

Ricordiamo che

$$T_{BH}(n) = \frac{n}{2} \sum_{i=1}^h i \left(\frac{1}{2}\right)^i$$

Si noti che $\frac{d}{dx} (x^i) = ix^{i-1}$ il secondo oggetto si differenzia solo di poco dal nostro argomento di sommatoria che puoi essere espresso come $i x^i$; moltiplicando però ambo i termini per x otterremo proprio $x \frac{d}{dx} (x^i) = ix^i$ e quindi

$$\sum x \frac{d}{dx} (x^i) = \sum (ix^{i-1})x = x \sum ix^{i-1} \quad \text{possiamo portare fuori } \frac{d}{dx}$$

$$x \frac{d}{dx} \sum x^i = x \sum ix^{i-1}$$

Si noti che (indifferentemente che i parta da 0 o da 1):

$$\sum_{i=0}^h ix^i \leq \sum_{i=0}^h ix^i \leq \sum_{i=0}^{\infty} ix^i \quad \rightarrow x \leq \sum_{i=0}^h ix^i \leq x \frac{d}{dx} \left(\sum_{i=0}^{\infty} x^i \right)$$

Se $x \in (0,1)$ $\sum_0^{\infty} x^i = \frac{1}{1-x}$ quindi sostituendo nella diseguazione

$$x \leq \sum_{i=0}^h ix^i \leq \frac{d}{dx} \left[x (1-x)^{-1} \right] \xrightarrow{\text{derivata del prodotto}} x \leq \sum_{i=0}^h ix^i \leq \frac{x}{(1-x)^2}$$

Nel nostro caso x è un valore costante ($\frac{1}{2}$) quindi $\sum_{i=0}^h ix^i$ è compreso tra 2 valori costanti, deduciamo che è una costante

Ricordiamo che il nostro oggetto originale però era $T_{BH} = \frac{m}{2} \sum_{i=0}^k i x^i$

Quindi in definitiva (moltiplicando) avremo che $T_{BH} = \Theta m$

COSTO HEAP SORT

Ricordiamo l' algoritmo

HEAPSORT(A, m)

BUILDHEAP(A, m)

$\Theta(m)$

FOR $j = m$ DOWNTON 2 DO

$\Theta(m)$

SWAP($A, 1, j$)

}

$\Theta(1)$

HEAPSIZE[A]--

HEAPIFY($A, 1$)

$O(\lg_2 m)$ (sorapprossimazione se heapify ogni volta scorrere tutto l'albero)

$$T_{HS} = \Theta m + \Theta m + O m \lg_2 m = \Theta m \lg_2 m \quad \left(\text{l'espressione esatta e } \sum_{i=1}^m \lg_2 i \right)$$

TEMPO DI HEAPIFY

ci sono 2 modi per dimostrare che $T_{HF} = \Theta m \lg_2 m$

① calcolando i costi di tutte le chiamate osserviamo che lo heap decresce dal basso (rimozione di un nodo). Heapify viene chiamato dalla radice in cima e il tempo di esecuzione è lineare sull'altezza.

Ci voranno (ipotizzando un albero pieno, ma vale sempre essendo un caso particolare) $n/2$ (n° di foglie) chiamate per ridurre l'altezza (rimuovere tutti i nodi di un livello). Queste $n/2$ chiamate costano quindi $n/2 \Theta(h) = \Theta(nh)$ ma $h = \lg_2 m \rightarrow \Theta(n \lg_2 m)$. Questo è il caso peggiore le successive costeranno sicuramente meno (h decresce) quindi il tempo sarà **ALMENO** $n \lg_2 m$ (dobbiamo aggiungere le chiamate successive), ma prima abbiamo visto che la stima per ecceso era proprio $n \lg_2 m$ quindi il costo è $n \lg_2 m$.

$$\begin{aligned} ② T_{HS} &= \sum_{i=1}^m \lg_2 i = (\lg_2 1 + \lg_2 2 + \dots + \lg_2 n) \text{ applico la proprietà dei logaritmi} \\ &= \lg_2 (1 \cdot 2 \cdot 3 \cdots n) = \lg_2 (n!) \cong \lg_2 n^n = n \lg_2 n \end{aligned}$$

In definitiva il $T_{HS} = \Theta(n \lg_2 m)$

QUICK SORT

Il quick sort a differenza dell'heap sort ha un caso medio ottimale ma non nel caso peggiore.

Il quick sort ha la stessa idea di base del merge sort ma differisce sul come dividere le sequenze e come riunirle.

L'algoritmo generale è

DIVIDE ET IMPERA (A, m)

IF !CASO_BASE (A, m) THEN

$((A_{m_1}, A_{m_2})) = \text{DECOMPONI} (A, m)$

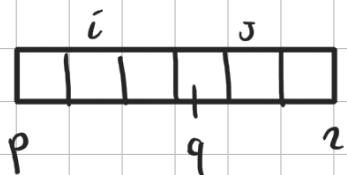
$S_1 = \text{DIVIDE_ET_IMPERA} (A, m_1)$

$S_2 = \text{DIVIDE_ET_IMPERA} (A, m_2)$

$S = \text{MERGE} (S_1, S_2)$

In particolare quicksort punta a rimuovere il merge immaginando che quando ordina le 2 sequenze tra loro poi già risultino ordinate.

Questo avviene aumentando la complessità dell'algoritmo di decomposizione



Vogliamo garantire due proprietà al termine del partizionamento ovvero:

① $p \leq q < r$

② $\forall p \leq i \leq q \quad \forall q+1 \leq j \leq r \quad A[i] \leq A[j]$

Queste ci garantiscono il corretto funzionamento dell'algoritmo.

Vediamo l'algoritmo

QUICKSORT (A, p, r)

IF $p < r$ THEN

$q = \text{PARTIZIONA} (A, p, r)$

QUICKSORT (A, p, q)

QUICKSORT ($A, q+1, r$)

non siamo nel caso base

PARTIZIONAMENTO individua q e forza le proprietà.