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 dont 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.

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

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

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

explained in next page (APPDX A)

Date / /

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

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

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

☆ Build is of order n.

<u>APPDX A</u> $\displaystyle\sum_{j=0}^{\infty} \frac{j}{2^j} = 2$

Proof $\displaystyle\sum_{j=0}^{\infty} x^j = \frac{1}{1-x}$ 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 Prooved

(2) ~~Amt~~ Cost Analysis INSERT in Priority Que

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

$\therefore$ $\boxed{\text{Worst Case complexity } O(\log n)}$

- Amortized Complexity Analysis

→ I will use potential method.

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

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

→ Define potential function ϕ

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

→ Assuming i^{th} insert on a non empty heap →

$$n_i = n_{i-1} + 1$$

~~(a)~~ Since heap was non empty, $n_i \geq 2$

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

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

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

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

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

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

$$= O(\log n_i)$$

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

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

Proof

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

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

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

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

$\therefore$ $\boxed{\text{Amortized Complexity for Insert} = O(\log n)}$ ☆

(3) Complexity for pop front.

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

$\therefore$ $\boxed{\text{Worst Case} - O(\log n)}$ ☆

→ Amortized Complexity
→ using same potential function as before. After i^{th} operation, number of nodes $n_i = n_{i-1} - 1$

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

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

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

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

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

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

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

$$\leq k\log 2 \quad \text{(assuming } n_{i-1} > 2\text{)}$$

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

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