OTT Today: 1:15 pm - 2:15 pm

Exam 1: Tue, March 2.

Stacks.

Doubling Strategy.

Push (obj)

\[ A(s) \leftarrow \text{obj} \]
\[ s \leftarrow s + 1 \]
\[ \text{if } s = a \text{ then } \]
\[ a \leftarrow a \times 2 \]

Copy elements from the old array into the new array.

Analysis: We will find the total time taken for \( n \) operations (Push).

\[
\begin{array}{c}
\text{worst case time of one push is } O(s) \text{ s} \leq n. \\
\text{This can happen for time.} \\
\therefore \text{total cost } \leq n \times \text{time}
\end{array}
\]
Total cost = time to push the elements +

\[ = c'n + c(1+2+2^2+2^3+\cdots+2^{\lg n}) + 2^{\frac{\lg n}{2}} + 2^{\frac{\lg n+1}{2}} \]

\[ = c'n + c(2^{\lg n+1}-1) + c(2^{\lg n+2}-1) \]

\[ = c'n + c(2n-1) + c(4n-1) \]

\[ = \Theta(n). \quad \text{(amortized analysis)} \]

\[ \text{\text{total time for a set} } f \]
Amortized cost of a push: $O(1)$. 

Amortized cost of each push: $7 \approx O(1)$. 

Pop() 

when the number of elements becomes $< \frac{a}{4}$ then we decrease the size of the array by half, i.e. $a \leftarrow \frac{a}{2}$. 

1 2 3 4 5 6 7 8
Queues:

Enque, Deque.

adds at the end, removes it from the front.
Heaps:

A Heap is an array object that can be viewed as an almost complete binary tree.

A binary tree where all levels except the last is full & the last level is full from left until where the last element is, i.e., there are no "holes" at the last level.

Heap size: # elements in the heap.
Array size: Size of the array.

Heap property (Max heap): For any element \( i \),

\[ A[\text{parent}(i)] > A[i] \]

For Min heap: \( A[\text{parent}(i)] \leq A[i] \).

Height of the heap: \( \Theta(\log n) \).

\( h \): height of the heap.

Total # nodes in all levels except the last
\[ = 2^0 + 2^1 + \cdots + 2^h = 2^h - 1 \]

The last level must have \( \geq 1 \) node.

\[ \therefore \text{Total \# nodes} \quad n \geq 2^h - 1 + 1 = 2^h. \]

\[ \therefore 2 \leq n \]

\[ \text{Total \# nodes} \quad n \leq 2^0 + 2^1 + \cdots + 2^h = 2^{h+1} - 1 \]

\[ \therefore n \leq 2^{h+1} \]

\[ \therefore 2 \leq n < 2^{h+1} \]

\[ \therefore h \leq \lfloor \log n \rfloor < h+1 \]

\[ \therefore h = \lfloor \log n \rfloor. \]

Max Heapify \((A, i)\)  

Fixes the heap property for the tree rooted at \(i\).

// the trees rooted at \(i\)'s children are max heaps.
\[
\begin{align*}
\text{l} & \leftarrow \text{left}(i) \\
\text{r} & \leftarrow \text{right}(i) \\
\text{if } \text{l} < \text{heapisze and } A(l) > A(c[i]) \text{ then} & \\
\text{largest} & \leftarrow \text{l} \\
\text{else} & \\
\text{largest} & \leftarrow i \\
\text{if } \text{r} < \text{heapisze and } A(r) > A(\text{largest}) \text{ then} & \\
\text{largest} & \leftarrow \text{r} \\
\text{if largest} \neq i \text{ then} & \\
\text{Swap } A[i] \text{ with } A[\text{largest}] & \\
\text{Max Heapify (A, largest)} &
\end{align*}
\]

\begin{align*}
\text{Running time} & \leq \text{ht of the heap} \\
& = \Theta(n) \\
\text{Runtime recurrence.}
\end{align*}
\[ T(n) = \frac{2n}{3} \cdot \left( \frac{n}{2} \right) + O(1). \]

Simplified Master Theorem:

\( a = 1, \ b = \frac{3}{2}, \ k = 0. \)

\( a = b^k \Rightarrow \text{Case II: } \Theta (a \cdot \log_b n) \)

\[ = \Theta \left( \log_{\frac{3}{2}} n \right) \]

\[ = \Theta (\log n). \]

BuildHeap \( A \)
\[
\text{heapsize} \leftarrow \lfloor \frac{\lfloor \frac{\lfloor \lfloor A \rfloor \rfloor}{2} \rfloor}{2} \rfloor
\]

For \( i \leftarrow \lfloor \frac{\lfloor \frac{\lfloor \lfloor A \rfloor \rfloor}{2} \rfloor}{2} \rfloor \) do

\[\text{MaxHeapify} (A, i)\]

Runtime: \( \text{MaxHeapify} : \Theta (\log n) \)

loop: \( n \) times

\( \Rightarrow \Theta (n \log n) \times O(n \log n) \leq \)
Lemma: At height $h$ there are at most \[
\left\lfloor \frac{n}{2^{h+1}} \right\rfloor \text{ nodes}.
\]

Runtime: $\Theta(n)$. 