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
```

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

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 $T(n) = \Theta(n^0 \log n) = \Theta(\log n)$.

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

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

Si vuole calcolare la quantità $m = \min\{c(i,j) : 0 \leq i \leq j \leq n-1\}$.

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di m.
2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

Soluzione:

(a) Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione in cui calcoliamo gli elementi in ordine crescente di indice di riga e, per ogni riga, in ordine decrescente di indice di colonna. Il codice è il seguente.

```
COMPUTE(a,b)
n <- length(a)
m = +infinito
for i=1 to n-1 do
   C[i,n-1] <- a_i
   m <- MIN(m,C[i,n-1])
for j=0 to n-1 do
   C[0,j] <- b_j
   m <- MIN(m,C[0,j])
for i=1 to n-2 do
   for j=n-2 downto i do
      C[i,j] <- C[i-1,j-1] * C[i,j+1]
      m <- MIN(m,C[i,j])
return m
```

(b)

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

Esercizio 2 (9 punti) Sia n un intero positivo. Si consideri la seguente ricorrenza $M(i, j)$ definita su tutte le coppie (i, j) con $1 \leq i \leq j \leq n$:

$$M(i,j) = \begin{cases} 2 & \text{se } i = j, \\ 3 & \text{se } j = i+1, \\ M(i+1, j-1) \cdot M(i+1, j) + M(i, j-1) & \text{se } j > i+1. \end{cases}$$

(a) Si scriva una coppia di algoritmi INIT_M(n) e REC_M(i, j) per il calcolo memoizzato di $M(1, n)$.

(b) Si calcoli il numero esatto $T(n)$ di moltiplicazioni tra interi eseguite per il calcolo di $M(1, n)$.

Soluzione:

(a)
```
INIT_M(n)
if n=1 then return 2
if n=2 then return 3
for i=1 to n-1 do
   M[i,i] = 2
   M[i,i+1] = 3
M[n,n] = 2
for i=1 to n-2 do
   for j=i+2 to n do
      M[i,j] = 0
return REC_M(1,n)


REC_M(i,j)
if M[i,j] = 0 then
   M[i,j] = REC_M(i+1,j-1) * REC_M(i+1,j) + REC_M(i,j-1)
return M[i,j]
```

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

Domanda A (7 punti) Definire formalmente la classe $\Omega(f(n))$. Dimostrare che la seguente ricorrenza ha soluzione $T(n) = \Omega(n)$
$$T(n) = \frac{1}{3}T(n-1) + 2n + 1$$

Soluzione: L'insieme $\Omega(f(n))$ è definito come:
$$\Omega(f(n)) = \{g(n) : \exists c > 0.\ \exists n_0.\ \forall n \geq n_0.\ 0 \leq cf(n) \leq g(n)\}.$$

Si deve provare che asintoticamente, per un'opportuna costante $c > 0$
$$T(n) \geq cn$$

Si procede per induzione:

$$\begin{array}{rcll} T(n) & = & \frac{1}{3}T(n-1) + 2n + 1 & \text{[per definizione della ricorrenza]} \\ & \geq & \frac{1}{3}c(n-1) + 2n + 1 & \text{[per ipotesi induttiva } T(n-1) \geq c(n-1)]] \\ & \geq & 2n & \\ & \geq & cn & \end{array}$$

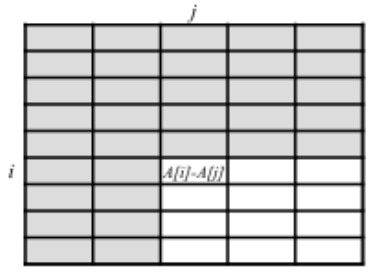
dove l'ultima disuguaglianza vale quando $c \leq 2$. Risulta dunque dimostrata la tesi.

Esercizio 1 (10 punti) Realizzare una funzione `Diff(A,k)` che, dato un array `A[1,n]` ordinato in senso decrescente, verifica se esiste una coppia di indici `i, j` tali che $\texttt{A[i]} - \texttt{A[j]} = \texttt{k}$. Restituisce la coppia di indici se esiste e (0,0) altrimenti. La funzione non deve alterare l'input e deve operare in spazio costante. Scrivere lo pseudocodice, provarne la correttezza e valutarne la complessità.

```
diff(A, n, k):
    i=1
    j=1
    while (i<=n) and (j<=n) and (A[i]- A[j] <> k)
        if (A[i]- A[j] < k)
            j++
        else
            i++

    if (i <= n) and (j<=n)
        return (i,j)
    else
        return (0,0)
```

È facile vedere che si mantiene l'invariante $\forall (i', j') \in [1, n].(i' < i) \vee (j' < j) \Rightarrow A[i] - A[j] \neq k$, ovvero una coppia (i', j') tale che $A[i] - A[j] = k$ può esistere solo tra le coppie ancora esplorabili ($i' \geq i$ e $j' \geq j$), ovvero, graficamente nella parte non grigia:



Infatti, inizialmente, con $i = j = 1$, l'invariante è vacuamente vero.
Ad ogni iterazione, se entro nel ciclo, ci sono due possibilità:

- Se $A[i] - A[j] < k$, allora incremento j. In questo modo, escludo dall'esplorazione le coppie (i', j) con $i' \geq i$ per le quali, dato che l'array è decrescente e quindi $A[i] \geq A[i']$, vale

$$A[i'] - A[j] \leq A[i] - A[j] < k.$$

  Dunque non escludo coppie utili, l'invariante continua a valere.

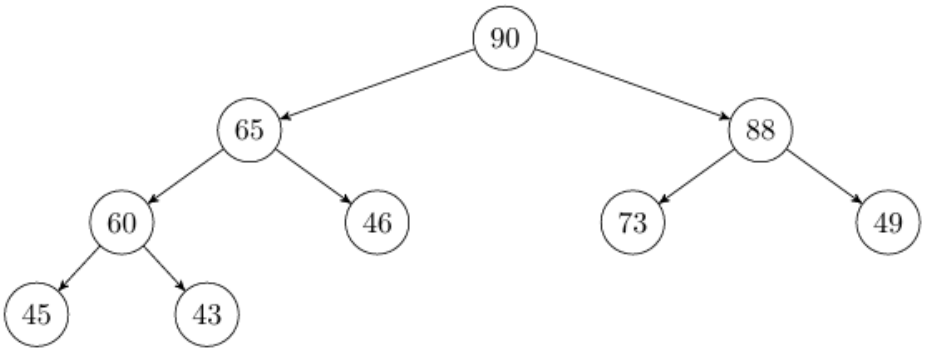
- Dualmente, $A[i] - A[j] > k$, allora incremento i. In questo modo, escludo dall'esplorazione le coppie (i, j') con $j' \geq j$ per le quali, dato che l'array è decrescente e quindi $A[j] \geq A[j']$, vale

$$A[i] - A[j'] \geq A[i] - A[j] > k.$$

  Dunque, anche in questo caso, non escludo coppie utili, l'invariante continua a valere.

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`.

Soluzione: Per la descrizione del processo di costruzione, si rimanda al libro. Si noti che la procedura **non** consiste in una serie di inserimenti. Il risultato è il seguente:



ovvero, in forma di array, $[90, 65, 88, 60, 46, 73, 49, 45, 43]$.

Esercizio 1 (9 punti) Realizzare una funzione `avgTree(T)` che dato un albero binario `T` con chiavi numeriche, verifica se, per ogni nodo che abbia discendenti, la chiave del nodo è maggiore o uguale della media delle chiavi dei discendenti e ritorna conseguentemente un valore booleano (la radice dell'albero è `T.root` e ogni nodo `x` ha i campi `x.left` `x.right` e `x.key`).

Soluzione: L'idea è quella di effettuare una visita dell'albero, in modo ricorsivo, raccogliendo per ogni sottoalbero le seguenti informazioni:

- se è soddisfatta la condizione sulla media richiesta
- la somma delle chiavi
- il numero dei nodi.

Quano mi trovo su di un nodo, analizzo i sottoalberi sinistro e destro. Quindi, avendo a disposizione la somma delle chiavi ed il numero dei nodi per i due sottoalberi posso verificare la condizione sulla media per il nodo in esame. Lo pseudocodice della soluzione può essere il seguente:

```
avgTree(T)
   v, n, s = avgTreeRec(T.root)
   return v


# avgTreeRec(x):
# verifica se il sottoalbero radicato in x e' un avgTree e ritorna tre valori:
# - un booleano,
# - il numero dei discendenti
# - la somma delle loro chiavi

avgTreeRec(x)

if x = nil
   return true, 0, 0
else
   # ispeziono i sottoalberi sinistro e destro
   vl, nl, sl = avgTreeRec(x.left)
   vr, nr, sr = avgTreeRec(x.right)

   # se non ci sono discendenti (nl+nr =0) oppure la chiave del nodo in
   # esame e' maggiore o uguale della media delle chiavi dei discendenti
   # la condizione e' soddisfatta
   v =   (nl+nr =0) or (x.key >= (sl+sr)/(nl+nr))

   # il numero di nodi dell'albero radicato in x e' dato dalla somma del
   # del numero di nodi nel sottoalbero dx e nel sottoalbero sx piu' uno
   # (il nodo x stesso)
   n =   nl + nr + 1

   # la somma delle chiavi dell'albero radicato in x e' dato dalla somma del
   # delle chiavi nel sottoalbero dx e in quello sx piu' x.key
   s =   sl + sr + x.key

   return v, n, s
```

Si tratta di una visita e quindi la complessità è $\Theta(n)$.

Esercizio 2 (9 punti) Lungo una strada ci sono, in vari punti, n parcheggi liberi e n auto. Un posteggiatore ha il compito di parcheggiare tutte le auto, e lo vuole fare minimizzando lo spostamento totale da fare. Formalmente, dati n valori reali $p_1, p_2, \ldots, p_n$ e altri n valori reali $a_1, a_2, \ldots, a_n$, che rappresentano le posizioni lungo la strada rispettivamente di parcheggi e auto, si richiede di assegnare ad ogni auto a_i un parcheggio $p_{h(i)}$ minimizzando la quantità

$$\sum_{i=1}^{n} |a_i - p_{h(i)}|.$$

1. Si consideri il seguente algoritmo greedy. Si individui la coppia (auto, parcheggio) con la minima differenza. Si assegni quell'auto a quel parcheggio. Si ripeta con le auto e i parcheggi restanti fino a quando tutte le auto sono parcheggiate. Dimostrare che questo algoritmo non è corretto, esibendo un controesempio.

2. Si consideri il seguente algoritmo greedy. Si assuma che i valori $p_1, p_2, \ldots, p_n$ e $a_1, a_2, \ldots, a_n$ siano ordinati in modo non decrescente. Si produca l'assegnazione $(a_1, p_1), (a_2, p_2), \ldots, (a_n, p_n)$. Dimostrare la correttezza di questo algoritmo per il caso $n = 2$.

Soluzione:

1. Si consideri il seguente input:

$$p_1 = 5, p_2 = 10 \quad \text{e} \quad a_1 = 9, a_2 = 14.$$

   L'algoritmo produce l'assegnazione $(a_1, p_2), (a_2, p_1)$, che ha costo $1 + 9 = 10$, mentre l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $4 + 4 = 8$.

2. Ci sono vari casi possibili:

   (a) Caso $a_1 \leq p_1 \leq p_2 \leq a_2$
   - l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $p_1 - a_1 + a_2 - p_2 = (a_2 - a_1) - (p_2 - p_1)$
   - l'assegnazione $(a_1, p_2), (a_2, p_1)$ ha costo $p_2 - a_1 + a_2 - p_1 = (a_2 - a_1) + (p_2 - p_1)$; siccome $p_2 - p_1 \geq 0$, questa assegnazione ha costo non inferiore rispetto alla precedente

   (b) Caso $a_1 \leq p_1 \leq a_2 \leq p_2$
   - l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $p_1 - a_1 + p_2 - a_2 = (p_2 - a_1) - (a_2 - p_1)$
   - l'assegnazione $(a_1, p_2), (a_2, p_1)$ ha costo $p_2 - a_1 + a_2 - p_1 = (p_2 - a_1) + (a_2 - p_1)$; siccome $a_2 - p_1 \geq 0$, questa assegnazione ha costo non inferiore rispetto alla precedente

   (c) Caso $a_1 \leq a_2 \leq p_1 \leq p_2$
   - l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $p_1 - a_1 + p_2 - a_2 = (p_2 - a_1) + (p_1 - a_2)$
   - l'assegnazione $(a_1, p_2), (a_2, p_1)$ ha costo $p_2 - a_1 + p_1 - a_2 = (p_2 - a_1) + (p_1 - a_2)$, uguale a quello precedente

   Tutti gli altri casi sono simmetrici e si dimostrano nella stessa maniera.