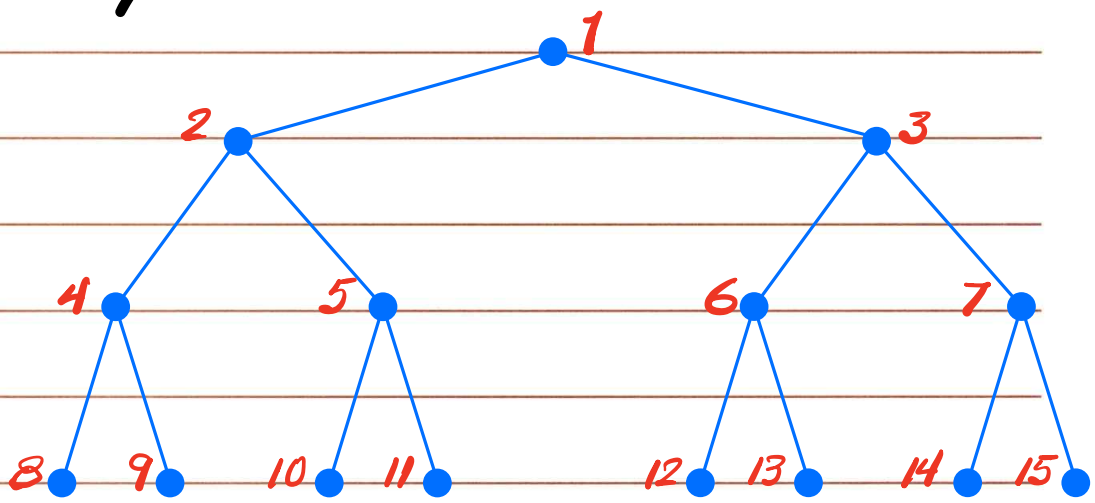
Priority Queues

A priority queue has to perform these two operations fast!

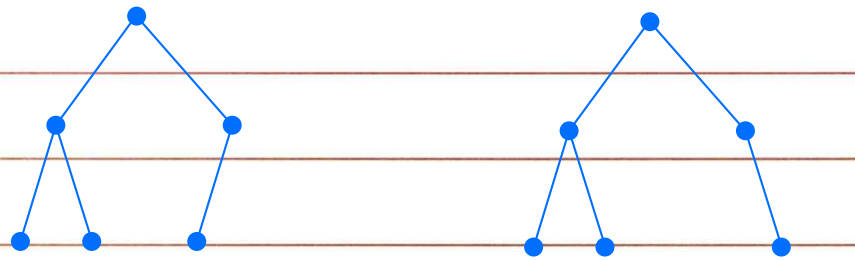
1. Insert an element into the set
2. Find the smallest element in the set

	Insert	Find Min
Array Implementation	O(1)	O(n)
Sorted array "	O(n)	O(1)
Linked list "	O(1)	O(n)
Sorted Linked list "	O(n)	O(1)

Def. A binary tree of depth k which has exactly $2^k - 1$ nodes is called a full binary tree.



Def. A binary tree with n nodes and of depth k is complete iff its nodes correspond to the nodes which are numbered 1 to n in the full binary tree of depth k.



Complete

Not Complete

Traversing a complete binary tree stored as an array.

Parent(i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$
if $i = 1$, i is the root

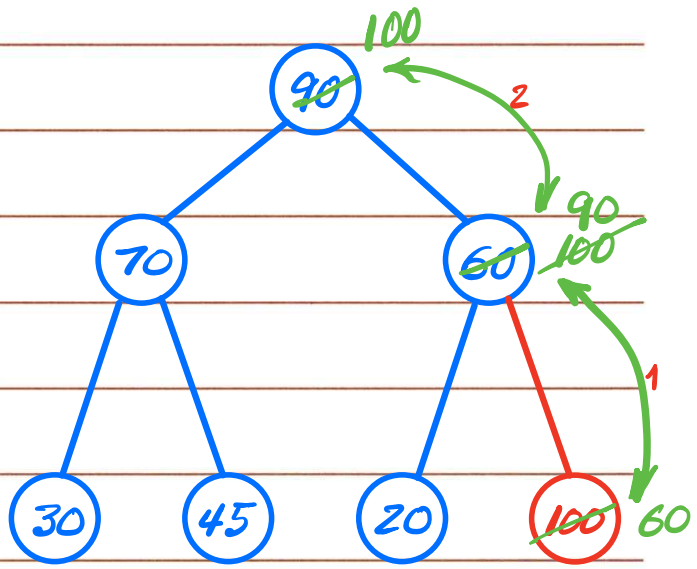
Lchild(i) is at $2i$ if $2i \leq n$
otherwise, it has no left child

Rchild(i) is at $2i+1$ if $2i+1 \leq n$
otherwise, it has no right child

Def. A binary heap is a complete binary tree with the property that the value of the key at each node is at least as large as the key values at its children (Max heap)

Find_Max

Takes O(1)



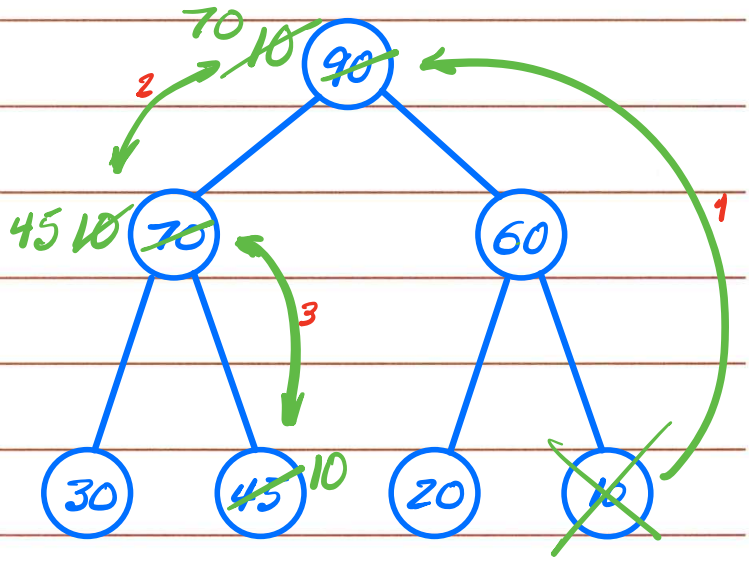
Insert

ex. insert (100)

Takes O(lgn)

Extract_Max

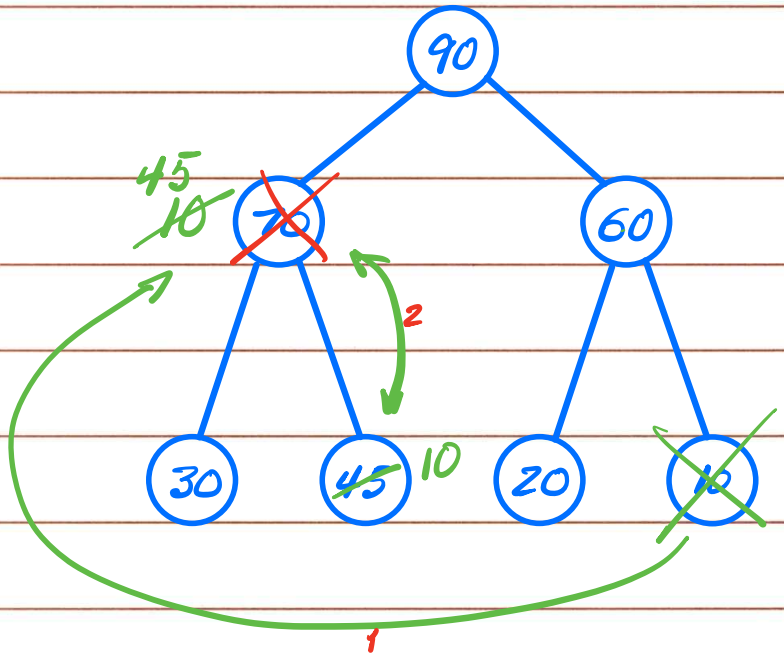
Takes O(lgn)



Delete

ex. Delete (70)

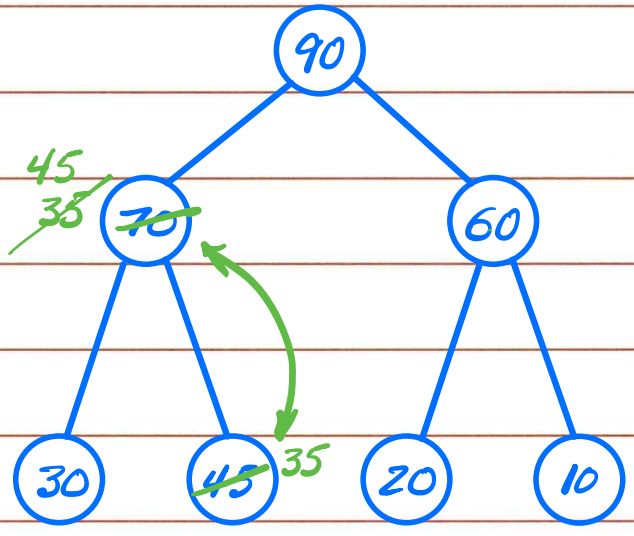
Takes O(lgn)



Decrease-key

ex. Decrease-key (70) to (35)

Takes O(lgn)

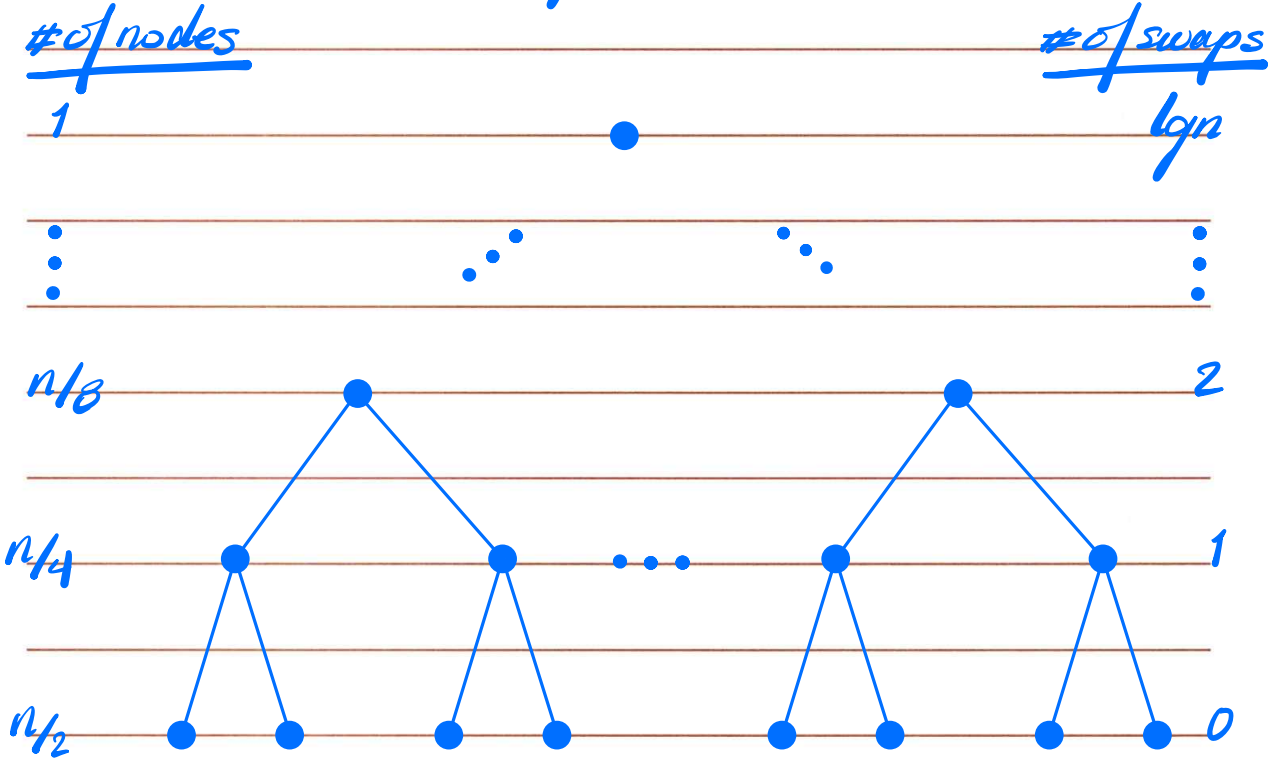


Construction

Can be done in $O(n \lg n)$ time using n insert operations.

Can we do better?

Bottom up construction



T = Maximum number of swaps needed

$T = n/4 * 1 + n/8 * 2 + n/16 * 3 + \dots$

$T/2 = n/8 * 1 + n/16 * 2 + n/32 * 3 + \dots$

$T - T/2 = \underbrace{n/4 * 1 + n/8 * 1 + n/16 * 1 + \dots}_{n/2}$

$T/2 = n/2$

$T = n$

Bottom up construction takes $O(n)$

Q: What is the best run time to merge two binary heaps of size n?

A: $O(n)$ using linear time construction

Finding the top k elements in an array.

Input: An unsorted array of length n.

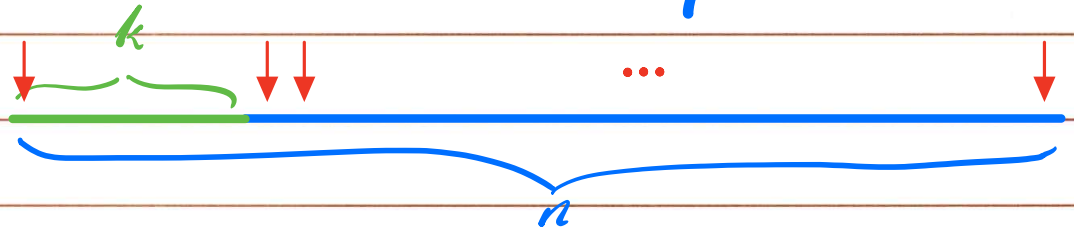
Output: Top k values in the array ($k < n$)

Constraints:

- You cannot use any additional memory
- Your algorithm should run in time $O(n \lg k)$

Solution:

- Construct a Minheap of size k.



- Move through the rest of the $(n-k)$ elements. If we encounter an element larger than the element at the top of our minheap, swap them. This involves extract-min and insert ($O(\lg k)$)

Overall complexity $= O(k) + O((n-k)\lg k) = O(n \lg k)$