23/11

Esercizio 2 (8 punti) Data una stringa di numeri interi $A = (a_1, a_2, \ldots, a_n)$, si consideri la seguente ricorrenza $z(i,j)$ definita per ogni coppia di valori (i,j) con $1 \le i, j \le n$:

$$z(i,j) = \begin{cases} a_j & \text{if } i = 1, 1 \le j \le n, \\ a_{n+1-i} & \text{if } j = n, 1 < i \le n, \\ (z(i-1,j) \cdot z(i,j+1)) - z(i-1,j+1) & \text{altrimenti.} \end{cases}$$

z[1,3]

1. Si fornisca il codice di un algoritmo iterativo bottom-up COMPUTE_Z(A) che, data in input la stringa A restituisca in uscita il valore $z(n,1)$.
2. Si valuti il numero esatto $T_{CZ}(n)$ di operazioni tra interi eseguite dall'algoritmo sviluppato al punto (a).

FOR j = 1 TO N
z[1,j] = A_j
FOR i = 2 TO N
z[i,N] = A_{N+1-i}
... RETURN z(N,1)

FOR j=N-1 DOWN TO 1
FOR i=2 TO N

```
COMPUTE_Z(A)
n = length(A)
for i=1 to n do
   z[1,i] = a_i
   z[i,n] = a_{n+1-i}
for j=n-1 downto 1 do
   for i=2 to n do
      z[i,j] = (z[i-1,j] * z[i,j+1]) - z[i-1,j+1]
return z[n,1]
```

$= \sum_{j=1}^{n-1} \sum_{i=2}^{n} 2$

$= \sum_{j=1}^{n-1} 2(n-1)$

(2) QUANTE OPERAZIONI?

$= 2(n-1)(n-1)$

$= 2(n-1)^2$

2. Si valuti il numero esatto $T_{CZ}(n)$ di operazioni tra interi eseguite dall'algoritmo sviluppato al punto (a).

LCS

$(n-1) \times (n-1)$

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

FOR j=0 TO N-1
C[0,j] = B[j]

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

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

M(n, 1)

FOR i = 1 TO N-1
C[i, N-1] = A[i]

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

M = MAX(M, C[0,j])

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

FOR i = 1 TO N-1
C[i, N-1] = A[i]
M = MAX(M, C[i, N-1]

=>

M = MAX(M, C[i,N-1]

(VA BENE PER IL TUO FOR)

```
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])
```

(ci prendiamo la lunghezza dell'array e inizializziamo il minimo a meno infinito (così, qualsiasi prima quantità massima sarà più grande)

Ricordandosi che:
- "i" va da i ad (n-1), quindi diventa perché abbiamo già inizializzato, (n-2) la scansione
- "j" va da (n-1) a 0 compresi, quindi dato che abbiamo inizializzato, parte da (n-2)

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

a quantità $M = \max\{c(i,j) : 0 \leq i \leq j \leq n-1\}$.

M(n, 1)

FOR I = 1 TO N-2
  FOR J = N-2 DOWNTO 1 (I>
    C[I,J] = C[I-1,J] * C[I,J+1]
    M = MAX(M, C[I,J])
// M = -∞ (PERCHÉ TROVI MAX)

RETURN M

```
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])
```

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

②
N. OGGETTO DI OPERAZIONI?

$= \sum_{i=1}^{n-2} n-1-i$

$\sum_{k=1}^{n} k = \frac{n(n-1)}{2} \Rightarrow \sum_{k=1}^{(n-2)} k$

GAUSS

$\frac{(n-1)(n-2)}{2}$

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

Esercizio 2 (9 punti) Supponiamo di avere un numero illimitato di monete di ciascuno dei seguenti valori: 50, 20, 1. Dato un numero intero positivo n, l'obiettivo è selezionare il più piccolo numero di monete tale che il loro valore totale sia n. Consideriamo l'algoritmo greedy che consiste nel selezionare ripetutamente la moneta di valore più grande possibile.

(a) Fornire un valore di n per cui l'algoritmo greedy *non* restituisce una soluzione ottima.

(b) Supponiamo ora che i valori delle monete siano 10, 5, 1. In questo caso l'algoritmo greedy restituisce sempre una soluzione ottima: dimostrare che ogni insieme ottimo $M^\star$ di monete di valore totale n contiene la scelta greedy.

$n = 60$ (50 monete da 1) (1 moneta da 50)

GREEDY

$60 \rightarrow [50 + 10 \cdot 1] \rightarrow$ NON OTTIMO!

$20 - 20 - 20$

(b) Supponiamo ora che i valori delle monete siano 10, 5, 1. In questo caso l'algoritmo greedy restituisce sempre una soluzione ottima: dimostrare che ogni insieme ottimo $M^\star$ di monete di valore totale n contiene la scelta greedy.

$M^* =$ OTTIMO

GREEDY = MIN. N. DI MONETE POSSIBILI

$\Sigma \min = n$

$\rightarrow [M' = M^* \setminus M \cup X]$

$X =$ SOL. OTTIMA

$60 = 50 + (10 \cdot 1)$

NO

[20] [20] $1 \leq 10$ 20

(b) Sia $M^\star$ una soluzione ottima. Sia x il valore maggiore tra 10, 5, e 1 che sia non superiore a n. Se $M^\star$ contiene una moneta di valore x, la proprietà è dimostrata. Altrimenti, sia $M \subseteq M^\star$ un insieme di (2 o più) monete di valore totale x (si osservi che tale insieme esiste sempre quando i valori delle monete sono 10, 5, 1); consideriamo $M' = M^\star \setminus M \cup X$, dove X è l'insieme contenente una moneta di valore x. M' è un insieme di monete di valore totale n e di cardinalità inferiore a quella di $M^\star$: assurdo, quindi questo secondo caso non può verificarsi, e quindi $M^\star$ contiene necessariamente una moneta di valore x.

Domanda A (7 punti) Si consideri la funzione ricorsiva `search(A,p,r,k)` che dato un array A, ordinato in modo decrescente, un valore k e due indici p, q con $1 \leq p \leq r \leq A.length$ restituisce un indice i tale che $p \leq i \leq r$ e $A[i] = k$, se esiste, e altrimenti restituisce 0.

```
search(A,p,r,k)
   if p <= r
      q = (p+r)/2
      if A[q] = k
         return q
      elseif A[q] > k
         return search(A,q+1,r,k)
      else
         return search(A,p,q-1,k)
   else
      return 0
```

CODICE ⇒ RICORRENZA (?)

⇒ CORRETTEZZA

Dimostrare induttivamente che la funzione è corretta. Inoltre, determinare la ricorrenza che esprime la complessità della funzione e risolverla con il Master Theorem.

DIVIDE ET IMPERA

A(1, N)

q = (p+r)/2

p q r

n/2 + n/2 ⇒ T(n) = 2T(n/2) + c

⇒ Θ(log(n))

[MASTER THEOREM]

DIMOSTRAZ. DI CORRETTEZZA

A[q] = k !

A[q] > k

(p+1, M) ...

Soluzione: La prova di correttezza avviene per induzione sulla lunghezza $l = r - p + 1$ del sottoarray $A[p..r]$ di interesse. Se $l = 0$, ovvero $p > r$, la funzione ritorna correttamente 0, dato che non ci sono elementi nel sottoarray e quindi non ci possono essere elementi $= k$. Se invece $l > 0$ si calcola $q = \lfloor (p + r)/2 \rfloor$ e si distinguono tre casi:

- se $A[q] = k$ si ritorna correttamente k
- se $A[q] > k$, dato che l'array è ordinato in modo decrescente certamente $A[j] \geq A[q] > k$ per $p \leq j \leq q$. Quindi il valore k può trovarsi solo nel sottoarray $A[q + 1, r]$. La chiamata $search(A, q + 1, r, k)$, dato che il sottoarray ha lunghezza minore di l, per ipotesi induttiva restituisce un indice i tale che $q + 1 \leq i \leq r$ e $A[i] = k$, se esiste, e altrimenti restituisce 0. Per l'osservazione precedente, questo è il valore corretto da restituire anche per $search(A, p, r, k)$ e si conclude.
- se $A[q] < k$ si ragiona in modo duale rispetto al caso precedente.

Per quanto riguarda la ricorrenza, ignorando i casi base, dato che la funzione ricorre su di un array la cui dimensione è la metà di quello originale, si ottiene:

$$T(n) = T(n/2) + c$$

Rispetto allo schema standard del master theorem ho $a = 1$, $b = 2$ e $f(n) = c$. Ho dunque che $f(n) = 1 = \Theta(n^{\log_2 1} = n^0 = 1$. Pertanto si conclude che $T(n) = \Theta(n^0 \log n) = \Theta(\log n)$.

[p ≤ q ≤ M]

p q M

← A[q]

(p+1, q) (q+1, M)

i ∈ ... ≤ M

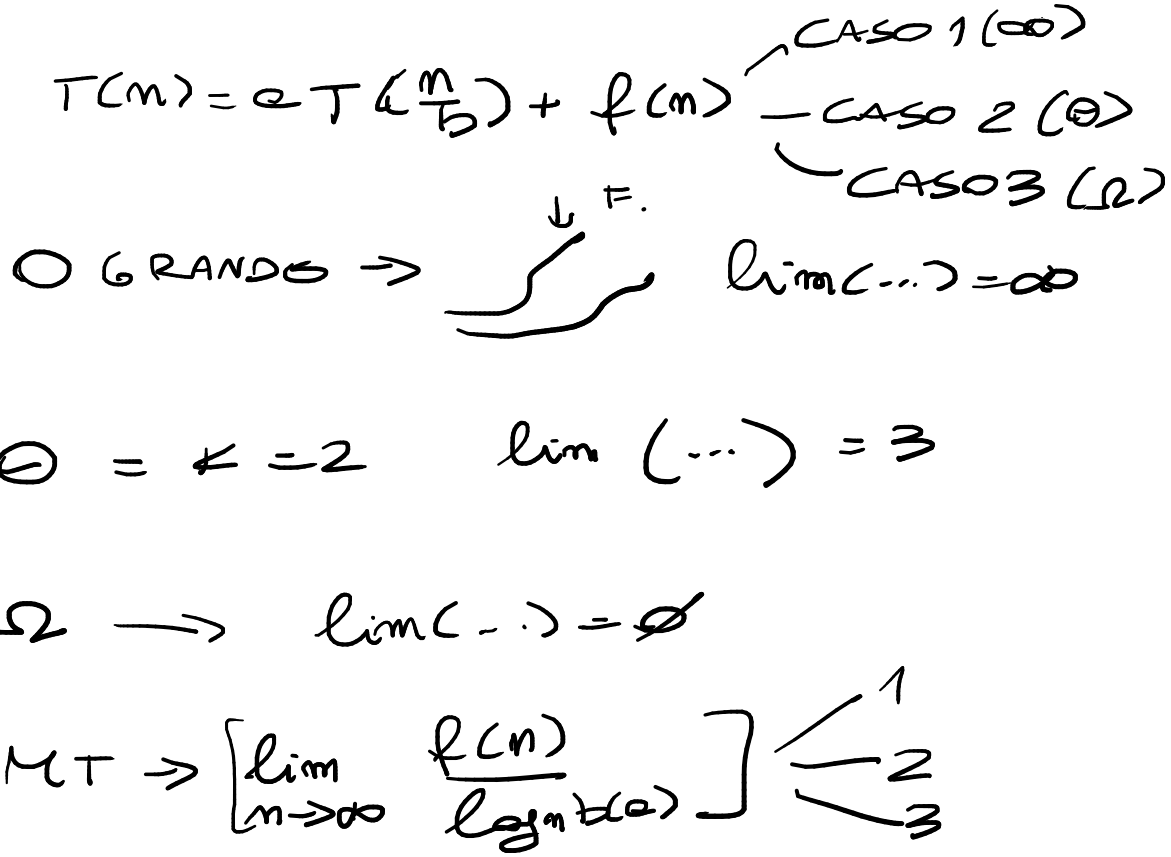
ORDINE

⇒ IT WORKS!

Domanda A (7 punti) Dare la definizione della classe $\Theta(f(n))$. Mostrare che la ricorrenza

$$T(n) = \frac{3}{4}T(n/3) + T(2n/3) + 2n$$

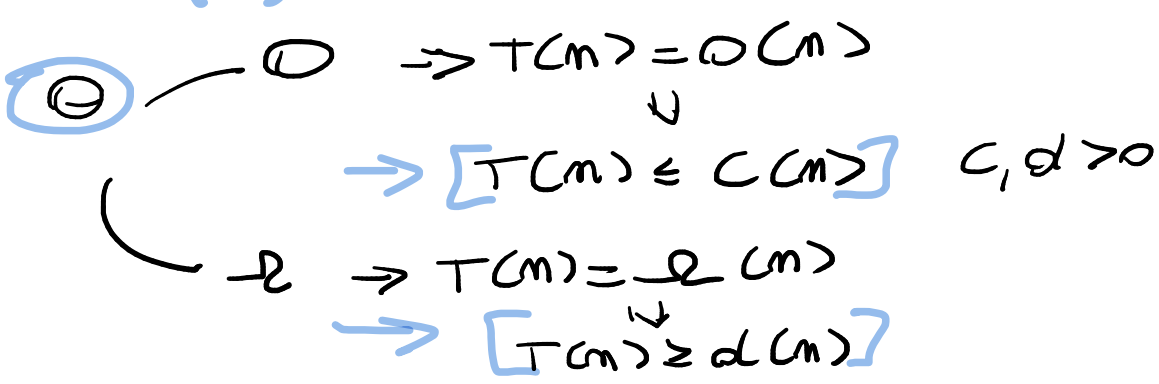
ha soluzione in $\Theta(n)$.



Domanda A (7 punti) Dare la definizione della classe $\Theta(f(n))$. Mostrare che la ricorrenza

$$T(n) = \frac{3}{4}T(n/3) + T(2n/3) + 2n$$

ha soluzione in $\Theta(n)$.



$$T(n) = \frac{3}{4}T(n/3) + T(2n/3) + 2n$$

$T(n) \leq c(n)$

$$T(n) = \frac{3}{4}T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + 2n \qquad > c$$

$$c(n) \leq \frac{3}{4}c\left(\frac{n}{3}\right) + c\left(\frac{2n}{3}\right) + 2n$$

$$c(n) \leq \frac{cn}{4} + \frac{2}{3}cn + 2n$$

$$\frac{c(n)}{n} \leq \frac{n\left(\frac{c}{4} + \frac{2}{3}c + 2\right)}{n} \qquad \forall n \geq 0$$

$$c \leq \frac{c}{4} + \frac{2}{3}c + 2 \quad \cdot 12 \quad \leq \frac{3c + 8c - 12c}{12}$$

$$-2 \leq \frac{c}{4} + \frac{2}{3}c - c \quad \cdot 12 \quad \leq -c$$

$$[\forall n \geq 0,\ c \geq 24]$$

$\rightsquigarrow [T(n) = \Omega(n) \Rightarrow T(n) \geq a(n)]$

$$T(n) = \frac{3}{4} T\left(\frac{n}{3}\right) + T\left(\frac{2}{3} n\right) + 2n$$

$$a n \geq \frac{3}{4} a\left(\frac{n}{3}\right) + a\left(\frac{2}{3} n\right) + 2n$$

$$[\forall n \geq 0] \quad a \geq \frac{a}{4} + \frac{2}{3} a + 2$$

$$12a \geq \frac{3a + 8a + 24 \cdot 12}{12}$$

$$120a \geq 11a + 24$$

$$[a \geq 24, \forall n \geq 0] \quad \ldots$$

Domanda A (5 punti) Risolvere la ricorrenza $T(n) = 4T(n/2) + n^2\sqrt{n}$ utilizzando il master theorem.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}$$

$a = 4$

$b = 2$

$f(n) = \sqrt{n}\, n^2$

$$= a T\left(\frac{n}{b}\right) + f(n)$$

$$\lim_{n \to \infty} \frac{f(n)}{n^{\log_b(a)}} \ldots \varnothing \Rightarrow \text{CASO 3}$$

CASO 3 $\Rightarrow$ COND. DI REGOLARITÀ

$$a f\left(\frac{n}{b}\right) \leq c f(n)$$

$\forall n$, TROVIAMO $c \ldots$

Soluzione: Rispetto allo schema generale si ha $a = 4$, $b = 2$, $f(n) = n^2\sqrt{n} = n^{\frac{5}{2}}$. Si osserva che $\log_b a = 2$ quindi $f(n) = \Omega(n^{\log_b a + \epsilon})$ (per $0 < \epsilon \leq \frac{1}{2}$). In aggiunta vale la condizione di regolarità, ovvero $af(\frac{n}{b}) \leq cf(n)$ per qualche $c < 1$. Infatti $af(\frac{n}{b}) = 4f(\frac{n}{2}) = 4\frac{n^2}{4}\sqrt{\frac{n}{2}} = n^2\sqrt{\frac{n}{2}} \leq cn^2\sqrt{n}$ per $c \geq \frac{1}{\sqrt{2}}$ (che è < 1). Quindi $T(n) = \Theta(n^2\sqrt{n})$.

Domanda B (4 punti) Dato l'array $A = [60, 90, 49, 65, 46, 73, 88, 45, 43]$, mostrare in forma di albero, il max-heap prodotto dalla procedura `BuildMaxHeap`.