

SELECTION SORT

Il selection sort partendo dall'ultima posizione cerca quale elemento

va nell'i-esimo posto (dalla fine sempre al più grande) fino all'inizio dell'array.

● SELECTION SORT (A, n)

FOR $j = n$ DOWN TO 2 DO

MAX = FINDMAX (A, j)

SWAP (A, j, MAX)

● FINDMAX (A, J)

MAX = 1

FOR $i = 2$ TO j DO

IF $A[i] > A[MAX]$ THEN

MAX = i

RETURN MAX

$$T_{SS}(n) = \Theta(n + \sum_{j=2}^n T_{FH}(j)) = \Theta(n + \sum_{j=2}^n \Theta_j) = \Theta(n + \Theta(\sum_{j=2}^n j)) = \Theta(n + \Theta(\frac{n(n+1)}{2} - 1)) = \Theta(n + \Theta(n^2)) = \Theta(n^2)$$

$$T_{FH}(m) = \Theta(1 + \Theta(j) + \Theta(j)) = \Theta(j)$$

Il tempo non dipende dall'ordine dell'array, risulta essere il peggiore

HEAP SORT

Si noti che FINDMAX scorre sempre la stessa sequenza, questo puo' essere sfruttato per migliorare l'algoritmo.

Con un albero possiamo rappresentare la relazione d'ordine parziale (tra fratelli non vi c'è relazione). Se ogni confronto viene salvato in un arco tra nodi di un albero possiamo sfruttare la transitività per evitare numerosi confronti.

HEAP BINARIO

La struttura dati che useremo è detta HEAP BINARIO con le seguenti caratteristiche:

- È un albero **COMPLETO**
- $\forall x \in T \quad \text{VAL}(x) \geq \text{VAL}(y)$ con y figlio di x

Costruendo quest' albero avremo come **RADICE** il massimo.

Il vincolo di completezza è fondamentale per il tempo di esecuzione.

Un albero è **PIENO** quando ogni nodo (che non è una foglia) ha grado massimo (il grado di un nodo è il numero di figli che ha, per **BINARIO** si intende che il grado è al massimo 2).

Per un albero binario pieno tutti i nodi interni hanno grado 2 e tutte le foglie sono alla stessa profondità (altezza).

Il nostro obiettivo è quello di trovare l'albero che ha l'altezza minore

TEOREMA

Dato un albero pieno con n nodi posso trovare l'altezza e viceversa

DimOSTRAZIONE

Trattiamo il numero di modi di un albero binario pieno di altezza h .

Per ogni livello abbiamo 2^i modi quindi i modi totali saranno:

$$n = \sum_{i=0}^h 2^i = \frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1 \Rightarrow 2^h \leq n < 2^{h+1} \rightarrow \lg(2^h) \leq \lg n < \lg(2^{h+1})$$

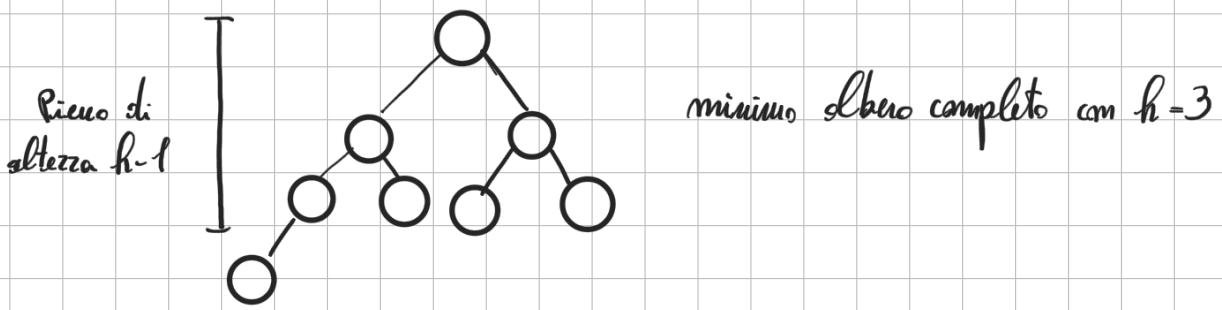
$$\rightarrow h \leq \lg n < h+1 \Rightarrow h = \log_2 n$$

Questa funzione di h è iniettiva ma **NON** è suriettiva, quindi esistono numeri non rappresentati dall'equazione $2^{h+1} - 1$, per questo motivo consideriamo un albero **COMPLETO** che ammette ogni cardinalità.

ALBERO BINARIO COMPLETO

E' un albero in cui:

- Tutti i nodi interni hanno grado 2 al massimo 1
- Tutte le foglie si trovano a profondità $h = h-1$



Il massimo sarà l'albero pieno.

Completeno l'ultimo livello otteniamo tutti i modi compresi tra $2^h - 1$ e $2^{h+1} - 1$.

NON è possibile ricavare il numero di nodi data l'altezza,
vole ancora $\lfloor \lg m \rfloor = h$ notando che l'albero completo è
composto da un albero pieno + le foglie.

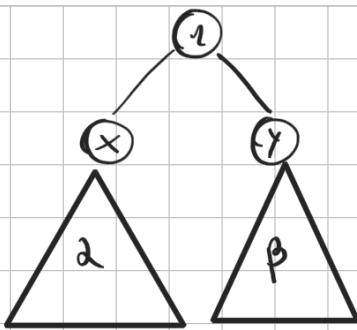
Quindi il numero di modi sarà:

$$2^h \leq m < 2^{h+1} - 1 \rightarrow 2^h \leq m < 2^{h+1} \text{ e di nuovo } \lfloor \lg m \rfloor = h$$

HEAPIFY

E' un algoritmo che trasforma in uno heap un albero binario
completo tale che i due sottosalberi sono heap.

Si ricordi che ogni albero completo è composto da alberi completi
quindi l'unico motivo per i quali l'albero dato non può essere
trasformato è perché la radice non rispetta la relazione d'ordine



α e β sono heap

$\forall z \in \alpha \quad val(x) \geq val(z)$

$\forall z \in \beta \quad val(y) \geq val(z)$

Se r è massimo non viola nessun vincolo, altrimenti uno tra x e y sarà max dell'albero.

Prendiamo $\max\{x, y\}$ e lo scambiamo con r , ma non sappiamo se l'albero con radice r è ancora un albero.

Questo passaggio lo possiamo iterare sul sott'albero modificato potenzialmente fino alle foglie.