Introduzione agli Algoritmi
Esame Scritto a canali unificati con spunti per la soluzione

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma

10 Aprile 2025

ATTENZIONE: I compiti in cui l'esercizio 1 sia valutato meno di 7/10 non saranno ritenuti sufficienti, indipendentemente dalla qualità delle soluzioni degli esercizi 2 e 3.

Esercizio 1 (10 punti): Si risponda alle seguenti domande, dando una giustificazione:

1. a. $n^2 + n \log n = O(\ldots)$?
   b. $\log n + \log \log n = O(\ldots)$?
2. a. è vero che $2^{2n} = O(2^n)$?
   b. è vero che $f(n) = O(g(n))$ implica che $2^{f(n)} = O(2^{g(n)})$?
3. Se un algoritmo ha tempo di esecuzione $\Theta(n^2)$ nel caso peggiore, possiamo dedurne che nel caso migliore il suo tempo di esecuzione sarà $\Theta(n^2)$?
4. Se un algoritmo ha tempo di esecuzione $\Omega(n^2)$ nel caso migliore, è possibile che in qualche caso il suo tempo di esecuzione sia $O(n)$?

5. Qual è l'equazione di ricorrenza relativa all'algoritmo ricorsivo della ricerca binaria? Qual è la sua soluzione?

Le seguenti risposte vogliono solo dare un'idea della soluzione e non vanno considerate come esaustive:

1. **a.** $n^2 + n \log n = O(n^2)$
   b. $\log n + \log \log n = O(\log n)$**.**

2. **a. non è vero che** $2^{2n} = O(2^n)$ **poiché** $2^{2n} = 2^n \cdot 2^n$ **e quindi non si trova una costante** c **tale che** $2^{2n} \leq c \cdot 2^n$**.**
   b. non è vero che $f(n) = O(g(n))$ **implica che** $2^{f(n)} = O(2^{g(n)})$**; per vederlo si può sfruttare il punto precedente, scegliendo** $f(n) = 2n$ **e** $g(n) = n$**.**

3. **La risposta è no, perché non è detto che i due casi abbiano lo stesso costo, come nel caso dell'insertionSort in cui il caso peggiore è in** $\Theta(n^2)$ **ma il caso migliore è in** $\Theta(n)$**.**

4. **La risposta è no. Se nel caso migliore si è dimostrato che il costo è in** $\Omega(n^2)$**, ogni altro caso ha questo stesso limite inferiore e quindi non può avere un costo in** $O(n)$**.**

5. **L'equazione di ricorrenza per l'algoritmo di ricerca binaria ricorsiva è** $T(n) = T(n/2) + \Theta(1)$ **e** $T(1) = \Theta(1)$**. La sua soluzione è** $T(n) = \Theta(\log n)$**, che rappresenta il costo computazionale dell'algoritmo nel caso peggiore.**

Esercizio 2 (10 punti): Si consideri la seguente funzione:

```
def esercizio_iter(n):
    i=s=1
    while i<=n:
        t+=i*i
        i+=1
    for j in range(t):
        k=1
        while k<=i:
            k=2*k
            s+=2
    return s
```

Se ne determini il costo computazionale, motivando bene la risposta.

In modo molto stringato possiamo dire che il primo while richiede $\Theta(n)$ tempo ed, alla fine, t risulta essere la sommatoria dei primi n quadrati e quindi dell'ordine $\Theta(n^3)$. Di conseguenza il for viene iterato $\Theta(n^3)$ volte e, per ogni iterazione, viene eseguito un ciclo while di costo $\Theta(\log n)$. Quindi il costo computazionale complessivo è $\Theta(n^3 logn)$.
E' necessario dettagliare i calcoli utili ad arrivare a questo risultato.

Esercizio 3 (10 punti): Si consideri la seguente funzione:

```
def esercizio_ric(n):
    if n <= 1:
        return 1
    k = n*n
    while k >=1:
        k = k//2
return k + 3*esercizio_ric(n//4)
```

Si scriva l'equazione di ricorrenza che ne definisce il costo computazionale, la si risolva con il metodo dell'albero, e si discuta se sia possibile applicarvi il teorema principale o no, motivando bene la risposta.

La relazione di ricorrenza è così definita:
il caso base è $T(0) = T(1) = \Theta(1)$ **mentre il caso generico è** $T(n) = T(n/4) + \Theta(\log n)$ **visto che il ciclo while è eseguito** $\log n^2$ **volte e che** $\log n^2 = 2 \log n$**. La soluzione è** $O(\log^2 n)$**.**

Non si può applicare il teorema principale perché $n^{\log_b a}$ **è asintoticamente più piccolo di** $f(n) = \Theta(\log n)$ **ma non polinomialmente.**