```java
public class Esame4febbraio {
	public static void main(String[] args) {
	//ESERCIZIO 1b
	// Cosa stampa il seguente frammento di codice Java?
	System.out.println ("STAMPA ESERCIZIO 1b");
	int conto=0;
	int[] a = {-5, 13, -6, 5, 10, -4};
	int n=-30, i;
	for (i=a.length-1; i>0 && conto <=2; i--) {
		if (a[i]*a[i-1]<=n){
			conto++;
			System.out.println(a[i]);
		}
	}
	System.out.println(conto);

	//ESERCIZIO 1a
	//Cosa stampa il seguente programma Java (rebus è scritto sotto)
	System.out.println ();
	System.out.println ("STAMPA ESERCIZIO 1a");
	System.out.println(rebus(8));
	System.out.println(rebus(14));
	System.out.println(rebus(9));
	System.out.println(rebus(17));


	// Stampa del risultato di 2 chiamate dei metodi listaPicchi
	// e listaPicchi2 (altra versione di listaPicchi).
	// Le 10 righe che seguono non andavano scritte
	// Solo per farvi vedere il comportamento dei metodi.

	int []b={1,3,5,4,6,5,6} ;
	int [] picchib=listaPicchi(b);
	int [] picchib2=listaPicchi2(b);
	stampaArray(picchib);
	stampaArray(picchib2);
	int []c={4,3,5,4,6,7,6};
	int [] picchic=listaPicchi(c);
	stampaArray(picchic);
	int [] picchic2=listaPicchi2(c);
	stampaArray(picchic2);

	// Stampa del risultato di 1 chiamata del metodi comuniDistinti.
	// Le 3 righe che seguono non andavano scritte
	// Solo per farvi vedere il comportamento dei metodi.

	int []d={2,5,3,4,2,3,5};
```

```
	int []e={2,2,3,3,6,3,6};
	System.out.println(comuniDistinti (d,e));
	}

/*************************************************************/
	/* stampaArray, mergeSort e binarySearch non si devono scrivere
	 * nel compito. Servono per provare i metodi listaPicchi e
	 * comuniDistinti
	*/

	static void stampaArray (int[]a) {
		if (a==null) {
			System.out.println("Array non inizializzato");
			return;
		}
		System.out.print("{ ");
		for (int i=0; i<a.length; i++){
			System.out.print(a[i]);
			if (i<a.length-1) System.out.print(", ");
		}
		System.out.println(" }");
	}

	static void mergeSort(int[]a){
		if (a==null|| a.length==0) return;
		mergeSort(a,0, a.length -1);
	}
	static void mergeSort(int[]a, int l, int r){
			//CASO BASE
		if (l==r) return;
		int p=(r+l)/2; //  l<=p<r
		mergeSort(a,l,p);
		mergeSort(a, p+1,r);
		merge(a,l,p,r);
	}

	static void merge(int[]a, int l, int p, int r){
		int n1=p-l+1;
		int n2=r-p;
		int[] sin= new int[n1];
		int[] des= new int[n2];
		for (int i=0; i<n1; i++) {sin[i]=a[l+i];}
		for (int i=0; i<n2; i++) {des[i]=a[p+1+i];}
		int i=0;  // per scorrere sin
		int j=0;  // per scorrere des
		for(int k=l; k<=r;k++){
			if (i==n1)  {a[k]=des[j]; j++;}
```

```
            else {if (j==n2) {a[k]=sin[i]; i++;}
                  else {if (sin[i]<des[j]) {a[k]=sin[i]; i++;}
                        else {a[k]=des[j]; j++;}
                        }
            }
      }
}

static int binarySearch (int[] a, int v){
      if (a==null || a.length==0) return -1;
      return binarySearch(a,v,0,a.length-1);
}

static int binarySearch (int[] a, int v,int l,int r){
      //CASO BASE
      if (l==r) {
            if (a[l]==v) return l; else return -1;
      }
      int p=(l+r)/2;  //l<=p<r
      if (a[p]>=v) return binarySearch(a,v,l,p);
      return binarySearch(a,v,p+1,r);
}

static int rebus (int x){
      if (x<=0 || x>=20) return x;
      if (x>=10) return x+ rebus (x+4);
      return -x-rebus(x-4);
}


/*Esercizio 2
* Dato un array a di n elementi interi, un elemento di a è un picco
* se è maggiore dell'elemento che lo precede (se esso esiste)
* ed è maggiore dell'elemento che lo segue (se esso esiste)
* nell'array a.
* Scrivere un metodo in Java
*        static int [] listaPicchi (int[] a)
* che, preso come parametro un array a di numeri interi,
* restituisce un nuovo array che contiene solo gli elementi
* che sono picchi in a, nell'ordine in cui appaiono.
* Ad esempio, se
*     a={1,3,5,4,6,5,6}
* l'esecuzione del metodo con input a dovrà restituire l'array
* {5,6,6}, mentre se b={4,3,5,4,6,7,6} l'esecuzione del metodo
* con input b dovrà restituire l'array {4,5,7}.
* Si analizzi la complessità temporale del metodo proposto.
*/
```

```java
static int [] listaPicchi (int[] a) {
	if (a==null || a.length<=1) return a;
	int contaPicchi=0;
	for (int i=0; i<a.length; i++) {
		if (i==0 && a[i+1]<a[i]) contaPicchi++;
		else {if (i==a.length-1 && a[i-1]<a[i]) contaPicchi++;
			else if (i!=0 && i!=a.length-1 &&
					a[i-1]<a[i] && a[i+1]<a[i]) contaPicchi++;
			}
	}
	System.out.println(contaPicchi);  //non è richiesto
	int []b=new int[contaPicchi];
	int j=0;
	for (int i=0; i<a.length; i++) {
		if (i==0 && a[i+1]<a[i]) {b[j]=a[i];j++;}
		else {if (i==a.length-1 && a[i-1]<a[i]){b[j]=a[i];j++;}
			 else if (i!=0 && i!=a.length-1 &&
					a[i-1]<a[i] && a[i+1]<a[i])
						{b[j]=a[i];j++;}
			}
	}
	return b;
}


// Altra possibile soluzione più compatta per listaPicchi

static int [] listaPicchi2 (int[] a)
	{if (a==null || a.length<=1) return a;
	int contaPicchi=0;
	for (int i=0; i<a.length; i++) {
		if((i==0||a[i-1]<a[i]) && (i==a.length-1 || a[i+1]<a[i]))
			contaPicchi++;
	}
	int []b=new int[contaPicchi];
	int j=0;
	for (int i=0; i<a.length; i++) {
		if((i==0||a[i-1]<a[i]) && (i==a.length-1 || a[i+1]<a[i]))
			{b[j]=a[i];j++;}
	}
	return b;
}


/* Esercizio 3
 * Scrivere un metodo in Java
 *     static int comuniDistinti (int[] a, int [] b)
```

```
     * che presi in input due array a e b di n elementi interi
     * possibilmente ripetuti, restituisce il numero di elementi
     * distinti in comune fra a e b.
     * Ad esempio, se a={2,5,3,4,2,3,5} e b={2,2,3,3,6,3,6}
     * il metodo restituirà 2.
     * Si analizzi la complessità temporale del metodo proposto:
     * tale metodo deve avere complessità temporale O(n logn)
     * (soluzioni con complessità temporale peggiore danno luogo a una
     * valutazione minore), dove n è la lunghezza degli array a e b.
     */

    static int comuniDistinti (int[] a, int [] b) {
         if (a==null||a.length==0) return 0;
         mergeSort(a);
         mergeSort(b);
         int contaComuni=0, x;
         if (a[0]==0) x=1; else x=0;
         for (int i=0; i<a.length; i++) {
              if(a[i]!=x) {
                   x=a[i];
                   int k= binarySearch(b, x);
                   if (k!=-1) contaComuni++;}
              }
              return contaComuni;
         }

    /*Esercizio 4
     * Si consideri il seguente array di numeri interi:
     * {46, 27, 25, 3, 19, 62, 70, 16, 1, 8, 14}
     * Mostrare passo-passo l'esecuzione del merge-sort
     * per ordinare l'array in modo non decrescente.
     * Si descriva la complessità computazionale del merge-sort
     * (anche scrivendo e risolvendo la ricorrenza T(n)).
    */

}
```