Priority Queues (Heaps)

Each job within a computer takes turns using the cpu. If a job comes earlier than another job it needs to be executed earlier. If a job has been scheduled, it should not be scheduled a second time if there exists a job that hasn't been scheduled once. The queue is the answer.

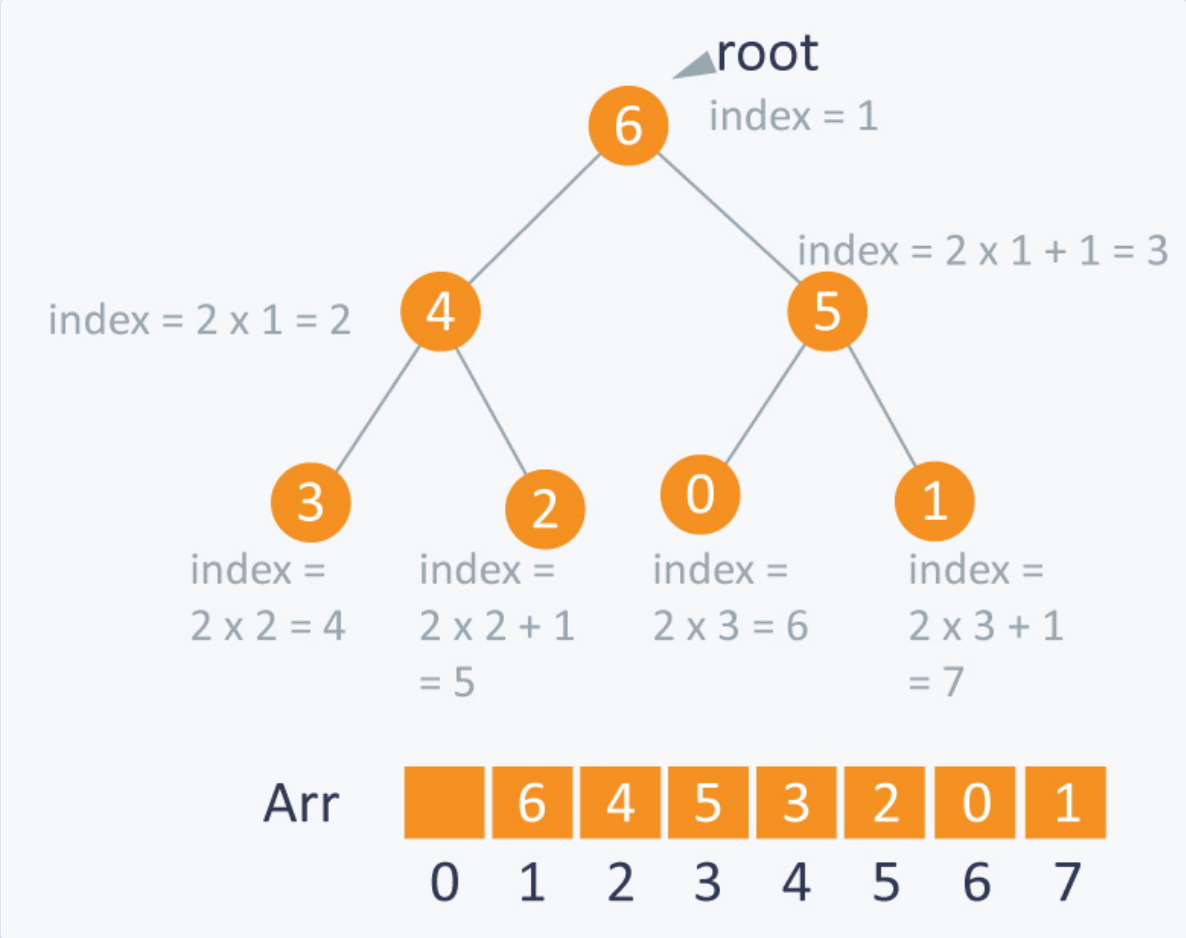
- first in first out
- when a new job comes in it is put at the end of the queue

Regular Queues

- Enqueue adds a new element
- Dequeue removes the eldest element
- Insert adds a new element
- deleteMin removes the minimum element in a priority queue

Application of Priority Queue

- Find the shortest job
- Scheduling next event based on time
- find greedy algorithms



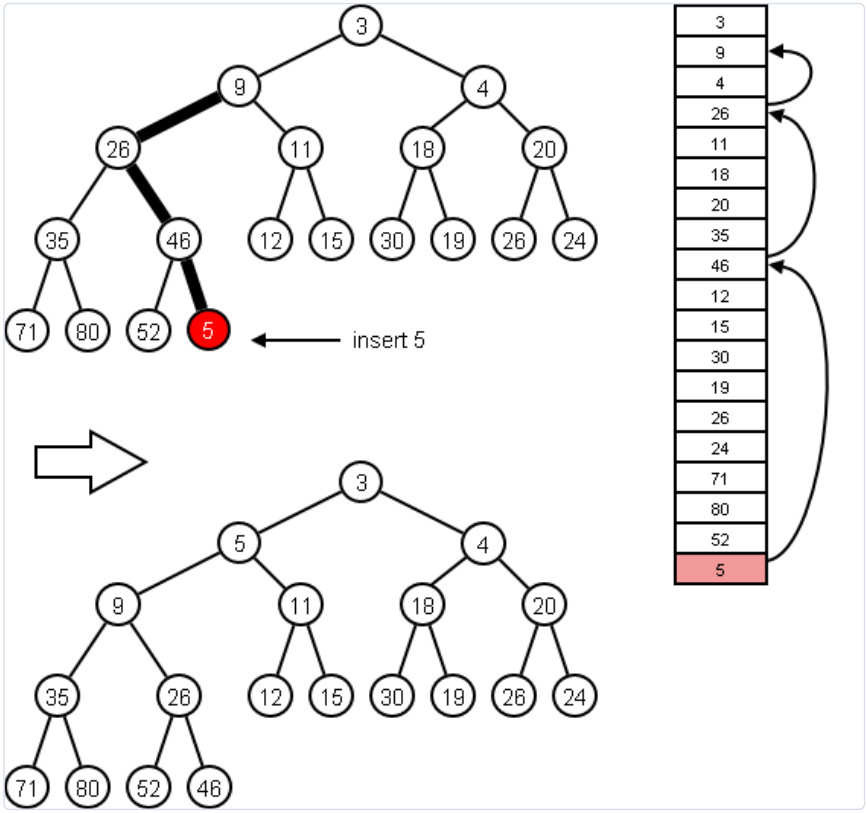
Applications of Priority Queue

Implemented as adaptor class around

- Linked list
  - O(N) worst case in insert or deleteMin
- AVL Tree
  - O(log(N)) worst case on insert and delete
  - Can be overkill when compared to heap, due to being sorted
- Heaps
  - O(log(N)) worst case for both insert and delete

Partially Ordered Trees (POT)

- There is an order relation <= defined for the vertices of T
- For any vertex p and any child c of p, p <= c
- smallest element is the root
- no conclusion can be drawn about the children



Binary Heaps

- A **binary heap** is a partially ordered complete binary tree
  - The tree is completely filled on all levels, with the exception of possibly the lowest
  - d-Heap, a parent node can have d children
  - just refer to as heaps

A vector representation of a complete binary tree is as follows...
Storing element in a vector in level order

- Parent of `v[k] = v[k/2]`
- Left Child of `v[k] = v[2*k]`
- Right Child of `v[k] = v[2*k + 1]`

Basic Heap Operations

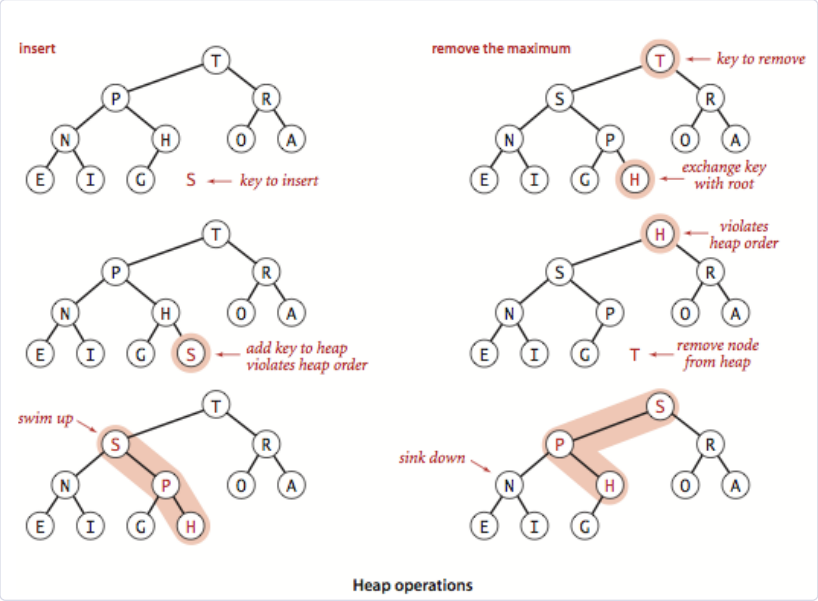
Insert

1. Create a hole at the next leaf
2. // Repair upward
3. Repeat
4. Locate Parent
5. if POT not satisfied (should x inserted in the hole)
    1. sliding parent element to hole
6. else
    1. stop

Delete Min

- Move the last element to the root and then fix the heap
    - Find the smaller child. If the new element is larger than the smaller child, swap with the small child (POT property violated), swap with the small child
    - Repeat in the new subtree

1. Delete the root
2. // root becomes a hole
3. Must move last element (leftmost node) to somewhere
4. let y be the last element (rightmost node)
5. Repeat
    1. find the smaller child of the node
    2. if POT not satisfied should y inserted in hole, sliding smaller child in whole
6. else
    1. stop
7. insert y into hole



Heap operations

Constructor

- Insert each element
- Worst Case O(N(Log(N))
- First insert all elements into a tree without worrying about POT, then satisfy the POT