		A c = new B(3);
		System.out.println (c.metodo(b).getN());
		System.out.println (b.metodo(a.metodo(b)).getN());
		System.out.println (c.metodo(c.metodo(a)).getN());
	}
}
```

Dire cosa stampa il programma, giustificando la risposta e mostrando: le firme associate a tempo di compilazione ad ogni chiamata di metodo e l'evoluzione della memoria (stack e heap).

Esercizio 2 (12 punti)

Si vogliono gestire, in Java, gli ordini di un negozio di fast food.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Personale** con

- una variabile di istanza **nome** (tipo String, private);
- una variabile di istanza **cognome** (tipo String, private);
- una variabile di istanza **matricola** (tipo int, final, private).

e i seguenti metodi di istanza:

- un costruttore che crea un oggetto dati **nome, cognome e matricola**;
- metodi pubblici accessori **`getNome()`**, **`getCognome(), getMatricola()`**;

Si assuma, senza scrivere nulla, l'esistenza di una classe **ProdottoOrdinato** con

- una variabile di istanza **descrizione** (tipo String, private)
- una variabile di istanza **costoUnitario** (tipo double, private)
- una variabile di istanza **quantita** (tipo int, private)
- un costruttore che prende i 3 parametri sopra descritti;
- metodi pubblici accessori per tutti e 3 i parametri sopra descritti.

Si scriva una classe astratta **Ordine** con

- una variabile di istanza **cameriere** (tipo Personale, private) che rappresenta il cameriere che gestisce il tavolo;
- una variabile di istanza **comanda** (tipo ArrayList<ProdottoOrdinato>, final, private);

e i seguenti metodi di istanza:

- un costruttore che prende come parametro un **cameriere** e crea una comanda vuota associandola all'ordine;
- metodi pubblici accessori **`getCameriere()`**
- metodo pubblico astratto **`double getTotale()`** che restituisce l'importo totale dell'ordine.

La classe astratta **Ordine** è estesa dalle sottoclassi (non astratte) **OrdineTavolo** e **OrdineAsporto.**

Si tenga conto del fatto che la classe **OrdineTavolo** deve avere:

- una variabile di istanza **tavolo** (tipo String, private) che contiene l'identificativo del tavolo;
- una variabile di istanza **coperti** (tipo int, final, private) che contiene il numero di coperti.

e la classe **OrdineAsporto** deve avere:

- una variabile di istanza **nominativo** (tipo String, private) che contiene il nominativo della persona che ha effettuato l'ordine d'asporto.

Per entrambe le sottoclassi vanno scritti opportunamente i costruttori che richiamano il costruttore della classe **Ordine** e va implementato il metodo **`double getTotale(),`** tenendo conto del fatto che ogni coperto ha un costo di € 1,50 (nel caso di OrdineTavolo), mentre non c'è nessun costo aggiuntivo per gli ordini da asporto.

Si progetti infine una classe **FastFood** con

- una variabile di istanza **persone** (tipo ArrayList<Personale>, final, private);
- una variabile di istanza **ordini** (tipo ArrayList<Ordine>, final, private);

Implementare i seguenti metodi di istanza:

- un costruttore senza parametri che crea arraylist vuoti per persone e ordini.
- un metodo **public double totaleAsporto ()** che restituisce il totale degli importi degli ordini da asporto.
- un metodo **public Personale getMigliore ()** che restituisce il cameriere (tra quelli nell'arraylist persone) che ha servito tavoli per il più alto importo complessivo di ordini a loro associati (senza contare gli ordini da asporto). Si assuma che i camerieri associati agli ordini si riferiscano allo stesso oggetto cui si riferiscono i camerieri presnti nell'arraylist persone.

Esercizio 3 (5 punti)

Dato un intero positivo x ed un array a di n interi positivi maggiori si 1, progettare un algoritmo di programmazione dinamica che ritorni il valore **true** se e solo se esistono o meno indici i_1, i_2,…, i_k (non necessariamente distinti) nell'array a tali che:

$$\prod_{j=1}^{k} a[i_j] = x$$

Esempio: Si consideri a= {3, 7, 5, 2). È possibile ottenere 12 con tre indici 0,3 e 3 (ovvero a[0]*a[3]*a[3]=12).

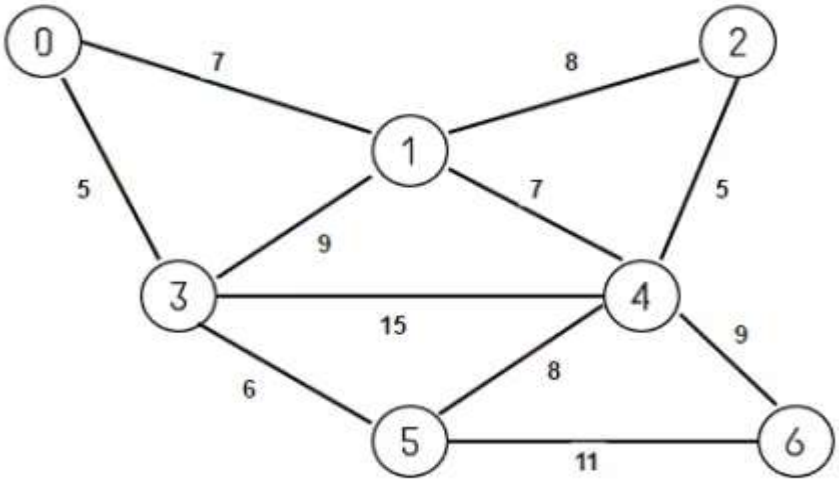
Suggerimento: indicare con F(i, x) la soluzione al problema che richiede di formare l'intero x utilizzando i primi i elementi dell'array a.

$$F(i,x) = \begin{cases} true & se\ x = 1 \\ false & se\ x \neq 1\ e\ i = 0 \\ ??\ or\ ?? & se\ x \neq 1\ e\ i > 0\ e\ x\ mod\ a[i-1] = 0 \\ ?? & altrimenti \end{cases}$$

Nella terza riga (caso ricorsivo) bisogna considerare due casi: il caso in cui l'i-esimo elemento di a (cioè a[i-1]) concorre (almeno una volta) a formare x, e il caso in cui non concorre; nella quarta riga (altro caso ricorsivo), poiché a[i-1] non è un divisore di x, c'è da analizzare solo il caso in cui si può ottenere x come prodotto dei precedenti elementi dell'array.
Si fornisca una implementazione ricorsiva in Java dell'algoritmo proposto.

Esercizio 4 (15 punti)

Si consideri il grafo non diretto in figura.



a. **[7 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **4**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.
b. **[8 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Floyd-Warshall** per i cammini minimi tra tutte le coppie, calcolando ad ogni passo la matrice delle distanze e la matrice dei padri π. Si dica se, nel grafo considerato, esiste, dal punto di vista computazionale, un modo più efficiente per risolvere il problema del calcolo dei cammini minimi tra tutte le coppie.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2019/2020

Appello del 18 febbraio 2020

Esercizio 1 [8 punti]

Si consideri il seguente programma Java:

```
class A {
	protected int n;

	public A(int n) {
		this.n=n;
	}

	public int metodo (int a) {
		return a+n;
	}
	public double metodo (double b) {
		return b+n+0.5;
	}
}

class B extends A {
	public B(int n) {
		super (2*n);
	}
	public int metodo (int a) {
		return a+n+5;
	}
	public double metodo (double b) {
		return b+n+2.3;
	}
}

public class MainClass {
	public static void main(String[] args) {
		A a = new A(1);
		B b = new B(2);
		A c = b;
		System.out.println (c.metodo(4));
		System.out.println (b.metodo(a.metodo(2.0)));
		System.out.println (b.metodo(a.metodo(2)));
		System.out.println (c.metodo(c.metodo(0.4)));
	}
}
```

Dire cosa stampa il programma, giustificando la risposta mostrando in particolare:

- le firme associate a tempo di compilazione e di esecuzione ad ogni chiamata di metodo
- l'evoluzione della memoria nella parte heap.

Esercizio 2 [12 punti]

Si scrivano le classi Java per gestire *il bilancio in un condominio*.
Si scriva una classe astratta **Spesa** con

- una variabile di istanza **descrizione** (tipo String, private);
- una variabile di istanza **importo** (tipo double, private);

Scrivere i seguenti metodi di istanza:

- un costruttore che crea una spesa dati **descrizione** e **importo**;
- metodi pubblici accessori **`getDescrizione()`**, **`getImporto()`**;
- metodo astratto **`getRiparto(Condomino c)`** che calcola l'ammontare che il condomino c deve pagare per la spesa.

La classe Spesa è estesa dalle sottoclassi **SpesaGenerale** e **SpesaScala,** per le quali vanno scritti opportunamente i costruttori che richiamano il costruttore della classe Spesa e va implementato il metodo **`getRiparto(Condomino c)`** tenendo conto del fatto che un'istanza di SpesaGenerale va ripartita secondo i millesimiProprietà, mentre un'istanza di SpesaScala secondo i millesimiScala (la formula per ripartire una spesa **s** per un condomino che ha **x** millesimi è $s\frac{x}{1000}$).

Si assuma, senza scrivere nulla, l'esistenza di una classe **Condomino** con

- una variabile di istanza **nome** (tipo String, final, private)
- una variabile di istanza **cognome** (tipo String, final, private)
- una variabile di istanza **millesimiProprietà** (tipo int, final, private)
- una variabile di istanza **millesimiScala** (tipo int, final, private)
- un costruttore che prende tutti e 4 i parametri sopra descritti;
- metodi pubblici accessori per tutti e 4 i parametri sopra descritti.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Versamento** con

- una variabile di istanza **eseguitoDa** (tipo Condomino, final, private)
- una variabile di istanza **importo** (tipo double, private);
- un costruttore che prende tutti e 2 i parametri sopra descritti;
- metodi pubblici accessori per tutti e 2 i parametri sopra descritti.

Si progetti infine una classe **Condominio** con

- una variabile di istanza **condomini** (tipo ArrayList<Condomino>, final, private);
- una variabile di istanza **spese** (tipo ArrayList<Spesa>, final, private);
- una variabile di istanza **entrate** (tipo ArrayList<Versamento>, final, private);

Implementare i seguenti metodi di istanza:

- un costruttore senza parametri che crea ArrayList vuoti per condomini, spese e entrate.
- un metodo **public double getDovuto (Condomino c)** che restituisce il totale degli importi dovuti dal condomino c per tutte le spese presenti nell'ArrayList spese.
- un metodo **public double getPagato (Condomino c)** che restituisce il totale degli importi pagati dal condomino c e registrati nell'ArrayList entrate.
- un metodo **public ArrayList<Condomino> getMorosi ()** che restituisce un ArrayList contenente tutti e soli i condomini che non hanno pagato a sufficienza per coprire i pagamenti dovuti.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Esercizio 3 (6 punti)

Dato un intero positivo x ed un array a di n interi positivi, la seguente ricorrenza ritorna il valore **true** se e solo se esistono o meno indici $i_1, i_2, \ldots, i_k$ (tra loro distinti) nell'array a tali che:

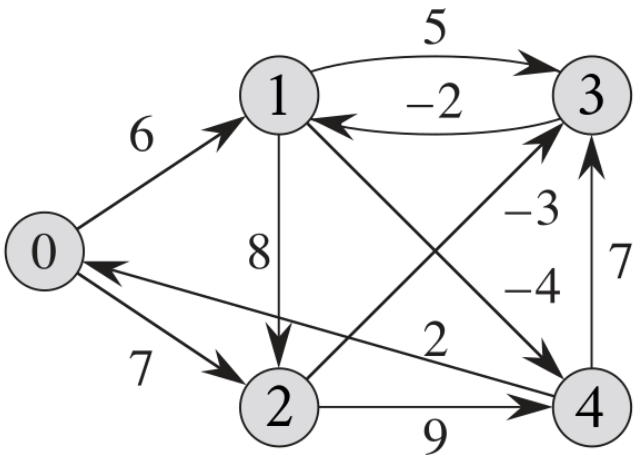
$$\sum_{j=1}^{k} a_{i_j} = x$$

In particolare, F(i,x) è la soluzione al problema che richiede di formare l'intero x utilizzando i primi i elementi dell'array a.

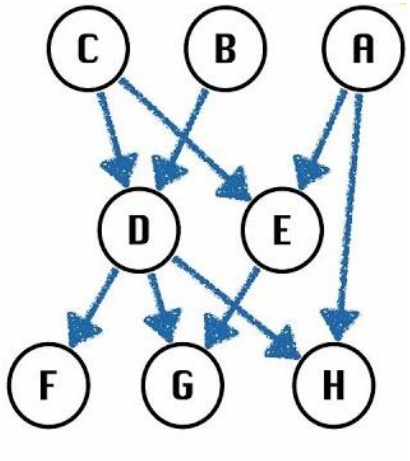
$$F(i,x) = \begin{cases} true & se\ x = 0 \\ false & se\ x > 0\ e\ i = 0 \\ F(i-1, x - a[i-1])\ or\ \ F(i-1,x) & altrimenti \end{cases}$$

a) **[3 punti]** Si fornisca una implementazione ricorsiva in Java dell'algoritmo proposto.
b) **[3 punti]** Si fornisca una implementazione ricorsiva con **memoization** in Java dello stesso algoritmo.

Esercizio 4 [14 punti]



a. **[7 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Bellman-Ford** per calcolare i cammini minimi a partire dalla sorgente **0.** Dire, giustificando la risposta, se il grafo possiede cicli di peso negativo.



b. **[7 punti]** Calcolare un ordinamento topologico usando il metodo della visita in profondità, evidenziando per ogni nodo i timestamp di inizio e fine visita.

Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2021/2022

Appello del 6 luglio 2022

Esercizio 1 [8 punti]

Si considerino le seguenti classi.

```
class A{
    int method(A a){
     return 10;
    }
    int method(B a){
     return 20;
    }
}
```

```
class B extends A{
    int method(A a){
     return 30;
    }
    int method(B a){
     return 40;
    }
 }
```

Si dica cosa viene stampato a video dal seguente codice:

```
      A a = new A();
      B b = new B();
      A c = new B();
      System.out.println(a.method(a));
      System.out.println(a.method(c));
      System.out.println(b.method(b));
      System.out.println(b.method(c));
      System.out.println(c.method(a));
      System.out.println(c.method(c));
```

Si giustifichi la risposta mostrando in particolare:

- le firme associate a tempo di compilazione e di esecuzione ad ogni chiamata di metodo
- l'evoluzione della memoria nella parte heap.

Esercizio 2 [6 punti)

a. Mostrare **l'heap di minimo** che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi **6, 32, 8, 45, 2, 80, 9, 7, 43** (mostrare l'heap che si ottiene dopo l'inserimento di ogni chiave);
b. Rimuovere per 3 volte la chiave minima dall'heap, mostrando l'heap che si ottiene dopo ogni estrazione;
c. Inserire nell'heap così ottenuto le chiavi **3, 72, 1.**

Regole per lo svolgimento della prova scritta:

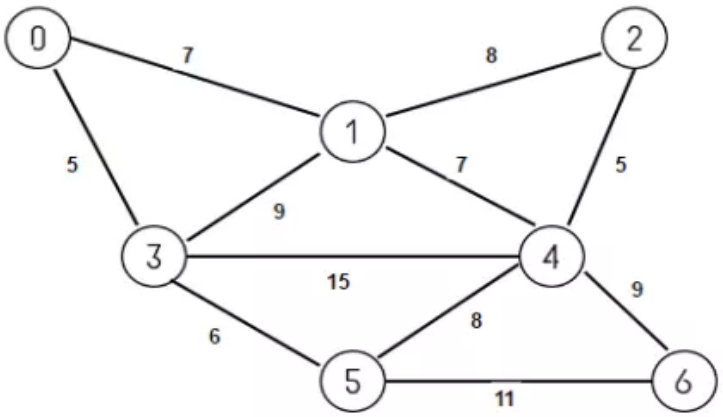
- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Esercizio 3 [9 punti]

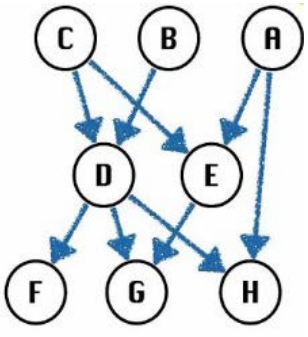
- Si scriva una classe astratta **Temperatura** per rappresentare valori di temperatura
- La classe deve avere un metodo **public abstract double toKelvin ();** che sarà implementato dalle classi che estendono **Temperatura** con lo scopo di fornire una valutazione della temperatura in gradi Kelvin.
- La classe deve inoltre implementare un metodo **public boolean LEQ (Temperatura b)** che ritorna true se e solo se la temperatura su cui è invocato è minore o uguale a quella di b. *[si sfrutti il metodo **toKelvin**]*
- *Si implementino infine le seguenti classi **concrete** che implementano* **Temperatura*, in ognuna delle quali bisogna implementare un costruttore che prende un double e il metodo toKelvin:***
    - La classe **TemperaturaCelsius** che possiede una variabile d'istanza di tipo double che memorizza la temperatura in gradi centigradi, ed il cui costruttore prende in input il valore della temperatura in gradi centigradi.
    - La classe **TemperaturaFahrenheit** che possiede una variabile d'istanza di tipo double che memorizza la temperatura in gradi Fahrenheit, ed il cui costruttore prende in input il valore della temperatura in gradi Fahrenheit.

[Si ricorda che:
°K = (°F + 459,67) / 1,8 *°K = °C + 273,15*
]

Esercizio 4 [11 punti]



a. **[6 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **6**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.



b. **[5 punti]** Calcolare un ordinamento topologico usando il metodo della visita in profondità, evidenziando per ogni nodo i timestamp di inizio e fine visita. Per effettuare la visita in profondità, si parta sempre dal nodo non ancora visitato identificato dalla lettera che viene prima nell'ordinamento alfabetico.

Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2021/2022

Appello del 12 settembre 2022

Esercizio 1 [5 punti]

Si considerino le seguenti classi.

```
class A{
    int method(A a){
     return 1;
    }
    int method(B a){
     return 2;
    }
}
```

```
class B extends A{
    int method(A a){
     return 3;
    }
}
```

Si dica cosa viene stampato a video dal seguente codice:

```
      A a = new A();
      B b = new B();
      A c = new B();
      System.out.println(b.method(b));
      System.out.println(b.method(c));
      System.out.println(c.method(a));
      System.out.println(c.method(c));
```

Si giustifichi la risposta mostrando in particolare le firme associate a tempo di compilazione e di esecuzione ad ogni chiamata di metodo.

Esercizio 2 [7 punti]

a. Mostrare **l'heap di massimo** che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi **6, 32, 8, 45, 2, 80, 9, 7, 43** (mostrare l'heap che si ottiene dopo l'inserimento di ogni chiave);

b. Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array **[6, 32, 8, 45, 2, 80, 9, 7, 43]** in modo non decrescente (compresa la fase iniziale di build-max-heap).

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Esercizio 3 [12 punti]

Si vogliono gestire, in Java, gli ordini di un negozio di fast food.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Prodotto** con

- una variabile di istanza **descrizione** (tipo String, private);
- una variabile di istanza **prezzoUnitario** (tipo double, private);

e i seguenti metodi di istanza:

- un costruttore che crea un oggetto dati **descrizione e prezzoUnitario**;
- metodi pubblici accessori **`getDescrizione() e getPrezzoUnitario()`**;

Si assuma, senza scrivere nulla, l'esistenza di una classe **ProdottoOrdinato** con

- una variabile di istanza **prodotto** (tipo Prodotto, private)
- una variabile di istanza **quantita** (tipo int, private)
- un costruttore che crea un oggetto dati **prodotto e quantita**;
- metodi pubblici accessori **`getProdotto() e getQuantita()`**.

Si scriva una classe astratta **Ordine** con

- una variabile di istanza **comanda** (tipo ArrayList<ProdottoOrdinato>, final, private);

e i seguenti metodi di istanza:

- un costruttore senza parametri che crea una comanda vuota associandola all'ordine;
- un metodo pubblico **`void aggiungi(Prodotto p, int quantita)`** che aggiunge alla comanda il prodotto indicato con la relativa quantità.
- metodo pubblico astratto **`double getTotale()`**che restituisce l'importo totale dell'ordine.

La classe astratta **Ordine** è estesa dalle sottoclassi (non astratte) **OrdineTavolo, OrdineAsporto e OrdineDelivery.**

Si tenga conto del fatto che la classe **OrdineTavolo** deve avere:

- una variabile di istanza **tavolo** (tipo String, private) che contiene l'identificativo del tavolo;
- una variabile di istanza **coperti** (tipo int, private) che contiene il numero di coperti.

La classe **OrdineAsporto** deve avere:

- una variabile di istanza **nominativo** (tipo String, private) che contiene il nominativo della persona che ha effettuato l'ordine d'asporto.

La classe **OrdineDelivery** deve avere:

- una variabile di istanza **destinazione** (tipo String, private) che contiene il nominativo e l'indirizzo di destinazione dell'ordine.

Per tutte e tre le sottoclassi vanno scritti opportunamente i costruttori che richiamano opportunamente il costruttore della classe **Ordine** e va implementato il metodo **`double getTotale(),`** tenendo conto del fatto che ogni coperto ha un costo di € 1,80 (nel caso di OrdineTavolo), la consegna ha un costo di € 4,00 (nel caso di OrdineDelivery) mentre non c'è nessun costo aggiuntivo per gli ordini da asporto.

Si progetti infine una classe **FastFood** con

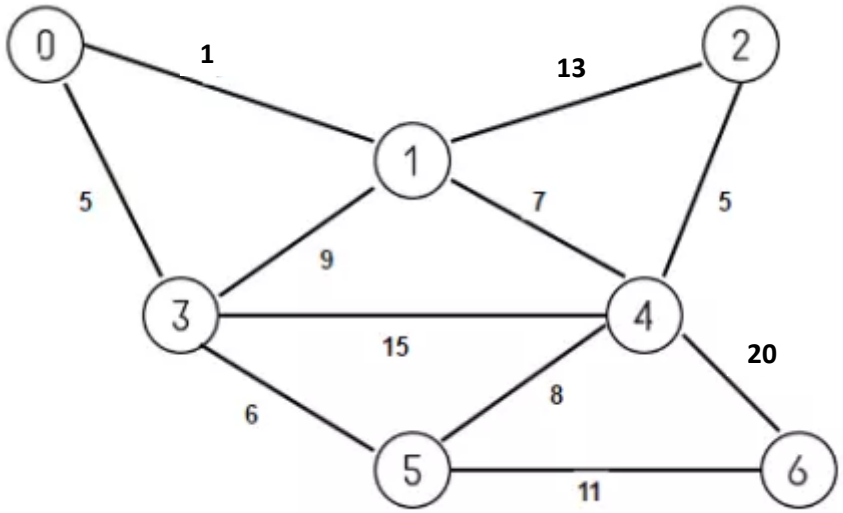
- una variabile di istanza **prodotti** (tipo ArrayList<Prodotto>, final, private);
- una variabile di istanza **ordini** (tipo ArrayList<Ordine>, final, private);

Implementare i seguenti metodi di istanza:

- un costruttore senza parametri che crea arraylist vuoti per i prodotti e gli ordini.
- un metodo **public double totaleAsporto ()** che restituisce il totale degli importi degli ordini da asporto.
- un metodo **public Prodotto piuVenduto ()** che restituisce il prodotto (tra quelli nell'arraylist prodotti) che è stato venduto di più negli ordini (come somma di tutte le quantità a lui associate nei vari ordini). *(Si assuma che i prodotti associati agli ordini si riferiscano allo stesso oggetto cui si riferiscono i prodotti presenti nell'arraylist prodotti.)*

Esercizio 4 [10 punti]

Si consideri il grafo non diretto in figura.



a. **[5 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente, partendo dal nodo sorgente **3**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

b. **[5 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **3**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 4 luglio 2017

Esercizio 1 (9 punti)

Cosa stampa il seguente programma Java?

```
class A {
	private int n;
	public A (int n){
		this.n=n;
	}
	public int getN(){
		return n;
	}
	void metodo (A a, int n){
		System.out.println ("metodo 1; " + n);
	}
	void metodo (B a, int n){
		System.out.println ("metodo 2; " + n);
	}
}

class B extends A{
	public B(int n){
		super (2*n);
	}
	void metodo (A a, int n){
		System.out.println ("metodo 3; " + n);
	}

}

public class MainClass {
	public static void main(String[] args) {
		A a = new A(1);
		B b = new B(2);
		A ab = new B(3);
		ab.metodo(a, ab.getN());
		ab.metodo(b, b.getN());
		ab.metodo(ab, a.getN());
	}
}
```

Esercizio 2 (10 punti)

Un array di **char** rappresenta in modo naturale una stringa. Ad esempio, l'array **{'c', 'a', 'r'}** rappresenta la stringa **"car"**.
Scrivere un metodo
`static boolean palindromo (char[] s)`
che, presi come parametro un array di char, restituisce true se e solo se la stringa rappresentata da **s** è palindroma, ovvero può essere letta indifferentemente da sinistra verso destra o da destra verso sinistra.
Se s vale *null*, viene restituito *false*.
Ad esempio se **s={'a', 't', 'e', 'l', 'e', 't', 'a'}**, viene restituito true.
Non è permesso usare nessun metodo/operatore della classe **String**.

Esercizio 3 (15 punti)

Si progetti una classe **Cliente** con

- una variabile di istanza **name** (tipo String, final, private)
- una variabile di istanza **surname** (tipo String, final, private).

La classe deve avere:

- un costruttore che crea un Cliente dati **nome** e **cognome**;
- metodi pubblici accessori **`getName()`**, **`getSurname()`**;
- metodo pubblico **`boolean equals (Cliente c)`** che restituisce true se e solo se **this** e **c** hanno name e surname uguali (secondo l'equals tra stringhe).

Si progetti una classe **ListaDiAttesa** con

- una variabile di istanza **list** di tipo ArrayList<Cliente>, private
  La classe deve avere:
- un costruttore che crea una lista di attesa vuota (creando l'arrayList **list** vuoto)
- un metodo **`public void add (Cliente c)`** che aggiunge alla fine della lista di attesa il cliente **c** (se c è già presente in lista, non viene aggiunto nuovamente; per effettuare il controllo, si sfrutti il metodo *equals* scritto nella classe Cliente)
- un metodo **`public Cliente extract()`** che elimina dalla lista il primo cliente in attesa e lo restituisce.
- un metodo **`public boolean isWaiting(Cliente c)`** che restituisce true se e solo se il cliente **c** è presente nella lista di attesa (si sfrutti il metodo *equals* scritto nella classe Cliente)

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Le risposte al primo esercizio devono essere date direttamente nel riquadro di questo foglio.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 7 febbraio 2017

Esercizio 1 (9 punti)

Cosa stampa il seguente programma Java?

```
class A {
	private double x;
	public A (double x){
		this.x=x;
	}
	public double getX(){
		return x;
	}
	void metodo (A a, double x){
		System.out.println ("metodo 1; " + x);
	}
	void metodo (B a, double x){
		System.out.println ("metodo 2; " + x);
	}
}

class B extends A{
	public B(double x){
		super (3+x);
	}
	void metodo (A a, double x){
		System.out.println ("metodo 3; " + x);
	}
	void metodo (B a, double x){
		System.out.println ("metodo 4; " + x);
	}
}

public class MainClass {
	public static void main(String[] args) {
		A a = new A(2.5);
		B b = new B(4.0);
		A ab = new B(7.0);
		a.metodo(ab, b.getX());
		ab.metodo(b, a.getX());
		ab.metodo(ab, ab.getX());
	}
}
```

Esercizio 2 (10 punti)

Scrivere un metodo

`static int[] espandi (int[] elementi, int[] molteplicita)`

che, presi come parametro due array di numeri interi *della stessa lunghezza n*, crea e restituisce un array che contenga, per ogni i=0,…,n-1, molteplicita[i] copie consecutive dell'elemento elementi[i] (se molteplicita[i] non è positivo elementi[i] non viene inserito nell'array). Se **elementi** o **molteplicita** vale **null**, oppure se non hanno la stessa lunghezza, viene restituito *null*.

Ad esempio se **elementi={1, 3, -4, 5, 6}**, e **molteplicita={2, 1 , 3, 0, -5}**, viene creato e restituito l'array **{1, 1, 3, -4, -4, -4}**.

Esercizio 3 (15 punti)

Si progetti una classe **Statino** con

- una variabili di istanza **matricola** (tipo int, final, private)
- una variabile di istanza **voto** (tipo int, final, private).
- una variabile di istanza **lode** (tipo boolean, final, private).

La classe deve avere:

- un costruttore che crea uno Statino dati **matricola** e **voto** (se voto è minore di 0 viene impostato a 0; se è maggiore di 30 viene impostato a 30 e viene assegnata la lode);
- metodi pubblici accessori **`getVoto()`**, **`getMatricola()`**, **`getLode()`**;
- un metodo pubblico **`toString()`** che restituisce la descrizione nel seguente modo (nell'esempio seguente il campo matricola è 31000000, il campo voto è 30 e il campo lode è true):

```
studente 31000000 voto 30 e lode
```

Si progetti una classe **Appello** con

- una variabile di istanza **statini** di tipo ArrayList<Statino> , private

La classe deve avere:

- un costruttore che crea un appello con nessuno statino (creando l'arrayList statini vuoto)
- un metodo **`public int contoSufficienti()`** che restituisce il numero di studenti che hanno riportato un voto maggiore o uguale a 18
- un metodo **`public int[] estraiSufficienti()`** che, facendo uso del metodo contoSufficienti, crea e restituisce un array contenente tutte le matricole che hanno riportato un voto sufficiente all'appello.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Le risposte al primo esercizio devono essere date direttamente nel riquadro di questo foglio.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Programmazione

Appello del 10/07/2012

Esercizio 1 (8 punti)

a. (4 punti) Scrivere l'output del seguente programma Java

```
public class Main {
public static void main(String[] args) {
        System.out.println ( enigma (10,0) );
        System.out.println ( enigma (3,1) );
        System.out.println ( enigma (4,-3) );
        System.out.println ( enigma (-2,10) );
    }


public static int enigma (int x, int y){
      if (y< 0) return enigma(-x, -y);
      if (y==0) return 0;
      if (y==1) return x;
      return  x + enigma(x, y-1);
}

}
```

b. (4 punti) Riscrivere il metodo enigma in modo equivalente **non** ricorsivo.

Esercizio 2 (8 punti)

Un array si dice **fortemente crescente** se ogni suo elemento (dal secondo all'ultimo) è almeno pari alla somma di tutti gli elementi che lo precedono.

Scrivere un metodo **public static boolean fc (int [] A)** che preso in input un array A di numeri interi, restituisce ***true*** se e solo se **A** è "fortemente crescente"

Ad esempio se A=[1,1,3,5,15] deve restituire **true**, mentre se A=[1,2,3,5,6] deve restituire **false**.

Attenzione:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**

SECONDA PARTE

Esercizio 3 (8 punti)

Si implementino in Java le classi **Prodotto** e **Carrello**.
La *classe* **Prodotto** ha i seguenti attributi:

- **codice** (una **stringa**)
- **descrizione** (una **stringa**)
- **prezzoUnitario** (un **intero** che esprima il prezzo in centesimi di euro)

ed i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Prodotto** assegnando descrizione, codice e prezzo unitario (si assuma che il prezzo unitario sia fornito al costruttore in euro, come un parametro di tipo *float*).
- metodi "*get*" per tutti gli attributi, cioè metodi che restituiscono i valori di ciascun attributo (anche in questo caso, il prezzo deve essere restituito in euro, di tipo *float*) ;
- metodo **public boolean equals (Object o)** che restituisce true se e solo se due prodotti hanno lo stesso codice e lo stesso prezzo unitario (indipendentemente dalla descrizione).

La *classe* **Carrello** ha i seguente attributi:

- **nome** (una Stringa)
- **prodotti** (un arrayList di Prodotto)
- **quantita** (un arrayList di Integer)

*Gli arraylist **prodotti** e **quantita** sono pensati per essere arrayList "paralleli", tali che l'arrayList **quantita** in posizione **i** contiene il numero di prodotti presenti nel carrello corrispondenti alla posizione **i** dell'arrayList **prodotti**.*
La *classe* **Carrello** ha i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Carrello** prendendo in input il nome del carrello
- metodo **public void *addProdotto (Prodotto p, int n),*** *che (sfruttando il metodo **equals** della classe prodotto)*
  - aggiunge il prodotto p all'arrayList **prodotti** (e corrispondentemente la quantità **n** all'arrayList **quantita**), se nel carrello non è già presente il prodotto in questione.
  - altrimenti aggiorna la quantita corrispondnete al prodotto **p** incrementandola di **n**.
- metod**o public String getNome ()** che restituisce il nome del carrello.
- metodo **public float getTotale ()** che restituisce il prezzo totale in euro dei prodotti presenti nel carrello.

Esercizio 4 (8 punti)

- Si scriva una classe astratta **Funzione** per rappresentare funzioni aventi come dominio l'insieme dei numeri interi
- La classe deve avere un metodo astratto **public abstract double calcola (int n);** che sarà implementato dalle classi che estendono **Funzione** con lo scopo di calcolare il valore della funzione sull'intero **n**.
- La classe deve inoltre implementare un metodo **public int min (int left, int right)** che restituisce uno dei valori del dominio della funzione che realizzano il **minimo** della funzione stessa nell'intervallo **[left, right]**. *[si sfrutti il metodo* **calcola***]*
- *Si implementino infine le seguenti classi **concrete** che estendono* **Funzione***,* ognuna con le opportune variabili di istanza, ed in ognuna delle quali bisogna implementare un ***costruttore*** ed **il** metodo **calcola***:*
  - La classe **IperboleEquilatera**, il cui costruttore prende quattro double (a,b,c,d) tali che la funzione rappresentata è **f(n)=(a n + b)/(c n + d)**
  - La classe **FunzionePolinomiale**, il cui costruttore prende un array A di **d+1** double (dove d è il grado del polinomio) tali che la funzione rappresentata è
    $f(n)=A[0] + A[1]\ n + A[2]\ n^2 + \ldots + A[i]\ n^i + \ldots + A[d]\ n^d$

Programmazione

Appello dell'11/09/2012

Esercizio 1 (8 punti)

a. (4 punti) Scrivere l'output del seguente programma Java

```
public class Main {
public static void main(String[] args) {
        System.out.println ( enigma (15,0) );
        System.out.println ( enigma (4,1) );
        System.out.println ( enigma (-4,-3) );
        System.out.println ( enigma (3,-100) );
    }


public static int enigma (int x, int y){
      if (y==0) return x;
      if (y>0) return enigma(x+1, y-1);
      else return  enigma(x-1, y+1);
}

}
```

b. (4 punti) Riscrivere il metodo enigma in modo equivalente **non** ricorsivo.

Esercizio 2 (8 punti)

Scrivere un metodo di classe ***search*** che prende in input due array A e B di int, e restituisce ***true*** se e solo se l'array B ha lunghezza minore o uguale dell'array A, ed è possibile individuare un sottoarray (di elementi consecutivi) di A uguale all'array B.

Per esempio, se A=`{1,55,83,21,13,43,10,56};` e B=`{21,13,43};` deve essere ritornato ***true***.

Attenzione:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**

SECONDA PARTE

Esercizio 3 (8 punti)

Si implementino in Java le classi **Autore** e **Libro**.
La *classe* **Autore** ha i seguenti attributi:

- **nome** (una stringa)
- **cognome** (una stringa)

ed i seguenti metodi:

- *costruttore* che crea un oggetto della *classe* **Autore** assegnando nome e cognome.
- metodi "*get*" per tutti gli attributi, cioè metodi che restituiscono i valori di ciascun attributo;
- metodo "toString"

La *classe* **Libro** ha i seguente attributi:

- **titolo** (una Stringa)
- **autori** (un ArrayList di Autore)

ed i seguenti metodi:

- *costruttore* che crea un oggetto della *classe* **Libro** senza autori (ma creando opportunamente un arrayList vuoto)
- metodo **public *addAutore (Autore a),*** che aggiunge l'autore a agli autori del libro
- metodo **public String getTitolo ()** che restituisce il titolo del libro.
- metodo **public int getNAutori ()** che restituisce il numero di autori del libro.
- metodo **public Autore getAutore (int k)** che restituisce il k-esimo autore del libro (k varia tra 1 e getNAutori()). Se k non è un intero valido viene restituito null.
- metodo "toString", che sfrutta il metodo toString della classe Libro

Scrivere infine nel main il codice che crea un libro a vostro piacimento e vi aggiunge almeno due autori.

Esercizio 4 (8 punti)

- Si scriva una classe astratta **Operazione** per rappresentare semplici operazioni tra numeri interi
- La classe deve avere un metodo **public abstract int risultato ();** che sarà implementato dalle classi che estendono **Operazione** con lo scopo di calcolare il risultato dell'operazione.
- La classe deve inoltre sovrascrivere il metodo **toString** in modo che stampi il risultato dell'operazione. *[si sfrutti il metodo **risultato**]*
- *Si implementino infine le seguenti classi **concrete** che estendono* **Operazione***,* in ognuna delle quali bisogna inserire le necessarie **variabili di istanza (gli operandi)** ed implementare un ***costruttore*** e il metodo **risultato***:*
    - La classe **addizione**, il cui costruttore prende i due interi da addizionare.
    - La classe **sottrazione**, il cui costruttore prende i due interi da sottrarre.
    - La classe **moltiplicazione**, il cui costruttore prende i due interi da moltiplicare.

Programmazione

Appello del 19/06/2012

Esercizio 1 (8 punti)

a. (4 punti) Scrivere l'output del seguente programma Java

```
public class Main {
public static void main(String[] args) {
        System.out.println ( enigma (10,0) );
        System.out.println ( enigma (3,1) );
        System.out.println ( enigma (4,3) );
        System.out.println ( enigma (2,10) );
    }


public static int enigma (int x, int y){
      if (y==0) return 1;
      return  x * enigma(x,y-1);
}

}
```

b. (4 punti) Riscrivere il metodo enigma in modo equivalente **non** ricorsivo.

Esercizio 2 (8 punti)

Scrivere un metodo **public static boolean monotono (int [] A)** che preso in input un array A di numeri interi, restituisce ***true*** se e solo se **A** verifica le seguenti proprietà:

- Ha lunghezza pari
- La sua prima metà è ordinata in modo **strettamente crescente**
- La sua seconda metà è ordinata in modo **strettamente decrescente**.

Ad esempio se A=[1,2,20,5] deve restituire **true**,
mentre se A=[1,2,1,5,6] o se A=[1,2,2,10] deve restituire **false**.

Attenzione:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**

SECONDA PARTE

Esercizio 3 (8 punti)

Si implementino in Java le classi **Prodotto** e **Magazzino**.
La *classe* **Prodotto** ha i seguenti attributi:

- **codice** (una **stringa**)
- **descrizione** (una **stringa**)

ed i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Prodotto** assegnando descrizione e codice
- metodi "*get*" per tutti gli attributi, cioè metodi che restituiscono i valori di ciascun attributo;
- metodo **public boolean equals (Object o)** che restituisce true se e solo se due prodotti hanno lo stesso codice (indipendentemente dalla descrizione).

La *classe* **Magazzino** ha i seguente attributi:

- **nome** (una Stringa)
- **prodotti** (un arrayList di Prodotto)
- **quantita** (un arrayList di Integer)

*Gli arraylist **prodotti** e **quantita** sono pensati per essere arrayList "paralleli", tali che l'arrayList **quantita** in posizione **i** contiene il numero di prodotti presenti nel magazzino corrispondenti alla posizione **i** dell'arrayList **prodotti**.*
La *classe* **Magazzino** ha i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Magazzino** prendendo in input il nome del magazzino
- metodo **public void *addProdotto (Prodotto p, int n),*** *che (sfruttando il metodo **equals** della classe prodotto)*
  - aggiunge il prodotto p all'arrayList **prodotti** (e corrispondentemente la quantità **n** all'arrayList **quantita**), se nel magazzino non è già presente il prodotto in questione.
  - altrimenti aggiorna la quantità corrispondente al prodotto **p** incrementandola di **n**.
- metod**o public String getNome ()** che restituisce il nome del magazzino.
- metodo **public float getQuantita (Prodotto p)** che *(sfruttando il metodo **equals** della classe prodotto)* restituisce la quantità di prodotto **p** presente nel magazzino.

Esercizio 4 (8 punti)

- Si scriva una classe astratta **Funzione** per rappresentare funzioni aventi come dominio l'insieme dei numeri interi
- La classe deve avere un metodo astratto **public abstract double calcola (int n);** che sarà implementato dalle classi che estendono **Funzione** con lo scopo di calcolare il valore della funzione sull'intero **n**.
- La classe deve inoltre implementare un metodo **public int max (int left, int right)** che restituisce uno dei valori del dominio della funzione che realizzano il **massimo** della funzione stessa nell'intervallo **[left, right]**. *[si sfrutti il metodo* **calcola***]*
- *Si implementino infine le seguenti classi **concrete** che estendono* **Funzione***,* ognuna con le opportune variabili di istanza, ed in ognuna delle quali bisogna implementare un ***costruttore*** ed **il** metodo **calcola***:*
  - La classe **IperboleEquilatera**, il cui costruttore prende quattro double (a,b,c,d) tali che la funzione rappresentata è **f(n)=(a n + b)/(c n + d)**
  - La classe **FunzionePolinomiale**, il cui costruttore prende un array A di **d+1** double (dove d è il grado del polinomio) tali che la funzione rappresentata è
    $f(n)=A[0] + A[1]\ n + A[2]\ n^2 + \ldots + A[i]\ n^i + \ldots + A[d]\ n^d$

Programmazione

Appello del 20/12/2011 – Compito n° 1

Esercizio 1 (8 punti)

a. (4 punti) Scrivere l'output del seguente programma Java

```
public class Main {
public static void main(String[] args) {
        int [] A = {55,83,21,16,43,10,56};
        stampa(A);
        enigma(A);
        stampa(A);
    }

public static void stampa (int [] A ){
        for (int x:A) {
            System.out.print (x + " ");
        }
        System.out.println();
    }

public static void enigma (int A[]){
      int i=0;
      while (i < A.length - 1){
         if (A[i] > A[i+1] && A[i]%2==1) A[i]=A[i+1];
            else A[i+1]=A[i]/2;
         i++;
        }
    }
}
```

b. (4 punti) Riscrivere la procedura enigma in modo equivalente senza far uso né del costrutto while né del costrutto for

Esercizio 2 (7 punti)

Scrivere un metodo **public static ArrayList<Integer> estrai (int [] A, int k)** che preso in input un array di numeri interi ed un intero **k**, crea e restituisce un ArrayList di Integer in cui sono presenti tutti e soli gli interi presenti in **A** esattamente **k** volte, nello stesso ordine della loro prima occorrenza in **A**.

Ad esempio, se **A=[3,1,5,3,9,1]** e **k=2**, il metodo deve restituire un arrayList contenente gli **Integer** 3 ed 1.

SECONDA PARTE

Esercizio 3 (5 punti)

Si implementino in Java le classi **Autore** e **Libro**.
La *classe* **Autore** ha i seguenti attributi:

- **nome** (una stringa)
- **cognome** (una stringa)

ed i seguenti metodi:

- *costruttore* che crea un oggetto della *classe* **Autore** assegnando nome e cognome.
- metodi "*get*" per tutti gli attributi, cioè metodi che restituiscono i valori di ciascun attributo;
- metodo "toString"

La *classe* **Libro** ha i seguente attributi:

- **titolo** (una Stringa)
- **autori** (un ArrayList di Autore)

ed i seguenti metodi:

- *costruttore* che crea un oggetto della *classe* **Libro** senza autori (ma creando opportunamente un ArrayList vuoto)
- metodo **public *addAutore (Autore a),*** che aggiunge l'autore a agli autori del libro
- metodo **public String getTitolo ()** che restituisce il titolo del libro.
- metodo **public int getNAutori ()** che restituisce il numero di autori del libro.
- metodo **public Autore getAutore (int k)** che restituisce il k-esimo autore del libro (k varia tra 1 e getNAutori()). Se k non è un intero valido viene restituito null.
- metodo "toString", che sfrutta il metodo toString della classe Libro

Scrivere infine nel main il codice che crea un libro a vostro piacimento e vi aggiunge almeno due autori.

Esercizio 4 (5 punti)

Si considerino le seguenti classi.

```
class A{
    int method(A a){
     return 1;
    }
    int method(B a){
     return 2;
    }
}

class B extends A{
    int method(A a){
     return 3;
    }
    int method(B a){
     return 4;
    }

 }
```

```
class C extends B{
    int method(A a){
     return 5;
    }
    int method(B a){
     return 6;
    }

 }
```

Si dica cosa viene stampato a video dal seguente codice, segnalando eventualmente eventuali errori a tempo di compilazione e **giustificando adeguatamente la risposta** indicando anche per ogni invocazione di metodo il metodo selezionato a tempo di compilazione (guardando solo i tipi statici):

```
      A a = new A();
      B b = new B();
      C c = new C();
      A d = new B();
      A e = new C();
      B f = new C();
      System.out.println(a.method(f));
      System.out.println(d.method(a));
      System.out.println(e.method(b));
      System.out.println(e.method(c));
      System.out.println(f.method(d));
      System.out.println(f.method(e));
```

Esercizio 5 (5 punti)

Data la seguente interfaccia:

```
interface Figura{
   double getArea();
   double getPerimetro();
}
```

scrivere le classi **Quadrato** e **TriangoloRettangolo** che implementano l'interfaccia Figura.
Ciascuna classe deve prevedere le opportune variabili di istanza, il costruttore e deve naturalmente implementare i metodi dell'interfaccia.
In particolare, la classe Quadrato deve avere un costruttore che prende in input la misura del lato, mentre la classe TriangoloRettangolo un costruttore che prende in input la misura dei 2 cateti.

[può essere utile utilizzare il metodo Math.sqrt(double x) che restituisce la radice quadrata di x]

ESERCIZIO FACOLTATIVO:

Esercizio 6 (3 punti)
Aggiungere alla classe Pila (implementata qui sotto), un metodo ricorsivo **public int contaPari ()** che restituisce il numero di interi pari presenti nella pila.

```
public class Nodo {

      private int elemento;
      private Nodo next;

      public Nodo
(int elemento, Nodo next) {
            this.elemento=elemento;
            this.next=next;
      }

      public int getElemento() {
            return elemento;
      }

      public Nodo getNext() {
            return next;
      }

      public void setNext(Nodo n) {
            next = n;
      }
}
```

```
public class Pila {
      private Nodo topElement;

      public Pila() {
            topElement=null;
      }

      public boolean isEmpty() {
            return topElement==null;
      }

      public void push(int elemento) {
            topElement = new
Nodo(elemento,topElement);
      }

      public int pop() {
  if(isEmpty())return(-1);

int top = topElement.getElemento();
topElement = topElement.getNext();
            return top;
      }
}
```

Attenzione:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO.
- Durante la prova scritta non è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**

Programmazione

Appello del 22/05/2012

Esercizio 1 (8 punti)

a. (4 punti) Scrivere l'output del seguente programma Java

```
public class Main {
public static void main(String[] args) {
        int [] A = {10,9,8,7,6,5,4,3,2,1};
        stampa(A);
        enigma(A);
        stampa(A);
    }

public static void stampa (int [] A ){
        for (int x:A) {
            System.out.print (x + ",");
        }
        System.out.println();
    }

public static void enigma (int A[]){
      for (int i=A.length -2; i>=0; i--)
      {
          A[i] += A[i+1];
      }
}
}
```

b. (4 punti) Riscrivere il metodo enigma in modo equivalente (ossia in modo che QUALSIASI sia l'array A lo modifichi nello stesso modo) facendo uso esclusivamente del costrutto iterativo **do while**.

Esercizio 2 (8 punti)

Scrivere un metodo **public static boolean monotono (int [] A)** che preso in input un array A di numeri interi, restituisce ***true*** se e solo se A è ordinato in modo non decrescente oppure non crescente.

Ad esempio se A=[1,2,2,4,5] o se A=[5,4,3,3,2] deve restituire true,
mentre se A=[1,2,1,5,6] deve restituire false.

Attenzione:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**

SECONDA PARTE

Esercizio 3 (8 punti)

Si implementino in Java le classi **Prodotto** e **Carrello**.
La *classe* **Prodotto** ha i seguenti attributi:

- **codice** (una **stringa**)
- **descrizione** (una **stringa**)
- **prezzoUnitario** (un **intero** che esprima il prezzo in centesimi di euro)

ed i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Prodotto** assegnando descrizione, codice e prezzo unitario (si assuma che il prezzo unitario sia fornito al costruttore in euro, come un numero in virgola mobile).
- metodi "*get*" per tutti gli attributi, cioè metodi che restituiscono i valori di ciascun attributo (anche in questo caso, il prezzo deve essere restituito in euro, come un numero in virgola mobile) ;
- metodo **public boolean equals (Object o)** che restituisce true se e solo se due prodotti hanno lo stesso codice e lo stesso prezzo unitario (indipendentemente dalla descrizione).

La *classe* **Carrello** ha i seguente attributi:

- **nome** (una Stringa)
- **prodotti** (un arrayList di Prodotto)
- **quantita** (un arrayList di Integer)

*Gli arraylist **prodotti** e **quantita** sono pensati per essere arrayList "paralleli", tali che l'arrayList **quantita** in posizione **i** contiene il numero di prodotti presenti nel carrello corrispondenti alla posizione **i** dell'arrayList **prodotti**.*
La *classe* **Carrello** ha i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Carrello** prendendo in input il nome del carrello
- metodo **public void *addProdotto (Prodotto p, int n),*** *che (sfruttando il metodo **equals** della classe prodotto)*
  - aggiunge il prodotto p all'arrayList **prodotti** (e corrispondentemente la quantità **n** all'arrayList **quantita**), se nel carrello non è già presente il prodotto in questione.
  - altrimenti aggiorna la quantita corrispondnete al prodotto **p** incrementandola di **n**.
- metod**o public String getNome ()** che restituisce il nome del carrello.
- metodo **public float getTotale ()** che restituisce il prezzo totale in euro dei prodotti presenti nel carrello.

Esercizio 4 (8 punti)

- Si scriva una classe astratta **Funzione** per rappresentare funzioni sul piano cartesiano
- La classe deve avere un metodo astratto **public abstract double calcola (double x);** che sarà implementato dalle classi che estendono **Funzione** con lo scopo di calcolare il valore della funzione nel punto **x**.
- La classe deve inoltre implementare un metodo **public boolean LEq (Funzione f, double x)** che ritorna ***true*** se e solo se il valore della funzione su cui è invocato calcolata nel punto **x** è minore o uguale al valore della funzione **f** calcolata sempre in **x** *[si sfrutti il metodo* **calcola***]*
  Ad esempio, se f e g sono due funzioni, f.LEq(g,3) deve ritornare true se e solo se f(3)≤g(3)
- *Si implementino infine le seguenti classi **concrete** che estendono* **Funzione***,* ognuna con le opportune variabili di istanza, ed in ognuna delle quali bisogna implementare un ***costruttore*** ed **il** metodo **calcola***:*
  - La classe **IperboleEquilatera**, il cui costruttore prende quattro double (a,b,c,d) tali che la funzione rappresentata è **f(x)=(ax+b)/(cx+d)**
  - La classe **FunzionePolinomiale**, il cui costruttore prende un array A di **d+1** double (dove d è il grado del polinomio) tali che la funzione rappresentata è
    $f(x)=A[0] + A[1]\,x + A[2]\,x^2 + \ldots + A[i]\,x^i + \ldots + A[d]\,x^d$

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 22 settembre 2017

Esercizio 1 (9 punti)

Cosa stampa il seguente programma Java?

```
class A {
	private int n;
	public A (int n){
		this.n=n;
	}
	public int getN(){
		return n;
	}
	void metodo (A a, int n){
		System.out.println ("metodo 1; " + n);
	}
	void metodo (B a, int n){
		System.out.println ("metodo 2; " + n);
	}
}

class B extends A{
	public B(int n){
		super (n+2);
	}
	void metodo (A a, int n){
		System.out.println ("metodo 3; " + n);
	}

}

public class MainClass {
	public static void main(String[] args) {
		A a = new A(2);
		B b = new B(3);
		A ab = new B(4);
		ab.metodo(a, ab.getN());
		ab.metodo(b, b.getN());
		ab.metodo(ab, a.getN());
	}
}
```

Esercizio 2 (10 punti)

Un array di **char** rappresenta in modo naturale una stringa. Ad esempio, l'array **{'c', 'a', 'r'}** rappresenta la stringa **"car"**.
Scrivere un metodo

`static boolean palindromoDispari(char[] s)`

che, presi come parametro un array di char, restituisce true se e solo se la stringa rappresentata da **s** ha lunghezza dispari ed è palindroma, ovvero può essere letta indifferentemente da sinistra verso destra o da destra verso sinistra.
Se s vale *null*, viene restituito *false*.
Ad esempio se **s={'a', 't', 'e', 'l', 'e', 't', 'a'}**, viene restituito true.
Non è permesso usare nessun metodo/operatore della classe **String**.

Esercizio 3 (15 punti)

Si progetti una classe **Cliente** con

- una variabile di istanza **name** (tipo String, final, private)
- una variabile di istanza **surname** (tipo String, final, private).

La classe deve avere:

- un costruttore che crea un Cliente dati **nome** e **cognome**;
- metodi pubblici accessori **`getName()`**, **`getSurname()`**;
- metodo pubblico **`boolean equals(Cliente c)`** che restituisce true se e solo se **this** e **c** hanno name e surname uguali (secondo l'equals tra stringhe).

Si progetti una classe **ListaDiAttesa** con

- una variabile di istanza **list** di tipo ArrayList<Cliente>, private

La classe deve avere:

- un costruttore che crea una lista di attesa vuota (creando l'arrayList **list** vuoto)
- un metodo **`public boolean isWaiting(Cliente c)`** che restituisce true se e solo se il cliente **c** è presente nella lista di attesa (si sfrutti il metodo *equals* scritto nella classe Cliente)
- un metodo **`public void add(Cliente c)`** che aggiunge alla fine della lista di attesa il cliente **c** (se c è già presente in lista, non viene aggiunto nuovamente: per effettuare il controllo, si sfrutti il metodo *isWaiting*)
- un metodo **`public Cliente[] extract(int n)`** che elimina dalla lista i primi n clienti in attesa e li restituisce in un array (la lunghezza dell'array da restituire deve essere il minimo tra n e il numero di elementi presenti in lista).

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Le risposte al primo esercizio devono essere date direttamente nel riquadro di questo foglio.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 22 Dicembre 2016

Compito n° 1

Prima parte

Esercizio 1 (10 punti)

Scrivere un metodo

`public static int contaOccorrenze (int[] a, int n)`

che, preso come parametro un array a di numeri interi e un intero n, senza modificare il contenuto dell'array, restituisce il numero di occorrenze dell'intero n all'interno dell'array a. Se **a** vale *null*, viene restituito *0*.
Ad esempio, se a= [10, 6, 5, 4, 5, 6] e n=5 deve essere restituito 2.

Esercizio 2 (10 punti)

Scrivere un metodo

`public static boolean primaOccorrenza (int[]a, int pos)`

che, preso come parametro un array di numeri interi **a** e un intero **pos**, senza modificare il contenuto dell'array, restituisce true se e solo se l'elemento in posizione **pos** non compare nell'array **a** nelle posizioni da 0 a **pos**-1.
Se **a** vale *null*, viene restituito *false*. Se **pos** è fuori dagli indici ammissibili per l'array, viene restituito *false*.
Ad esempio, se a={5,3,7,7,3,10,5} e **pos** = 2, il metodo deve restituire *true* in quanto l'elemento 7 compare per la prima volta in posizione 2.

Esercizio 3 (13 punti)

Scrivere un metodo

`public static int[] multiIntersezione (int[]a, int[] b)`

che, presi come parametri due array di numeri interi **a** e **b**, senza modificare il contenuto degli array, crea e restituisce un nuovo array contenente l'intersezione dei multinsiemi rappresentati dagli array **a** e **b**.
Nell'intersezione di due multinsiemi **a** e **b** compaiono tutti e soli gli elementi comuni ai multinsiemi con un numero di ripetizioni uguale al minimo tra il numero di ripetizioni dell'elemento in **a** e quello in **b**.
Se **a** o **b** valgono *null*, viene restituito *null*.
Ad esempio, se a={5, 3, 7, 7, 3, 10, 5} e b={20, 5, 7, 5, 30}, il metodo deve restituire {5, 5, 7}.
Il metodo **deve** richiamare i metodi contaOccorrenze e primaOccorrenza e potrà applicare il seguente algoritmo:

- prima bisogna contare quanti elementi vanno inseriti nell'array da restituire:
  - per ogni posizione **i** dell'array **a**, se **i** è la prima occorrenza dell'elemento **a[i]**, devono essere contati un numero di elementi in numero uguale al minimo tra le occorrenze di **a[i]** in **a** e le occorrenze di **a[i]** in **b**
- bisogna creare l'array da restituire dell'opportuna lunghezza
- bisogna infine riempire l'array creato:
  - per ogni posizione **i** dell'array **a**, se **i** è la prima occorrenza dell'elemento **a[i]**, devono essere inseriti un numero di elementi uguali ad **a[i]** in numero uguale al minimo tra le occorrenze di **a[i]** in **a** e le occorrenze di **a[i]** in **b**

Seconda parte

Esercizio 4 (10 punti)

Cosa stampa il seguente programma Java?

```
class A {
      private double x;

      public A (double x){
            this.x=x;
      }
      public double getX(){
            return x;
      }
      double metodo (A a){
            return getX()-3*a.getX();
      }
}


class B extends A{
      public B(double x){
            super (2*x);
      }
      double metodo (A a){
            return this.getX()+
                  5*a.getX();
      }
}
```

```
public class MainClass {
      public static void main
                  (String[] args) {
       A a = new A(3);
       A b = new B(2);
       double y = a.metodo(b);
       double z = b.metodo(a);
       System.out.println(a.getX());
       System.out.println(b.getX());
       System.out.println(y);
       System.out.println(z);
      }
}
```

Esercizio 5 (23 punti)

Si progetti una classe **MultiElemento** con le variabili di istanza **valore** (tipo double, final) e **ripetizioni** (tipo int). La classe deve avere:

- un costruttore che crea un MultiElemento dati valore e ripetizioni (se ripetizioni è minore di 1, viene impostato ad 1);
- un costruttore che crea un MultiElemento dato il valore (impostando ripetizioni ad 1);
- un metodo getValore();
- un metodo getRipetizioni();
- un metodo setRipetizioni (int ripetizioni) che imposta il campo ripetizioni al valore dato come parametro (se ripetizioni è minore di 1, viene impostato ad 1);
- un metodo toString() che restituisce la descrizione nel seguente modo (nell'esempio seguente il campo valore è 3.14 e il campo ripetizioni è 5):
  `elemento 3.14 occorrenze 5`

Si progetti una classe **MultiInsieme** con una variabile di istanza **set** di tipo ArrayList<MultiElemento> che rappresenti un multinsieme. La classe deve avere:

- un costruttore che crea un MultiInsieme vuoto
- un metodo getRipetizioni (double valore) che restituisce il numero di ripetizioni relative all'elemento valore contenuto nell'insieme
- un metodo addElemento (double valore, int ripetizioni) che aggiunge al multiInsieme l'elemento valore con il numero di ripetizioni indicato. ***Attenzione***: se nel multinsieme c'è già un elemento con lo stesso valore, viene solo aggiornato il suo campo ripetizioni, altrimenti viene aggiunto all'arraylist un nuovo MultiElemento.

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 22 Dicembre 2016

Compito n° 2

Prima parte

Esercizio 1 (10 punti)

Scrivere un metodo

`public static int contaOccorrenze (int[] a, int n)`

che, preso come parametro un array a di numeri interi e un intero n, senza modificare il contenuto dell'array, restituisce il numero di occorrenze dell'intero n all'interno dell'array a. Se **a** vale *null*, viene restituito *0*.
Ad esempio, se a= [10, 6, 5, 4, 5, 6] e n=5 deve essere restituito 2.

Esercizio 2 (10 punti)

Scrivere un metodo

`public static boolean ultimaOccorrenza (int[]a, int pos)`

che, preso come parametro un array di numeri interi **a** e un intero **pos**, senza modificare il contenuto dell'array, restituisce true se e solo se l'elemento in posizione **pos** non compare nell'array **a** nelle posizioni da **pos**+1 in poi.
Se **a** vale *null*, viene restituito *false*. Se **pos** è fuori dagli indici ammissibili per l'array, viene restituito *false*.
Ad esempio, se a={5,3,7,7,3,10,5} e **pos** = 3, il metodo deve restituire *true* in quanto l'elemento 7 compare per l'ultima volta in posizione 3.

Esercizio 3 (13 punti)

Scrivere un metodo

`public static int[] multiIntersezione (int[]a, int[] b)`

che, presi come parametri due array di numeri interi **a** e **b**, senza modificare il contenuto degli array, crea e restituisce un nuovo array contenente l'intersezione dei multinsiemi rappresentati dagli array **a** e **b**.
Nell'intersezione di due multinsiemi **a** e **b** compaiono tutti e soli gli elementi comuni ai multinsiemi con un numero di ripetizioni uguale al minimo tra il numero di ripetizioni dell'elemento in **a** e quello in **b**.
Se **a** o **b** valgono *null*, viene restituito *null*.
Ad esempio, se a={5, 3, 7, 7, 3, 10, 5} e b={20, 5, 7, 5, 30}, il metodo deve restituire {5, 5, 7}.
Il metodo **deve** richiamare i metodi contaOccorrenze e ultimaOccorrenza e potrà applicare il seguente algoritmo:

- prima bisogna contare quanti elementi vanno inseriti nell'array da restituire:
  - per ogni posizione **i** dell'array **a**, se **i** è l'ultima occorrenza dell'elemento **a[i]**, devono essere contati un numero di elementi in numero uguale al minimo tra le occorrenze di **a[i]** in **a** e le occorrenze di **a[i]** in **b**
- bisogna creare l'array da restituire dell'opportuna lunghezza
- bisogna infine riempire l'array creato:
  - per ogni posizione **i** dell'array **a**, se **i** è l'ultima occorrenza dell'elemento **a[i]**, devono essere inseriti un numero di elementi uguali ad **a[i]** in numero uguale al minimo tra le occorrenze di **a[i]** in **a** e le occorrenze di **a[i]** in **b**

Seconda parte

Esercizio 4 (10 punti)

Cosa stampa il seguente programma Java?

```
class A {
      private double x;

      public A (double x){
            this.x=x;
      }
      public double getX(){
            return x;
      }
      double metodo (A a){
            return getX()-2*a.getX();
      }
}


class B extends A{
      public B(double x){
            super (3*x);
      }
      double metodo (A a){
            return this.getX()+
                  7*a.getX();
      }
}
```

```
public class MainClass {
      public static void main
                  (String[] args) {
       A a = new A(5);
       A b = new B(2);
       double y = a.metodo(b);
       double z = b.metodo(a);
       System.out.println(a.getX());
       System.out.println(b.getX());
       System.out.println(y);
       System.out.println(z);
      }
}
```

Esercizio 5 (23 punti)

Si progetti una classe **MultiElemento** con le variabili di istanza **valore** (tipo double, final) e **ripetizioni** (tipo int). La classe deve avere:

- un costruttore che crea un MultiElemento dati valore e ripetizioni (se ripetizioni è minore di 1, viene impostato ad 1);
- un costruttore che crea un MultiElemento dato il valore (impostando ripetizioni ad 1);
- un metodo getValore();
- un metodo getRipetizioni();
- un metodo setRipetizioni (int ripetizioni) che imposta il campo ripetizioni al valore dato come parametro (se ripetizioni è minore di 1, viene impostato ad 1);
- un metodo toString() che restituisce la descrizione nel seguente modo (nell'esempio seguente il campo valore è 3.14 e il campo ripetizioni è 5):
  `elemento 3.14 occorrenze 5`

Si progetti una classe **MultiInsieme** con una variabile di istanza **set** di tipo ArrayList<MultiElemento> che rappresenti un multiinsieme. La classe deve avere:

- un costruttore che crea un MultiInsieme vuoto
- un metodo getRipetizioni (double valore) che restituisce il numero di ripetizioni relative all'elemento valore contenuto nell'insieme
- un metodo addElemento (double valore, int ripetizioni) che aggiunge al multiInsieme l'elemento valore con il numero di ripetizioni indicato. Attenzione: se nel multinsieme c'è già un elemento con lo stesso valore, viene solo aggiornato il suo campo ripetizioni, altrimenti viene aggiunto all'arraylist un nuovo MultiElemento.

Programmazione

Appello del 24/01/2012

Esercizio 1 (8 punti)

a. (4 punti) Scrivere l'output del seguente programma Java

```
public class Main {
public static void main(String[] args) {
        int [] A = {55,83,21,16,43,10,56};
        stampa(A);
        enigma(A);
        stampa(A);
    }

public static void stampa (int [] A ){
        for (int x:A) {
            System.out.print (x + " ");
        }
        System.out.println();
    }

public static void enigma (int A[]){
      int i=A.length-2;
      do {
        A[i+1]=A[i]+1;
        i--;
      } while (i>=0);
    }
}
```

b. (4 punti) Riscrivere il metodo enigma in modo equivalente (ossia in modo che QUALSIASI sia l'array A lo modifichi nello stesso modo) facendo uso esclusivamente del costrutto iterativo **for**

ESERCIZIO FACOLTATIVO:
Riscrivere il metodo enigma in modo equivalente facendo uso della ricorsione, senza utilizzare alcun costrutto iterativo (né for, né while, né do-while). Se necessario, si possono utilizzare anche altri metodi statici di appoggio, a patto che non usino alcun costrutto iterativo.

Esercizio 2 (8 punti)

Scrivere un metodo **public static boolean verifica (int [][] A, int [][] B, int [][] C)** che presi in input tre array bidimensionali di numeri interi restituisce ***true*** se e solo se valogono tutte le seguenti condizioni:

- A, B e C sono tutte e tre matrici rettangolari
- A, B e C hanno le stesse dimensioni
- A+B = C

Attenzione:

- Per svolgere il compito si hanno a disposizione **90** minuti (**50** minuti per chi deve svolgere solo una parte).
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO.
- Durante la prova scritta non è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**

SECONDA PARTE

Esercizio 3 (8 punti)

Si implementino in Java le classi **Articolo** e **Legge**.
La *classe* **Articolo** ha i seguenti attributi:

- **numero** (un intero)
- **testo** (una stringa)

ed i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Articolo** assegnando numero e testo.
- metodi "*get*" per tutti gli attributi, cioè metodi che restituiscono i valori di ciascun attributo;
- metodo "toString"

La *classe* **Legge** ha i seguente attributi:

- **titolo** (una Stringa)
- **articoli** (un arrayList di Articolo)

ed i seguenti metodi di istanza:

- *costruttore* che crea un oggetto della *classe* **Legge** prendendo in input il titolo della legge
- metodo **public void *addArticolo (Articolo a),*** che aggiunge l'articolo a all'arrayList articoli, **solo se** nella legge non è già presente un articolo con lo stesso numero.
- metodo **public String getTitolo ()** che restituisce il titolo della legge.
- metodo **public int getNArticoli ()** che restituisce il numero di articoli della legge.
- metodo **public Articolo getArticolo (int k)** che restituisce il k-esimo articolo della legge, ovvero l'articolo avente k come numero (non necessariamente quello in posizione k). Se un tale articolo non esiste, viene restituito null.
- metodo "toString", che sfrutta il metodo toString della classe Articolo

ESERCIZIO FACOLTATIVO:
Fare in modo che il metodo toString della classe Legge stampi gli articoli in ordine crescente di numero (si possono utilizzare, se lo si ritiene necessario, anche variabili di istanza ausiliarie).

Esercizio 4 (8 punti)

- Si scriva una classe astratta **Funzione** per rappresentare funzioni sul piano cartesiano
- La classe deve avere un metodo astratto **public abstract double calcola (double x);** che sarà implementato dalle classi che estendono **Funzione** con lo scopo di calcolare il valore della funzione nel punto **x**.
- La classe deve inoltre implementare un metodo **public boolean LEq (Funzione f, double x)** che ritorna ***true*** se e solo se il valore della funzione su cui è invocato calcolata nel punto **x** è minore o uguale al valore della funzione **f** calcolata sempre in **x** *[si sfrutti il metodo* **calcola***]*
  Ad esempio, se f e g sono due funzioni, f.LEq(g,3) deve ritornare true se e solo se f(3)≤g(3)
- *Si implementino infine le seguenti classi* ***concrete*** *che estendono* **Funzione***,* ognuna con le opportune variabili di istanza, ed in ognuna delle quali bisogna implementare un ***costruttore*** ed **il** metodo **calcola***:*
  - La classe **Retta**, il cui costruttore prende due double (m e q) tali che la funzione rappresentata è **f(x)=mx+q**
  - La classe **Parabola**, il cui costruttore prende tre double (a,b,c) tali che la funzione rappresentata è **$f(x)=ax^2+bx+c$**

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 24 gennaio 2017

Esercizio 1 (6 punti)

1.1 (3 punti) Cosa stampa il seguente frammento di codice Java?

```
boolean trovato=false;
int n=25,i;
for (i=1; i<10 && !trovato; i++) {
  if (i*i==n){
     trovato=true;
  } else {
      System.out.println ("- " + i);
  }
}
System.out.println ("+ " + (i-1));
```

1.2 (3 punti) Cosa stampa il seguente programma Java?

```
public class Main {
   public static void main(String[] args) {
        System.out.println(enigma(1));
        System.out.println(enigma(6));
        System.out.println(enigma(10));
        System.out.println(enigma(11));
   }
    static int enigma (int x){
        if (x<=0) return 0;
        if (x%2==0) return enigma(x-1) + x;
        return enigma (x-1) - x;
    }
}
```

Esercizio 2 (10 punti)

- **(5 punti)** Scrivere un metodo
  `static int contoCompresi (int[] a, int min, int max)`
  che, preso come parametro un array **a** di numeri interi e due interi **min** e **max**, restituisce il numero di elementi di a che sono compresi tra **min** e **max** (estremi inclusi). Se **a** vale **null**, viene restituito 0.
  Ad esempio se **a={5,7,3,2,9,8,10,8}**, **min=7** e **max=9**, il metodo deve restituire **4**.
- **(5 punti)** Scrivere un metodo
  `static int[] estraiCompresi (int[] a, int min, int max)`
  che, preso come parametro un array **a** di numeri interi e due interi **min** e **max**, sfruttando il metodo contoCompresi, restituisce un nuovo array della opportuna lunghezza contenente tutti e soli gli elementi dell'array **a** che sono compresi tra **min** e **max** (estremi inclusi), preservando lo stesso ordine che tali elementi hanno nell'array a.
  Ad esempio se **a={5,7,3,2,9,8,10,8}**, **min=7** e **max=9**, il metodo deve restituire {7,9,8,8}.

Esercizio 3 (16 punti)

Si progetti una classe **Statino** con

- una variabili di istanza **matricola** (tipo int, final, private)
- una variabile di istanza **voto** (tipo int, private).
- una variabile di istanza **lode** (tipo boolean, private).
- una variabile di istanza **domandeEsame** (tipo String, private).

La classe deve avere:

- un costruttore che crea uno Statino dati **matricola** e **voto** (se voto è minore di 0 viene impostato a 0; se è maggiore di 30 viene impostato a 30 e viene assegnata la lode);
- un costruttore che crea uno Statino dati matricola, voto e domandeEsame;
- un metodo getVoto();
- un metodo getMatricola();
- un metodo getLode();
- un metodo setDomandeEsame (String s) che imposta il campo **domandeEsame** al valore dato come parametro
- un metodo toString() che restituisce la descrizione nel seguente modo (nell'esempio seguente il campo matricola è 31000000, il campo voto è 24 e il campo domandeEsame è "comando For"):

```
studente 310100000 voto 24 domande:comando For
```

Si progetti una classe **Appello** con

- una variabile di istanza **statini** di tipo ArrayList<Statino> , private

La classe deve avere:

- un costruttore che crea un appello con nessuno statino (creando l'arrayList statini vuoto)
- un metodo getVoto (int matricola) che restituisce il voto associato allo studente cono la matricola indicata
- un metodo addStatino (Statino s) che aggiunge all'arrayList statini lo statino s
- un metodo contaLodi che restituisce il numero di lodi presenti negli statini.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Le risposte al primo esercizio devono essere date direttamente nel riquadro di questo foglio.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Cognome e Nome _____

Matricola _____

Programmazione
A.A. 2016/2017

Appello del 30 maggio 2017

Esercizio 1 (9 punti)

Cosa stampa il seguente programma Java?

```
class A {
	private int n;
	public A (int n){
		this.n=n;
	}
	public int getN(){
		return n;
	}
	void metodo (A a, int n){
		System.out.println ("metodo 1; " + n);
	}
	void metodo (B a, int n){
		System.out.println ("metodo 2; " + n);
	}
}

class B extends A{
	public B(int n){
		super (2*n);
	}
	void metodo (A a, int n){
		System.out.println ("metodo 3; " + n);
	}

}

public class MainClass {
	public static void main(String[] args) {
		A a = new A(5);
		B b = new B(4);
		A ab = new B(3);
		a.metodo(ab, ab.getN());
		ab.metodo(b, a.getN());
		ab.metodo(ab, b.getN());
	}
}
```

Esercizio 2 (10 punti)

Un array di **char** rappresenta in modo naturale una stringa. Ad esempio, l'array **{'c', 'a', 'r'}** rappresenta la stringa **"car"**.

Scrivere un metodo

```
static boolean prefisso (char[] s1, char[] s2)
```

che, presi come parametro due array di char, restituisce true se e solo se la stringa rappresentata da **s1** è un prefisso di quella rappresentata da **s2**. Se **s1** o **s2** vagono **null**, viene restituito *false*.

Ad esempio se **s1={'c', 'a'}**, e **s2={'c', 'a' , 's', 'a'}**, viene restituito true.

Non è possibile usare nessun metodo/operatore della classe **String**.

Esercizio 3 (15 punti)

Si progetti una classe **Cliente** con

- una variabile di istanza **nome** (tipo String, final, private)
- una variabile di istanza **cognome** (tipo String, final, private).

La classe deve avere:

- un costruttore che crea un Cliente dati **nome** e **cognome**;
- metodi pubblici accessori **`getNome()`**, **`getCognome()`**;
- un metodo pubblico **`toString()`** che restituisce la descrizione del cliente concatenando il nome e il cognome (separati da uno spazio)

Si progetti una classe **ListaDiAttesa** con

- una variabile di istanza **lista** di tipo ArrayList<Cliente> , private

  La classe deve avere:
- un costruttore che crea una lista di attesa vuota (creando l'arrayList **lista** vuoto)
- un metodo **`public void aggiungi(Cliente c)`** che aggiunge alla fine della lista di attesa il cliente **c**
- un metodo **`public Cliente estrai()`** che elimina dalla lista il primo cliente in attesa e lo restituisce.
- un metodo **`public int inAttesa()`** che restituisce il numero dei clienti presenti nella lista d'attesa.
- un metodo pubblico **`toString()`** che restituisce la descrizione della lista di attesa elencando uno dopo l'altro i clienti in attesa preceduti dal numero d'ordine.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **90** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Le risposte al primo esercizio devono essere date direttamente nel riquadro di questo foglio.
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- **Non** è possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Appello di Programmazione 2
16 dicembre 2013

Esercizio 1.
Si progetti una classe `Articolo` con tre variabili di istanza: descrizione, prezzo unitario e quantità. Si progetti una classe `Magazzino` con due variabili d'istanza: un nome di tipo String e una lista di articoli di tipo array di `Articolo`.
Si scrivano i metodi della classe `Magazzino` per calcolare:
- dato un parametro intero n, il numero di articoli la cui quantità è minore o uguale a n (ritorna un intero);
- il valore totale degli articoli del magazzino (ritorna un double);
- se la quantità di tutti gli articoli è diversa da 0 (ritorna un boolean).

Infine si progetti una classe di test che costruisce un magazzino con tre articoli, invoca tutti i metodi, e ne stampa i risultati.

Esercizio 2.
Si considerino l'interfaccia `Archiviabile` e la classe `Biblioteca`:

```
public interface Archiviabile {
     int numeroOggettiInArchivio();
     int numeroPosizioniLibere();
}

public class Biblioteca {
          private String[] descrizioneOggetto;
}
```

dove l'array `descrizioneOggetto` contiene la descrizione del libro oppure `null` se la posizione è libera.

Si modifichi la classe `Biblioteca` affinché implementi l'interfaccia `Archiviabile`, dove il primo metodo calcola il numero totale di libri presenti in biblioteca, il secondo metodo le posizioni libere. Si progetti una classe `BibliotecaPerBambini` sottoclasse di `Biblioteca` con, in aggiunta, una variabile d'istanza `etaConsigliata` (di tipo array di interi) che implementi l'interfaccia `Archiviabile`.

Esercizio 3.
Si dica cosa stampa il seguente programma, motivando la risposta.

```
public class A {

     private int n;

     public A(int n) {
          this.n=n;
     }

     public int getN() {
          return n;
     }

     public void setN(int n) {
          this.n=n;
     }

     public double calcola(int m) {
          n= n + m;
          return n;
     }

     public double calcola(double m) {
          return n + m + 5;
     }
}
```

```
public class B extends A {

     public B(int n) {
          super(n);
     }

     public double calcola(int m) {
          setN(getN() * m);
          return getN();
     }
}
```

```
public class Appello1 {

     public static void main(String[] args) {

          int i = 2;
          int k = 3;
          B b = new B(i);
          A a = b;
          System.out.println(a.calcola(k));
          System.out.println(b.calcola(b.calcola(k)));
     }
}
```

Appello di Programmazione 2
20 gennaio 2014

Esercizio 1.
Un multinsieme è una generalizzazione del concetto di insieme che permette elementi ripetuti. Ad esempio [[1, 2, 2, 2, 5, 6, 6]] è un multinsieme dove il numero 2 è ripetuto tre volte.

Si progetti una classe `Multinsieme` con una variabile d'istanza di tipo array di interi che conterrà gli elementi del multinsieme.
Si scrivano un costruttore ed i metodi della classe `Multinsieme` per:
- inserire un nuovo elemento (non è necessario mantenere l'ordinamento tra gli elementi);
- calcolare il numero di elementi contenuti nel multinsieme, tenendo conto delle ripetizioni (esempio: il multinsieme [[1, 2, 2, 2, 5, 6, 6]] contiene 7 elementi);
- eliminare una occorrenza di un elemento: il metodo deve ritornare `true` se è stato possibile eliminare l'elemento, `false` altrimenti (esempio: eliminando 2 da [[1, 2, 2, 2, 5, 6, 6]] otteniamo [[1, 2, 2, 5, 6, 6]] ed il metodo ritorna `true`);
- dato un numero "n", calcolare il numero di occorrenze di "n" (esempio: le occorrenze di 2 in [[1, 2, 2, 2, 5, 6, 6]] sono tre).

Infine si progetti una classe di test che costruisce un multinsieme, invoca tutti i metodi, e ne stampa i risultati.

Esercizio 2.
Data l'interfaccia:

```
public interface FormaGeometrica {

      int numeroLati();
      double calcolaPerimetro();
}
```

si progettino due classi Rettangolo (con due varibili di istanza) e Quadrato che implementino l'interfaccia `FormaGeometrica`, in cui Quadrato è una sottoclasse di Rettangolo.

Esercizio 3.
Si dica cosa stampa il seguente programma, motivando la risposta.

```
public class A {

     private int n;

     public A() {
          n=0;
     }

     public int calcola(int a) {
          return n+a;
     }
}
```

```
public class B extends A {

     public B() {
          super();
     }

     public int calcola(int a) {
          return super.calcola(a)+1;
     }
}
```

```
public class Test {

     public static void main(String[] args) {
          B b = new B();
          A a = b;

          System.out.println(b.calcola(10));
          System.out.println(a.calcola(10));
     }

}
```

Appello di Programmazione 2
20 maggio 2014

Esercizio 1.
Un numero razionale (o frazione) è un numero che può essere espresso come frazione di due numeri interi. Ad esempio 3/4 e 7/100 sono numeri razionali.

Si progetti una classe `Razionale` con due variabili d'istanza intere per rappresentare il numeratore ed il denominatore del numero razionale. (Nota: non è necessario che i numeri razionali siano ridotti, ad esempio 1/2 e 2/4 sono ugualmente accettabili).

Si progettino un costruttore ed i seguenti metodi della classe `Razionale`:
- `int getNumeratore()` : restituisce il numeratore;
- `int getDenominatore()` : restituisce il denominatore;
- `boolean maggioreDi(int n)`: restituisce `true` se il numero razionale è maggiore di `n`, `false` altrimenti;
- `Razionale somma(Razionale r)`: restituisce un nuovo numero razionale ottenuto sommando il numero `r`.

Si progetti una classe di test che contenga i seguenti metodi statici:
- `Razionale[] serie(int n)`: restituisce una array di numeri razionali i cui elementi sono 1/n, 2/n, 3/n,... n/n;
- `Razionale sommatoria(Razionale[] a)`: calcola la somma di tutti i numeri razionali presenti nell'array `a`.

Infine si scriva il metodo main nel quale si calcoli la sommatoria 1/100 + 2/100 + 3/100 + 4/100+ … + 100/100 e si dica se è maggiore di 2 sfruttando i metodi statici sopra esposti.

Esercizio 2.
Si progettino una nuova classe `Razionale2,` sottoclasse di `Razionale,` ed una eccezione `EccezioneRazionale`.

Si progetti un costruttore della classe `Razionale2` affinché restituisca l'eccezione `EccezioneRazionale` se si cerca di costruire un numero razionale il cui denominatore è zero.

Si progetti un metodo per calcolare la divisione tra numeri razionali:
`Razionale2 divisione(Razionale2 r)`: effettua la divisione del numerro razionale ed `r`, e solleva l'eccezione `EccezioneRazionale` se `r` è zero.

Esercizio 3.
Si dica cosa stampa il seguente programma, motivando la risposta.

```
public class A {
```

```
	private int n=1;

	public int calcola(int a) {
		return n+a;
	}
}


public class B extends A {

	public B() {
		super();
	}

	public int calcola(int a) {
		return super.calcola(a)+1;
	}
}


public class Test {

	public static void main(String[] args) {
		A[] a = new A[2];
		a[0] = new A();
		a[1] = new B();

		System.out.println(a[0].calcola(10));
		System.out.println(a[1].calcola(10));
	}
 }
```

Appello di Programmazione 2
9 giugno 2014

Esercizio 1.
Si progetti una classe `InsiemeLimitato` che rappresenti un insieme di al più `n` elementi di tipo intero. La classe `InsiemeLimitato` utilizza un array di interi di lunghezza `n` per memorizzare gli elementi dell'insieme ed una variabile `numElementi` per tenere traccia del numero di elementi contenuti nell'insieme.

Si progetti un costruttore con un parametro `n` che indica il numero massimo di elementi che l'insieme può contenere.

Si progettino i seguenti metodi della classe `InsiemeLimitato`:
- `getNumElementi()`: restituisce il numero di elementi contenuti nell'insieme;
- `getSize()`: restituisce il numero massimo di elementi che l'insieme può contenere;
- `boolean contenuto(int e)`: restituisce `true` se l'elemento `e` è contenuto nell'insieme, `false` altrimenti;
- `boolean vuoto()`: restituisce `true` se l'insieme è vuoto, `false` altrimenti;
- `boolean pieno()`: restituisce `true` se l'insieme è pieno, `false` altrimenti;

Si progetti una classe di test che contenga i seguenti metodi statici:

- `boolean confronta(InsiemeLimitato i, InsiemeLimitato j)`: restituisce `true` se gli insiemi `i` e `j` hanno gli stessi elementi, `false` altrimenti;

- `InsiemeLimitato differenza(InsiemeLimitato i, InsiemeLimitato j)`: restituisce l'insieme differenza formato da tutti gli elementi che appartengono ad `i` e non appartengono a `j`.

Infine si scriva il metodo main nel quale si creano due insiemi e si chiamano tutti i metodi.

Esercizio 2.
Si progettino una eccezione `EccezioneInsiemeLimitato` e due metodi `inserisci` e `max` da aggiungere alla classe `InsiemeLimitato`:

- `void inserisci(int e)`: inserisce l'elemento `e` nell'insieme. Se l'insieme è pieno solleva l'eccezione `EccezioneInsiemeLimitato`. (Nota: ovviamente se un elemento appare già nell'insieme non deve essere inserito nuovamente).

- `int max()`: restituisce il massimo dell'insieme. Se l'insieme è vuoto solleva l'ecezione `EccezioneInsiemeLimitato`.

Esercizio 3.
Si dica cosa stampa il seguente programma, motivando la risposta.

```
public class A {

      private int n;
```

```
	public A(int n) {
		this.n=n;
	}

	public void raddoppia() {
		n=n*2;
	}

	public int getN() {
		return n;
	}
}

public class B extends A{

	public B(int n) {
		super(n);
	}

	public void raddoppia() {
		super.raddoppia();
		super.raddoppia();
	}
}

public class Test {

	public static void main(String[] args) {
		A[] vettore = new A[3];
		vettore[0] = new A(1);
		vettore[1] = new B(1);
		vettore[2] = vettore[0];

		for(int i=0;i<vettore.length;i++) {
			A a = vettore[i];
			a.raddoppia();
			System.out.println(a.getN());
		}
	}
}
```

Appello di Programmazione 2
7 luglio 2014

Esercizio 1.
Si progettino le classi `Libro, LibroPerBambini` e `Biblioteca` per la gestione di una biblioteca.

La classe `Libro` ha 3 variabili di istanza `titolo, autore, inPrestito`. La variabile booleana `inPrestito` indica se il libro è attualmente in prestito. Si inseriscano nella classe `Libro` i segunti metodi:
- `String getDescrizione()`: restituisce una descrizione contenente il titolo e l'autore del libro;
- `boolean getInPrestito()`: restituisce `true` se il libro è in prestito, `false` altrimenti;
- `void setInPrestito(boolean stato)`: cambia lo stato del libro (in prestito oppure disponibile).

La classe `LibroPerBambini`, sottoclasse di `Libro`, deve contenere una variabile d'istanza `etaConsigliata` ed il metodo `getDescrizione()` deve restituire nella descrizione, oltre al titolo ed all'autore, anche l'età consigliata.

La classe `Biblioteca` gestisce la collezione di libri con una struttura dati di tipo `ArrayList`, che può contenere qualsiasi oggetto di tipo `Libro` e `LibroPerBambini`. La biblioteca implementa i seguenti metodi:
- `void addLibro(Libro l)`: aggiunge un nuovo libro alla biblioteca;
- `boolean inBiblioteca(Libro t)`: restituisce true se il libro è presente in biblioteca;
- `void prestato(Libro l)`: consente di dare in prestito un libro;
- `void restituito(Libro l)`: consente di restituire un libro dato in prestito;
- `int totInPrestito()`: conta il numero di libri in prestito;
- `boolean nessunPrestito()`: restituisce true se nella biblioteca non vi sono libri in prestito;
- `int perBambini(int n)`: conta quanti libri per bambini di età minore o uguale a n sono presenti in biblioteca.

Si scriva una classe di test che crea un libro, un libro per bambini, una biblioteca e chiama tutti i metodi.

Esercizio 2.
Si riscriva il metodo `void prestato(Libro l)` affinché sollevi l'eccezione EccezioneLibro nel caso in cui il prestito non sia possibile (il libro non è presente in biblioteca oppure è già in prestito).

Esercizio 3.
Si dica cosa stampa il seguente programma, motivando la risposta.

```java
public class A {
	private int n;

	public A(int n) {
		this.n=n;
	}

	public int getN() {
		return n;
	}

	public double calcola(int m) {
		return n+m;
	}

	public double calcola(double m) {;
		return n+m+1;
	}
}

public class B extends A {

	public B(int n) {
		super(n);
	}

	public double calcola(int m) {
		return getN() + m + 10;
	}
}

public class C extends B{

	public C(int n) {
		super(n);
	}

	public double calcola(int m) {
		return getN() + m + 100;
	}
}

public class Test {

	public static void main(String[] args) {
		C c = new C(2);
		B b = new B(5);
		A a = b;
		System.out.println(c.calcola(a.calcola(0)));
	}
}
```

Appello di Programmazione 2
11 settembre 2014

Esercizio 1.

Si progetti una classe `RegistroVendite` per registrare le vendite di un negozio (vettore di importi double).
Si progetti un costruttore in modo tale che un `RegistroVendite` possa registrare al massimo 10 vendite e che l'utente debba specificate quante vendite vuole inserire (max 10) e fornisca gli importi.
Si progettino inoltre i seguenti metodi per calcolare alcune statistiche sui dati contenuti nel vettore;

- `double totaleVendite()`: restituisce l'importo totale delle vendite
- `int venditeAlDiSopra(double valoreDiRiferimento)` restituisce il numero di vendite che hanno un valore superiore a valoreDiRiferimento, richiesto in input all'utente
- `double mediaVendite()`: restituisce il valore medio di tutte le vendite
- `double importoPiùAlto()`: restituisce il valore più alto fra tutte le vendite
- `RegistroVendite inserisci(double nuovoEl)`: inserisce una nuova vendita, se è possibile, altrimenti non fa niente.

Infine si progetti una classe di test che costruisce un `RegistroVendite`, invoca tutti i metodi, ed eventualmente ne stampa i risultati.

Esercizio 2.
Si progetti una nuova eccezione `EccezioneRegistroVendite`. Si modifichino il costruttore ed il metodo `inserisci` della classe `RegistroVendite` in modo da utilizzare l'eccezione. Il nuovo costruttore deve controllare che il numero di vendite da inserire sia al più 10 e che siano forniti gli importi per tutte e sole le vendite specificate, in caso contrario solleva l'eccezione `EccezioneRegistroVendite`. Il metodo `inserisci` effettua l'inserimento se possibile, altrimenti solleva l'eccezione `EccezioneRegistroVendite`.

Esercizio 3.
Si dica cosa stampa il seguente programma, motivando la risposta.

```
public class A {

	private int n=1;

	public int calcola(int a) {
		return n+a;
	}
}


public class B extends A {

	public B() {
		super();
	}
```

```
	public int calcola(int a) {
		return super.calcola(a)+1;
	}
}


public class Test {

	public static void main(String[] args) {
		A[] a = new A[2];
		a[0] = new A();
		a[1] = new B();

		System.out.println(a[0].calcola(10));
		System.out.println(a[1].calcola(10));
	}
 }
```

Appello di Programmazione 2
23 gennaio 2015

Esercizio 1.
Si progettino le classi `Ristorante, Tavolo, Cameriere` e `Sommelier` per la gestione di un ristorante.

Le classi `Cameriere` e `Sommelier` implementano entrambe l'interfaccia `Personale` così definita:

```
public interface Personale {
     String getNome();
}
```

Ad ogni tavolo sono associati un cameriere ed un sommelier. La classe `Tavolo` ha 3 variabili di istanza: cameriere, sommelier e listaOrdini che tiene traccia degli ordini per il tavolo ed è implementata con un array di double di dimensione 100. La classe `Tavolo` ha inoltre i seguenti metodi:

- `void addOrdine(double costo)`: aggiunge un nuovo ordine alla listaOrdini del tavolo;
- `double calcolaTotaleTavolo()`: calcola il totale degli ordini presenti (o fatturato) nella listaOrdini del tavolo.

La classe `Ristorante` ha 3 variabili di istanza: `nome` (di tipo stringa), `listaTavoli` e `listaPersonale`, entrambe implementate con array list, ed i seguenti metodi:

- `calcolaTotale(Personale p)`: restituisce il totale fatturato dalla persona `p` calcolato sommando il totale del fatturato per tutti i tavoli a cui è associato;
- `double calcolaTotale()`: calcola il fatturato totale del ristorante, per tutti i tavoli;
- `Tavolo migliorTavolo()`: restituisce il tavolo con il più alto fatturato;
- `Personale migliore()`: restituisce il cameriere o sommelier con il più alto fatturato; il fatturato di un cameriere o di un sommelier è dato dalla somma dei fatturati dei tavoli a cui è associato;
- `int eccedenzaPersonale()`: calcola il numero di camerieri e sommelier che non hanno nessun tavolo associato;
- `int tavoliVuoti()`: calcola il numero di tavoli che non sono stati occupati (il cui fatturato è zero).

Si scriva una classe di test che crea un ristorante con due tavoli, un cameriere ed un sommelier. Si facciano due ordini per tavolo e si richiamino tutti i metodi del ristorante.

Esercizio 2.
Si progetti una nuova classe `SmartTavolo`, simile alle classe `Tavolo`, nella quale si tiene traccia non solo del costo delle cosumazioni effettuate, ma anche del loro nome. Ad esempio, ad un tavolo posso essere associate le seguenti consumazioni:
“pasta al pomodoro” 5.00 €
“sformato di verdure” 7.50 €
“tortino al cioccolato” 3.50 €
Si implementi un nuovo metodo che individui la consumazione più costosa ordinata nel tavolo.

Esercizio 3.

Si dica cosa stampa il seguente programma, motivando la risposta.

```
public class A {

   private int a=10;

   int calcola(int n) throws Exception {
      if (n==0) throw new Exception("ERRORE A");
      return n+a;
   }
}
```

```
public class B extends A {

   int calcola(int n) throws Exception {
      if (n==1)
         throw new Exception("ERRORE B");
      return 2*n+super.calcola(n+1);
   }
}
```

```
public class Test {

   public static void main(String[] args) {

      B b = new B();
      A a = b;
      try {
         System.out.println(a.calcola(0));
         System.out.println(a.calcola(1));
         System.out.println(b.calcola(0));
         System.out.println(b.calcola(1));
      }
      catch (Exception e) {
         System.out.println(e.getMessage());
      }
      System.out.print("END");
   }
}
```

Appello di Programmazione 2
11 giugno 2015

Esercizio 1.

Si realizzi una applicazione per gestire il rifornimento di un magazzino merci. Si progettino:

- una interfaccia Articolo con contenga i metodi:
  - String getCodice()
  - int getGiacenza()
  - int getMin()
  - double getPrezzoUnitario()
- una classe Prodotto che implementi l'interfaccia Articolo. Per ogni prodotto vogliamo memorizzare il codice (univoco), la giacenza (numero di pezzi in magazzino), la giacenza minima che deve essere sempre presente in magazzino ed il prezzo unitario.
- una classe Servizio che implementi l'interfaccia Articolo. Per i servizi, la giacenza ed il numero minimo di pezzi sono entrambi 0.
- una classe Magazzino con una struttura dati per memorizzare sia prodotti che servizi. La classe Magazzino deve implementare i seguenti metodi:
  - vuoto(): restituisce true se il magazzino è vuoto, false altrimenti;
  - valore(Articolo a): il valore di tutti i pezzi dell'articolo "a" presenti in magazzino (dato dal prezzo unitario per il numero di pezzi in giacenza);
  - valore(): il valore di tutti gli articoli in magazzino;
  - daAcquistare(): la lista degli articoli da acquistare (per cui la giacenza è inferiore al livello minimo)
  - costo(Articolo a): il costo da sostenere per acquistare l'articolo "a" affinché la giacenza raggiunga il minimo richiesto;
  - costo(): il costo da sostenere affinché tutti gli articoli raggiungano la giacenza minima.

Esercizio 2.

Si progetti una nuova eccezione MagazzinoException e si scriva il metodo controlloMagazzino() che restituisce true se per tutti gli articoli è presente la giacenza minima, altrimenti solleva l'eccezione MagazzinoException.

Esercizio 3.

Si dica cosa stampa il seguente programma motivando la risposta e indicando, per ogni chiamata ad un metodo, la lista delle firme candidate.

```java
public class A {
	public int m(A a, B b) {return 2; };
	public int m(B a, B b) {return 10; };
	public B   m(B c, A b) {return c; }
}

public class B extends A {
	public int m(A a, B b) {return 3; }
	public A   m(C c, B b) {return b; }
	public Object    m(C c, Object o) {return o; }
}

public class C extends B {
	public A   m(C c, A a) {return c; }
	public int m(B a, B b) {return 5; }
}
public class Test {
	public static void main(String[] args) {
		C gamma = new C();
		B beta = gamma;
		A alfa = gamma;
		System.out.println(alfa.m(beta, beta));
		System.out.println(gamma.m(beta.m(gamma, beta),beta));
	}
}
```

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2017/2018

Appello del 21 dicembre 2017

Esercizio 1 (6 punti)

Sia $T(n) = \begin{cases} 8T(n/2) + 2n^3 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso sia del **teorema generale** che del metodo di **sostituzione e induzione.**

Esercizio 2 (6 punti)

Si consideri il problema di ordinare in modo non decrescente un array di **n** numeri interi positivi sapendo che il massimo intero presente nell'array è $\mathbf{n^3}$. Si fornisca un algoritmo che ordini l'array con una complessità asintotica temporale **O(n)** nel caso peggiore. Tale algoritmo può essere basato su confronti? Giustificare la risposta.

Esercizio 3 (9 punti)

Si consideri la sequenza di chiavi intere

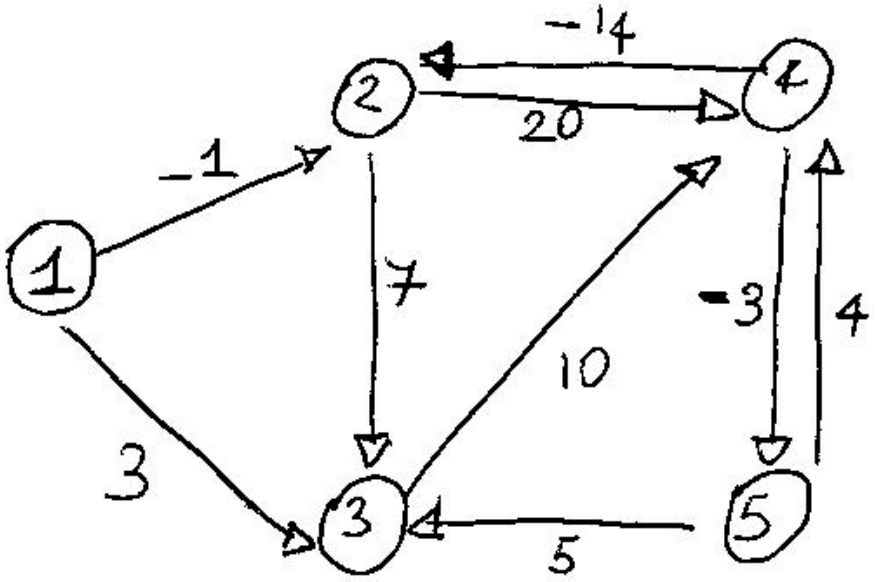
50, 32, 84, 5, 2, 8, 90, 76, 43, 35, 27, 66, 88

a. Mostrare l'heap di minimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi sopra elencate

b. Mostrare la tabella Hash di dimensione **23** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate

c. Mostrare la tabella Hash di dimensione **23** con indirizzamento aperto (tramite Hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'Hashing doppio sia **22**).

Esercizio 4 (11 punti)
Si consideri il grafo in figura.



a. Mostrare l'esecuzione **passo-passo** di un algoritmo (tra quelli studiati a lezione) per cammini minimi da singola sorgente a partire dal nodo sorgente 1. Si giustifichi anche la scelta dell'algoritmo.

b. Mostrare le matrici **D** e **π** prodotte ad ogni passo dall'algoritmo di Floyd-Warshall.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2017/2018

Appello del 16 gennaio 2018

Esercizio 1 (4 punti)

Sia $T(n) = \begin{cases} 2T(n/4) + n & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso del **teorema generale** e giustificando la scelta del caso opportuno attraverso la definizione di notazione asintotica.

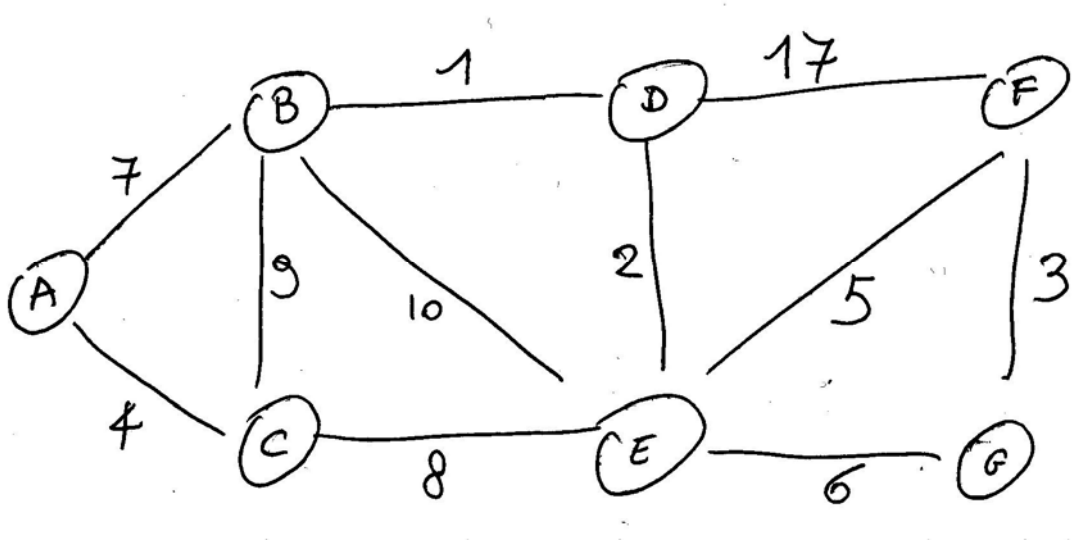
Esercizio 2 (8 punti)

Data un array di numeri interi, un elemento **x** si dice **abbondante** se e solo se compare nell'array più di x volte. Si consideri il problema di contare il numero di elementi abbondanti in un array di numeri interi. Si fornisca un algoritmo (in pseudocodice o in Java) che risolva il problema con complessità temporale **O(n log n)** nel caso peggiore. Tale algoritmo non deve modificare l'array ricevuto in input e può richiamare qualsiasi algoritmo visto a lezione.

Esercizio 3 (10 punti)

a. Mostrare l'heap di massimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi **50, 32, 84, 5, 2, 8, 90, 76, 43** (mostrare l'heap che si ottiene dopo l'inserimento di ogni chiave)
b. Rimuovere per 3 volte la chiave massima dall'heap, mostrando l'heap che si ottiene dopo ogni estrazione
c. Inserire nell'heap così ottenuto le chiavi **35, 27, 66, 88**

Esercizio 4 (10 punti)
Si consideri il grafo in figura.



a. Mostrare l'esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente a partire dal nodo sorgente A.
b. Mostrare l'esecuzione **passo-passo** dell'algoritmo di **Kruskal** per il minimo albero ricoprente.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2017/2018

Appello del 6 febbraio 2018

Esercizio 1 (6 punti)

Sia $T(n) = \begin{cases} 2T(n/2) + c & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso sia del **teorema generale** che del metodo di **sostituzione e induzione.**

Esercizio 2 (8 punti)

Si scriva un algoritmo (in pseudocodice o in Java) che, presi in input due array di numeri interi, restituisca il numero degli elementi comuni ai 2 array (senza tenere in conto le eventuali ripetizioni) e **se ne analizzi la complessità computazionale**. Tale algoritmo non deve modificare gli array ricevuti in input, può richiamare qualsiasi algoritmo visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*algoritmi con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti*), dove n è la somma delle lunghezze dei due array in input.

Se ad esempio gli array sono {1, 3, 1, 7, 5, 7} e {10, 1, 7, 8, 7}, l'algoritmo deve restituire **2** in quanto i due array hanno 2 elementi (**1** e **7**) in comune.

Esercizio 3 (10 punti)

Si consideri una tabella Hash di dimensione 17 con indirizzamento aperto implementato tramite la tecnica dell'Hashing doppio (modulo 17-1=16).

A partire dalla tabella vuota (contenente NIL in ogni posizione) e mostrando la tabella ottenuta dopo ogni inserimento, si inseriscano nella tabella Hash le chiavi

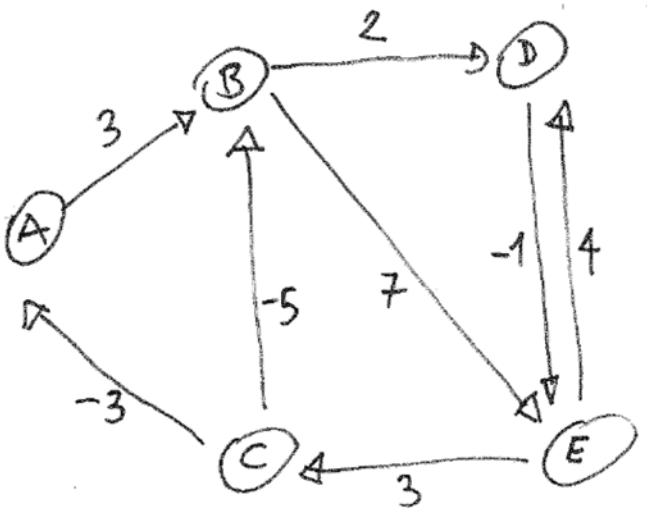
3, 10, 21, 30, 5, 8, 20, 50

Se si volesse cancellare una o più chiavi si potrebbe inserire direttamente NIL al posto delle chiavi da cancellare? Motivare la risposta.

Esercizio 4 (10 punti)

Quale algoritmo visto a lezione è possibile impiegare per determinare se un grafo (diretto) possiede cicli di peso negativo? In che modo è possibile utilizzarlo per ottenere tale risposta?

Dopo aver risposto alla domanda, si applichi l'algoritmo indicato nella risposta e se ne simuli l'esecuzione sul grafo seguente, evidenziando tutti i passaggi necessari per far decidere all'algoritmo circa la presenza di un ciclo di peso negativo nel grafo.



Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2017/2018

Appello del 2 luglio 2018

Esercizio 1 (8 punti)

Si consideri il seguente algoritmo di ricerca binaria ricorsiva:

```
function binarySearchRic(int[] A, int p, int r, int v)
     if p > r
      return -1
     if v < A[p] or v > A[r]
      return -1
     q=p+((r-p)/2)
     if A[q] == v
      return q
     if A[q] > v
      return binarySearchRic(A, p, q-1, v)
     else
      return binarySearchRic(A, q+1, r, v)
```

Scrivere la ricorrenza *T(n)* che indica il tempo di esecuzione dell'algoritmo nel caso peggiore. Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso di un metodo a scelta.

Esercizio 2 (8 punti)

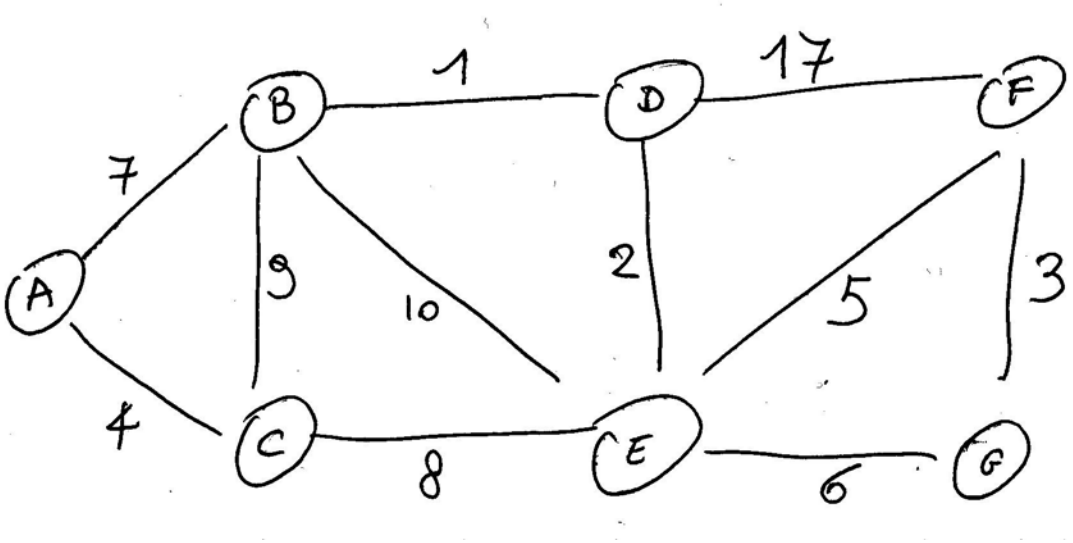
Si scriva un algoritmo (in pseudocodice o in Java) che, presi in input due array **a1** e **a2** di numeri interi, restituisca il numero degli elementi di a1 che non sono presenti in a2 (si può assumere che né a1 né a2 contengano elementi ripetuti) e **se ne analizzi la complessità computazionale**. Tale algoritmo non deve modificare gli array ricevuti in input, può richiamare qualsiasi algoritmo visto a lezione e deve avere complessità temporale **$O((n_1 + n_2) \log n_2)$** nel caso peggiore (*algoritmi con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti*), dove n_1 e n_2 sono le lunghezze dei due array in input a1 e a2, rispettivamente.

Se ad esempio gli array sono {1, 3, 9, 5, 7} e {10, 1, 7, 8, 4}, l'algoritmo deve restituire **3** in quanto gli elementi 3, 9 e 5 del primo array non sono presenti nel secondo array.

Esercizio 3 (8 punti)

a. Mostrare **l'heap di minimo** che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi **50, 32, 84, 5, 2, 8, 90, 76, 43** (mostrare l'heap che si ottiene dopo l'inserimento di ogni chiave)
b. Rimuovere per 3 volte la chiave minima dall'heap, mostrando l'heap che si ottiene dopo ogni estrazione
c. Inserire nell'heap così ottenuto le chiavi **35, 27, 6, 88**

Esercizio 4 (12 punti)
Si consideri il grafo non diretto in figura.



- Mostrare la rappresentazione del grafo tramite liste di adiacenza
- Mostrare la rappresentazione del grafo tramite matrice di adiacenza
- Si consideri il semplice problema di rispondere alla domanda: *Esiste e che peso ha l'arco tra due nodi dati in input?* Discutere quale rappresentazione del grafo rende possibile risolvere tale problema in tempo costante nel caso peggiore (non dipendente dalla dimensione del grafo).
- Eseguire **passo passo** la visita in ampiezza a partire dalla sorgente **F**, indicando l'ordine di visita dei nodi e le distanze calcolate dalla sorgente.
- Eseguire **passo passo** la visita in profondità a partire dalla sorgente **B**, indicando il tempo di inizio e fine visita per ogni nodo.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2017/2018

Appello del 17 settembre 2018

Esercizio 1 (8 punti)

Si consideri il seguente algoritmo di ricerca binaria ricorsiva, da eseguire su un array ordinato in modo non decrescente:

```
function ternarySearchRic(int[] A, int p, int r, int v)
     if r-p+1 <= 2
        if A[p] == v
            return p
        else if A[r] == v
            return r
        else
            return -1
     q1=p+((r-p+1)/3)
     q2=p+2*((r-p+1)/3)
     if v>=A[p] and v<=A[q1]
        return ternarySearchRic (A, p, q1, v)
     else if v>A[q1] and v<=A[q2]
        return ternarySearchRic (A, q1+1, q2, v)
     else
        return ternarySearchRic (A, q2+1, r, v)
```

Scrivere la ricorrenza *T(n)* che indica il tempo di esecuzione dell'algoritmo nel caso peggiore. Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso di un metodo a scelta.

Esercizio 2 (8 punti)

Si scriva un algoritmo (in pseudocodice o in Java) che, presi in input

- un intero **M**,
- un array **estratti** di numeri interi (compresi tra 1 e **M**) di dimensione **k**,
- un array bidimensionale **giocati** di numeri interi (compresi tra 1 e **M**) con **n** righe e **k** colonne,

restituisca il numero di righe dell'array **giocati** che contengono gli stessi elementi dell'array estratti (non necessariamente nello stesso ordine).

Se ad esempio M=90 e k=6, tale problema trova quanti 6 sono stati totalizzati a Superenalotto.

Si analizzi la complessità computazionale dell'algoritmo proposto in funzione di M, n e k. Tale algoritmo non deve modificare gli array ricevuti in input e deve avere complessità temporale **O(M + nk)** nel caso peggiore (*algoritmi con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti*).

Suggerimento: utilizzare un array ausiliario di dimensione M per memorizzare la *funzione caratteristica* dell'insieme dei numeri estratti.

Esercizio 3 (9 punti)

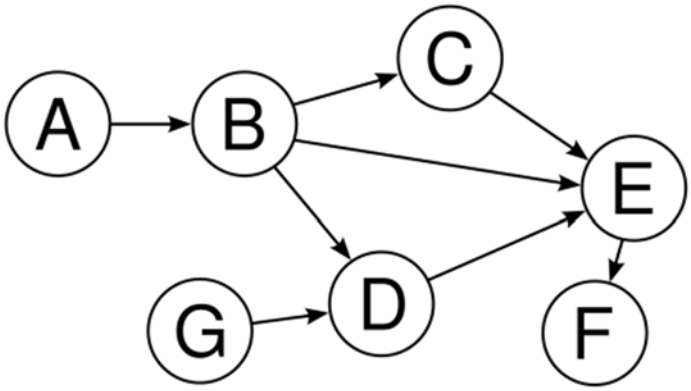
Si consideri la sequenza di chiavi intere

65, 94, 10, 140, 115, 24, 75, 35, 60, 125

a. Mostrare **passo-passo** l'heap di minimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi sopra elencate
b. Mostrare la tabella hash di dimensione **13** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate
c. Mostrare la tabella hash di dimensione **13** con indirizzamento aperto (tramite hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'hashing doppio sia **12**).

Esercizio 4 (11 punti)

Si consideri il grafo non diretto aciclico (DAG) in figura.



- Mostrare la rappresentazione del grafo tramite liste di adiacenza
- Mostrare la rappresentazione del grafo tramite matrice di adiacenza
- Eseguire **passo-passo** la visita in profondità del grafo, indicando il tempo di inizio e fine visita per ogni nodo (si fa riferimento all'algoritmo DFS che prende in input un grafo e richiama al suo interno l'algoritmo DFS-VISIT su vari nodi sorgente).
- Sfruttando la visita del passo precedente, individuare un ordinamento topologico dei nodi del DAG.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Compito n° 1

Appello del 10 gennaio 2019

Esercizio 1 (8 punti)

Sia $T(n) = \begin{cases} 4T(n/2) + cn^2 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso sia del **teorema generale** che del metodo di **sostituzione e induzione.**

Esercizio 2 (10 punti)

(7 punti) Scrivere un metodo (in Java o in pseudocodice)

```
static boolean sommaDiDue (int[] a, int x)
```

che, preso come parametro un array **a** di numeri interi ed un intero **x**, restituisce *true* se e solo se esistono due posizioni i e j di a tali che x=a[i]+a[j]. Se **a** vale **null**, viene restituito false. Il metodo **non** deve modificare l'array a.

(3 punti) Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti su 7*), dove n è la lunghezza dell'array a in input.

Ad esempio, se **a={5, 7, 3, 2, 9, 8, 10, 8}** e **x=9** il metodo deve restituire *true*, mentre se **x=-3** il metodo deve restituire *false*.

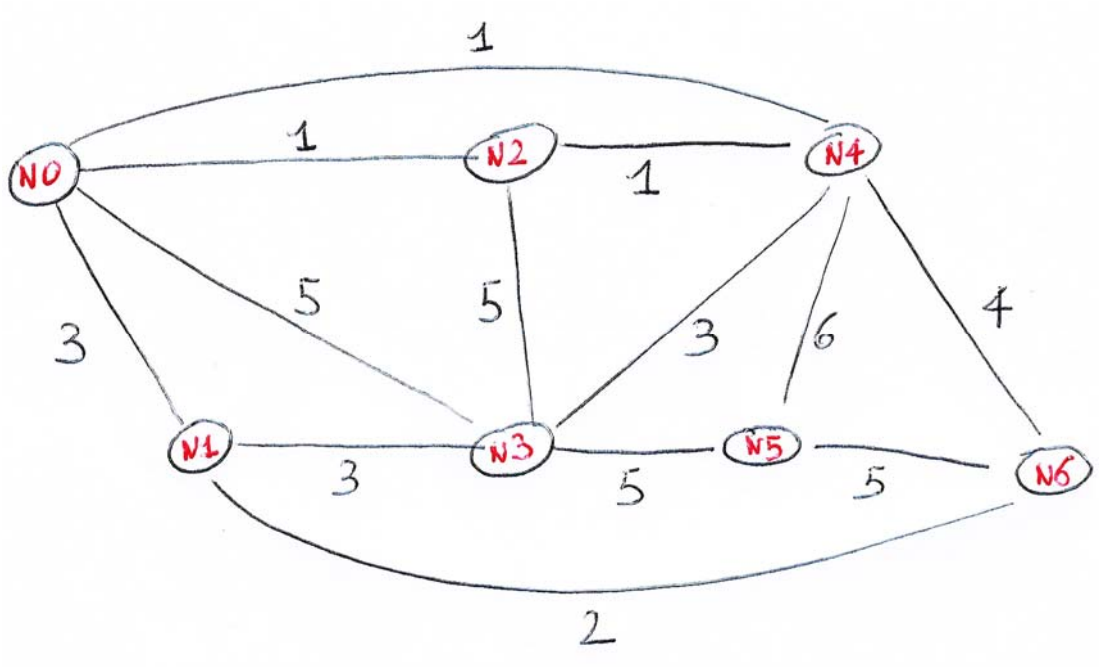
Esercizio 3 (9 punti)

Si consideri la sequenza di chiavi intere

5, 49, 10, 15, 11, 134, 75, 135, 6, 145

a. Mostrare **passo-passo** l'heap di massimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi sopra elencate
b. Mostrare la tabella hash di dimensione **13** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate
c. Mostrare la tabella hash di dimensione **13** con indirizzamento aperto (tramite hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'hashing doppio sia **12**).

Esercizio 4 (11 punti)
Si consideri il grafo in figura.



a. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente a partire dal nodo sorgente **Nx**, dove x è ottenuto *prendendo l'ultima cifra del numero di matricola e calcolando il resto della divisione per 7.*
b. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Kruskal** per il minimo albero ricoprente.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Compito n° 2

Appello del 10 gennaio 2019

Esercizio 1 (8 punti)

Sia $T(n) = \begin{cases} 4T(n/2) + cn^2 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso sia del **teorema generale** che del metodo di **sostituzione e induzione.**

Esercizio 2 (10 punti)

(7 punti) Scrivere un metodo (in Java o in pseudocodice)

```
static boolean sommaDiDue (int[] a, int x)
```

che, preso come parametro un array **a** di numeri interi ed un intero **x**, restituisce *true* se e solo se esistono due posizioni i e j di a tali che x=a[i]+a[j]. Se **a** vale **null**, viene restituito false. Il metodo **non** deve modificare l'array a.

(3 punti) Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti su 7*), dove n è la lunghezza dell'array a in input.

Ad esempio, se **a={5, 7, 3, 2, 9, 8, 10, 8}** e **x=9** il metodo deve restituire *true*, mentre se **x=-3** il metodo deve restituire *false*.

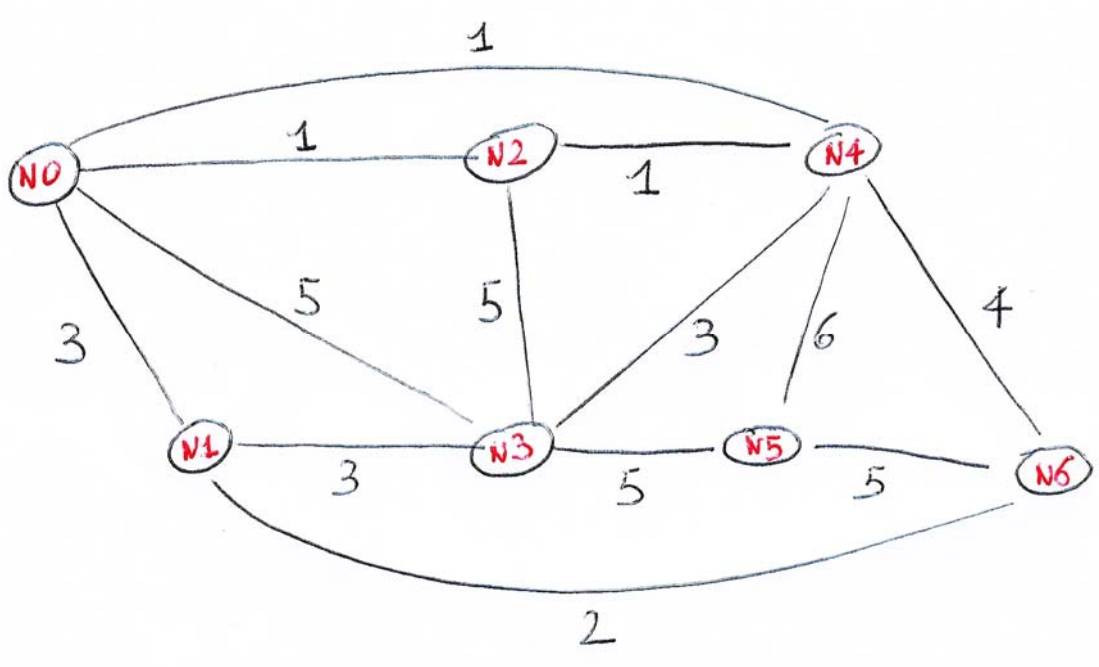
Esercizio 3 (9 punti)

Si consideri la sequenza di chiavi intere

75, 135, 5, 49, 10, 145, 15, 11, 134, 6

d. Mostrare **passo-passo** l'heap di massimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi sopra elencate
e. Mostrare la tabella hash di dimensione **13** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate
f. Mostrare la tabella hash di dimensione **13** con indirizzamento aperto (tramite hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'hashing doppio sia **12**).

Esercizio 4 (11 punti)
Si consideri il grafo in figura.



c. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente a partire dal nodo sorgente **Nx**, dove x è ottenuto *prendendo l'ultima cifra del numero di matricola e calcolando il resto della divisione per 7.*
d. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Kruskal** per il minimo albero ricoprente.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Compito n° 1

Appello del 30 gennaio 2019

Esercizio 1 (4 punti)

Sia $T(n) = \begin{cases} 9T(n/2) + cn^3 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso del **teorema generale**.

Esercizio 2 (10 punti)

(2 punti) Si scriva un metodo di ricerca binaria (iterativo o ricorsivo) in pseudocodice o in Java
`static int binarySearch (int[] a, int x)`
che, preso come parametro un array **a** di numeri interi (ordinato in modo non decrescente) ed un intero **x**, restituisce la posizione di un elemento dell'array a uguale a x, se esiste, e **-1** altrimenti.

(8 punti) Due array a e b della stessa lunghezza n (ognuno dei quali non contiene elementi ripetuti) si dicono ***equiposizionali*** se per ogni posizione i=0,…,n-1, vale che a[i] è il j-esimo elemento più piccolo in a se e solo se b[i] è il j-esimo elemento più piccolo in b. Ad esempio, a={1, 8, 3} e b={10, 13, 11} sono equiposizionali.
Scrivere un metodo (in pseudocodice o in Java)
`static boolean equiposizionali (int[] a, int[] b)`
che, presi come parametro due array **a** e **b** di numeri interi, restituisce *true* se e solo se a e b sono della stessa lunghezza e sono ***equiposizionali***. Se entrambi valgono *null*, viene restituito *true*. Se solo uno dei due vale *null*, viene restituito *false*. Il metodo **non** deve modificare gli array.
Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti su 8*), dove n è la lunghezza degli array a e b in input.

Esercizio 3 (7 punti)

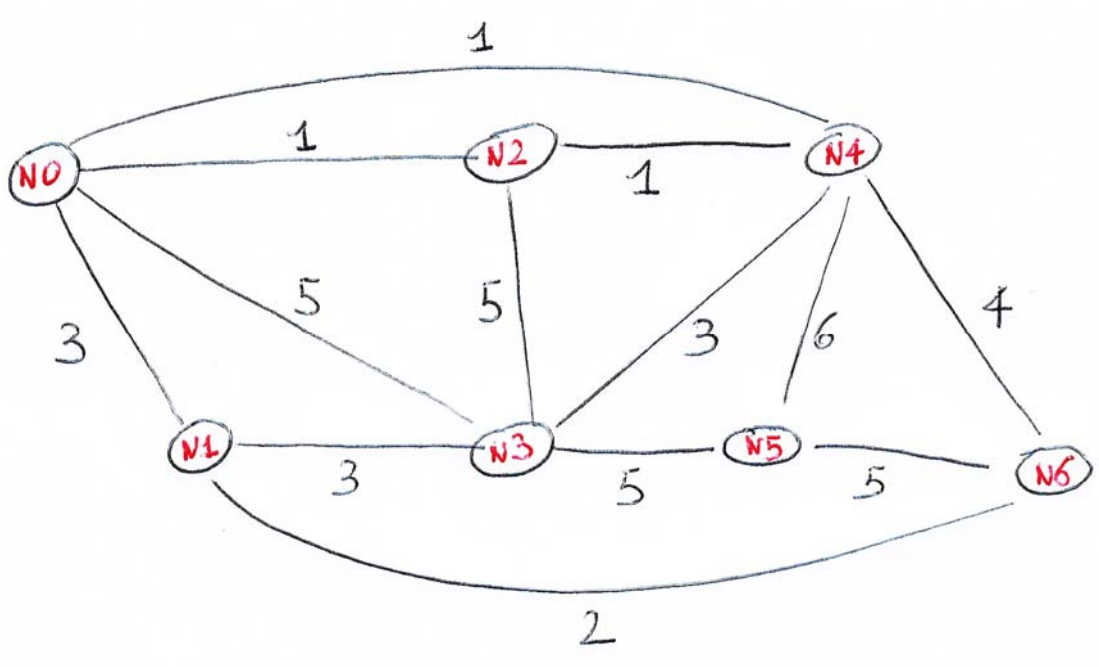
Si consideri il seguente array di numeri interi:

{5, 49, 10, 15, 11, 134, 75, 135, 6, 145}

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array in modo non decrescente (compresa la fase iniziale di build-max-heap).

Esercizio 4 (14 punti)

Si consideri il grafo in figura.



a. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per l'albero dei cammini minimi a partire dal nodo sorgente **Nx**, dove x è ottenuto *prendendo l'ultima cifra del numero di matricola e calcolando il resto della divisione per 7.*
   Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.
b. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Floyd-Warshall** per i cammini minimi tra tutte le coppie, mostrando ad ogni passo la matrice delle distanze.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Compito n° 2

Appello del 30 gennaio 2019

Esercizio 1 (4 punti)

Sia $T(n) = \begin{cases} 9T(n/4) + cn^2 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso del **teorema generale**.

Esercizio 2 (10 punti)

(2 punti) Si scriva un metodo di ricerca binaria (iterativo o ricorsivo) in pseudocodice o in Java
`static int binarySearch (int[] a, int x)`
che, preso come parametro un array **a** di numeri interi (ordinato in modo non decrescente) ed un intero **x**, restituisce la posizione di un elemento dell'array a uguale a x, se esiste, e **-1** altrimenti.

(8 punti) Due array a e b della stessa lunghezza n (ognuno dei quali non contiene elementi ripetuti) si dicono ***equiposizionali*** se per ogni posizione i=0,…,n-1, vale che a[i] è il j-esimo elemento più piccolo in a se e solo se b[i] è il j-esimo elemento più piccolo in b. Ad esempio, a={1, 8, 3} e b={10, 13, 11} sono equiposizionali.
Scrivere un metodo (in pseudocodice o in Java)
`static boolean equiposizionali (int[] a, int[] b)`
che, presi come parametro due array **a** e **b** di numeri interi, restituisce *true* se e solo se a e b sono della stessa lunghezza e sono ***equiposizionali***. Se entrambi valgono *null*, viene restituito *true*. Se solo uno dei due vale *null*, viene restituito *false*. Il metodo **non** deve modificare gli array.
Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo 4 punti su 8*), dove n è la lunghezza degli array a e b in input.

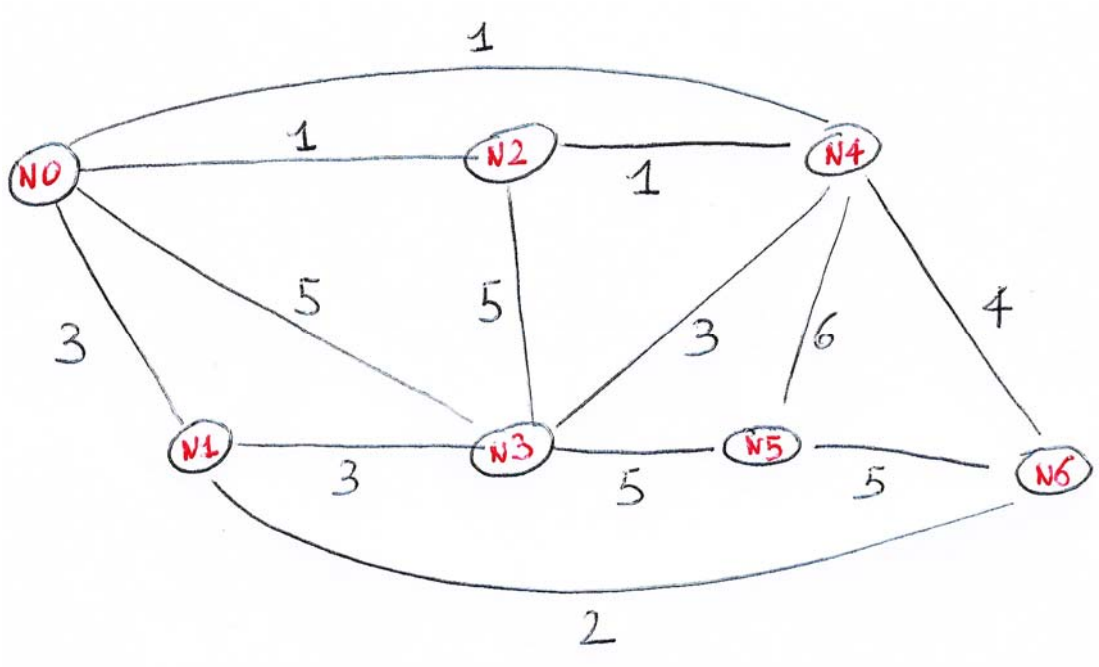
Esercizio 3 (7 punti)

Si consideri il seguente array di numeri interi:

{15, 49, 1, 15, 21, 13, 175, 35, 60, 14}

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array in modo non decrescente (compresa la fase iniziale di build-max-heap).

Esercizio 4 (14 punti)
Si consideri il grafo in figura.



a. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per l'albero dei cammini minimi a partire dal nodo sorgente **Nx**, dove x è ottenuto *prendendo l'ultima cifra del numero di matricola e calcolando il resto della divisione per 7.*
   Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.
b. Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Floyd-Warshall** per i cammini minimi tra tutte le coppie, mostrando ad ogni passo la matrice delle distanze.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Compito n° 1

Appello del 18 febbraio 2019

Esercizio 1 (6 punti)

Sia $T(n) = \begin{cases} T(n-1) + cn & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

1. **[2 punti]** Si dica, giustificano la risposta, se il **teorema generale** è utilizzabile per dare una stima esplicita (non ricorsiva) di T(n)
2. **[4 punti]** Ricordando che $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, si dia una stima esplicita (non ricorsiva) di T(n) facendo uso di un metodo a scelta.

Esercizio 2 (10 punti)

Scrivere un metodo (in pseudocodice o in Java)

`static int selezione (int[] a, int k)`

che, presi come parametro un array **a** di numeri interi e un intero **k**, restituisce il k-esimo valore più piccolo presente tra gli elementi dell'array (senza contare gli elementi ripetuti). Se k è minore di 1, viene restituito l'elemento più piccolo dell'array, mentre se k è maggiore del numero di elementi distinti presenti nell'array a, viene restituito l'elemento più grande dell'array. Il metodo **non** deve modificare l'array.

Ad esempio, se a={1, 8, 3, 1, 5, 10, 1, 4} e k=2, il metodo deve restituire 3 (poiché 3 è il secondo valore più piccolo presente nell'array.

Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo di ordinamento e/o di ricerca visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 5 punti*), dove n è la lunghezza dell'array a in input.

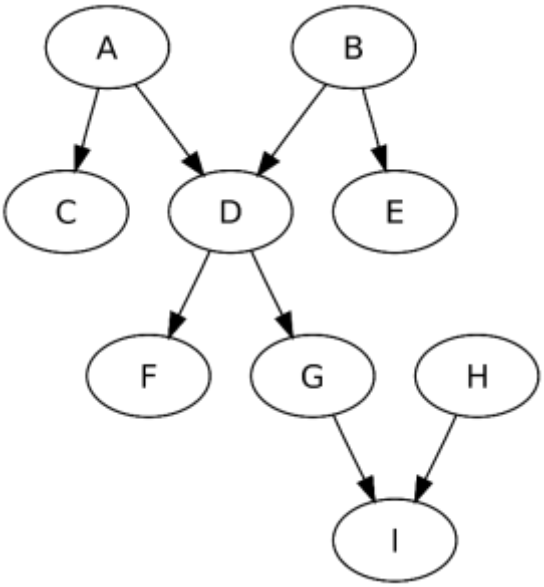
Esercizio 3 (9 punti)

Si consideri la sequenza di chiavi intere

5, 49, 10, 15, 27, 134, 73, 135, 6, 146

a. **[3 punti]** Mostrare **passo-passo** l'heap di minimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi sopra elencate
b. **[3 punti]** Mostrare la tabella hash di dimensione **17** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate
c. **[3 punti]** Mostrare la tabella hash di dimensione **17** con indirizzamento aperto (tramite hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'hashing doppio sia **16**).

Esercizio 4 (10 punti)
Si consideri il grafo diretto aciclico (DAG) in figura.



- **[2 punti]** Mostrare la rappresentazione del grafo tramite liste di adiacenza
- **[2 punti]** Mostrare la rappresentazione del grafo tramite matrice di adiacenza
- **[4 punti]** Eseguire **passo-passo** la visita in profondità del grafo, indicando il tempo di inizio e fine visita per ogni nodo (si fa riferimento all'algoritmo DFS che prende in input un grafo e richiama al suo interno l'algoritmo DFS-VISIT su vari nodi sorgente).
- **[2 punti]** Sfruttando la visita del passo precedente, individuare un ordinamento topologico dei nodi del DAG.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Compito n° 2

Appello del 18 febbraio 2019

Esercizio 1 (6 punti)

Sia $T(n) = \begin{cases} T(n-1) + cn & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

1. **[2 punti]** Si dica, giustificano la risposta, se il **teorema generale** è utilizzabile per dare una stima esplicita (non ricorsiva) di T(n)
2. **[4 punti]** Ricordando che $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, si dia una stima esplicita (non ricorsiva) di T(n) facendo uso di un metodo a scelta.

Esercizio 2 (10 punti)

Scrivere un metodo (in pseudocodice o in Java)

```
static int selezione (int[] a, int k)
```

che, presi come parametro un array **a** di numeri interi e un intero **k**, restituisce il k-esimo valore più piccolo presente tra gli elementi dell'array (senza contare gli elementi ripetuti). Se k è minore di 1, viene restituito l'elemento più piccolo dell'array, mentre se k è maggiore del numero di elementi distinti presenti nell'array a, viene restituito l'elemento più grande dell'array. Il metodo **non** deve modificare l'array.

Ad esempio, se a={1, 8, 3, 1, 5, 10, 1, 4} e k=2, il metodo deve restituire 3 (poiché 3 è il secondo valore più piccolo presente nell'array.

Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo di ordinamento e/o di ricerca visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 5 punti*), dove n è la lunghezza dell'array a in input.

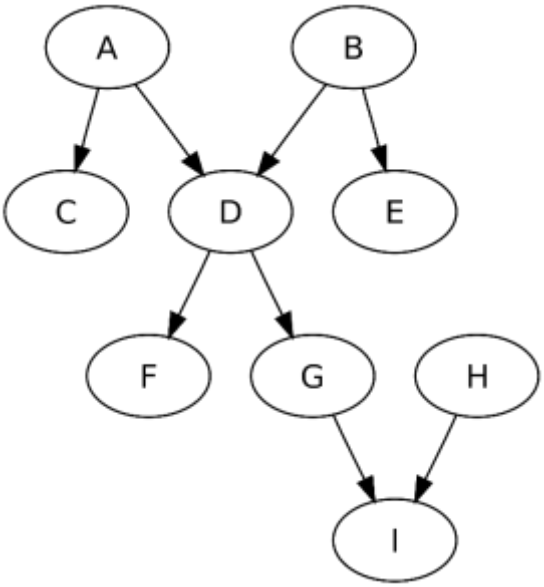
Esercizio 3 (9 punti)

Si consideri la sequenza di chiavi intere

8, 52, 13, 137, 18, 76, 138, 9, 149, 30

a. **[3 punti]** Mostrare **passo-passo** l'heap di minimo che si ottiene a partire dall'heap vuoto inserendo nell'ordine indicato le chiavi sopra elencate
b. **[3 punti]** Mostrare la tabella hash di dimensione **17** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate
c. **[3 punti]** Mostrare la tabella hash di dimensione **17** con indirizzamento aperto (tramite hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'hashing doppio sia **16**).

Esercizio 4 (10 punti)
Si consideri il grafo diretto aciclico (DAG) in figura.



- **[2 punti]** Mostrare la rappresentazione del grafo tramite liste di adiacenza
- **[2 punti]** Mostrare la rappresentazione del grafo tramite matrice di adiacenza
- **[4 punti]** Eseguire **passo-passo** la visita in profondità del grafo, indicando il tempo di inizio e fine visita per ogni nodo (si fa riferimento all'algoritmo DFS che prende in input un grafo e richiama al suo interno l'algoritmo DFS-VISIT su vari nodi sorgente).
- **[2 punti]** Sfruttando la visita del passo precedente, individuare un ordinamento topologico dei nodi del DAG.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Appello del 4 giugno 2019

Esercizio 1 (6 punti)

Sia $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + cn^2 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

1. **[2 punti]** Si dica, giustificano la risposta, se il **teorema generale** è utilizzabile per dare una stima esplicita (non ricorsiva) di T(n)
2. **[4 punti]** Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso di un metodo a scelta.

Esercizio 2 (10 punti)

Scrivere un metodo in Java

```
static boolean equivalenti (int[] a, int[] b)
```

che, presi come parametro due array **a** e **b** di numeri interi, restituisce *true* se e solo se a e b contengono gli stessi elementi (non tenendo in conto le ripetizioni degli elementi e/o l'ordine in cui essi compaiono).
Se a e b valgono entrambi *null*, viene restituito *true*; se solo uno dei due array vale *null*, viene restituito *false*.
Ad esempio, se a={1, 8, 3, 1, 5, 10, 1, 4} e b={8, 1, 5, 10, 4, 3, 3}, il metodo deve restituire *true*.
Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo di ordinamento e/o di ricerca visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 5 punti*), dove n è la lunghezza dell'array a in input.

Esercizio 3 (9 punti)

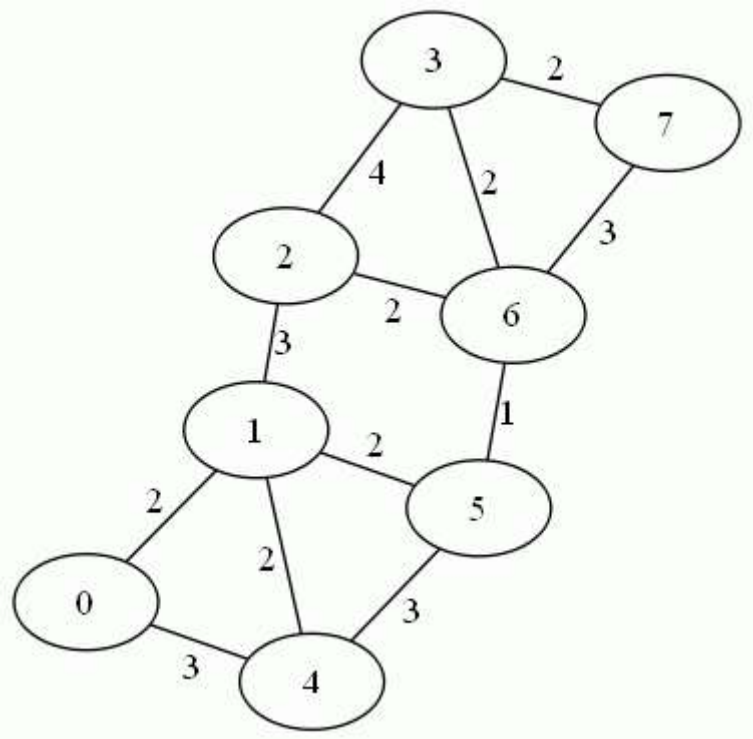
Si consideri la sequenza di chiavi intere

10, 20, 5, 50, 31, 14, 2, 6, 1, 8, 25

a. **[3 punti]** Mostrare **passo-passo** l'heap di minimo che si ottiene applicando la procedura **buildMinHeap** all'array contenente le chiavi nella sequenza sopra indicata.
b. **[3 punti]** Mostrare la tabella hash di dimensione **13** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate
c. **[3 punti]** Mostrare la tabella hash di dimensione **13** con indirizzamento aperto (tramite hashing doppio) che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate (si assuma che il secondo parametro della funzione dell'hashing doppio sia **12**).

Esercizio 4 (10 punti)

Si consideri il grafo non diretto in figura.



a. **[2 punti]** Mostrare la rappresentazione del grafo tramite liste di adiacenza
b. **[2 punti]** Mostrare la rappresentazione del grafo tramite matrice di adiacenza
c. **[6 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per l'albero dei cammini minimi a partire dal nodo sorgente **x**, dove x è ottenuto *prendendo l'ultima cifra del numero della propria matricola e calcolando il resto della divisione per 8.*
   Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Appello del 2 luglio 2019

Esercizio 1 (8 punti)

Sia $T(n) = \begin{cases} 8T(n/2) + cn^3 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso sia del **teorema generale** che del metodo di **sostituzione e induzione.**

Esercizio 2 (12 punti)

L'array delle ripetizioni di un dato array **a** di numeri interi è definito come un array avente come lunghezza il numero di elementi distinti dell'array **a**, e contenente in posizione **i** (i=0,1,…) il numero di ripetizioni nell'array **a** dell'(i+1)-esimo elemento più piccolo di **a**. Ad esempio, se **a**={8, 1, 3, 1, 5, 10, 1, 3}, il suo array di ripetizioni è {3,2,1,1,1} in quanto in **a** ci sono 5 elementi distinti, e l'elemento più piccolo di **a** (1) compare 3 volte, il secondo elemento più piccolo (3) compare 2 volte, e i rimanenti elementi compaiono solo una volta.
Scrivere un metodo in Java o in pseudocodice

```
static int[] ripetezioni (int[] a)
```

che, preso come parametro un array **a** di numeri interi, crea e restituisce il suo array delle ripetizioni.
Se a vale *null*, viene restituito *null*; se a è vuoto, viene restituito un array vuoto.
Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo di ordinamento e/o di ricerca visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 6 punti totali*), dove n è la lunghezza dell'array **a** in input.

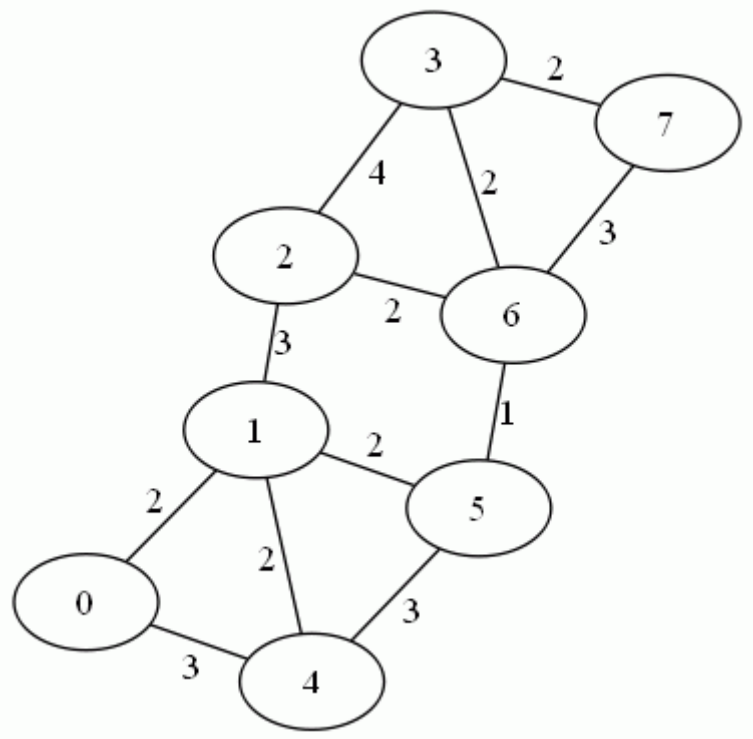
Esercizio 3 (6 punti)

Si consideri la sequenza di chiavi intere

11, 21, 15, 50, 32, 14, 2, 6, 1, 8, 27

a. **[3 punti]** Mostrare **passo-passo** l'heap di massimo che si ottiene applicando la procedura **buildMaxHeap** all'array contenente le chiavi nella sequenza sopra indicata.
b. **[3 punti]** Mostrare la tabella hash di dimensione **19** con liste di trabocco che si ottiene a partire dalla tabella vuota inserendo nell'ordine indicato le chiavi sopra elencate

Esercizio 4 (9 punti)
Si consideri il grafo non diretto in figura.



a. **[2 punti]** Mostrare la rappresentazione del grafo tramite liste di adiacenza
b. **[7 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente a partire dal nodo sorgente **x**, dove x è ottenuto *prendendo l'ultima cifra del numero della propria matricola e calcolando il resto della divisione per 8.*
   Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Appello del 9 settembre 2019

Esercizio 1 (6 punti)

Sia $T(n) = \begin{cases} 9T(n/2) + cn^3 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso del **teorema generale**.

Esercizio 2 (10 punti)

Scrivere un metodo in Java o in pseudocodice

```
static void ordinamentoRelativo (int[] a, int rif)
```

che, preso come parametro un array **a** di numeri interi e un intero **rif**, ordina gli elementi di **a** in modo non decrescente rispetto alla loro distanza (in valore assoluto) dall'intero **rif**.

Ad esempio, se **a**={8, 1, 3, 1, 5, 10, 1, 3} e **rif=**4, a dovrà essere ordinato così: {3, 5, 3, 1, 1, 1, 8, 10}.

Si analizzi la complessità temporale del metodo proposto: tale metodo deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 6 punti totali*), dove n è la lunghezza dell'array **a** in input.

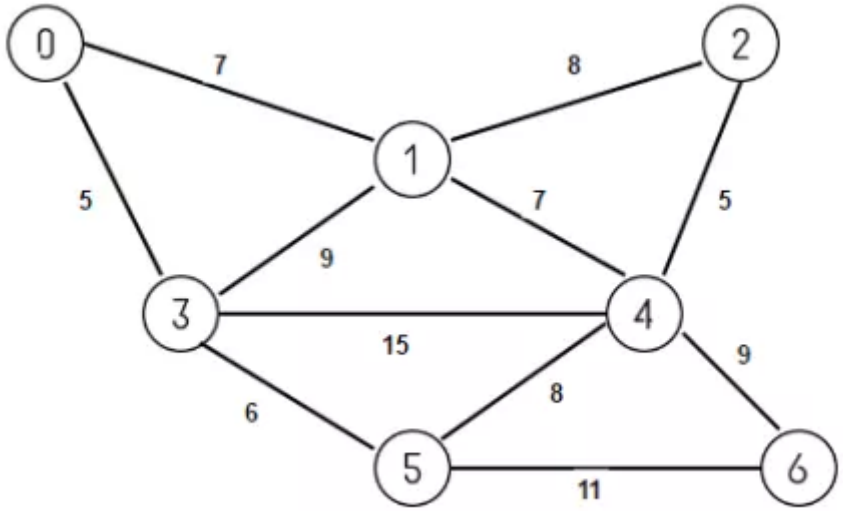
Esercizio 3 (10 punti)

Si consideri il seguente array di numeri interi:

{2, 20, 35, 12, 30, 1, 6, 55, 4, 22, 18, 7}

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array in modo non decrescente (compresa la fase iniziale di build-max-heap).

Esercizio 4 (9 punti)
Si consideri il grafo non diretto in figura.



a. **[3 punti]** Mostrare la rappresentazione del grafo tramite liste di adiacenza
b. **[6 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per l'albero dei cammini minimi a partire dal nodo sorgente **x**, dove x è ottenuto *prendendo l'ultima cifra del numero della propria matricola e calcolando il resto della divisione per 7.*
   Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Appello del 18 febbraio 2020

Esercizio 1 (5 punti)

Sia $T(n) = \begin{cases} 10T(n/3) + cn^3 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso del **teorema generale**.

Esercizio 2 (10 punti)

Scrivere un metodo in Java o pseudocodice

`static void ordinamentoModulo (int[] a, int k)`

che, preso come parametro un array **a** di numeri interi, ordina gli elementi di **a** in modo

- prioritariamente non decrescente rispetto al valore modulo k
- a parità di valore modulo k, in modo non crescente.

Ad esempio, se **a={8, 1, 3, 1, 5, 10, 2, 6}**, e k=3, a dovrà essere ordinato così: {6, 3, 10, 1, 1, 8, 5, 2}.

Si analizzi la complessità temporale del metodo proposto: tale metodo deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 6 punti totali*), dove n è la lunghezza dell'array **a** in input.

Suggerimento: Si modifichi un algoritmo noto nella parte in cui effettua il confronto tra due elementi dell'array, avvalendosi di un metodo ausiliario che effettua tale confronto.

Esercizio 3 (7 punti)

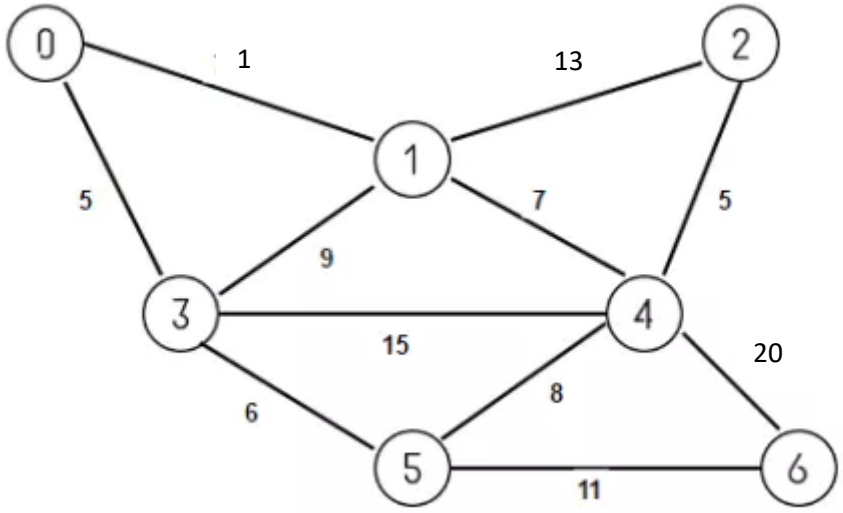
Si consideri il seguente array di numeri interi:

{5, 49, 10, 51, 11, 13, 75, 15, 6, 4}

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array in modo non decrescente (compresa la fase iniziale di build-max-heap).

Esercizio 4 (13 punti)

Si consideri il grafo non diretto in figura.



a. **[6 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **2**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

b. **[7 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Floyd-Warshall** per i cammini minimi tra tutte le coppie, calcolando ad ogni passo la matrice delle distanze e la matrice dei padri π. Si dica se, nel grafo considerato, esiste, dal punto di vista computazionale, un modo più efficiente per risolvere il problema del calcolo dei cammini minimi tra tutte le coppie.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Cognome e Nome _____

Matricola _____

Algoritmi e strutture dati A.A. 2018/2019

Appello del 12 settembre 2022

Esercizio 1 (5 punti)

Sia $T(n) = \begin{cases} 9T(n/3) + cn^2 & se\ n > 1 \\ d & se\ n = 1 \end{cases}$

con *c* e *d* costanti.

Si dia una stima esplicita (non ricorsiva) di T(n) facendo uso del **teorema generale**.

Esercizio 2 (12 punti)

L'array delle ripetizioni di un dato array **a** di numeri interi è definito come un array avente come lunghezza il numero di elementi distinti dell'array **a**, e contenente in posizione **i** (i=0,1,…) il numero di ripetizioni nell'array **a** dell'(i+1)-esimo elemento più piccolo di **a**. Ad esempio, se **a**={8, 1, 3, 1, 5, 10, 1, 3}, il suo array di ripetizioni è {3,2,1,1,1} in quanto in **a** ci sono 5 elementi distinti, e l'elemento più piccolo di **a** (1) compare 3 volte, il secondo elemento più piccolo (3) compare 2 volte, e i rimanenti elementi compaiono solo una volta.
Scrivere un metodo in Java o in pseudocodice

```
static int[] ripetezioni (int[] a)
```

che, preso come parametro un array **a** di numeri interi, senza modificare **a**, crea e restituisce il suo array delle ripetizioni.
Se a vale *null*, viene restituito *null*; se a è vuoto, viene restituito un array vuoto.
Si analizzi la complessità temporale del metodo proposto: tale metodo può richiamare qualsiasi algoritmo di ordinamento e/o di ricerca visto a lezione e deve avere complessità temporale **O(n log n)** nel caso peggiore (*soluzioni con complessità temporale peggiore danno luogo a una valutazione minore, pari a un massimo di 6 punti totali*), dove n è la lunghezza dell'array **a** in input.

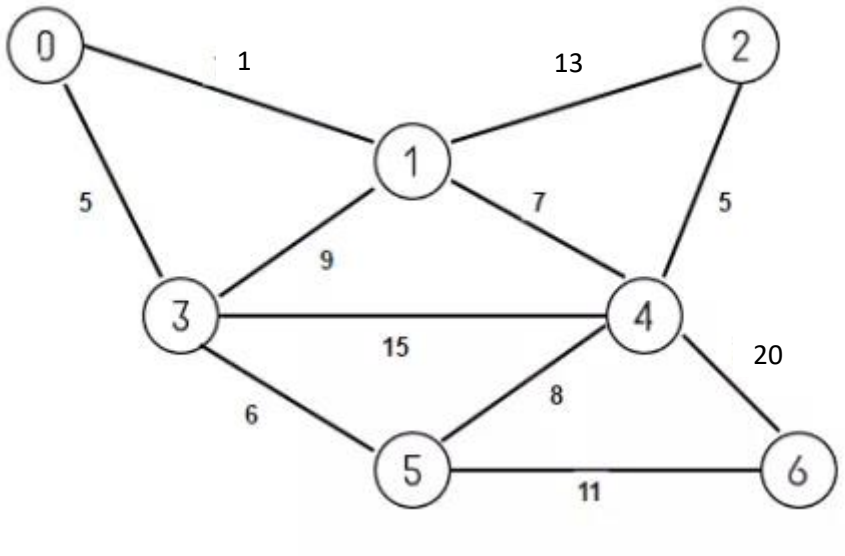
Esercizio 3 (7 punti)

Si consideri il seguente array di numeri interi:

{51, 4, 12, 5, 11, 100, 7, 15, 6, 1}

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array in modo non decrescente (compresa la fase iniziale di build-max-heap).

Esercizio 4 (10 punti)
Si consideri il grafo non diretto in figura.



a. **[5 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente, partendo dal nodo sorgente **3**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

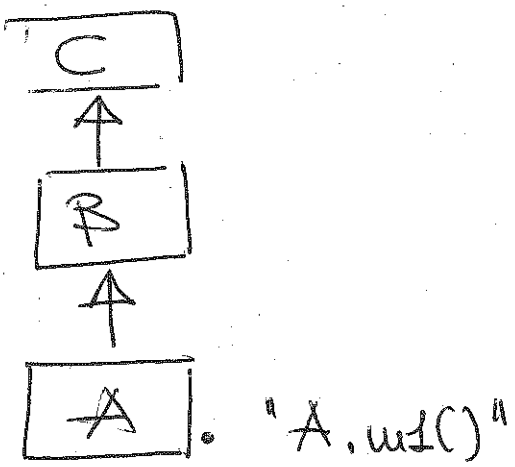
b. **[5 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **3**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo.

Regole per lo svolgimento della prova scritta:

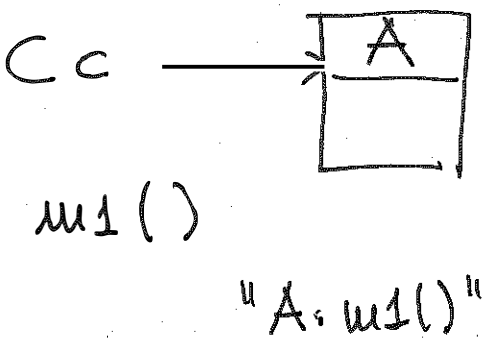
- Per svolgere il compito si hanno a disposizione **120** minuti.
- Scrivere **subito** nome, cognome, matricola e numero del compito su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri, dispense o qualsiasi altro materiale.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco il proprio libretto (o altro documento di identità).

Quiz: Costrutti OO 1

1) Date le seguenti classi:



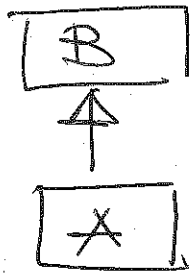

```
public class C {
public void m1() {
System.out.println("C.m1()");
}
}
public class B extends C{
public void m1() {
System.out.println("B.m1()");
}
}
public class A extends B{
public void m1() {
System.out.println("A.m1()");
}
}
public class P
{
        public static void test(C c){
           c.m1();
        }
}
public class MainClass{
public static void main(String args[]){
        C c=new A();
        P.test(c);
        }
}
```



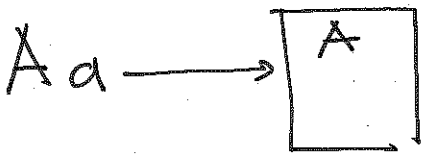
Qual è il risultato della compilazione ed esecuzione del programma ?

1. Errore di compilazione
2. Viene stampato "A.m1()"
3. Viene stampato "B.m1()"
4. Viene stampato "C.m1()"

2) Date le seguenti classi:




```
public class B
{
private int z;
        public B(int m){
           z=2*m;
        }
        public B(float m){
           z=3*(int) m;
        }
        public int getZ() {return z;}
}
public class A extends B
{
private int x;
private int y;
        public A(int i,int j){
           super((float)i/4);
           x=i;
           y=j;
        }
        public A(int i){
           this(i,10);
        }
        public void print(){
           System.out.println(x+y+getZ());
        }
}
public class MainClass{
public static void main(String args[]){
                A a=new A(5);
                a.print();
        }
}
```



Cosa viene stampato ?:

5. 17
6. 18
7. 15
8. 0

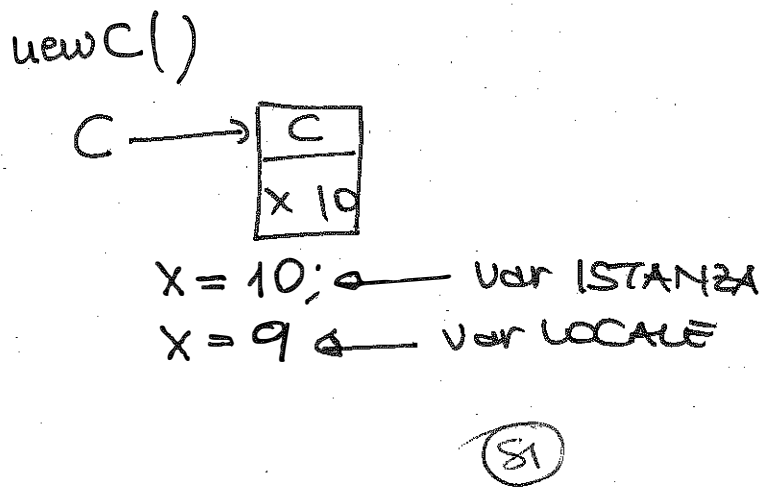
this (5,10)
super((float) 5/4)
z = 3
x = 5 y = 10
3+5+10 → 18

Quiz: Costrutti OO 1	Corso: "Programmazione orientata agli oggetti"	Docente: Andrea Bei

3) Date le seguenti classi:

```
public class C {
private int x;
public C(){
x=10;
}
public void m1(int h) {
System.out.print("x="+x+" ");
  {
  int x=3*h;
  System.out.println("x="+x);
 }
}
}
public class MainClass{
public static void main(String args[]){
      (new C()).m1(3);

          }
}
```

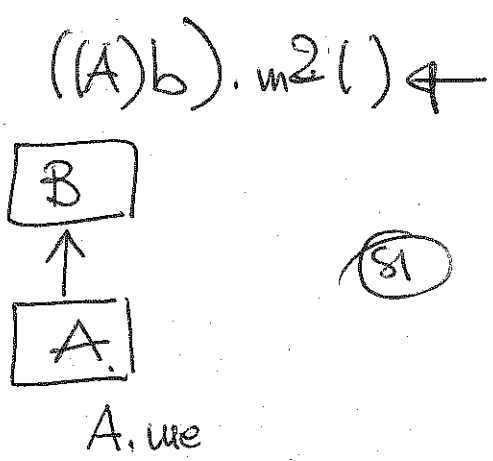


Qual è il risultato della compilazione ed esecuzione del programma ?

1. Errore di compilazione
2. stampa x=10 x=9
3. stampa x=10 x=10
4. stampa x=0 x=9

4)Date le seguenti classi:

```
public class B{
public void m1(){
System.out.println("B.m1()");
}
}
public class A extends B {
          public void m2(){
             System.out.println("A.m2()");
          }
}
public class MainClass{
public static void main(String args[]){
      B b=new A();
      b.m2();
          }
}
```



Qual è il risultato della compilazione ed esecuzione del programma ?

1. stampa: "A.m2()"

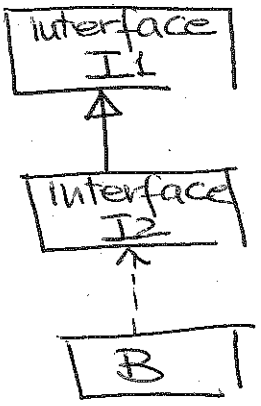
2. stampa: "B.m1()"
3. errore di compilazione
4. errore a tempo di esecuzione

5) Date le seguenti classi:

```
public abstract class B{
public abstract void m1();
public abstract void m2();
}
public class A extends B{
        public void m2(){
           System.out.println("A.m2()");
        }
}
public class MainClass{
public static void main(String args[]){
     B b=new A();
     b.m2();
}
}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. stampa: "A.m2()"
2. stampa: ""
3. errore a tempo di esecuzione
4. errore di compilazione



S1

Non implementa m1()
ERRORE a tempo di compilazione

6) Date le seguenti classi:

```
public interface Int1 {
        public void m1();
}
public interface Int2 extends Int1{
        public void m2();
}
public class B implements Int2{
        public void m2(){
        System.out.println("B.m2()");
        }
}
public class MainClass{
public static void main(String args[]){
     B b=new B();
     b.m2();
        }
}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. stampa: "B.m2()"
2. stampa: ""
3. Errore di compilazione
4. errore a tempo di esecuzione

7) Date le seguenti classi:

```
public class B
{
private int x;
public B(){
x=10;
}
public int getX(){
        return x;
        }
}
public class A extends B
{
        public void m2(){
         System.out.println("x= "+x);
        }
}
public class MainClass{
public static void main(String args[]){
    (new A()).m2();
        }
}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Errore di compilazione
2. stampa x=10
3. stampa x=0

8) Date le seguenti classi:

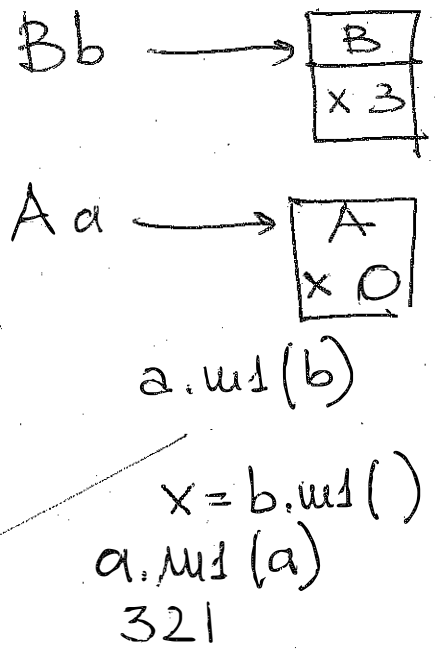
```
public class B
{
<??????> int x;
public B (){
x=10;
}
public int getX() {
        return x;
        }
}
public class A extends B
{
        public void m2() {
        x=0;
        }
}
public class C
{
        public void m2() {
        B b=new B() ;
        b.x=2;
        }
}
```

Come deve essere dichiarato l'attributo x di B affinchè la compilazione della classe A vada a buon fine e la compilazione della classe C dia errore ?

1. protected
2. public
3. private

Quiz: Costrutti OO 1	Corso: "Programmazione orientata agli oggetti"	Docente: Andrea Bei

9) Date le seguenti classi:




```
public class B {
int x;
public B(int i){
x=i;
}
public int m1(){
return x;
}
}
public class A
{
int x;
public void m1(B b) {
x=b.m1();
m1(this);
}
public void m1(A a) {
System.out.print(x--);
if (x>0) a.m1(this);
}
}
public class MainClass{
public static void main(String args[]){
        B b= new B(3);
                A a=new A();
                a.m1(b);
        }
}
```

Cosa viene stampato ?:

1. nulla
2. 3
3. 1
4. 321

10) Date le seguenti classi:

```
public class A{
private B b;
public A(B b){
this.b=b;
}
public void m1() {
b=new B();
}
public int getValue(){
return b.getValue();
}
}
public class B{
private int y ;
        public B(){
                y=10;
        }
        public int getValue(){
                return y;
        }
        public void setValue(int j){
                y=j;
        }
}
public class MainClass{
public static void main(String args[]){
        B b=new B();
        A a=new A(b);
        b.setValue(15);
        System.out.print(a.getValue());
        a.m1();
        System.out.print(a.getValue());
        }
}
```

Cosa viene stampato ?

1. nulla
2. 1010
3. 1015
4. 1510
5. 1515

11) Data la seguente classe

```
1: public class Q
2: {
3:    int maxElements;
4:
5:    void Q()
6:    {
7:       maxElements = 100;
8:       System.out.println(maxElements);
9:    }
10:
11:   Q(int i)
12:   {
13:      maxElements = i;
14:      System.out.println(maxElements);
15:   }
16:
17:   public static void main(String[] args)
18:   {
19:      Q a = new Q();
20:      Q b = new Q(999);
21:   }
22: }
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa 100 e 999.
2. Stampa 999 e 100.
3. Errore di compilazione a linea 3 (variabile maxElements non inizializzata).
4. Errore di compilazione a linea 19.

12) Date le seguenti classi:

```
1  class A
2  {
3     public int r=10;
4     void callme()
5     {
6        System.out.println("A");
7     }
8  }
9  class B extends A {
10
11    public void callme()
12    {
13       System.out.println("B");
14    }
15
16 }
17 class Q
18 {
19    public static void main(String args[])
20    {
21       B a = new B();
22       a.callme();
23       System.out.println(b.r);
24    }
25 }
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Errore di compilazione
2. Stampa B e 10
3. Stampa A e 10

Quiz: Costrutti OO 1	Corso: "Programmazione orientata agli oggetti"	Docente: Andrea Bei

13) Date le seguenti classi:

```
1  class Messaggio {
2  String text;
3  public Messaggio() { text ="Hello1"; }
4  }
5  class Super {
6   Messaggio msg;
7     public Super() { msg=new Messaggio(); }
8  }
9  class Ered extends Super
10 {
11    public static void main(String arg[])
12    {
13       Ered i=new Ered();
14       i.print();
15    }
16    public void print()
17    {
18       //Inserire il codice QUI !
19    }
20 }
```

Quale dei seguenti è il modo più semplice di stampare il valore della variabile text a linea 18 ?

1. System.out.println(msg.text);
2. System.out.println(super.msg.text);
3. System.out.println(Messaggio.text);
4. System.out.println(text);

14) Quali delle seguenti dichiarazioni di classe è corretta ?

```
1. public class Fred {
     public int x = 0;
     public Fred (int x) {
       this.x = x;
     }
   }

2. public class fred
     public int x = 0;
     public fred (int x) {
       this.x = x;
     }
   }

3. public class Fred extends MiaClasseBase, MiaAltraClasseBase{
     public int x = 0;
     public Fred (int xval) {
```

Non può ereditare da 2 classi

```
    x = xval;
  }
}
```

15) Date le seguenti classi:

```
1.  class Veicolo {
2.  public void guida() {
3.  System.out.println("Veicolo: guida");
4.  }
5.  }
6.  class Automobile extends Veicolo {
7.  public void guida() {
8.  System.out.println("Automobile: guida");
9.  }
10. }
11. public class Test {
12. public static void main (String args []) {
13. Veicolo v;
14. Automobile c;
15. v = new Veicolo();
16. c = new Automobile();
17. v.guida();
18. c.guida();
19. v = c;
20. v.guida();
21. }
22. }
```

Quali delle seguenti affermazioni è vera ?

1. Errore di compilazione su v= c;
2. Errore a tempo si esecuzione su v= c;
3. Stampa:

   Veicolo: guida
   Automobile: guida
   Automobile: guida

5. Stampa:

   Veicolo: guida
   Automobile: guida
   Veicolo: guida

16) Dove, in un costruttore, deve essere inserita l'istruzione super per chiamare il costruttore della superclasse ?

1. Ovunque
2. Deve essere la prima istruzione del costruttore

3. Deve essere l'ultima istruzione del costruttore
4. L'istruzione super non può essere inserita nel costruttore

17) Da quale istruzione nel codice seguente l'oggetto Impiegato("Roberto",48) può essere eliminato dal garbage collector ?

```
1. public class Test {
2. public static void main (String args []) {
3. Impiegato e = new Impiegato("Roberto", 48);
4. e.calcolaPaga();
5. System.out.println(e.stampaDettagli());
6. e = null;
7. e = new Impiegato("Federica", 36);
8. e.calcolaPaga();
9. System.out.println(e.stampaDettagli());
10. }
11. }
```

qui

1. Linea 10
2. Linea 11
3. Linea 6
4. Linea 8
5. Mai

18) Data la seguente classe:

```
public class A {int i1; void m1() {}}
```

Quali delle seguenti affermazioni è vera ?

1. La classe A eredita dalla classe Object. ✗
2. Il compilatore inserisce implicitamente un costruttore di default. ✗
3. Il costruttore di default accetta un parametro per ogni attributo di A. no
4. Il costruttore di default invoca il costruttore della superclasse ✗ (Object)

19) Date le seguenti classi:

```
class A {A(int i) {}}  // 1
class B extends A {}   // 2
```

B() { super(); }

Quali delle seguenti affermazioni sono vere ?

1. Il compilatore crea un costruttore di default per la classe A no
2. Il compilatore crea un costruttore di default per la classe B sì ma dà errore a tempo di COMPILAZIONE
3. Errore di compilazione a linea 1. no
4. Errore di compilazione a linea 2. sì

20) Quali delle seguenti affermazioni è vera ?

1. Il compilatore crea un costruttore di default solo se non esiste già un costruttore
2. Il costruttore di default ha zero argomenti.
3. Se la classe A ha una superclasse allora il costruttore di default di A invoca il costruttore a zero argomenti della superclasse.

21) Dato il seguente codice :

```
class Q {
 public static void main (String[] args) {
   private int x = 1;
   System.out.println(x);
}}
```

Quali delle seguenti affermazioni sono vere ?

1. Stampa: 1
2. Errore di esecuzione
3. Errore di compilazione
4. Nessuna

22) Data la seguente classe:

```
class Rosso {
 public int a;
 public static int b;
 public static void main (String[] in) {
   Rosso r1 = new Rosso(), r2 = new Rosso();
   r1.a++; r1.b++;
   System.out.print(r1.a+", "+r1.b+", "+r2.a+", "+r2.b);
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: 0, 0, 0, 0
2. Stampa: 0, 1, 1, 1
3. Stampa: 1, 1, 1, 0
4. Stampa: 1, 1, 0, 1
5. Stampa: 1, 1, 0, 0
6. Stampa: 1, 1, 1, 1
7. Compile-time error
8. Run-time error
9. None of the above

23) Quali delle seguenti affermazioni sono vere ?

1. Un metodo final non può essere sovrascritto
2. Tutti i metodi dichiarati in una classe final sono implicitamente final
3. Tutti i metodi dichiarati in una classe final devono essere esplicitamente dichiarati final altrimenti avviene un errore di compilazione

24) Data la seguente classe:

```
class Q {
  static int m1(int x) {return ++x ;}
  public static void main (String[] args) {
    int x = 1;
    int y = m1(x);
    System.out.println(x + "," + y);
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: 1,1
2. Stampa: 1,2
3. Stampa: 2,1
4. Stampa: 2,2
5. Errore di esecuzione
6. Errore di compilazione
7. Nessuna delle precedenti

25) Data la seguente classe:

```
class Q {
  private static int x=1;
  static void m1(int i) {x++; i++;}
  public static void main (String[] args) {
    int y=3; m1(y);
    System.out.println(x + "," + y);
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: 1,3
2. Stampa: 2,3
3. Stampa: 1,4
4. Stampa: 2,4
5. Errore di esecuzione
6. Errore di compilazione
7. Nessuna delle precedenti

26) Data la seguente classe:

```
class Q {
 private String name;
 public Q(String name) {this.name = name;}
 public void setName(String name) {this.name = name;}
 public String getName() {return name;}
 public static void m1(Q r1, Q r2) {
  r1.setName("Uccello");
  r2 = r1;
 }
 public static void main (String[] args) {
  Q animale1 = new Q("Cane");
  Q animale2 = new Q("Gatto");
  m1(animale1,animale2);
  System.out.println(animale1.getName() + "," + animale2.getName());
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: Cane,Gatto
2. Stampa: Cane,Uccello
3. Stampa: Uccello,Gatto
4. Stampa: Uccello,Uccello
5. Errore di esecuzione
6. Errore di compilazione
7. Nessuna delle precedenti

27) Data la seguente classe:

```
class Q {
 private String name;
 public Q(String name) {this.name = name;}
 public void setName(String name) {this.name = name;}
 public String getName() {return name;}
 public static void m1(Q animale1, Q animale2) {
  animale1 = new Q("Pesce");
  animale2 = null;
 }
 public static void main (String[] args) {
  Q animale1 = new Q("Cane");
  Q animale2 = new Q("Gatto");
  m1(animale1,animale2);
  System.out.println(animale1.getName() + "," + animale2.getName());
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: Cane,Gatto
2. Stampa: Cane,Pesce
3. Stampa: Pesce,Gatto
4. Stampa: Pesce,Pesce
5. Errore di compilazione
6. Errore di esecuzione
7. Nessuna delle precedenti

28) Data la seguente classe:

```
class Q {
 static void m1(int[] i1, int[] i2) {
  int[] i3 = i1; i1 = i2; i2 = i3;
 }
 public static void main (String[] args) {
  int[] i1 = {1}, i2 = {3}; m1(i1, i2);
  System.out.print(i1[0] + "," + i2[0]);
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: 1,1
2. Stampa: 1,3
3. Stampa: 3,1
4. Stampa: 3,3
5. Errore di esecuzione
6. Errore di compilazione
7. Nessuna delle precedenti

29) Date le seguenti classi:

```
class A {
 void m1() {System.out.print("A.m1");}
}
class B extends A {
 void m1() {System.out.print("B.m1");}
 static void m1(String s) {System.out.print(s+",");}
}
class C {
 public static void main (String[] args) {
  B.m1("main");
  (new B()).m1();
}}
```

Qual è il risultato della compilazione ed esecuzione del programma ?

1. Stampa: main,B.m1
2. Errore di compilazione

3. Errore a Run-time
4. Nessuna delle precedenti

30) Quali delle seguenti affermazioni è vera ?

1. La relazione tra una classe e la superclasse è una relazione di composizione/aggregazione
2. La relazione tra una classe e la superclasse è una relazione di ereditarietà
3. La relazione tra una classe e un oggetto referenziato da un attributo della classe è una relazione di composizione/aggregazione
4. La relazione tra una classe e un oggetto referenziato da un attributo della classe è una relazione di ereditarietà

31) Date le seguenti classi:

```
class Zampa{}
abstract class Animale {
  public abstract void mangia() ;
  public abstract void dorme() ;
}
class Cane extends Animale {
  Zampa sinistraAnteriore;
  Zampa destraAnteriore;
  Zampa sinistraPosteriore;
  Zampa destraPosterire;
   Cane()
   {
   sinistraAnteriore= new Zampa();
   destraAnteriore=new Zampa();
   sinistraPosteriore= new Zampa();
   destraPosteriore=new Zampa();
   }
  public void mangia() {}
  public void dorme() {}
}
class Gatto extends Cane {
  public void disobbediente () {}
  public void siArrampicaSugliAlberi() {}
}
```

Quali delle seguenti affermazioni non è vera ?

1. Un Gatto eredita 4 istanze di Zampa dalla superclasse Cane
2. Un Gatto può mangiare e dormire
3. Un Gatto si arrampica sugli alberi
4. La relazione tra Cane e Animale è un esempio di uso approrpiato dell'ereditarietà
5. La relazione tra Gatto e Cane è un esempio di uso inappropriato dell'ereditarietà
6. Nessuna delle precedenti.

Soluzioni: Costrutti OO

Quesito	Soluzione	Commenti
1	2	Per il binding dinamico viene eseguita l'implementazione del metodo m1 della classe A
2	2	L'ordine di esecuzione è il seguente: • Invocazione del costruttore di A con un solo parametro (5) • Invocazione (tramite this) del costruttore di A con 2 parametri (5,10) • Invocazione del costruttore di B con il parametro di tipo float (5/4). Il cast a (int) rende 5/4 uguale a 1. Quindi z=3 • Esecuzione del metodo print 10+5+3
3	2	All'interno del metodo m1, per le regole sulla visibilità delle variabili, la prima istanza delle variabile x è l'attributo, la seconda è il valore della variabile dichiarata nel blocco interno al metodo m1 e definito dalle parentesi graffe.
4	3	L'errore di compilazione è dovuto al fatto che il tipo di dato B non definisce l'operazione m2. La variabile b punta però ad un oggetto il cui tipo di dato associato definisce l'operazione m2 (oggetto della classe A). Se si vuole invocare il metodo m2 si deve effettuare un cast e quindi sostituire all'istruzione b.m2();l'istruzione ((A) b).m2(); Quest'ultima istruzione non genera errori di compilazione.
5	4	La classe A deve implementare sia m1 che m2, altrimenti deve essere dichiarata astratta.
6	3	L'interfaccia Int2 ha i metodi m1 (ereditato) e m2. B dichiara di implementare Int2 quindi deve implementare sia m1 che m2 altrimenti si ha un errore di compilazione.
7	1	L'attributo x della classe A è di tipo private quindi non è visibile dalla classe B
8	1	Un attributo con modificatore di visibilità protected è visibile solo dalle sottoclassi.
9	4	Il metodo m1 della classe A è ricorsivo e chiama se stesso decrementando l'attributo x ad ogni chiamata fino a che x non diventa uguale a 0.
10	4	La sequenza delle operazioni è questa: • Viene creato un oggetto di tipo B (attributo y = 10) • L'oggetto di tipo B viene passato alla classe A • b.setValue(15) imposta y=15 • La prima stampa di a.getValue() restituisce 15 • a.m1() crea un altro oggetto di tipo B che assegna all'attributo b. Invocando di nuovo il costruttore di B ho nuovamente y=10 • La seconda stampa di a.getValue() restituisce 10
11	4	"void Q()" non è un costruttore perchè specifica il tipo ritornato (void) e nei costruttori questo non deve essere specificato. Dato che la classe Q ha già un costruttore "Q(int i)" non viene creato il costruttore di default. Quindi la classe non ha un costruttore a zero argomenti e quando viene invocato costruttore di questo tipo (riga 19) viene emesso un errore a tempo di compilazione.
12	1	La variabile b non è stata dichiarata

13	1	La variabile msg è visibile dalla classe Ered (friendly). Posso scrivere msg.text perché le classi sono tutte nello stesso package
14	1	La 2 non è corretta perché mancano le parentesi, la 3 non è corretta perché Fred eredita da due classi e in Java questo non è possibile,
15	3	• A riga 17 l'istruzione v.guida (); stampa: "Veicolo:guida" • A riga 18 l'istruzione c.guida(); stampa: "Automobile: guida" (nella classe Automobile viene fatto l'override del metodo guida) • A riga 19 l'istruzione v=c (possibile per il polimorfismo) fa si che il riferimento v punti allo stesso oggetto puntato dal riferimento c. Quindi v punta ad un oggetto di tipo Automobile. • A riga 20 l'istruzione v.guida(); provoca l'invocazione del metodo guida di Automobile() (binding dinamico) e quindi la stampa di "Automobile: guida"
16	2	Se non si inserisce come prima istruzione viene restituito un errore a tempo di compilazione.
17	3	A partire dalla linea 6 (*e=null*) il riferimento *e* non punta più all'oggetto considerato. Inoltre non esistendo, oltre ad *e*, nessun altro riferimento a tale oggetto questo può essere distrutto dal garbage collector che libera la memoria occupata.
18	1,2,4	• La classe A eredita implicitamente dalla classe Object. Questo vale in Java per tutte le classi che non ereditano esplicitamente da una classe. Infatti tutte le classi (anche le proprie) ereditano direttamente o indirettamente dalla classe Object. • Il compilatore inserisce un costruttore di default se la classe non ne definisce uno. • Il costruttore di default invoca il costruttore a zero argomenti della superclasse.
19	4	L'errore di compilazione a riga 2 deriva dal fatto che il compilatore inserisce un costruttore di default nella classe B. Tale costruttore invoca il costruttore a zero argomenti della superclasse. Dato che nella superclasse un costruttore di questo tipo non esiste, viene restituito un errore di compilazione a riga 2.
20	1,2,3	
21	3	Il modificatore di visibilità "private" può essere specificato per la dichiarazione degli attributi di una classe; non ha senso per le variabili locali di un metodo.
22	4	• r1.a vale 1 perché l'attributo a dell'oggetto r1 è stato incrementato dall'istruzione r1.a++ • r1.b vale 1 perché l'attributo statico b della classe Rosso è stato incrementato dall'istruzione r1.b++ • r2.a vale 0 perché l'attributo a dell'oggetto r1 non è stato ma inizializzato • r2.b vale 1 ed è lo stesso valore di r1.b dato che b è statico ed è un attributo della classe Rosso e non dei suoi oggetti. Quindi tutti gli oggetti della classe Rosso "vedono" lo stesso valore.
23	1,2	Una classe final non può essere ereditata e quindi i suoi metodi non possono essere sovrascritti ed in questo senso sono implicitamente final.
24	2	