PAOLO ROSSI mat. 123456

ESERCIZIO (1)

Calcolo il costo $T'(n)$ della funzione misteryZ(n,k). E' una funzione ricorsiva con equazione di ricorrenza:

$$T'(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2\,T'\left(\frac{n}{2}\right) + 1 & \text{se } n > 1 \end{cases}$$

Usando master theorem si ha:

$$\alpha = \frac{\log 2}{\log 2} = 1 \qquad \beta = 0$$

e avendo $\alpha > \beta$ si ottiene $T'(n) = \Theta(n^{\alpha}) = \Theta(n)$

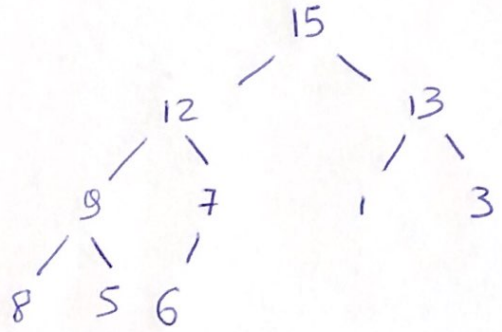
Ora considero il costo $T(n)$ della funzione misteryZ(n). Il ciclo while viene eseguito $\Theta(\log n)$ volte in quanto inizialmente $k = n$, e poi k viene dimezzato fino ad arrivare a 1
Tutte le operazioni hanno costo costante ad esclusione dell'invocazione misteryZ(n, k) che ha sempre costo $\Theta(n)$.
Il costo complessivo risulta quindi $\Theta(n \log n)$

PAOLO ROSSI mat. 123456

ESERCIZIO (2)

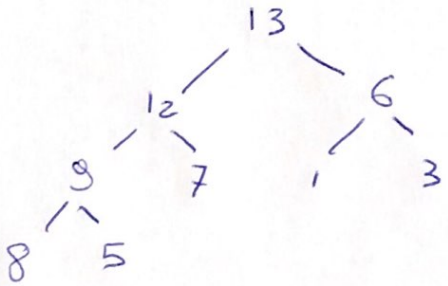
stato iniziale H

[15, 12, 13, 9, 7, 1, 3, 8, 5, 6]



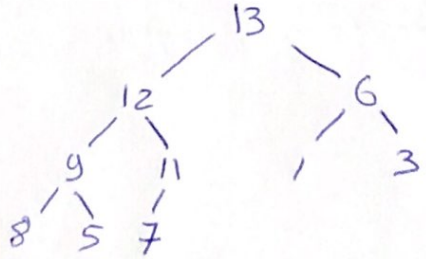
dopo H.deleteMax()

[13, 12, 6, 9, 7, 1, 3, 8, 5]



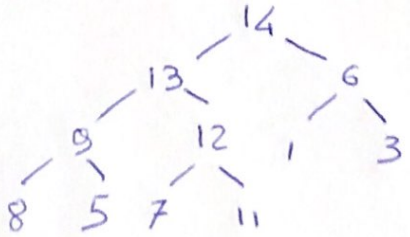
dopo H.insert(11)

[13, 12, 6, 9, 11, 1, 3, 8, 5, 7]



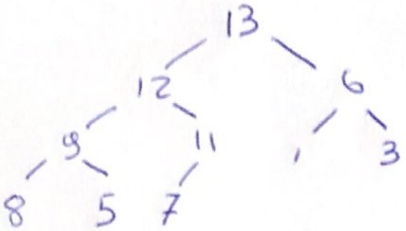
dopo H.insert(14)

[14, 13, 6, 9, 12, 1, 3, 8, 5, 7, 11]



dopo H.deleteMax()

[13, 12, 6, 9, 11, 1, 3, 8, 5, 7]



PAOLO ROSSI mat. 123456

ESERCIZIO (3)

Utilizzo programmazione dinamica risolvendo i sottoproblemi $maxH(i)$, con $i \in [1..n]$, definito come segue:

$maxH(i)$ = numero massimo di hamburger nel tratto $[1..i]$ prendendo anche l'hamburger i-esimo

La soluzione del problema iniziale sarà quindi:

$$max\{ maxH(i) \mid i \in [1..n] \}$$

I sottoproblemi possono essere risolti come segue:

$$maxH(i) = \begin{cases} 1 & \text{se } i = 1 \\ 1 & \text{se per ogni } j<i \;.\; K[j] > K[i] \\ max\{maxH(j) \mid j \in [1..i-1] \text{ tale che } K[j] \leq K[i]\} + 1 & \text{altrimenti} \end{cases}$$

Scriviamo ora l'algoritmo function maxHamburger(K[1..n]) che risolve i sottoproblemi e restituisce la soluzione del problema iniziale.

Variabili usate:

- maxH[1..n] conterrà in maxH[i] la soluzione del problema maxH(i)
- max conterrà il valore più grande finora inserito in maxH
- prevMax mentre si calcola un nuovo maxH(i) contiene il più grande maxH(j) per j tali che K[j] ≤ K[i]

```
function maxHamburger(K[1..n]) → Int
    maxH[1] ← 1        max ← 1
    for i ← 2..n
    |   prevMax ← 0
    |   for j ← 1..i-1
    |   |   if K[j] ≤ K[i] and maxH[j] > prevMax then
    |   |       prevMax ← maxH[j]
    |   maxH[i] ← prevMax + 1
    |   if maxH[i] > max then max ← maxH[i]
    return max
```

PAOLO ROSSI mat. 123456

Calcoliamo ora il costo computazionale $T(n)$.
Il ciclo esterno viene eseguito $n-1$ volte.
All'i-esimo ciclo, il for ~~esterno~~ interno viene eseguito i volte. Considerando che tutte le singole operazioni hanno costo costante ottengo:

$$T(n) = \Theta(1+2+\cdots+(n-1)) = \Theta\left(\frac{n\cdot(n-1)}{2}\right) = \Theta(n^2)$$

PAOLO ROSSI mat. 123456

ESERCIZIO ④

Per il calcolo della profondità di un albero è possibile usare una visita in profondità modificata. Al momento di chiusura della visita di un nodo, gli si associa profondità pari alla profondità massima dei nodi adiacenti già chiusi incrementata di 1.

Il seguente algoritmo `function profondità(G=(V,E), r)` utilizza i campi `mark` e `depth` associati ad ogni vertice:

- `v.mark` — booleano uguale a true se e solo se il vertice v è già stato incontrato durante la visita
- `v.depth` — profondità a partire dal vertice v

L'algoritmo usa una funzione ausiliaria `DFS(v)` che utilizza una variabile

- `max` — massima v.depth per i vertici adiacenti già chiusi al momento della chiusura di v

```
function profondità( G=(V,E), r ) → INT
    for each v∈V
        v.mark ← false      v.depth ← 0
    r.mark ← true
    DFS(r)
    return r.depth

function DFS(v)
    max ← 0
    for each u adiacente a v
        if u.mark = false
            u.mark ← true
            DFS(u)
        if u.depth > max then max ← u.depth
    v.depth ← max + 1
```

Assumendo implementazione tramite liste di adiacenza, tale algoritmo esegue una quantità costante di operazione di costo costante per ogni vertice quindi $T(n) = \Theta(n)$, con $n = |V|$