# Test Person

**Email: testperson@example.com**
**Class: E1a**
**Teacher: Professor Teacher**

2020-06-20

## 2020-06-11 pm Esame Introduzione agli Algoritmi [Caminiti]

**Word count: 1058**

**Introduzione agli algoritmi**

**Proff. T. Calamoneri – S. Caminiti - E. Fachini**

**11 Giugno 2020 - Testo B (pomeriggio)**

**Soluzioni (prof. S. Caminiti)**

*Testo scritto direttamente su Exam.net (con i suoi limiti e errori di formattazione).*

***Es 1.*** *Si dimostri, utilizzando la definizione di Θ, che*

$$f(n) = 50n^2 + lg^2\ n = \Theta(n^2)$$

*mettendo in evidenza e commentando con chiarezza i passi seguiti.*

Th. Esistono c1>0, c2>0, n0>=0 t.c. $c1\ n^2 <= 50n^2 + lg^2\ n <= c2\ n^2$

$c1\ n^2 <= 50n^2 + lg^2\ n <= c2\ n^2$

scelgo c1=50 e c2=51

$50n^2 <= 50n^2 + lg^2\ n <= 51n^2$

sottraggo $50n^2$ da ogni parte

$0 <= lg^2\ n <= n^2$

La prima disequazione è vera quando n0 >= 1.

La seconda è vera quando lg n <= n, ovvero sempre.

Quindi scegliendo c1=50, c2=51 e n0=1 la th è vera.

qed.

*In alternativa si poteva dimostrare semparatamente O e Omega, fare 2 dim e alla fine dire che O+Omega => Theta.*

***Es 2.*** *Si imposti la relazione di ricorrenza che definisce il tempo di esecuzione della seguente funzione e la si risolva usando il metodo della sostituzione e si commentino opportunamente i passaggi del calcolo; si imposti l'induzione con chiarezza, sia nello scrivere quanto si vuole dimostrare sia nel formulare l'ipotesi induttiva.*

*Per ottenere l'ipotesi da verificare si utilizzi l'albero della ricorsione.*

```
fun test(A, i, j) {
  n = j - i + 1      c1
  if n < 1 then return 0;     c2
  if n = 1 then return (A[1]);     c3
```

```
    m = n/3;      c4
    for k = 1..m do {      (n/3)-volte
        A[k] = A[k] − A[1];    c5
    }
    x = test(A,i,i+m-1) + test(A,i+m,i+2*m-1) + test(A,i+2*m,j);     3T(n/3)
    return x;   c6
}
```

L'eq di ricorrenza che descrive il costo della funzione test è:
T(n) = 3T(n/3) + c12346 + (n/3)*c5 = 3T(n/3) + Theta(n)
T(1) = c123 = O(1)

Ora cerco un ipotesi con l'albero della ricorsione.
Lavoro sulla funzione T semplificata, senza notazione asintotica.
T(n) = 3T(n/3) + cn
T(1) = c'

| | | | | | | | | | #nodi | costo nodo | costo livello |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | (n) | | | | | 1 | cn | 1cn |
| | | / | | \| | | \ | | | | |
| | (n/3) | | | (n/3) | | | (n/3) | | 3 | cn/3 | 3cn/3 |
| / | \| | \ | / | \| | \ | / | \| | \ | | | |
| (n/9) | (n/9) | (n/9) | (n/9) | (n/9) | (n/9) | (n/9) | (n/9) | (n/9) | 9 | cn/9 | 9cn/9 |
| \| | | | | | | | | | | | |
| (n/3^i) | ... | | | | | | | | 3^i | cn/3^i | 3^i * cn/3^i |
| \| | | | | | | | | | | | |
| (1) | foglie | | | | | | | | 3^h | c' | 3^h c' |

L'altezza dell'albero è h = $\log_3 n$

Sommando il contributo di tutti i livelli ottengo:
T(n) = $\sum_{i=0..h-1}$ [ cn ] + 3^h c' = cn $\log_3 n$ + c'n = Theta(n log n)

Ora dimostro per sostituzione prima T(n) = O(n log n) e poi T(n) = Omega(n log n).

Th. T(n) <= d n $\log_3 n$
Passo base: n = 3  // vincolo su n0
T(3) = 3T(1) + c3 = 3c' + 3c <= d * 3 * 1
d >= c+c'  // vincolo su d
Passo induttivo:
hp ind. per ogni m < n vale T(m) <= d m $\log_3 m$
T(n) = 3T(n/3) + cn <= 3 [ d (n/3) $\log_3(n/3)$ ] + cn =
= dn $\log_3 n$ - dn1 + cn
dn $\log_3 n$ - dn1 + cn <= d n $\log_3 n$
cn <= dn

d >= c  // vincolo su d

Quindi quando n >= 3, scegliendo d >= c+c' la th è dimostrata.

qed.

Th. $T(n) >= d\ n \log_3 n$

Passo base: n = 3  // vincolo su n0

$T(3) = 3T(1) + c3 = 3c' + 3c => d * 3 * 1$

d <= c+c'  // vincolo su d

Passo induttivo:

hp ind. per ogni m < n vale $T(m) >= d\ m \log_3 m$

$T(n) = 3T(n/3) + cn >= 3 [ d\ (n/3) \log_3(n/3) ] + cn =$

$= dn \log_3 n - dn1 + cn$

$dn \log_3 n - dn1 + cn >= d\ n \log_3 n$

cn >= dn

d <= c  // vincolo su d

Quindi quando n >= 3, scegliendo 0 < d = c la th è dimostrata.

qed.

***Es. 3*** *Dati due numeri interi x ed y, definiamo la loro distanza come la loro differenza in valore assoluto: dist(x, y) = |x − y|.*
*Sia dato un Albero Binario di Ricerca T, contenente chiavi intere positive. Si progetti un algoritmo il più efficiente possibile che determini le chiavi di T aventi distanza massima e se ne calcoli il tempo di esecuzione asintotico.*
*Si descriva a parole l'idea algoritmica, si produca lo pseudocodice e si analizzi il tempo di esecuzione asintotico.*

Cerco il valore min e il valore max tra le chiavi dell'ABR e questi hanno dist d massima. Basta pensare all'elenco ordinato di tutte le chiavi dell'albero (ottenibile con una visita inorder) per vedere che ogni altra coppia di chiavi nell'ABR ha dist <= d.

```
fun maxDist(T) {
  if (T == null) return null;
  min = abr_min(T);
  max = abr_max(T);
  return (min, max)
}
```

Riportiamo le funzioni viste a lezione per calcolare mine max su ABR:

```
fun abr_min(T) {
  return T.sx ? abr_min(T.sx) : T.key;
}

fun abr_max(T) {
  return T.dx ? abr_max(T.dx) : T.key;
}
```

abr_min e abr_max scendono lungo una singola diramazione e quindi richiedono O(h) ciascuna, potenzialmente anche meno di h se la diramazione è corta.
Quindi maxDist richiede al più 2O(h) = O(h).
Su un ABR h = O(n) perchè l'albero può essere sbilanciato.
Quindi su ABR maxDist costa T(n) = O(n).

*Si discuta brevemente sul modo in cui (eventualmente) cambiano l'algoritmo ed il costo computazionale nel caso in cui T sia:*

- *un albero bilanciato (AVL o rosso-nero);*
- *un max-heap.*

Su AVL l'altezza è h = Theta(log n), quindi maxDist costa al più T(n) = O(log n).

Su max-heap la ricerca del max richiede O(1) è la radice, la ricerca del min è più costosa perchédevo visitarlo tutto (o quanto meno scorrere tutte le sue n/2 folgie) quindi Theta(n).
maxDist costa T(n) = O(1) + Theta(n) = Theta(n).

Test Person