EX.5: $T(n) = 8T(\frac{n}{2}) + \Theta(1)$ (Eq. di ricorrenza dell'alg. per la molt. tra matrici)

$f(n) = \Theta(1)$ (driving function)

$w(n) = n^{\log_2 8} = n^3$ (watershed function)

$f(n) = \Theta(1) = O(n^{3-\varepsilon})$ $0 < \varepsilon < 3$ $\rightarrow$ caso 1

$T(n) = \Theta(n^3)$

EX6.: $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$ (eq. di ricorrenza dell'alg. di Strassen per la molt. tra matrici)

$f(n) = \Theta(n^2)$

$w(n) = n^{\log_2 7}$

$\log 7 = 2.807\ldots$

$\varepsilon < 0.807$

$f(n) = \Theta(n^2) = \Theta(n^{\log 7 - \varepsilon}) = O(n^{\log 7 - \varepsilon}) \rightarrow$ caso 1 $\rightarrow$

$T(n) = \Theta(n^{\log 7})$.

⚠ SITUAZIONI IN CUI NON È POSSIBILE APPLICARE IL METODO MASTER:

- es. $w(n)$ e $f(n)$ non sono asintoticamente comparabili.
  - C'è un gap tra caso 1 e caso 2: $f(n) = o(w(n))$
  - C'è un gap tra il caso 2 e il caso 3: $f(n) = \omega(w(n))$
  - per il caso 3 potrebbe non essere soddisfatta la cond. di regolarità.

Es. 7. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log n}$ $f(n) = \frac{n}{\log n}$

$$w(n) = n^{\log_2 2} = n$$

$$f(n) = \frac{n}{\log n} = O(n) \rightarrow$$ $f(n)$ cresce più lentamente di $w(n)$

$$\Updownarrow$$

$$\lim_{n\to\infty} \frac{n}{\frac{n}{\log n}} = \lim_{n\to\infty} \log n = \infty$$

Precisamente,

$$\forall \varepsilon>0, \quad \log(n) = o(n^{\varepsilon}) \iff \frac{1}{\log n} = \omega(n^{-\varepsilon})$$

$$\Rightarrow \frac{n}{\log n} = \omega(n^{1-\varepsilon}) = \omega(n^{\log_b a - \varepsilon})$$

$$\Downarrow$$

$$\nexists \varepsilon>0 : \frac{n}{\log n} = O(n^{\log_b a - \varepsilon})$$ ~~caso 1~~

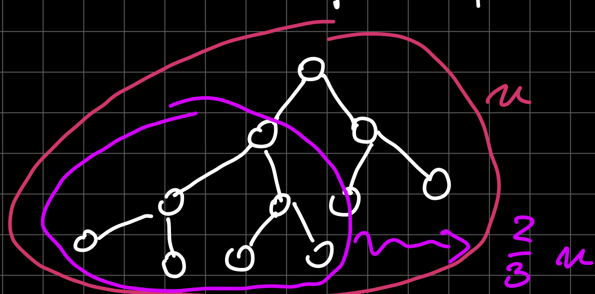
$$\frac{n}{\log n} = \Theta(\underbrace{n^{\log_b a}}_{n} \log^k n)$$ solo $k = -1 \ngeq 0$. ~~caso 2~~

4.1. MAX-HEAPIFY

Serve a conservare la condizione $\forall\, i \neq root$

$A[parent(i)] \geq A[i]$. (Max-heap condition)

Assumiamo che gli alberi binari con radice LEFT(i) e RIGHT(i) siano max-heap.
Però $A[i]$ potrebbe essere più piccolo dei suoi figli.



$T(n)$

$n = 1 \qquad T(1) = \Theta(1)$

$$T(n) \leq \underbrace{\Theta(1)}_{\substack{\text{tempo necessario}\\ \text{a correggere}\\ \text{la relazione}\\ \text{tra } A[i],\ A[\text{LEFT}(i)]\\ A[\text{RIGHT}(i)]}} + \underbrace{T\left(\frac{2}{3} n\right)}_{\substack{\text{tempo per}\\ \text{eseguire MAX-HEAPIFY}\\ \text{sul sottoalbero}\\ \text{radicato a uno}\\ \text{dei due nodi}\\ \text{figli}}}$$

$$T(n) \leq \Theta(1) + T\left(\frac{2}{3} n\right)$$

$f(n) = \Theta(1)$

$w(n) = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$

$f(n) = \Theta(1) = \Theta(1 \cdot \log^0 n)$

$\Theta(n^{\log_b a} \cdot \log^k n) \xrightarrow[\text{caso 2 Teo Master}]{} T(n) = O(\log n)$

Se eseguo max-heapify su un sottoalbero radicato ad un nodo di altezza h

$$T(h) = O(h)$$

<u>3.4</u> BUILD MAX-HEAP

1. A.heapsize = n
2. for $i = \lfloor \frac{n}{2} \rfloor$ to 1, i--
3. MAX-HEAPIFY (A, i)

"All'inizio di ogni iterazione del for loop alle linee 2-3, ogni nodo $\{i+1, \dots, n\}$ è la radice di un max-heap".

Proprietà invariante

Inizializzazione: ✓ alla prima iterazione $i = \lfloor \frac{n}{2} \rfloor$ e $\{\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \dots, n\}$ sono foglie ⇝ max-heap banali.

Mantenimento: alla i-esima iterazione $\{i+1, \dots, n\}$ sono radici di max-heap; in più max-heapify fa sì che l'albero radicato ad i sia un max-heap. Quindi alla successiva iterazione vale la prop. invariante.

Terminazione: L'alg. esegue $\lfloor \frac{n}{2} \rfloor$ iterazioni. Al termine $i = 0$. Per la propr. inv. $\{1, 2, \dots, n\}$ sono radici di max heap. In part. è vero per il nodo 1.

Calcoliamo un upper bound al tempo computazionale di BUILD-MAX-HEAP:

$$T(n) = O(\log n) \cdot \underbrace{O(n)}_{\#\text{itera}} = O(n \log n)$$

Non è preciso!!

max-heapify

OSS: Un heap di n elementi ha altezza $\lfloor \log n \rfloor$ e ha al più $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodi che sono radici di sottoalberi di altezza h.

Es: $h=0$ sono le foglie e sono $\left\lfloor \frac{n}{2} \right\rfloor$.

costante nascosta in $O(h)$ → tempo di max-heapify

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil c \cdot h \ (=)$$

$$0 \le h \le \lfloor \log n \rfloor \Rightarrow \frac{n}{2^{h+1}} \ge \frac{1}{2}$$

$$\downarrow$$

$$h \le \log n$$

$$\downarrow$$

$$2^h \le n \to \frac{n}{2 \cdot 2^h} \ge \frac{1}{2} \quad \uparrow$$

Inoltre, $\lceil x \rceil \le 2x \quad \forall x \ge \frac{1}{2} \to \left\lceil \frac{n}{2^{h+1}} \right\rceil \le \frac{n}{2^h}$ (⊛)

$$(=) \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil c h \overset{(⊛)}{\le} \sum_{h=0}^{\lfloor \log n \rfloor} \frac{n}{2^h} c h = c n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \le$$

$$c n \sum_{h=0}^{\infty} \frac{h}{2^h} = c n \sum_{h=0}^{\infty} h \left(\frac{1}{2}\right)^h$$

$$= c n \frac{1/2}{1/4} = 2 c n = O(n).$$

↓ lineare

$$\sum_{n=0}^{\infty} n x^n = \frac{x}{(1-x)^2}$$

4.3 ANALISI DELLE TABELLE HASH

Una tabella hash è una struttura dati efficace a implementare dizionari quando il # delle chiavi memorizzate $|K|$ è più piccolo di quello di tutte le chiavi possibili $|U|$.

Usa solo $\Theta(|K|)$ di memoria pur mantenendo tempo $O(1)$ per la ricerca di un elemento, nel caso medio.

⚠

DEF: Una funzione hash mappa l'universo U delle possibili chiavi negli slot di una tabella hash $T[0, \dots, m-1]$,

$$h : U \longrightarrow \{0, \dots, m-1\} \quad , \quad m << |U|$$

Collisioni: $\exists\, k_1, k_2 \in K \subseteq U \;:\; k_1 \neq k_2$ ma $h(k_1) = h(k_2)$.

Una funzione hash è uniforme e indipendente (aka random oracle)

- uniforme: $\forall\, k \in U, \quad \forall\, j \in \{0, \dots, m-1\}, \; Pr[h(k) = j] = \frac{1}{m}$

- indipendente $\forall\, k_1, k_2 \in U, \; k_1 \neq k_2$

$$Pr[\, h(k_1) = j_1 \wedge h(k_2) = j_2 \,] = Pr[\, h(k_1) = j_1 \,]\; Pr[\, h(k_2) = j_2 \,].$$