

Esercizio 1 (10 punti):

Si consideri la seguente funzione:

funzione Exam(n):

```

tot ← n;   Θ(1)
if n ≤ 1:  return tot; Θ(1)
tot ← tot + Exam(n-1); T(n-1)
b ← n-1;   Θ(1)
j ← n;     Θ(1)
while j ≥ 0 do: M/2 VOLTE
    tot ← tot + j;
    j ← j-2;
return tot + Exam(b) T(0) → B = M-1
    
```

- Si imposti la relazione di ricorrenza che ne definisce il tempo di esecuzione giustificando dettagliatamente l'equazione ottenuta.
- Si risolva la ricorrenza usando il **metodo di sostituzione** e si dimostri così che la soluzione è $O(n \cdot 2^n)$, commentando opportunamente i passaggi.

la funzione, se $n=1$ si ferma in $\Theta(1)$, altrimenti richiama se stessa con parametro $n-1$, dopodiché esegue un ciclo per $n/2$ volte, e richiama se stessa con parametro $b=n-1$.

$$T(n) = 2T(n-1) + \Theta(n) \text{ con } T(1) = \Theta(1)$$

VOGLIO DIMOSTRARE CHE $T(n) = O(n2^n)$, $\begin{cases} T(n) = 2T(n-1) + CM \\ T(1) = d \end{cases}$

QUINDI $T(n) \leq Kn2^n$ con K COSTANTE DA DETERMINARE.

CASO BASE:

$$T(1) \leq 2K \Rightarrow d \leq 2K \rightarrow K \geq \frac{d}{2}$$

IPOTESI INDUTTIVA:

$$\forall m < n \text{ VALE CHE } T(m) \leq Km2^m$$

PASSO INDUTTIVO

$$T(n) \leq Km2^n \Rightarrow \underbrace{2T(n-1)}_{\text{IPOTESI INDUTTIVA}} + CM \leq Km2^n$$

$$2[K(n-1)2^{n-1}] + CM \leq Km2^n \Rightarrow 2\left[\left(Kn-K\right)\frac{2^n}{2}\right] + CM \leq Km2^n$$

$$Km2^n - K2^n + CM \leq Km2^n \Rightarrow CM \leq K2^n \Rightarrow C \leq \boxed{Kn} \quad \checkmark$$

TENDE A INFINITO

Esercizio 2 (10 punti):

Sia dato un array A ordinato di n interi distinti ed un intero x ; si vuole trovare l'indice in A del più piccolo intero maggiore di x . Progettare un **algoritmo iterativo** efficiente che risolva il problema. Se l'array contiene solo elementi minori o uguali ad x , l'algoritmo deve restituire -1 .

Ad esempio: per $A = [1, 2, 8, 10, 11, 12, 19]$, assumendo che le posizioni dell'array partano da 0, per $x = 7$ l'algoritmo deve restituire 2 (cioè l'indice dell'elemento 8), per $x = 30$ l'algoritmo deve restituire -1 .

Dell'algoritmo proposto

- a) si dia la descrizione a parole,
- b) si scriva lo pseudocodice,
- c) si giustifichi il costo computazionale.

Come prima cosa occorre controllare l'ultimo elemento dell'Array A, se esso è minore o uguale ad x, si ritorni -1 (si ricordi che l'Array è ordinato).

Si esegua una ricerca binaria iterativa, servendosi di due indici "a" e "b", quando l'elemento ispezionato è minore di x, o l'elemento successivo è minore di x, si controllerà la metà destra, se l'elemento precedente del valore ispezionato è maggiore di x si controllerà la metà sinistra. Se per l'indice i si verifica che : $A[i] > x$ and $A[i-1] < x$, si ritorni i.

Pseudocodice :

Def Esercizio2(A,x):

b=len(A)-1;

a=0;

if(a[b]<=x) : return -1;

while(a!=b):

mid=abs((a+b)*0.5); //abs(x) -> parte intera di x

if((A[mid]>x) && (A[mid-1]<x)):

return mid;

if((A[mid]<x) || A[mid+1]<x):

a=mid;

if(A[mid-1]>x):

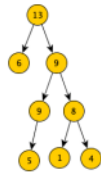
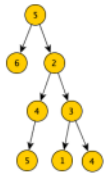
b=mid;

Essendo una ricerca binaria su un Array di n elementi, il costo è chiaramente : $O(\log(n))$.

Esercizio 3 (10 punti):

Si consideri un albero binario radicato T , i cui nodi hanno un campo `val` contenente un intero e i campi `left` e `right` con i puntatori ai figli.

Bisogna modificare il campo `val` di ciascun nodo in modo che il nuovo risulti la somma del valore originario incrementata dal valore originario degli eventuali figli. Si consideri ad esempio l'albero T in figura a sinistra, a destra viene riportato il risultato della modifica di T .



Progettare un **algoritmo ricorsivo** che, dato il puntatore r alla radice di T memorizzato tramite record e puntatori, effettui l'operazione di modifica in tempo $\mathcal{O}(n)$ dove n è il numero di nodi presenti nell'albero.

Dell'algoritmo proposto

- si dia la descrizione a parole,
- si scriva lo pseudocodice,
- si giustifichi il costo computazionale.

Def Esercizio3(T):

sum=0;

if(T->Right):

sum+=T->Right->Val;

Esercizio3(T->Right);

if(T->Left):

sum+=T->Left->Val;

Esercizio3(T->Left);

T->Val+=Sum;

Essendo che controllo ogni nodo dell'albero, ed essendo che ogni controllo viene eseguito in tempo costante, il costo computazionale sarà :

$T(n)=\Theta(n)$ dove n è il numero dei nodi.

Avrei dovuto fare l'equazione di ricorrenza ma non mi andava.