Priority Queues

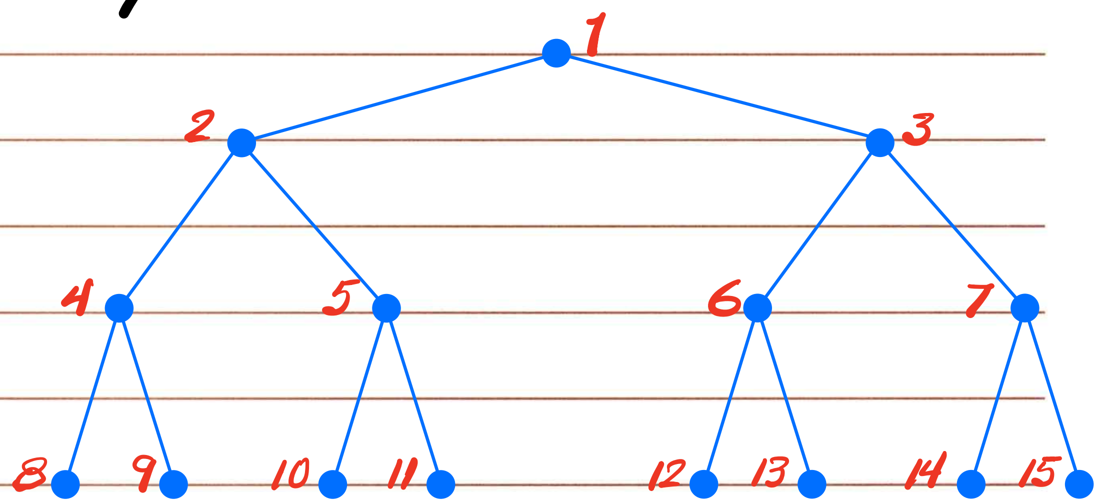
A priority queue has to perform these two operations fast!

1_ Insert an element into the set

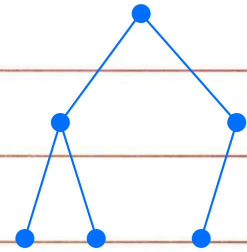
2. Find the smallest element in the set

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

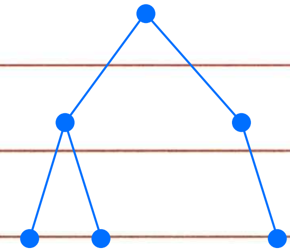
Def. A binary tree of depth k which has exactly $2^k - 1$ nodes is called a full binary tree.



Def. A binary tree with $\underline{n}$ nodes and of depth k is complete iff its nodes correspond to the nodes which are numbered $\underline{1}$ to $\underline{n}$ 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 (Maxheap)

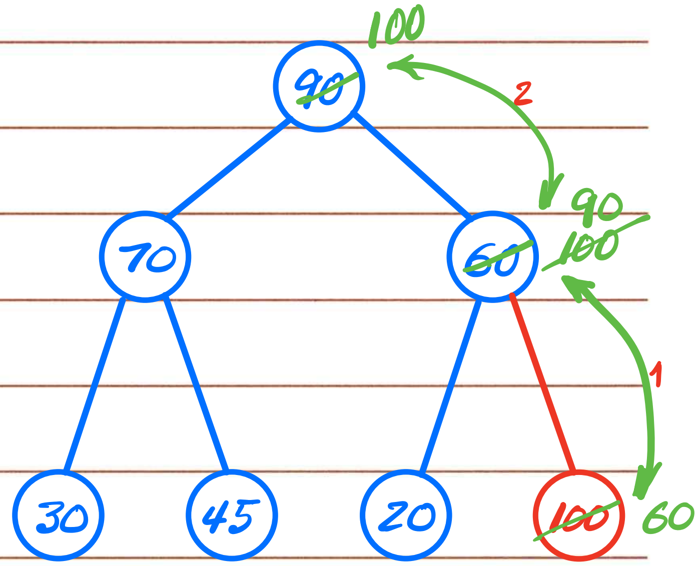
Find_Max

Takes O(1)

Insert

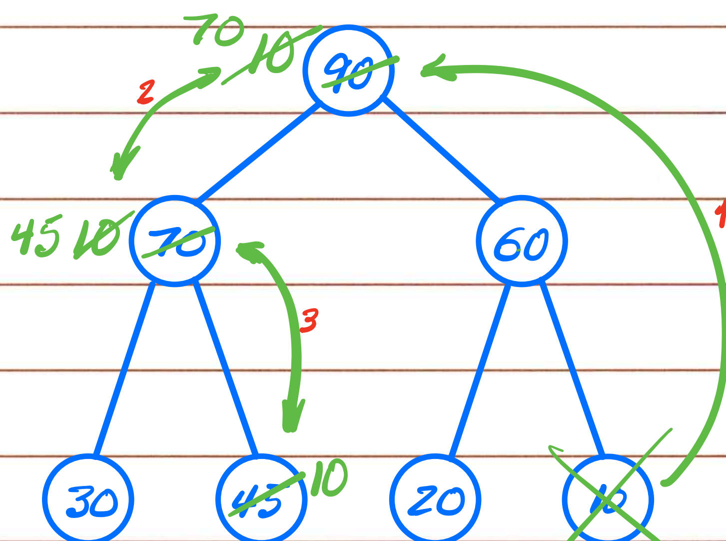
ex. insert 100

Takes O(lgn)



Extract_Max

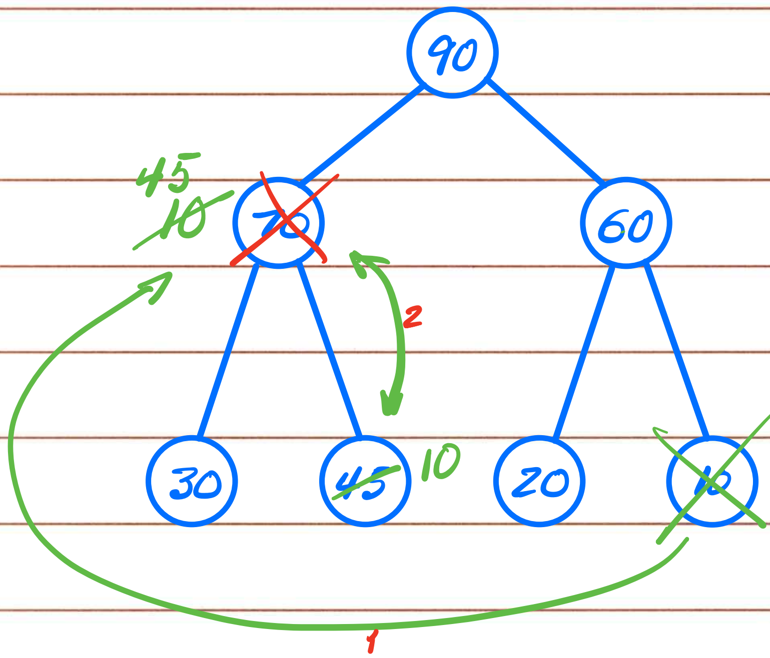
Takes O(lgn)



Delete

ex. Delete 70

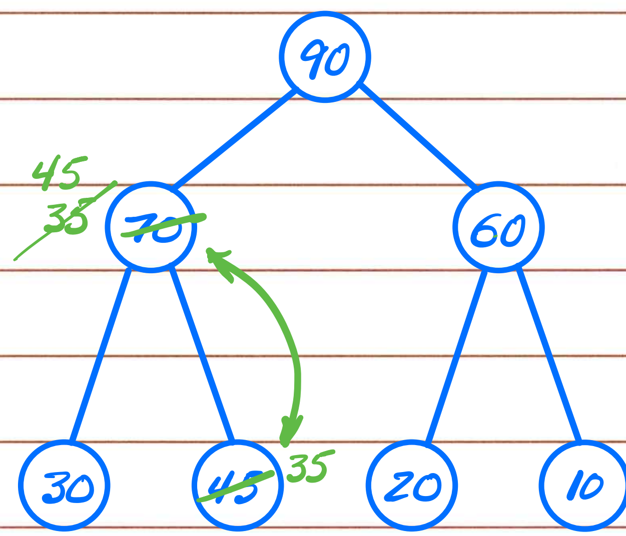
Takes O(lgn)



Decrease-key

ex. Decrease-key 70 to 35

Takes O(lgn)

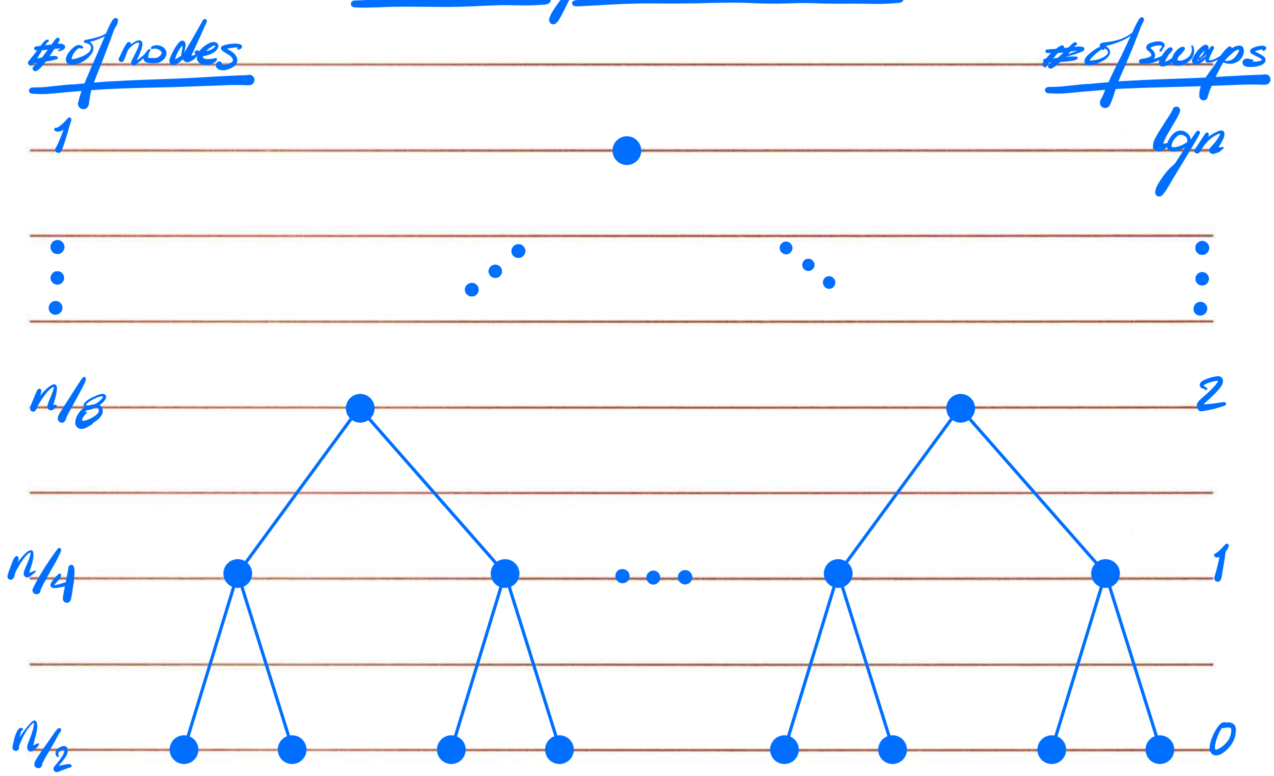


Construction

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

$$T = \frac{n}{4} * 1 + \frac{n}{8} * 2 + \frac{n}{16} * 3 + \dots$$

$$\frac{T}{2} = \frac{n}{8} * 1 + \frac{n}{16} * 2 + \frac{n}{32} * 3 + \dots$$

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

$$\frac{T}{2} = \frac{n}{2}$$

$$T = n$$

Bottom up construction takes $O(n)$

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

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

Finding the top $\underline{k}$ elements in an array.

Input: An unsorted array of length $\underline{n}$.

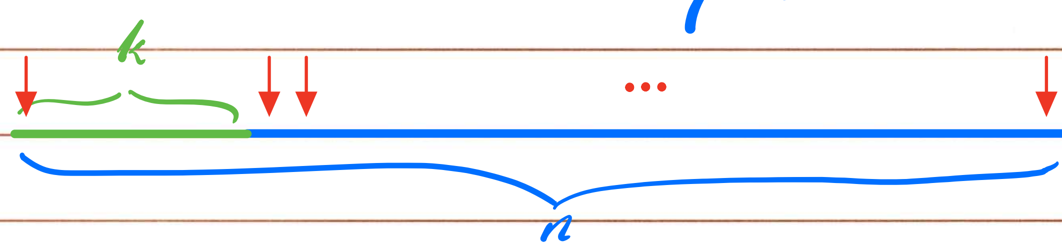
Output: Top $\underline{k}$ values in the array $(k < n)$

Constraints:

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

Solution:

- Construct a Minheap of size k.

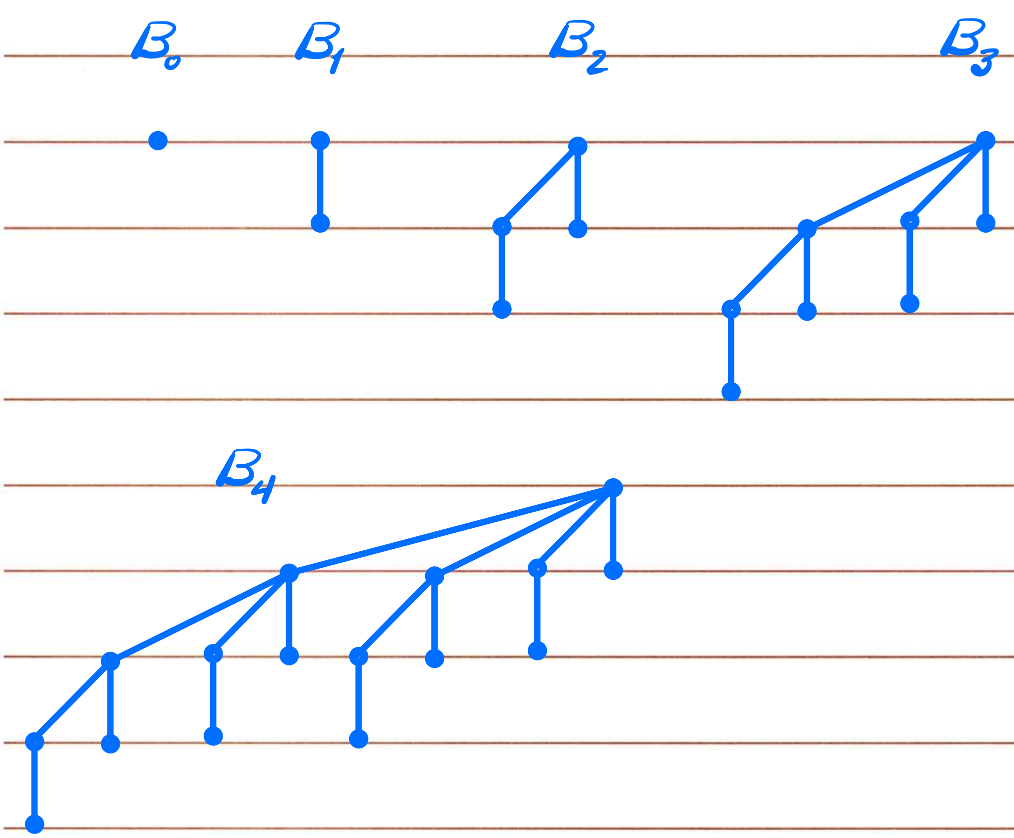


- Move through the rest of the $(n-k)$ elements. If we encounter an element larger than the element at the top of our minheap, swap them. This involves extract-min and insert $(O(\lg k))$ Overall complexity $= O(k) + O((n-k)\lg k) = O((n\lg k))$

Def. A binomial tree B_k is an ordered tree defined recursively

- Binomial tree B_0 consists of one node

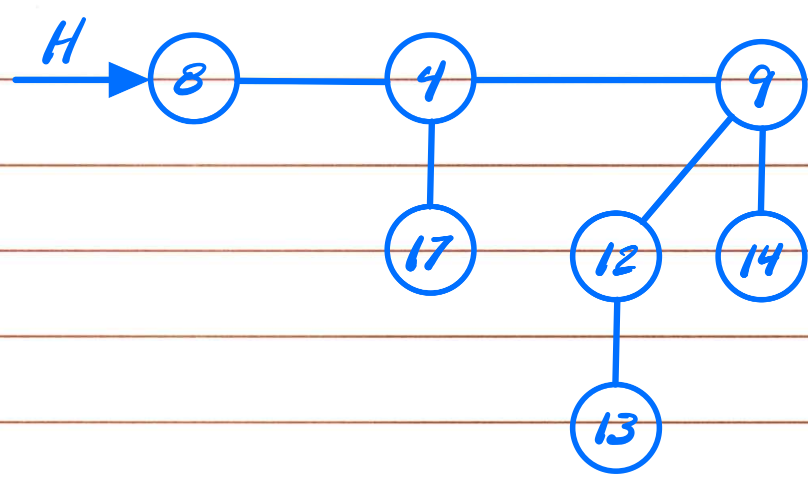
- Binomial tree B_k consists of 2 binomial trees B_{k-1} that are linked together such that root of one is the leftmost child of the root of the other.



Def. A binomial heap H is a set of binomial trees that satisfies the following properties:

1. Each binomial tree in H obeys the min-heap property.

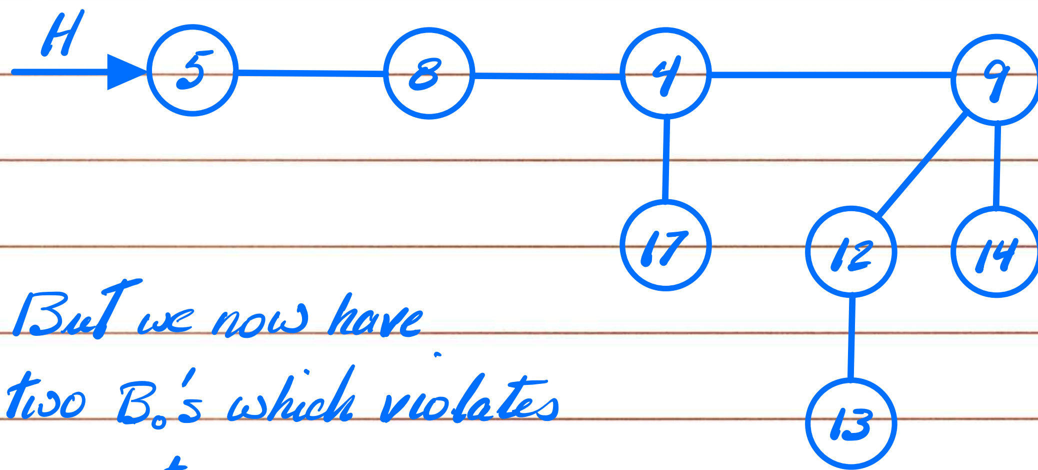
2. For any non-negative integer k, there is at most one binomial tree in H whose root has degree k.



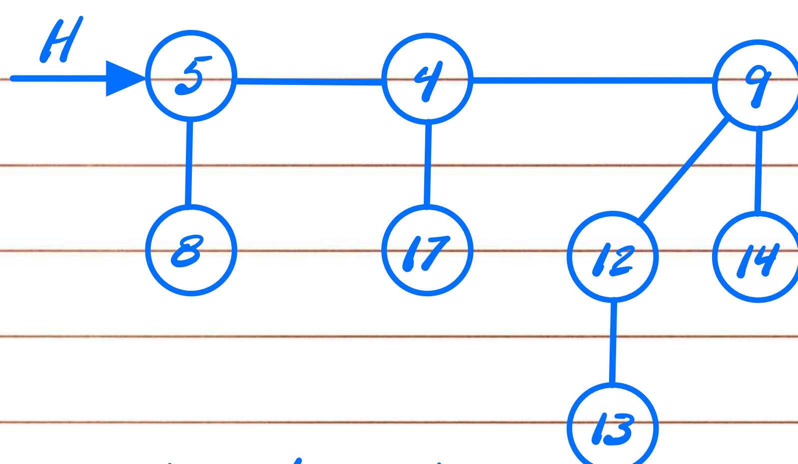
Find-min takes $O(\lg n)$

Insert

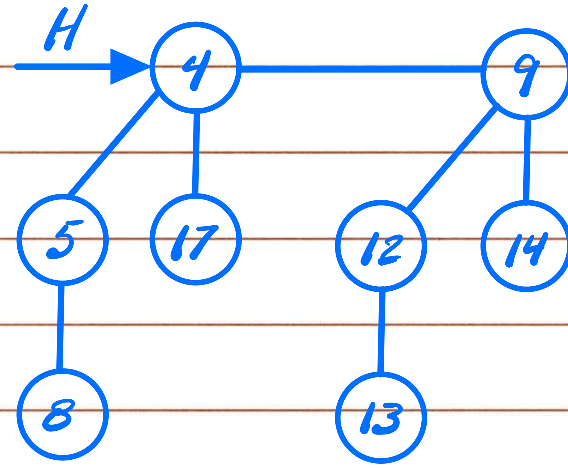
ex. insert 5



But we now have two B_0's which violates property #2.



Now we have two B_1's...



Now we have two B_2's...

- Insert requires at most lgn binomial tree merge operations
- Each tree merge operation takes O(1)
- So the worst case complexity of the insert operation in a binomial heap is O(lgn)