Algoritmi e Strutture Dati
13 – 09 – 2019

> *Note*
> 1. La leggibilità è un prerequisito: parti difficili da leggere potranno essere ignorate.
> 2. Quando si presenta un algoritmo è fondamentale spiegare l'idea e motivarne la correttezza.
> 3. L'efficienza e l'aderenza alla traccia sono criteri di valutazione delle soluzioni proposte.
> 4. Si consegnano i fogli di bella copia.

Domanda A

Si consideri la ricorrenza $T(n) = T(n-1) + T(n-2)$.
Si dimostri che abbia soluzione del tipo $O(a^n)$ per $a > 1$

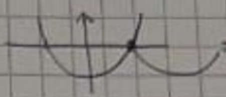
DOMANDA A

$T(n) = T(n-1) + T(n-2) \stackrel{?}{=} O(a^n)$ per $a > 1$

$T(n) \le ca^n$ $(c > 0)$

$T(n) = T(n-1) + T(n-2) \le ca^{n-1} + ca^{n-2} = c\frac{a^n}{a} + c\frac{a^n}{a^2} = ca^n\left(\frac{1}{a} + \frac{1}{a^2}\right) \le ca^n$

$\Rightarrow \frac{1}{a} + \frac{1}{a^2} \le 1$ $\quad a + 1 \le a^2 \quad a^2 - a - 1 \ge 0$



$\Rightarrow a = \frac{1+\sqrt{5}}{2}$ (prendo solo la costante più bassa)

Verifico $T(n) = \Omega(b^n)$: passaggi come gli stessi, basta cambiare il segno di disuguaglianza

$\Rightarrow b = a = \frac{1+\sqrt{5}}{2}$

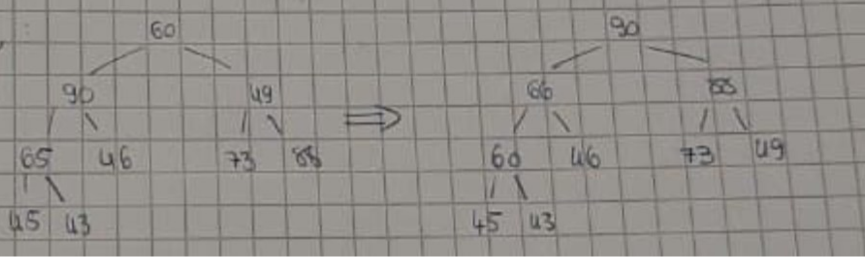
Domanda B

Dare la definizione di max-heap. Dato un array $A[1..12]$ con sequenza di elementi $[60, 90, 49, 65, 46, 73, 88, 45, 63]$ si indichi il risultato della procedura $BuildMaxHeap$ applicata ad A. Si descriva sinteticamente come si procede per arrivare al risultato.

DOMANDA B

A[60, 90, 49, 65, 46, 73, 88, 45, 63]

$\Rightarrow$ lo vedo come uno heap, poi partendo dai nodi di livello inferiore non foglia e scambio il nodo con il massimo dei figli



Domanda B (6 punti) Si consideri un insieme di 7 attività $a_i, 1 \leq i \leq 7$, caratterizzate dai seguenti vettori **s** e **f** di tempi di inizio e fine:

$$\mathbf{s} = (1, 4, 2, 3, 7, 8, 11) \qquad \mathbf{f} = (3, 6, 9, 10, 11, 12, 13).$$

Determinare l'insieme di massima cardinalità di attività mutuamente compatibili selezionato dall'algoritmo greedy GREEDY_SEL visto in classe. Motivare il risultato ottenuto descrivendo brevemente l'algoritmo.

Soluzione: Si considerano le attività ordinate per tempo di fine, e ad ogni passo si sceglie l'attività che termina prima, rimuovendo quelle incompatibili. Si ottiene così l'insieme di attività $\{a_1, a_2, a_5, a_7\}$.

<u>Esercizio 1</u>

Si consideri una variante degli alberi binari di ricerca nella quale i nodi x hanno due campi aggiuntivi:

- un campo pos che permette di capire la posizione del nodo inserito
- un campo $delta$ che permette di capire se dobbiamo ancora mappare le posizioni di altri nodi

Realizzare la procedura di $Insert(T, z)$ che inserisce un nodo z nell'albero e la procedura di stampa $Print(T)$ che dato un albero lo stampa considerando i campi aggiunti.

```
insert (T, z)
  y = nil
  x = T.root     if (z.key>0){delta = 1} else {delta=0}
  while (x != nil)
    y = x
    if (z.key < x.key)
      x = x.left
    else
      x = x.right
    x.pos = x.pos + delta
  z.p = y
  if (y ≠ nil) {
    if (z.key < y.key)
      y.left = z
    else
      y.right = z
  }
  z.pos = delta
```

```
Print (T)
  x = T.root
  if (x ≠ nil) and (x.pos>0) {
    PrintPos (x.left)
    if (x.key>0)
      Print (x.key)
    PrintPos (x.right)
  }
```

Esercizio 2 (8 punti) Per $n > 0$, siano dati due vettori a componenti intere $\mathbf{a}, \mathbf{b} \in \mathbf{Z}^n$. Si consideri la quantità $c(i,j)$, con $0 \le i \le j \le n-1$, definita come segue:

$$c(i,j) = \begin{cases} a_i & \text{se } 0 < i \le n-1 \text{ e } j = n-1, \\ b_j & \text{se } i = 0 \text{ e } 0 \le j \le n-1, \\ c(i-1,j) \cdot c(i,j+1) & 0 < i \le j < n-1. \end{cases}$$

Si vuole calcolare la quantità $M = \max\{c(i,j) : 0 \le i \le j \le n-1\}$.

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di M.

2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

Soluzione:

1. Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione in cui calcoliamo gli elementi in ordine crescente di indice di riga e, per ogni riga, in ordine decrescente di indice di colonna. Il codice è il seguente.

```
COMPUTE(a,b)
n <- length(a)
M = -infinito
for i=1 to n-1 do
   C[i,n-1] <- a_i
   M <- MAX(M,C[i,n-1])
for j=0 to n-1 do
   C[0,j] <- b_j
   M <- MAX(M,C[0,j])
for i=1 to n-2 do
   for j=n-2 downto i do
      C[i,j] <- C[i-1,j] * C[i,j+1]
      M <- MAX(M,C[i,j])
return M
```

2. $$T(n) = \sum_{i=1}^{n-2} \sum_{j=i}^{n-2} 1 = \sum_{i=1}^{n-2} n-1-i = \sum_{k=1}^{n-2} k = (n-1)(n-2)/2.$$