UNIVERSITÀ DEGLI STUDI DI BERGAMO

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE E METODI MATEMATICI

Corso di Laurea in Ingegneria Informatica

Esame di **Informatica III B – Progettazione e algoritmi a.a. 2012/13**

PRE-appello Dicembre 2012

1. Quale delle seguenti affermazioni è vera? **[0,5 pt]**

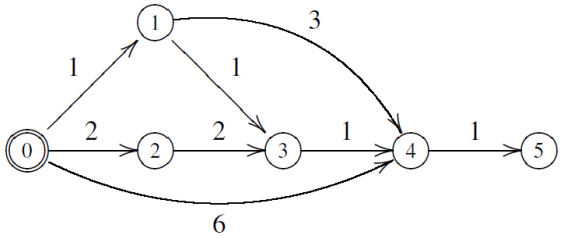
a. $n! = O(n)$
b. $\log n = \Omega(n)$
c. $\log n = O(\log \log n)$
d. Nessuna delle precedenti è vera

2. Nell'algoritmo di ordinamento *Insertion sort*, nel caso peggiore quant'è il numero di spostamenti effettuato? Motivare la risposta. **[1 pt]**

3. Data la seguente funzione ricorsiva foo, trovate la corrispondente relazione di ricorrenza e risolvetela utilizzando il *teorema Principale* (*Master*). **[1,5 pt]**

```
foo( n ){
        if (n < 10) return 1;
        else {
                tmp = n;
                for (i = 1; i <= n; i = i+1)
                  for (j = 1; j <= n; j = j+1)
                    tmp = tmp + i * j;
                }
        return tmp + foo( n/2 ) * foo( n/2 ) + foo( n/2 );
}
```

4. Dato il grafo orientato in figura, **[1,5pt]**



a. determinare le componenti fortemente connesse;
b. calcolare l'albero DFS prodotto visitando in profondità (visita DFS) il grafo a partire dal nodo sorgente **s=0** e seguendo un ordine numerico;
c. classificare, inoltre, tutti gli archi in base all'albero DFS prodotto dalla visita.

5. Dato un array di n elementi con ripetizioni, sia k il numero di elementi distinti in esso contenuti. Progettare e analizzare (la complessità tempo!) un algoritmo efficiente che restituisca un array di k elementi contenente una e una sola volta gli elementi distinti dell'array originale. **[1,5 pt]**

6. Descrivere un algoritmo *CostruisciOracolo (X, k)* di pre-processamento che, dato un array X di n interi nell'intervallo [1,k], processa l'array in modo che dati due interi a e b in [1,k] è possibile rispondere a interrogazioni del tipo *InterrogaOracolo (X, a, b)*= "quanti interi di X cadono nell'intervallo [a,b]?" in tempo costante O(1)! L'algoritmo di pre-processamento deve richiedere tempo O(n + k). **[4pt]**

Suggerimento: Una possibile soluzione è una variante dell'algoritmo di ordinamento lineare *IntegerSort*.

SOLUZIONE:

1. **Risposta**: d.
2. **Risposta**:

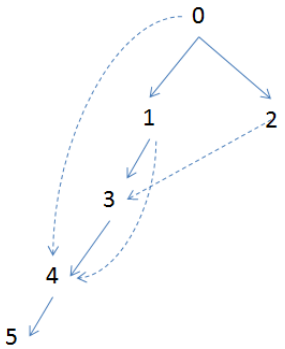
   Il caso peggiore si ha quando l'array è ordinato in ordine decrescente.

```
InsertionSort (A)
1.  for j=2 to n do
2.      x = A[j] //elemento da inserire nella sotto-sequenza ordinata A[1..j-1]
3.      i= j-1 //indice di scansione da destra verso sinistra (da j-1 ad 1)
4.      while i>0 and A[i] > x do
5.          A[i+1]= A[i]  //sposta di una posizione tutti gli elementi A[i] > x
6.          i= i-1
7.      A[i+1]=x //Inserisce x nella locazione che gli compete
```

   Il numero di spostamenti nel caso peggiore è quadratico: $\mathbf{S_{worst}(n)=\Theta(n^2)}$. Infatti, il numero di spostamenti per ogni j-esima iterazione (j=2..n) sono 2 (per caricare/scaricare x), più j-1 spostamenti per fare spazio al j-esimo elemento più piccolo. In totale:

$$S_{worst}(n)=\sum_{j=2..n}(2+j-1)=\Theta(n^2)$$

3. **Risposta**: La relazione di ricorrenza ha come caso base di costo costante il ramo then. Il ramo else e le chiamate ricorsive della seconda istruzione return contribuiscono al costo asintotico: $\mathbf{T(n) = 3\,T(n/2) + \Theta(n^2)}$. Quindi rientriamo nel terzo caso del teorema e si ha: $\mathbf{T(n) = \Theta(n^2)}$ per $\varepsilon>0$ e $3/4<= c <1$.
4. **Risposta**:
   a. Ci sono 6 componenti fortemente connesse formate dai singoli nodi del grafo.
   b. L'albero DFS:



   c. Tutti gli archi del grafo sono *archi dell'albero* DFS. Fanno eccezione:
      - (2,3): arco *trasversale a sinistra*;
      - (0,4) e (1,4): *archi in avanti*.
5. **Risposta**:

Sia a l'array di input, una soluzione naive consiste nel verificare, per ogni elemento a[i], che a non contenga nessuna altro elemento a[j] = a[i] per j diverso da i. Questo induce un algoritmo di costo

quadratico. Alternativamente a può essere prima ordinato in tempo O(n log n) in modo che elementi uguali siano contigui, quindi, con una semplice scansione lineare, possono essere selezionati i k elementi distinti.

```
SelezionaDistinti( a ):
  Sort( a );
  b[0] = a[0];
  j = 1;
  FOR (i = 1; i < n; i = i+1) {
    IF (a[i] != a[i-1]) {
      b[j] = a[i]; j = j+1;
    }
  }
  RETURN b;
```

Il costo dell'algoritmo è dominato dal costo dell'algoritmo Sort utilizzato per l'ordinamento e quindi può essere eseguito in tempo O(n log n).

6. Risposta: Variante dell'algoritmo di IntegerSort. Dopo la costruzione dell'array Y in fase 1, procedere in fase 2 con la creazione di un nuovo array Z di k elementi ed eseguire lo pseudocodice:

Z[1] = Y[1] //caso limite
for i = 2 to k do
  Z[i]= Z[i-1] + Y[i] //Z[i] contiene il numero di elementi dell'array X nell'intervallo [1,i]
return Z

A questo punto è possibile rispondere all'interrogazione "interi nell'intervallo [a,b]":
InterrogaOracolo (X, a, b)= Z[b]-Z[a-1] se a>1,
InterrogaOracolo (X, a, b) = Z[b] se a=1 //caso limite

Esempio:
X= 5 1 6 8 6
Y= 1 0 0 0 1 2 0 1
Z= 1 1 1 1 2 4 4 5
int(1,8)= Z[8]= 5
int(4,6)= Z[6]-Z[3]= 4-1 =3
int(8,8)= Z[8]- Z[7]= 5-4=1
int(5,8)= Z[8]- Z[4]= 5-1=4

In verità Z è superfluo, basta il solo array Y. In pseudocodice:

CostruisciOracolo (X, k) //X di dimensione n
Sia Y un array di dimensione k
for i=1 to k do Y[i]=0
for i=1 to n do incrementa Y[X[i]]
for i=2 to k do Y[i]=Y[i]+Y[i-1] //Y[i] contiene il numero di elementi di X in [1,i]
return Y

InterrogaOracolo (X, a, b)
if a = 1 then return Y[b]
else return (Y[b]-Y[a-1])