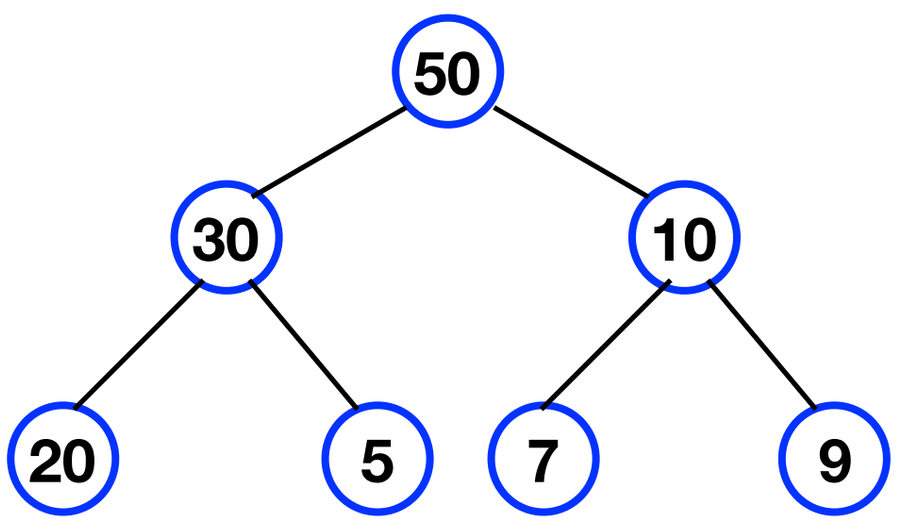
Es. 1

a. Si illustri l'operazione di ripristino della proprietà del Max-Heap violata in un solo elemento dell'array A. Se ne analizzi il tempo di esecuzione asintotico, nei casi migliore, peggiore e generale.

b. Si dimostri la falsità della seguente affermazione: siano x e y due chiavi in un Max-Heap A con chiavi intere positive e distinte.
Siano h(x) e h(y) le altezze dei sotto alberi radicati nei nodi di chiave x e y rispettivamente in T. Se x > y allora h(x) >= h(y).

Sol. a. Si veda il libro di testo, considerando i due casi possibili: violazione rispetto ai figli o rispetto al padre.

Sol. b. Basta costruire un contro esempio:



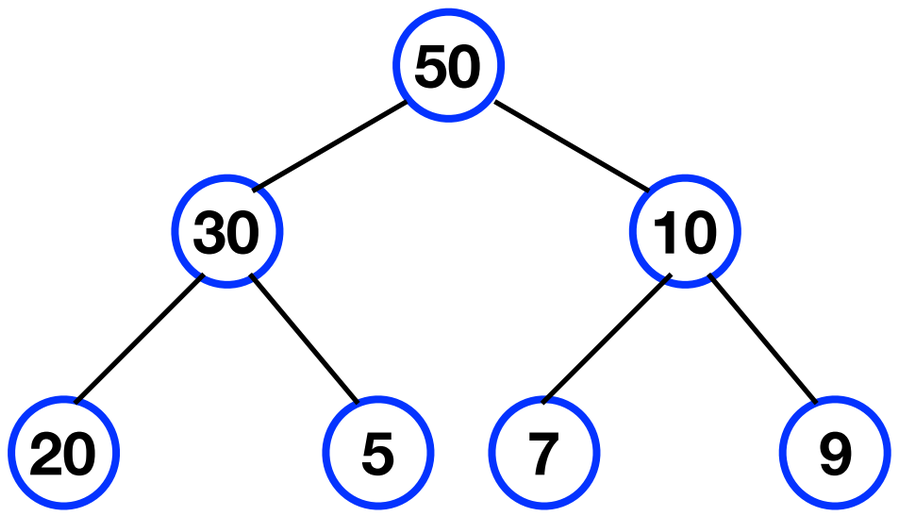
Se x = 20 e y = 10, si ha che h(x) = 0 e h(y) = 1, contraddicendo l'assunto.

Es. 1

a. Si illustri l'operazione di ripristino della proprietà del Max-Heap violata in un solo elemento dell'array A. Se ne analizzi il tempo di esecuzione asintotico, nei casi migliore, peggiore e generale.

b. Si dimostri la falsità della seguente affermazione: siano x e y
due chiavi in un Max-Heap A con chiavi intere positive e distinte.
Siano h(x) e h(y) le altezze dei sotto alberi radicati nei nodi di chiave x e y rispettivamente in T. Se x > y allora h(x) >= h(y).

Sol. b. Basta costruire un contro esempio:



Se x = 20 e y = 10, si ha che h(x) = 0 e h(y) = 1, contraddicendo l'assunto.

Es. 2

a. Dato un vettore A, si consideri il seguente frammento di pseudocodice, se ne determini l'output, illustrando il contenuto di B e se ne calcoli il tempo di esecuzione asintotico:

```
for i=1 to n do B[i]=0
B[1]=A[1]
for i=2 to n do
    for j=1 to i do
        B[i]=B[i]+A[j]
for i =1 to n do stampa B[i]
```

b. Si descriva come si può ottenere lo stesso risultato in Θ(n).

a. B[i] contiene la somma dei primi i elementi di A. Il frammento è eseguito in $\Theta(n^2)$, infatti il ciclo più interno viene eseguito tante volte quanto è la somma dei primi n interi, a partire da 2.

Es. 2

b. Si descriva come si può ottenere lo stesso risultato in Θ(n).

b. Poichè in B[i] si calcola la somma degli elementi A[1] + ….A[i], per ottenere la somma richiesta in B[i+1] basta sommare B[i] ad A[i+1], senza bisogno di calcolare la somma ogni volta partendo da A[1]. La seguente versione ha tempo di esecuzione asintotico Θ(n):

```
for i=1 to n do B[i]=0
B[1]=A[1]
for i =2 to n do
    B[i]=B[i-1]+A[i]
for i =1 to n do stampa B[i]
```

Es. 3

3. Dato un array di n elementi ordinati in ordine crescente A e un intero k si determini la coppia di indici della prima e dell'ultima occorrenza di k in A, in O(lg n).

Sol. Si tratta di usare i metodi RBisect(A,k) e LBisect(A,k). Il primo, visto in aula, è un algoritmo che dà in output la posizione di inserimento ordinato di k. Se k è già presente, dà la posizione immediatamente a destra dell'ultima occorrenza di x in A.
Il secondo algoritmo, LBisect(A,k), dà in output la posizione di inserimento ordinato di k. Se k è già presente dà la posizione della prima occorrenza di k in A.
Quindi gli indici cercati sono quello dato in output da RBisect, diminuito di uno e quello dato in output da LBisect(A,k).
In entrambi i casi lo spazio della ricerca viene dimezzato ad ogni passo a seconda dell'esito del confronto di k con l'elemento centrale della porzione di array esaminata. Quindi per entrambi gli algoritmi il tempo di esecuzione è Θ(lg n), in tutti i casi.

Lo pseudocodice

```
RBisect( A,x)
input: A è un array di  n interi, x è un intero
prec: A è ordinato
output: se x non è presente dà la posizione di inserimento
ordinato di x, altrimenti dà la posizione immediatamente a destra
dell'ultima occorrenza di  x in A.
lo=0
hi = A.size
while lo < hi do
se l'elemento cercato è presente ha un indice in [lo,hi)
      m = midpoint(lo,hi)
    if   x < A[m] then hi = m  else lo = m+1
return lo
```

20	20	40	40	40	50	60	60
0	1	2	3	4	5	6	7

x = 40, lo=0, hi = 8 m= 4, poichè 40 non è minore di A[4] lo=5.
Poi m= 6 e poichè 40 è minore di A[6] hi= 6. Ora m= 5 e x è minore di A[5], quindi hi =5. Poichè hi=lo viene dato in output lo=5

Lo pseudocodice

```
LBisect( A,x)
input: A è un array di  n interi, x è un intero
prec: A è ordinato
output: se x non è presente dà la posizione di inserimento ordinato
di x, altrimenti dà la posizione più a sinistra dell'ultima occorrenza
di  x in A.
lo=0
hi = A.size
while lo < hi do
se l'elemento cercato è presente ha un indice in [lo,hi)
        m = midpoint(lo,hi)
        if   x ≤ A[m] then hi = m else lo = m + 1
return lo
```

20	20	40	40	40	50	60	60
0	1	2	3	4	5	6	7

x = 40, lo=0, hi = 8 m= 4, poichè 40 è minore o uguale di A[4] hi=4. Poi m= 2 e poichè 40 è minore o uguale di A[2] hi = 2. Ora m= 1 e x è maggiore di A[1], quindi lo = 2. Poichè hi=lo viene dato in output lo=2

Lo pseudocodice

IndexOcc(A,x)
input: A è un array di interi, x è un intero
prec: A è ordinato
output: gli indici della prima e dell'ultima occorrenza di x in A,se presente, (-1,-1) altrimenti
i = LBisect(A,x)
j = RBisect(A,x)
if i==j then return (-1,-1)
else return (i,j-1)

20	20	40	40	40	50	60	60
0	1	2	3	4	5	6	7

Se x = 45 LBisect(A,x) e RBisect(A,x) danno in output 5, che è la posizione giusta per l'inserimento di 45, che non è presente.