Appello di Programmazione e Algoritmi 1
26 gennaio 2021

Esercizio 1
Cosa stampa il seguente frammento di codice Java?

```
int [] a= {2,3,6,3,7};
boolean condizione = true;
for (int i = 0; i < a.length-1; i++) {
if (a[i] != 2*a[i+1]) System.out.println(a[i]);
else condizione = false;
}
System.out.println(condizione);
```

NON SARANNO CONSIDERATE VALIDE LE RISPOSTE NON GIUSTIFICATE DAI CALCOLI EFFETTUATI PER OTTENERE IL RISULTATO

Il programma al momento dell'esecuzione del for si comporta come segue:

i=0: la condizione del for è true, si esegue il corpo del for, la condizione dell'if è true e si stampa 2
i=1: la condizione del for è true, si esegue il corpo del for, la condizione dell'if è true e si stampa 3
i=2: la condizione del for è true, si esegue il corpo del for, la condizione dell'if è false e si setta condizione a false
i=3: la condizione del for è true, si esegue il corpo del for, la condizione dell'if è true e si stampa 3
i=4: la condizione del for è false e si esce dal for
si stampa false

Riassumendo: il programma stampa

2
3
3
false

Cosa stampa il seguente programma Java?

```
public static void main(String[] args) {
      System.out.println(enigma(2));
      System.out.println(enigma(7));
      System.out.println(enigma(-5));
      }
static int enigma (int x){
      if (x==0) return x;
      if (x>0) return x+enigma(x-3);
      return x+ enigma (x+1);
      }
```

NON SARANNO CONSIDERATE VALIDE LE RISPOSTE NON GIUSTIFICATE DAI CALCOLI o DAI RAGIONAMENTI EFFETTUATI PER OTTENERE IL RISULTATO

enigma(2)= 2+*enigma*(2-3)= 2+*enigma*(-1)= 2+ (-1 + *enigma*(-1+1))=
2+ (-1 + *enigma*(0))= 2+ (-1 + 0)= 1

enigma(7)=
7+*enigma*(7-3)= 7+*enigma*(4)=
7+ (4 + *enigma*(4-3))= 7+ (4 + *enigma*(1)) =
7+ (4 + (1+ *enigma*(1-3))) = 7+ (4 + (1+ *enigma*(-2))) =
= 7+ (4 + (1+ (-2 + *enigma*(-2+1))))= 7+ (4 + (1+ (-2 + *enigma*(-1))))=
7+ (4 + (1+ (-2 + (-1 + *enigma*(-1+1)))))=7+ (4 + (1+ (-2 + (-1 + *enigma*(0)))))=
7+ (4 + (1+ (-2 + (-1 + *0*))))= 9

enigma(-5)=
-5 + *enigma* (-5+1) = -5 + *enigma* (-4) =
-5 + (-4 + *enigma* (-4+1)) = -5 + (-4 + *enigma* (-3)) =
-5 + (-4 + (-3 + *enigma* (-3+1))) = -5 + (-4 + (-3 + *enigma* (-2))) =
-5 + (-4 + (-3 + (-2 + *enigma* (-2+1)))) = -5 + (-4 + (-3 + (-2 + *enigma* (-1)))) =
-5 + (-4 + (-3 + (-2 + (-1 + *enigma* (-1+1))))) = -5 + (-4 + (-3 + (-2 + (-1 + *enigma* (0))))) =
-5 + (-4 + (-3 + (-2 + (-1 + 0)))) = -15

Esercizio 2
Scrivere un metodo iterativo
public static int[] noMin (int[] a)
che, presi come parametro l'array **a** di numeri interi restituisce un nuovo array che contiene tutti i valori dell'array a, nell'ordine, escludendo però il valore minimo (che in **a** può anche ripetersi). Ad esempio se a={ 8, 16, 5, 6, 12, 5, 10, 5, 16, 9 } il metodo restituisce { 8, 16, 6, 12, 10, 16, 9 } (vengono eliminati i valori uguali a 5, che è il minimo); se a= { 3, 10, 2, 4, 10, 8, 5, 2 } il metodo restituisce { 3, 10, 4, 10, 8, 5} (vengono eliminati i valori uguali a 2, che è il minimo); se a= { 3, 3,3 } il metodo restituisce { } (vengono eliminati i valori uguali a 3, che è il minimo). Il metodo deve restituire l'array vuoto nel caso in cui a sia vuoto o nullo. **Si analizzi la complessità temporale del metodo proposto (soluzioni con complessità minore verranno valutate maggiormente)**.

```
public static int[] noMin (int[] a)
      {if (a==null) return null;
      if (a.length==0) return new int[0];
      int min=a[0];
      int conta=1;
      for (int i=1; i<a.length; i++)
            {if (a[i]== min) conta++;
            if (a[i]< min) {min=a[i]; conta=1;}
            }
      int[] risultato =new int [a.length-conta];
      int k=0;
      for (int i=0; i<a.length; i++)
            {if (a[i]!= min) {risultato[k]=a[i]; k++;}
      }
      return risultato;
      }

      // altra soluzione

      public static int[] noMin2 (int[] a)
      {if (a==null) return null;
      if (a.length==0) return new int[0];
      int min=a[0];
      int conta=0;
      for (int i=1; i<a.length; i++)  //trova il minimo
            {
            if (a[i]< min) {min=a[i];}
            }
      for (int i=0; i<a.length; i++)//conta gli elementi uguali al minimo
      {if (a[i]== min) conta++;
      }
      int[] risultato =new int [a.length-conta];
      int k=0;
      for (int i=0; i<a.length; i++)
            {if (a[i]!= min) {risultato[k]=a[i]; k++;}
      }
      return risultato;
      }
```

La complessità è θ(n) dove n=a.length.

Esercizio 3
Due array a e b si dicono simili se contengono gli stessi elementi. Ad esempio, l'array {1,1,3} è simile all'array {1,3,1}. L'array {1,1,3} è simile anche all'array {3,3,3,1}, ma non all'array {1,2,3}.
Scrivere un metodo
static boolean simili (int[] a, int[] b)
che, presi come parametri due array **a** e **b** di numeri interi verifica se due array sono simili oppure no. Se i due array sono entrambi null o sono entrambi vuoti, viene restituito *true*.

Altri esempi, se **a**={1, 4, 5, 2} e **b**={1, 4, 4, 5} viene restituito false (perché 2 compare in a, ma non in b).
Se **a**={1, 4, 5, 5} e **b**={1, 4, 2,2} viene restituito false (perché 5 compare in a, ma non in b e 2 compare in b, ma non in a).
Se **a**={1, 4, 5, 5,2 ,2 } e **b**={1, 4, 4, 5, 5,2} viene restituito true.
Si analizzi la complessità temporale del metodo proposto.

```
static boolean simili (int[] a, int[] b)
      {if (a==null&& b== null) return true;
      if (a==null || b== null) return false;
      if (a.length==0 && b.length== 0) return true;
      if (a.length==0 || b.length== 0) return false;
      mergeSort(a);
      mergeSort(b);
      int k;
      for (int i=0; i<a.length; i++) //verifica se tutti gli elementi di a sono in b
            {k=binarySearch(b, a[i]);
            if (k==-1) return false;
            }
      for (int i=0; i<b.length; i++) //verifica se tutti gli elementi di b sono in a
      {k=binarySearch(a, b[i]);
      if (k==-1) return false;
      }
      return true;
}
```

La complessità è θ(n * log n) dove n è il massimo fra a.length e b.length.

Esercizio 4

Si consideri il seguente array di numeri interi:

{36,12,33,66,21,59,74,22,37}

Mostrare **passo-passo l'esecuzione del merge-sort** per ordinare l'array in modo non decrescente.
Si descriva la complessità computazionale del merge-sort.

Come al solito