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

docenti: T. Calamoneri, A. Monti
Sapienza Università di Roma
15 Gennaio 2026

Esercizio 1 (10 punti): Nella funzione riportata qui sotto, i valori delle costanti a, b corrispondono rispettivamente **alla seconda e terza cifra che compaiono nella propria matricola**.

```
def es1(n):
    if n <= 2:
        return 1
    s, t = (a+1)*n, 0
    for i in range(s):
        t+= i
    for i in range (b+1):
        t  +=  (b+1)*es1(n//(a+2))
    return t
```

Si sostituiscano i parametri a e b con i valori corrispondenti dati dalla propria matricola (evitando di ragionare in forma parametrica) e si risponda alle seguenti domande:

1. Qual è l'equazione di ricorrenza che descrive il costo computazionale della funzione $es1(n)$? Perché?

2. Risolvere l'equazione di ricorrenza con con un metodo a vostra scelta, dettagliando i calcoli.

Diamo una soluzione generale, che serve per capire come procedere. Tuttavia, sottolineiamo che il testo richiede di svolgere l'esercizio sostituendo i valori di a e b PRIMA di scrivere l'equazione di ricorrenza.

1. **il primo for ha costo $\Theta(n)$.**
   Il secondo for itera $b+1$ volte ed ogni iterazione fa una chiamata ricorsiva su $n//(a+2)$ per un costo di $(b+1)T\left(\frac{n}{a+2}\right)$.
   La ricorrenza è dunque:

$$T(n) = \begin{cases} \Theta(1) & \textbf{se } n \leq 2, \\ (b+1)\,T\left(\dfrac{n}{a+2}\right) + \Theta(n) & \textbf{se } n > 2. \end{cases}$$

2. **Possiamo risolvere l'equazione con il teorema principale, infatti la ricorrenza è della forma:**

$$T(n) = A\,T\left(\frac{n}{B}\right) + f(n),$$

   dove

$$A = b+1, \qquad B = a+2, \qquad f(n) = \Theta(n);$$

   inoltre, il caso base è $T(1) = \Theta(1)$ come richiesto dall'enunciato del teorema.
   Si confronta $f(n)$ con la funzione critica

$$n^{\log_B A} = n^{\log_{a+2}(b+1)}$$

   ottenendo così:

$$T(n) = \begin{cases} \Theta(n) & \textbf{se } b+1 < a+2, \\ \Theta(n \log n) & \textbf{se } b+1 = a+2, \\ \Theta\left(n^{\log_{a+2}(b+1)}\right) & \textbf{se } b+1 > a+2. \end{cases}$$

Esercizio 2 (10 punti): Siano dati un array A di n elementi (con possibili ripetizioni) ordinato in ordine non decrescente ed un intero non negativo k.

Progettare un algoritmo **iterativo** che restituisca la coppia di indici della prima e dell'ultima occorrenza di k in A. I due indici dovranno essere uguali se k occorre solo una volta ed entrambi pari a -1 se k non è presente in A. Il costo computazionale deve essere $O(\log n)$.

Dell'algoritmo proposto:

a) si scriva lo pseudocodice opportunamente commentato, in modo da chiarire il senso delle istruzioni,

b) si giustifichi il costo computazionale.

a. **È immediato capire che bisogna sfruttare due leggere modifiche della ricerca binaria, in modo che determinino rispettivamente la prima e l'ultima occorrenza di k. Ecco un possibile pseudocodice:**

```
def es2(A, k):
    # A è un array ordinato e k è una chiave
    # si richiama la funzione che cerca la prima occorrenza
    i = PrimaOccorrenza(A,k)
    # se k non presente inutile proseguire
    if i == -1:
       return (-1,-1)
    # si richiama la funzione che cerca l'ultima occorrenza
    j = UltimaOccorrenza(A,k)
   return(i,j)
```

dove le due funzioni richiamate sono:

```
def PrimaOccorrenza(A, k):
    # A è un array ordinato e k è una chiave
    # restituisce l'indice della prima occorrenza di k,
    # o -1 se k non è presente
    min, max = 0, len(A)-1
    risultato = -1
    while min<=max:
        med = (min+max)//2
        if k == A[med]:
           risultato = med
           max = med -1
        elif A[med] < k:
           min = med + 1
        else:
          max = med-1
    return risultato

def UltimaOccorrenza(A, k):
    # A è un array ordinato e k è una chiave
    # restituisce l'indice dell'ultima occorrenza di k,
    # o -1 se k non è presente
    min, max = 0, len(A)-1
    risultato = -1
    while min <= max:
        med = (min+max)//2
        if k == A[med]:
           risultato = med
           min = med +1
        elif A[med] < k:
           min = med + 1
        else:
          max = med-1
    return risultato
```

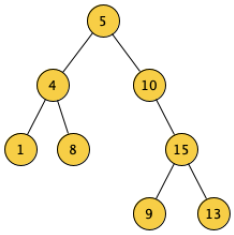
b. **Il costo computazionale è dato dalla somma dei costi delle due funzioni, RicBinSin e RicBinDes, entrambi pari al costo della ricerca binaria, cioè $O(\log n)$. È richiesto di calcolare il costo formalmente.**

Esercizio 3 (10 punti):
Sia dato un *albero binario* con n nodi tramite il puntatore r alla sua radice, in cui ogni nodo contiene:

- un campo *key*, che rappresenta la chiave del nodo;
- un puntatore *left* al figlio sinistro;
- un puntatore *right* al figlio destro.

Progettare un algoritmo **ricorsivo** che, presi r ed un intero non negativo k, stampi tutte le chiavi dei nodi che si trovano al *livello* k dell'albero (assumendo che la radice si trovi al livello 0). Le chiavi devono essere stampate *nell'ordine in cui i nodi vengono incontrati durante una visita* **inorder** *dell'albero*.
Si assuma che l'albero possa essere vuoto.
Il costo computazionale deve essere $O(n)$.

Ad esempio per l'albero in figura con $k = 1$ vanno stampate le chiavi 4 e 10; con $k = 2$ vanno stampate 1, 8 e 15 mentre con $k = 5$ non va stampato nulla.



Dell'algoritmo proposto:
a) si scriva lo pseudocodice opportunamente commentato, in modo da chiarire il senso delle istruzioni,

b) si giustifichi il costo computazionale, scrivendo la ricorrenza che lo caratterizza.
NOTA BENE: nello pseudocodice della funzione ricorsiva **non** si deve far uso di variabili globali.

a. **L'algoritmo esegue una visita inorder dell'albero binario. Durante la discesa ricorsiva, il parametro k viene decrementato a ogni livello. Quando $k = 0$, il nodo corrente si trova al livello richiesto e la sua chiave viene stampata. In questo modo vengono stampate solo le chiavi dei nodi al livello k nell'ordine inorder.**
Ecco un possibile pseudocodice:

```
def es3(r, k):
    if r is None:
        return
    # visita al sottoalbero sinistro (inorder)
    es3(r.left, k - 1)
    # visita del nodo
    if k == 0:
        print(r.key)
    # visita al sottoalbero destro
    es3(r.right, k - 1)
```

b. **Il costo computazionale è quello di una visita. L'equazione di ricorrenza relativa alla visita è:**

   - $T(n) = T(k) + T(n-1-k) + \Theta(1)$
   - $T(0) = \Theta(1)$

   dove k, è il numero di nodi presenti nel sottoalbero sinistro, $0 \leq k < n$. L'equazione si può risolvere con il metodo di sostituzione e dà come soluzione $\Theta(n)$.