

Time Complexity Analysis

Priority Queues

① Time Complexity for Building Priority Queue using a MAX HEAP.

→ We call the max-heapify function in a bottom to top fashion.

→ Amortized Cost Analysis (AGGREGATE METHOD) → assuming a complete tree with $n = 2^{h+1} - 1$ where h is height tree

→ Bottom most level there are 2^h nodes, but we don't call max-heapify on any of this node. So Cost = 0

→ At next to bottom level, 2^{h-1} nodes, each may move 1 level down

→ In general, level j from bottom → 2^{h-j} nodes & each may move j levels.

$$\therefore \text{Total time} = T(n) = \sum_{j=0}^n j 2^{h-j} = \sum_{j=0}^n j \frac{2^n}{2^j}$$

$$T(n) = 2^h \sum_{j=0}^h \frac{j}{2^j}$$

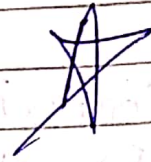
$$\leq 2^h \sum_{j=0}^{\infty} \frac{j}{2^j} < 2^h \cdot 2 = 2^{h+1}$$

explained in next page (APPENDIX A)

$$\therefore T(n) \leq 2^{n+1}$$

We assumed $n = 2^{n+1} - 1$ (complete tree)

$$\boxed{\therefore T(n) \in O(n)}$$



★ Build in of order n .

APPDX A $\sum_{j=0}^{\infty} \frac{1}{2^j} = 2$

Proof $\sum_{j=0}^{\infty} x^j = \frac{1}{1-x}$ holds for $x < 1$

Take derivative on both sides

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

Multiply by x

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

★ Put $x = 1/2$

$$\sum_{j=0}^{\infty} j/2^j = 2$$

Hence Proved

② Am Cost Analysis INSERT in Priority Queue

- Worst case analysis \rightarrow We push the new value at the end of array, if ~~it~~ it has highest priority it will have to shift $\log n$ levels.

\therefore Worst case complexity $O(\log n)$ ★

• Amortized Complexity Analysis

\rightarrow I will use potential method.

\rightarrow Let ~~be~~ S_i be the state of pq after i^{th} operation & consisting of n_i nodes.

\rightarrow Let each individual insert take at most (worst) $K \log n_i$ time.

\rightarrow Define potential function ϕ

$$\phi = \begin{cases} 0 & \text{if } n_i = 0 \\ K n_i \log n_i & \text{else} \end{cases}$$

\rightarrow Assuming i^{th} insert on a non empty heap \rightarrow
 $n_i = n_{i-1} + 1$

~~for~~ Since heap was non empty, $n_i \geq 2$

$$\hat{C}_i = C_i + \phi(D_i) - \phi(D_{i-1})$$

$$\leq K \log n_i + K n_i \log n_i - K n_{i-1} \log n_{i-1}$$

$$= K \log n_i + K n_i \log n_i - K(n_i - 1) \log(n_i - 1)$$

$$= K \log n_i + K n_i \log n_i - K n_i \log(n_i - 1) + K \log(n_i - 1)$$

$$\leq 2K \log n_i + K n_i \log \frac{n_i}{n_i - 1}$$

$$\leq 2K \log n_i + 2K \quad [\text{details APPDX}]$$

$$= O(\log n_i)$$

PS: For insert on empty heap, cost will be 0.

APPDX B. Prove $n \log \frac{n}{n-1} \leq 2$

Prove $n \log \frac{n}{n-1} = n \log \left(1 + \frac{1}{n-1} \right)$

$$= \log \left(1 + \frac{1}{n-1} \right)^n$$

$$\leq \log(e^{1/n-1})^n$$

$$= \log e^{n/n-1} = n/n-1 \leq 2$$

\therefore Amortized Complexity for Insert = $O(\log n)$ ★

③ Complexity for pop front.

Worst Case \rightarrow We replace top of the ~~array~~ queue with last element of array. To maintain the Priority Queue structure, the last \times max heapify call may have $\log n$ swaps.

\therefore Worst Case = $O(\log n)$ ★

\rightarrow Amortized Complexity

\rightarrow using same potential function as before. After i th operation, number of nodes $n_i = n_{i-1} - 1$

$$C_i = C_i + \phi(D_i) - \phi(D_{i-1})$$

Ex

$$\leq K \log n_{i-1} + K n_i \log n_i - K n_{i-1} \log n_{i-1}$$

$$= K \log n_{i-1} + K (n_{i-1} - 1) \log (n_{i-1} - 1) - K n_{i-1} \log n_{i-1}$$

$$= K \log n_{i-1} + K n_{i-1} \log (n_{i-1} - 1) - K \log (n_{i-1} - 1) - K n_{i-1} \log n_{i-1}$$

$$= K \log \frac{n_{i-1}}{n_{i-1} - 1} + K n_{i-1} \log \frac{n_{i-1} - 1}{n_{i-1}}$$

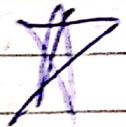
$$\leq K \log \frac{n_{i-1}}{n_{i-1} - 1} + K (\log 1) (n_{i-1})$$

$$= K \log \frac{n_{i-1}}{n_{i-1}-1}$$

$$\leq K \log 2$$

(assuming $n_{i-1} > 2$)

$$\boxed{\hat{c}_i \in O(1)}$$



✱ For $O(1)$ amortized insertion complexity, one can use Fibonacci Heaps where costly operations like heapify are done LAZILY