1 Esercizio.

Riferimento ai capitoli: 1. Introduzione; 2. Notazione asintotica

Si consideri il seguente programma:

```
def somma(n):
    s=0
    for i in range(1,n+1):
        s+= i
    return s
```

e si calcoli il costo computazionale asintotico nel caso peggiore.

Si dica, inoltre, se l'algoritmo descritto dal codice dato è efficiente per il problema che esso si prefigge di risolvere.

il costo nel caso peggiore e' $\Theta(n)$, e non e' efficiente. Si tratta di una sommatoria del tipo $\sum_{i=1}^{n} i$, anche dando per scontato che non si e' a conoscenza del risultato noto, tutti i valori da 1 a n possono essere "VISITATI" in un tempo $\Theta(\log(n))$ tramite ricerca binaria.

2 Esercizio.

Riferimento ai capitoli: 5. Equazioni di ricorrenza

Si risolva la seguente equazione di ricorrenza con due metodi diversi, per ciascuno dettagliando il procedimento usato:

$$\begin{cases} T(n) = T\left(\frac{n}{2}\right) + \Theta\left(n^2\right) & \text{se } n > 1 \\ T(1) = \Theta(1) \end{cases}$$

tenendo conto che *n* è una potenza di 2.

Metodo principale

$f(n) = \Theta(n^2)$ ESSENDO $f(n) = \Omega(1)$ E $f\left(\frac{n}{2}\right) \leq c \cdot f(n) \Rightarrow T(n) = f(n)$

$n^{\log_b a} = 1$

$T(n) = \Theta(n^2)$

Metodo iterativo

$$T(n) = \left[T\left(\frac{n}{2^2}\right) + \Theta\left(\left[\frac{n}{2}\right]^2\right)\right] + \Theta(n^2) = T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \Theta\left(\left[\frac{n}{2^i}\right]^2\right) \qquad k = \log(n)$$

$$\Theta(\log(n)) + \Theta(1)\sum_{i=0}^{\log(n)-1} \left[\frac{n}{2^i}\right]^2 = \Theta(\log(n)) + \Theta(1)\sum \frac{n^2}{2^{2i}} = \Theta(\log(n)) + \Theta(1)\sum \frac{n^2}{4^i}$$

$$= \Theta(\log(n)) + \Theta(n^2)\sum \frac{1}{4^i} = \Theta(\log(n)) + \Theta(n^2) \cdot \frac{\left(\frac{1}{4}\right)^{\log(n)} - 1}{\frac{1}{4} - 1}$$

$$= \Theta(\log(n)) + \Theta(1)\, n^2 \left[\frac{1/n^2}{\frac{1}{4}-1} - \frac{1}{\frac{1}{4}-1}\right] = \Theta(\log(n)) + \Theta(1)\left[\frac{1}{\frac{1}{4}-1} - \frac{n^2}{\frac{1}{4}-1}\right] = \Theta(n^2)$$

3 Esercizio.

Riferimento ai capitoli: 5. Equazioni di ricorrenza

Si risolva la seguente equazione di ricorrenza con tutti e quattro i metodi, per ciascuno dettagliando il procedimento usato:

$$\begin{cases} T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n) \text{ se } n > 1 \\ T(1) = \Theta(1) \end{cases}$$

Metodo iterativo

SICURAMENTE ① $T(n) \geq 2T\left(\frac{n}{3}\right) + \Theta(n)$ E

② $T(n) \leq 2T\left(\frac{2n}{3}\right) + \Theta(n)$

①

$$T(n) \geq 2^k T\left(\frac{n}{3^k}\right) + \sum_{i=0}^{k-1} 2^i \Theta\left(\frac{n}{3^i}\right)$$

$$T(n) \geq \Theta(n^{\log_3 2}) + \Theta(n) \sum \left(\frac{2}{3}\right)^i \geq \Theta\left(n^{\log_3 2}\right)$$

②

$$T(n) \leq 2\left[2T\left(4\frac{n}{3^2}\right) + \Theta\left(\frac{2n}{3}\right)\right] + \Theta(n)$$

$$T(n) \leq 2^k T\left(\left(\frac{2}{3}\right)^k n\right) + \sum 2^i \Theta\left(\left(\frac{2}{3}\right)^i n\right)$$

$$T(n) \leq \Theta(n^{\log_{(\frac{3}{2})}(2)}) + \Theta(n) \underbrace{\sum \left(\frac{4}{3}\right)^i}_{\left[\left[\left(\frac{4}{3}\right)^{\log n} - 1\right] 3\right]}$$

$$= \Theta(n^{\log_3 2}) \leq T(n) \leq \Theta\left(n^{\log_{(\frac{3}{2})}(2)}\right)$$

4 Esercizio.

Riferimento ai capitoli: 2. Notazione asintotica; 5. Equazioni di ricorrenza

Si consideri il seguente programma:

```
def Analizzami(A,n):
    if n<= 3: return A[0]
    j= 1
    while j < n:
        A[j] = A[j] - A[n-j]
        j *= 3
    for i in range(3):
        A[i] += A[i+1]
        Analizzami(A, n//3)
```

e si mostri tramite il metodo iterativo che il costo computazionale asintotico è O(n log n).

$$T(n) = 3T\left(\frac{n}{3}\right) + \Theta(\log_3(n))$$

$$T(n) = 3\left[3T\left(\frac{n}{3^2}\right) + \Theta\left(\log_3\left(\frac{n}{3}\right)\right)\right] + \Theta(\log(n))$$

$$T(n) = 3^K T\left(\frac{n}{3^K}\right) + \sum_{i=0}^{K-1} 3^i \log_3\left(\frac{n}{3^i}\right) \qquad K = \log_3(n)$$

$$T(n) = \Theta(n) + \sum 3^i \left[\log(n) - i\right]$$

$$T(n) = \Theta(n) + \sum 3^i \log_3(n) - 3^i i = \Theta(n) + \Theta(1)\sum 3^i \log(n) - \Theta(1)\sum 3^i i$$

$$= \Theta(n) + \log(n) \sum 3^i - \sum 3^i i$$

$$\Theta(n) + \Theta\left(\log(n)\left[\frac{n-1}{2}\right]\right) - \Theta\left(\left[\log(n)\, n\right]\right) =$$

$$\Theta(n) + \Theta(n\log(n)) - \Theta(n\log(n)) = \Theta(n\log(n))$$

7 Esercizio.

Riferimento ai capitoli: 4. Ricorsione; 5. Equazioni di ricorrenza

Dati in input un intero n ed un array A=($a_0, \dots, a_{n-1}$) di n numeri reali, si scrivano due funzioni, una iterativa ed una ricorsiva, che restituiscano il valore 1 se il array è palindromo, il valore 0 altrimenti.

```
DEF IT(A):
    i = 0;
    J = LEN(A) - 1;
    WHILE ((J - i) > 0):
        IF (A[J] != A[i]):
            RETURN 0;
        i += 1;  J -= 1;
    RETURN 1;

DEF RIC(A, i, J):
    IF (J - i) < 0:
        RETURN 1;
    IF (A[J] ! A[i]):
        RETURN 0;
    RETURN RIC(A, i+1, J-1);
```

8 Esercizio.

Riferimento ai capitoli: 4. Ricorsione; 5. Equazioni di ricorrenza

Dato un array A di n interi, progettare un algoritmo ricorsivo che restituisce il massimo ed il minimo di A in tempo O(n). Verificare tale costo computazionale tramite l'impostazione e la risoluzione di un'equazione di ricorrenza.

```
DEF IT(A):
    MIN, MAX = A[0];
    FOR (i IN LEN(A)-1):
        IF A[i] < MIN:
            MIN = A[i]
        IF A[i] > MAX:
            MAX = A[i]
    RETURN MIN, MAX;
```

DEFAULT

```
DEF RIC(A, MIN, MAX, S=TRUE)
    IF S: MIN, MAX = A[0];
    IF A[0] > MAX:
        MAX = A[0]
    IF A[0] < MIN:
        MIN = A[0]
    IF LEN(A)=1:
        RETURN MIN, MAX
    RETURN RIC(A[1:], MIN, MAX, FALSE);
```

$\Theta(1)$

$T(n-2)$

$$T(n) = T(n-2) + \Theta(1) = \Theta(n)$$

13 Esercizio.

Riferimento ai capitoli: 4. Ricorsione; 5. Equazioni di ricorrenza; 6. Il problema dell'ordinamento

Si consideri il codice del seguente algoritmo di ordinamento. dove la funzione viene richiamata la prima volta con *i=1* e *j=len(A)-1*.

```
def  UnAltroSort (A, i, j):
    if A[i] > A[j]:
        A[i],A[j]= A[j],A[i]
    if i+1 >= j: return
    k=(j-i+1)//3
    UnAltroSort(A, i, j-k) # si ordinano i primi 2/3 dell'array
    UnAltroSort(A, i+k, j) # si ordinano gli ultimi 2/3 dell'array
    UnAltroSort(A, i, j-k) # si ordinano di nuovo i primi 2/3 dell'array.
```

$$T(n) = 3T\left(\frac{2}{3}n\right) + \Theta(1)$$

Teorema principale

$a = 3$ $\qquad b = \frac{3}{2}$

$$T(n) = \Theta\left(n^{\log_{\frac{3}{2}} 3}\right)$$

$n^{\log_{\frac{3}{2}} 3}$ $\qquad f(n) = \Theta(1)$

metodo iterativo

$$T(n) = 3^k T\left(\left(\frac{2}{3}\right)^k n\right) + \sum_{i=0}^{k-1} 3^i \Theta(1) \qquad k = \log_{3/2} n$$

$$T(n) = \Theta\left(3^{\log_{\frac{3}{2}} n}\right) + \Theta(1)\left[\frac{3^{\log_{\frac{3}{2}} n} - 1}{2}\right] = \Theta\left(n^{\log_{\left(\frac{3}{2}\right)} 3}\right)$$

14 Esercizio.

Riferimento ai capitoli: 4. Ricorsione; 5. Equazioni di ricorrenza; 6. Il problema dell'ordinamento

Si consideri la seguente modifica dell'algoritmo di Merge Sort, che prende come parametri un array V di interi e due indici `primo` e `ultimo` (che alla prima chiamata valgono 0 ed len(V) − 1 rispettivamente):

```
def  MergeSortModificato(V, primo, ultimo):
     n=ultimo-primo+1
     if n <= 1: return
     medio1=primo+n//3
     medio2=medio1+n//3
     MergeSortModificato(V, primo, medio1)
     MergeSortModificato(V, medio1 + 1, medio2)
     MergeSortModificato(V, medio2 + 1, ultimo)
     Fondi(V, primo, medio1, medio2, ultimo)
     return
```

$\Theta(1)$

$3T\left(\frac{n}{3}\right)$

$\Theta(n)$

Tenendo conto che il costo computazione della funzione `Fondi` è comunque Θ(n), si ricavi l'equazione di ricorrenza che esprime il costo computazionale della funzione `MergeSortModificato`, specificando i contributi delle varie istruzioni, e si risolva l'equazione di ricorrenza trovata utilizzando il metodo iterativo.

Infine, si discuta se sia possibile includere questo procedimento tra gli algoritmi di ordinamento efficienti, e se si possa ulteriormente generalizzare l'idea di suddividere l'array di partenza in un numero qualsiasi *k* di sottoarray.

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

$$T(n) = 3^k T\left(\frac{n}{3^k}\right) + \sum_{i=0}^{k-1} 3^i \,\Theta\left(\frac{n}{3^i}\right)$$

$$T(n) = \Theta\left(3^{\log_3(n)}\right) + \sum^{\log_3(n)} \Theta(n) = \Theta(n) + \Theta(n \log(n)) = \Theta(n \log n)$$

15 Esercizio.

Riferimento ai capitoli: 6. Il problema dell'ordinamento

Sia dato un array *A* contenente *n* interi distinti e ordinati in modo crescente.

Progettare un algoritmo che, in tempo *O(log n)*, individui la posizione più a sinistra nell'array per cui si ha $A[i] \neq i$,l'algoritmo restituisce -1 se una tale posizione non esiste.

Ad esempio, per A = $[0, 1, 2, 3, 4]$, l'algoritmo deve restituire -1, per A = $[0, 5, 6, 20, 30]$, la risposta deve essere 1 e, per A = $[-3, 1, 2, 3, 6]$, la risposta deve essere 0.