Complexities

Array

Append	
Sorted Append	
Delete (+shift)	
Binary Search	

Linked List

Insert	
Insert (Enhanced)	$\mathcal{O}(1)$
Delete (+shift)	$\mathcal{O}(n)$
Traverse(Search)	$\mathcal{O}(n)$

Enhanced Linked Lists give pointers to both head and tail

Tree/Heap

Insert	
Delete	

Heapify/Heap sort

Top Down	$\mathcal{O}(n \cdot log_2 n)$
Bottom up	$\mathcal{O}(n)$

For the $T(n)$ for the number of sinks run by Heapify, the sequence is:

$$T(n) = \frac{n}{2} \cdot 0 + \frac{n}{4} \cdot 1 + \frac{n}{8} \cdot 2 + \cdots + 1 \cdot h$$

Big O no

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \neq 0, \infty \Rightarrow f \in \Theta(g)$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \neq \infty \Rightarrow f \in O(g)$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \neq 0 \Rightarrow f \in \Omega(g)$$

worst case $f(n) \in \mathcal{O}(g(n)) : f(n) \leq Cg(n)$

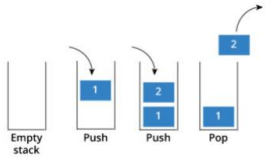
best case $f(n) \in \Omega(g(n)) : f(n) \geq Cg(n)$ $\forall n > n_0$

average case $f(n) \in \Theta(g(n)) : C_1 g(n) \leq f(n) \leq C_2 g(n)$

Stacks

Push $\mathcal{O}(1)$

Pop $\mathcal{O}(1)$



Queue

Enqueue $\mathcal{O}(1)$

Dequeue $\mathcal{O}(1)$

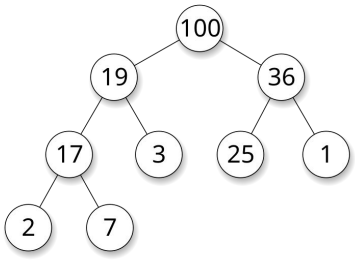
Heaps

These are the *only* conditions:

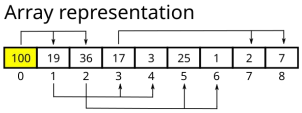
Max Heap $parent \geq child$

Min Heap $parent \leq child$

Tree representation



Array representation



Array, node at index i:

$$\text{left}(i) = 2i + 1$$

$$\text{right}(i) = 2i + 2$$

$$\text{parent}(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$$

Height and Size

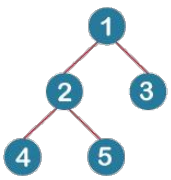
$$h(T) = \begin{cases} -1 & \text{if } T \text{ is null} \\ 1 + \max(h(T_{\text{left}}), h(T_{\text{right}})) & \text{otherwise} \end{cases}$$

$$s(T) = \begin{cases} 0 & \text{if } T \text{ is null} \\ 1 + s(T_{\text{left}}) + s(T_{\text{right}}) & \text{otherwise} \end{cases}$$

Number of nodes

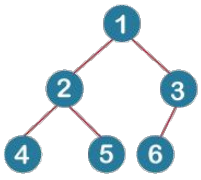
$$N = 2^{h(T)+1} - 1$$

Strict (or Full)



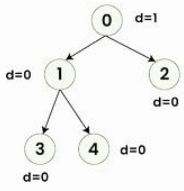
Each node has 0 or 2 children

Complete

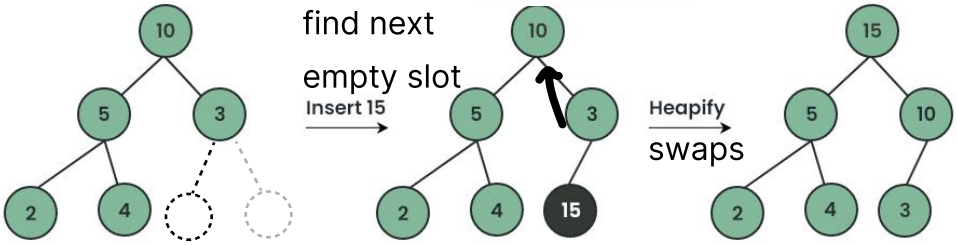


Each before the last level is filled

Balanced Binary Tree



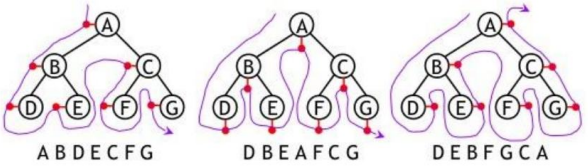
d = Depth of left subtree - Depth of right subtree = 0 or 1



Trees

Imagine a flag attached to each node:

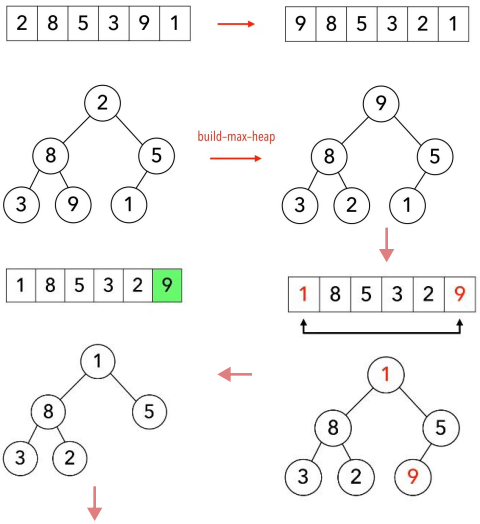
To traverse the tree, collect the flags:



LNR NLR LRN

Heap Sort

Top Down (siftDown) Approach:



heapify and repeat until sorted

Bottom up (siftUp)

Pick last unsorted node, heapify up. Repeat until sorted.