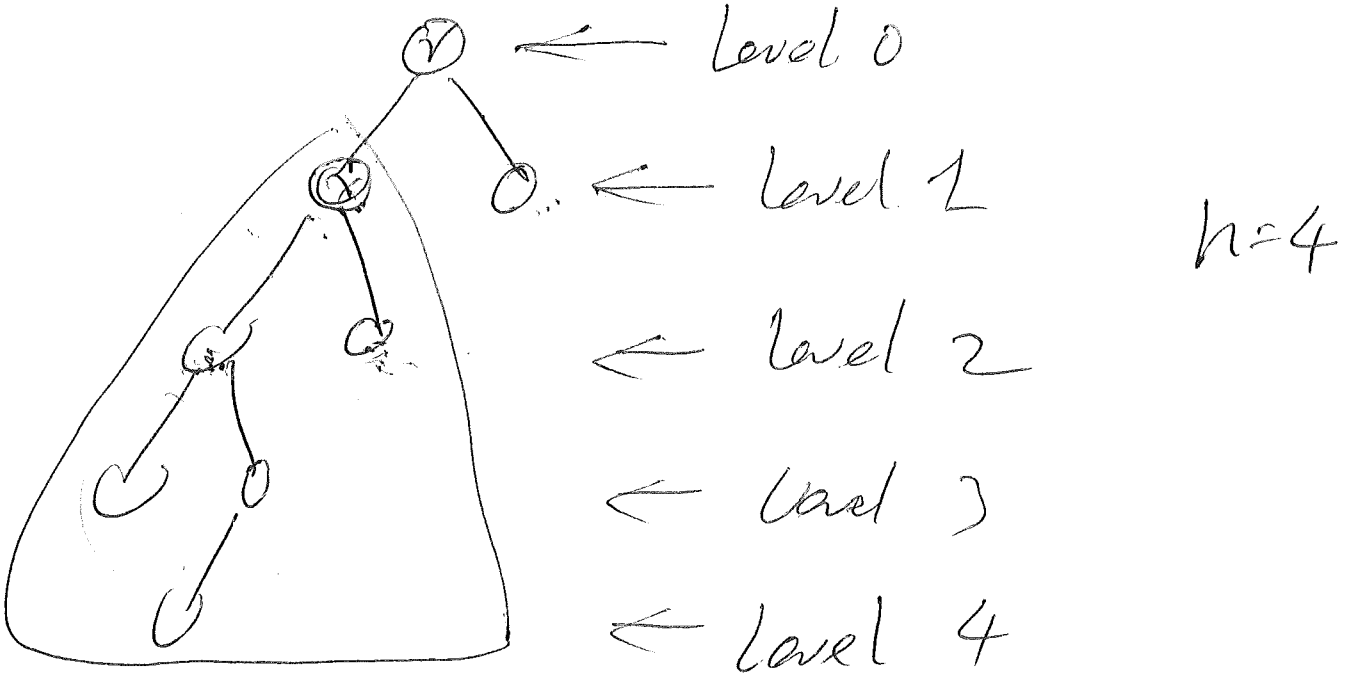
Feb 2, 2016

Sorting using a Heap and a BST

Height of a tree ← maximum level

The level of a node

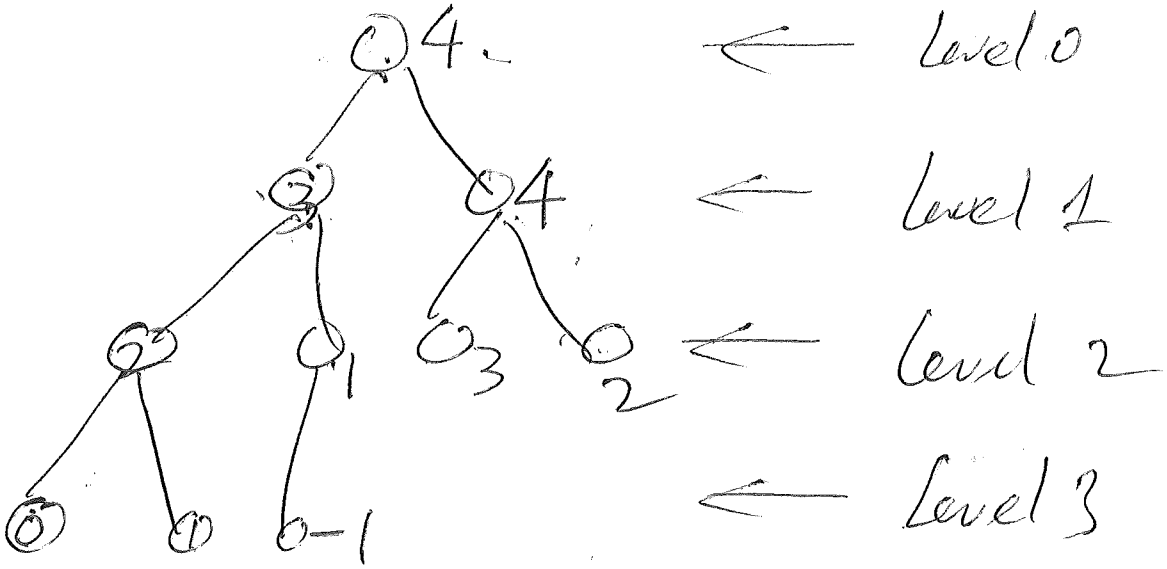


~~A tree~~ The degree of a tree is the maximum number of children of any node in a tree

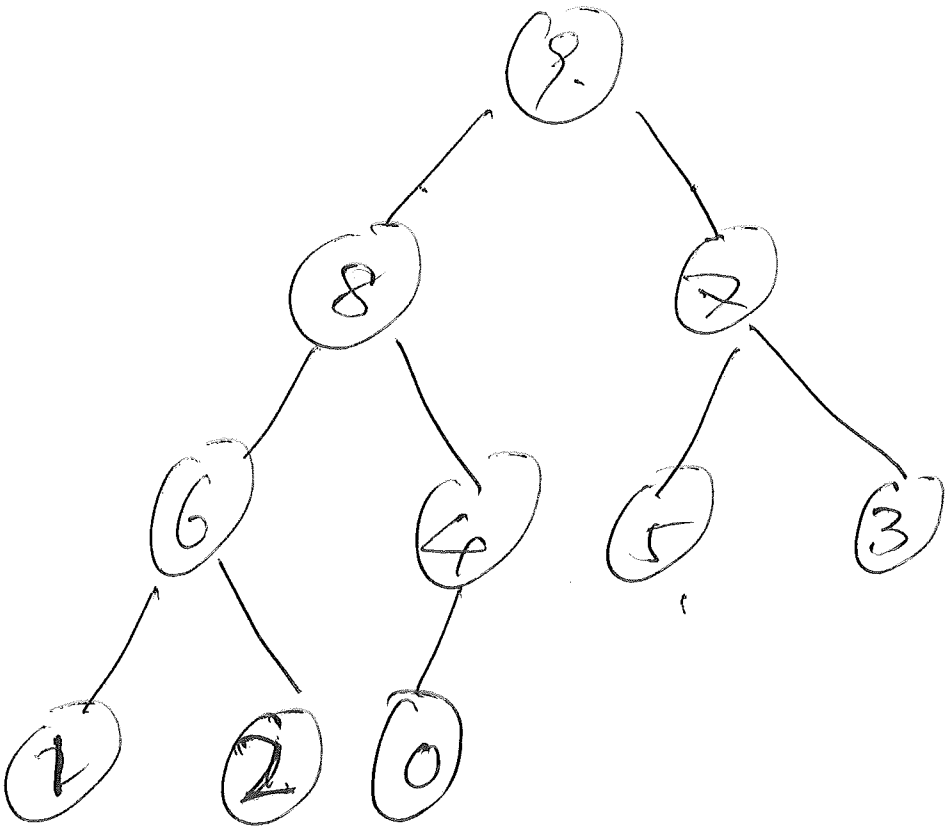
A max heap (also called a priority queue)

is a Binary tree that satisfies the following properties:

(1) The Binary tree is completely filled in all levels except the maximum, which is filled from left to right upto a certain point



(2) the key of any node is $\geq$ the keys of its two children



① The height of the heap is $O(\log n)$

② We can store a heap in an array

Pf of ① Let h be the height of a heap H.

Let n be the number of nodes in H

$$\left(\sum_{j=0}^{h-1} 2^j\right) + 1 \leq n \leq \sum_{j=0}^{h} 2^j$$

$$1 + \sum_{j=0}^{h-1} 2^j = 1 + (2^h - 1)$$

$$S = \sum_{j=0}^{h-1} 2^j = 2^0 + 2^1 + 2^2 + \cdots + 2^{h-1}$$

$$2S = 2^1 + 2^2 + \cdots + 2^{h-1} + 2^h$$

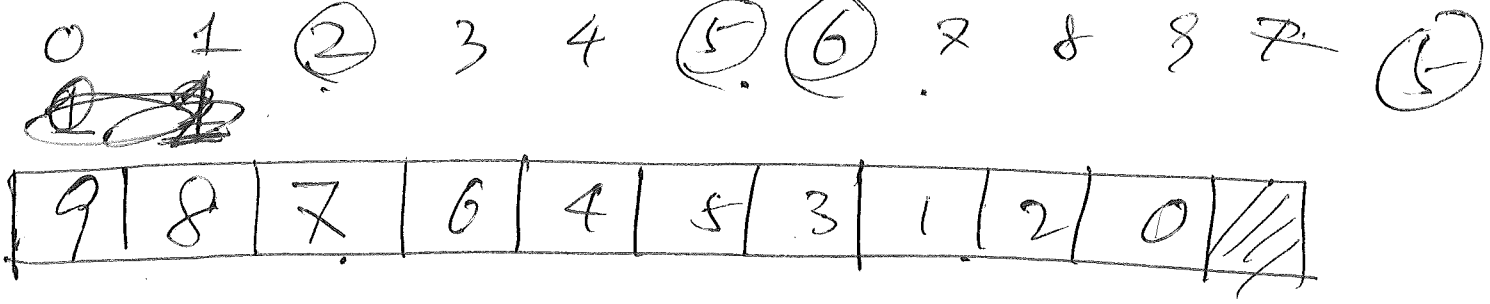
$$-S = -(2^0 + 2^1 + 2^2 + \cdots + 2^{h-1})$$

$$S = 2^h - 2^0 = 2^h - 1$$

Recall $1 + \sum_{j=0}^{h-1} 2^j \leq n$

$$1 + (2^h - 1) = 2^h \leq n$$

$$h \leq \lg n$$



The relation between parent and child are stored implicitly.

$$j \begin{cases} \text{parent} & \lfloor \frac{j-1}{2} \rfloor \\ \text{left child} & 2j+1 \\ \text{right child} & 2j+2 \end{cases}$$

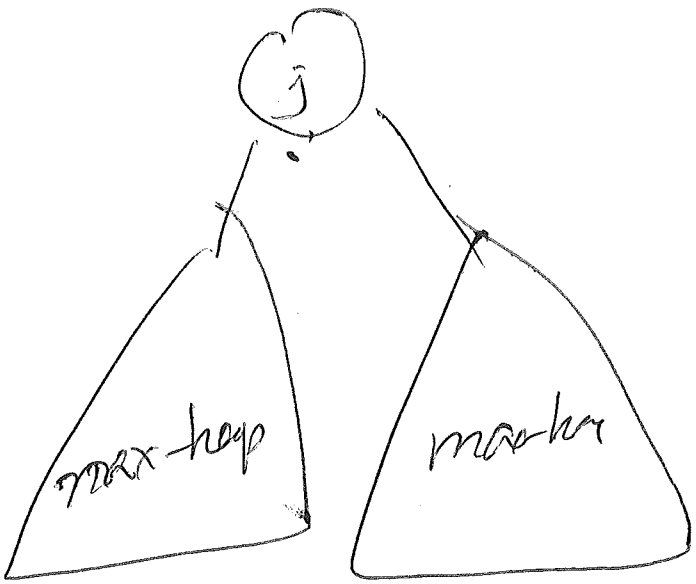
If the array index starts with 1,

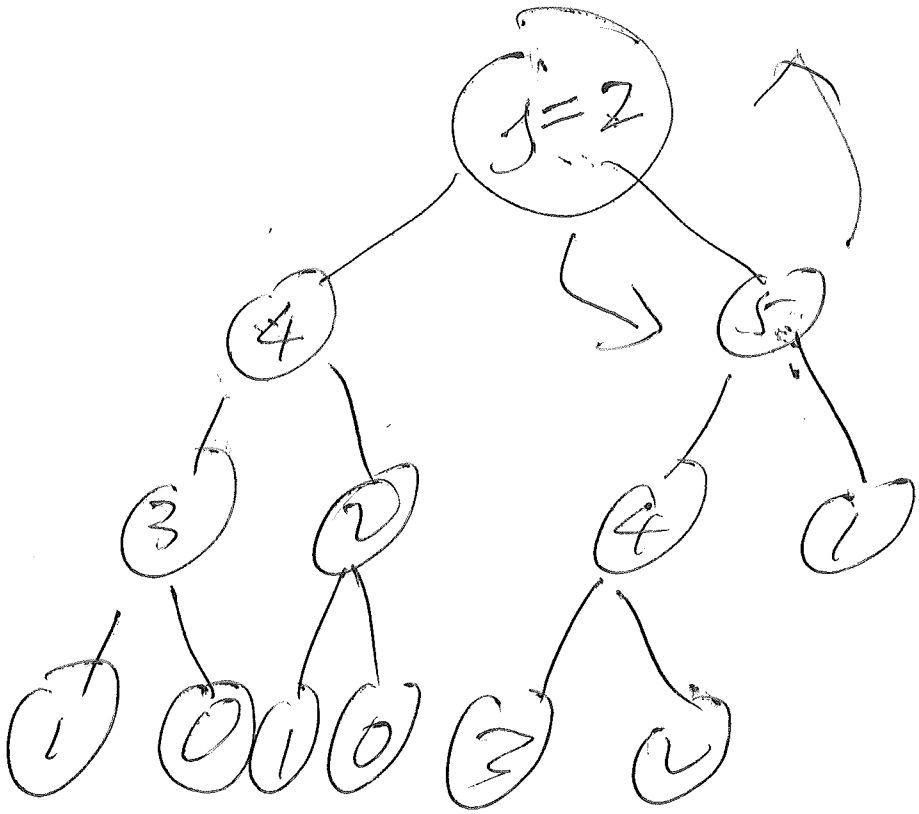
$$j \begin{cases} \text{parent} & \lfloor \frac{j}{2} \rfloor \\ \text{left child} & 2j \\ \text{right child} & 2j+1 \end{cases}$$

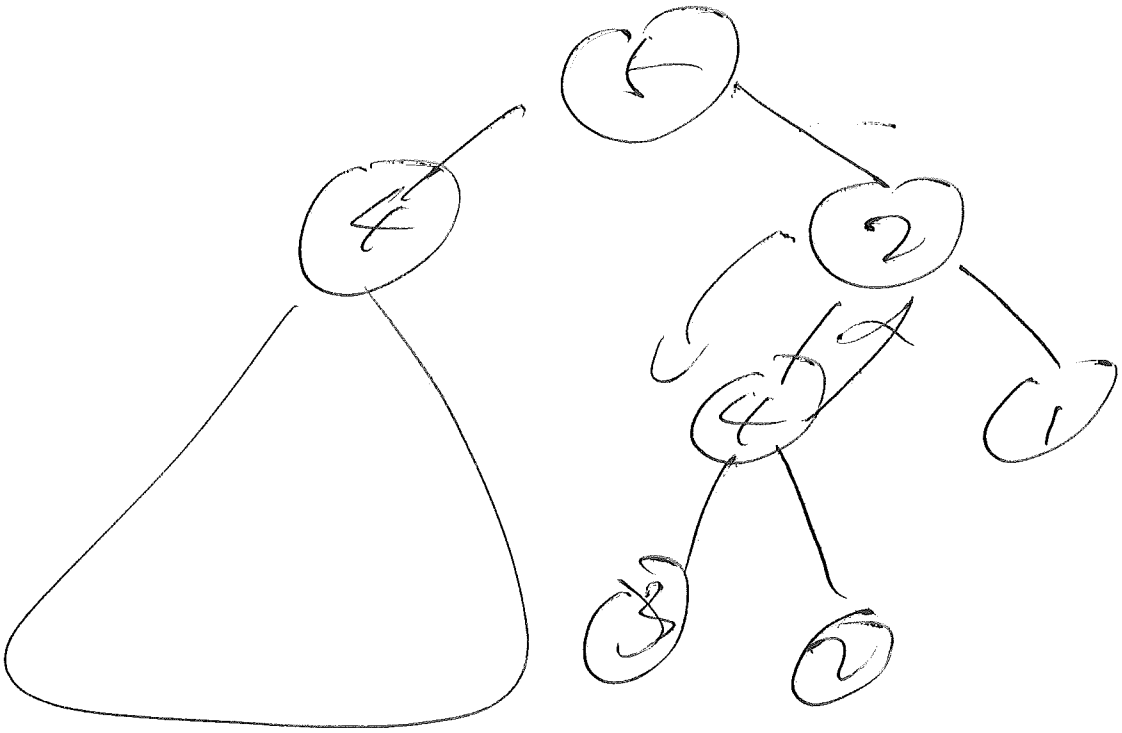
Heap operations.

1. heapify
2. extract-max
3. ~~xxxxx~~ insert.

Assume a max heap H., ma_heapify(j)_ ~~end~~ assumes that the two subtrees of j are max-heaps, and ensures that the subtree rooted at j is a ma-heap.




j=2
4
5
3
2
4
1
1
0
1
0
2
2


4
2
4
1
3
2

max-heapify (j, H)

Assumptions: ① the max heap H is stored in an array H[1..n]

② $1 \leq j \leq n$

③ not is 1

left = 2j

right = 2j+1

~~If H[left] or H[right] is > H[j]~~

~~Swap H[j] with~~

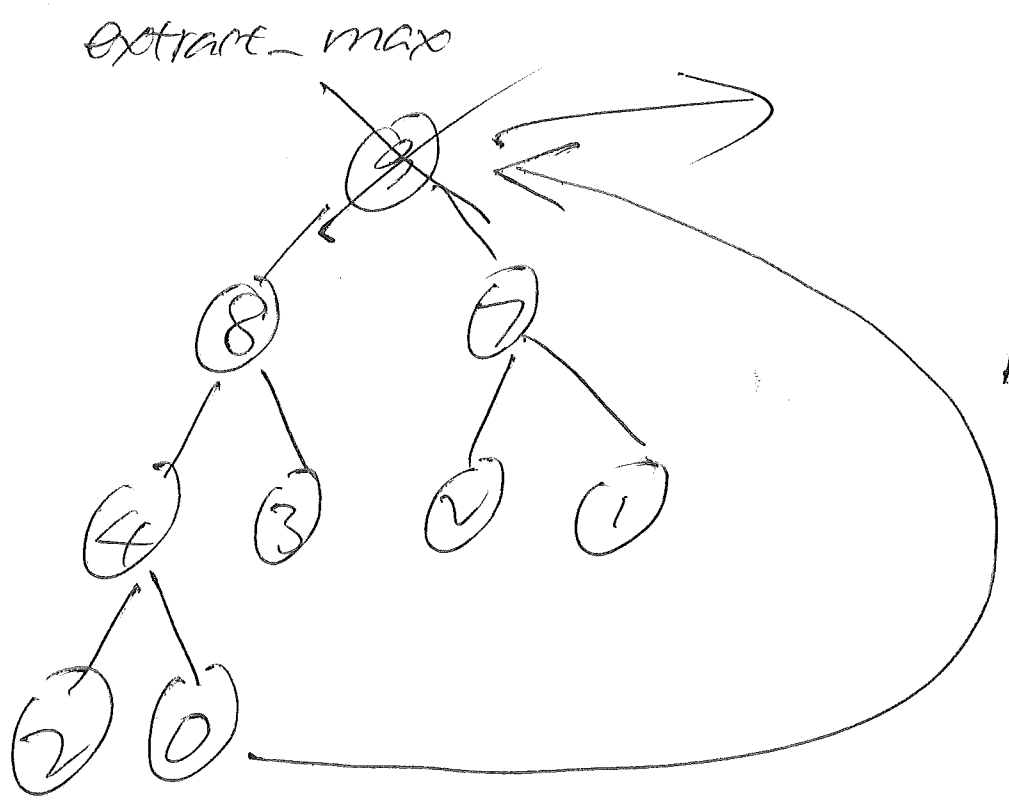
Let k be the index of max{H[left], H[right]}

If H[k] > H[j]

  Swap H[k] and H[j]

Heapify [k, H]

endif



Heap Sort.

input: an array $A[1..n]$

output: array A sorted

Create an empty max heap H.

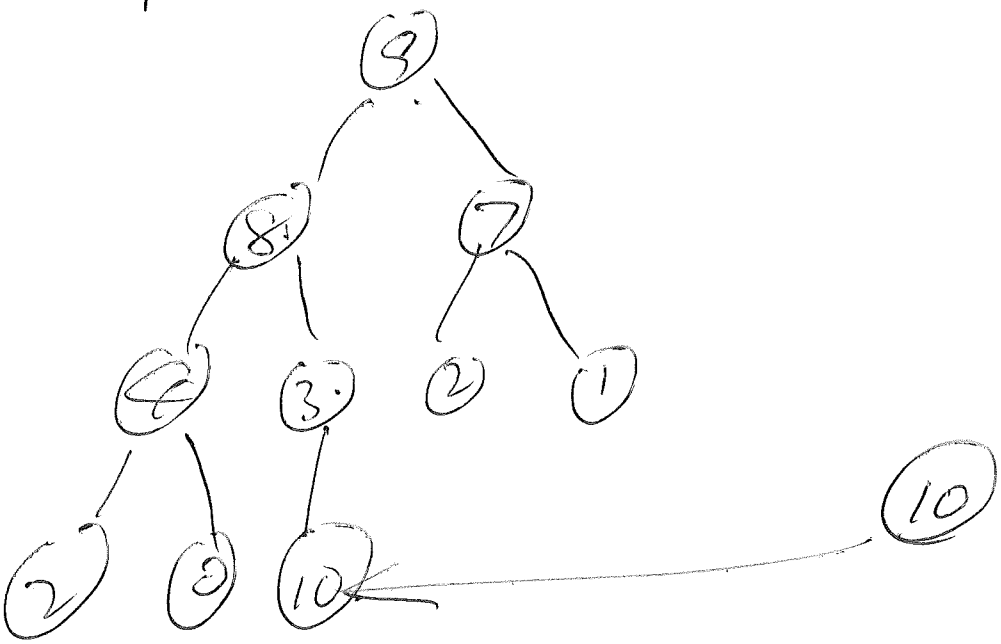
For j = 1 to n

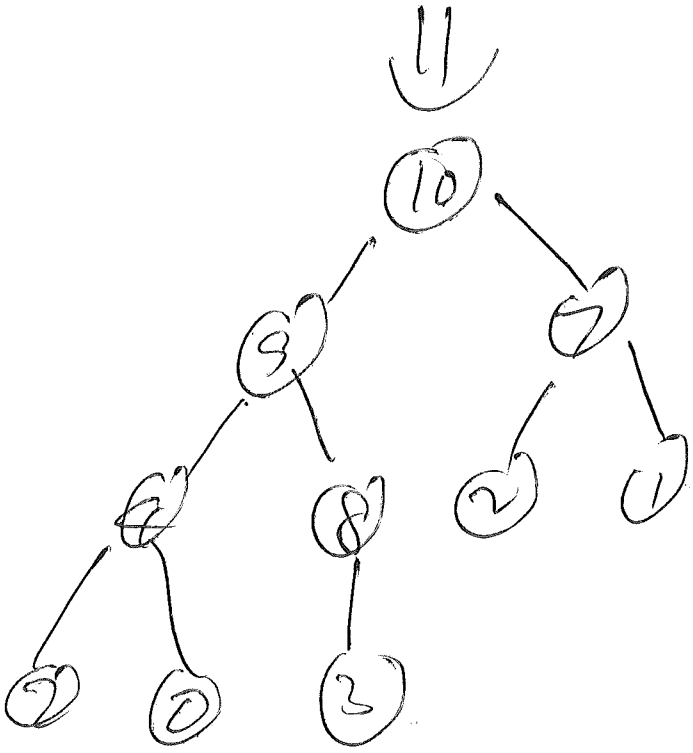
    insert (A[j], H)

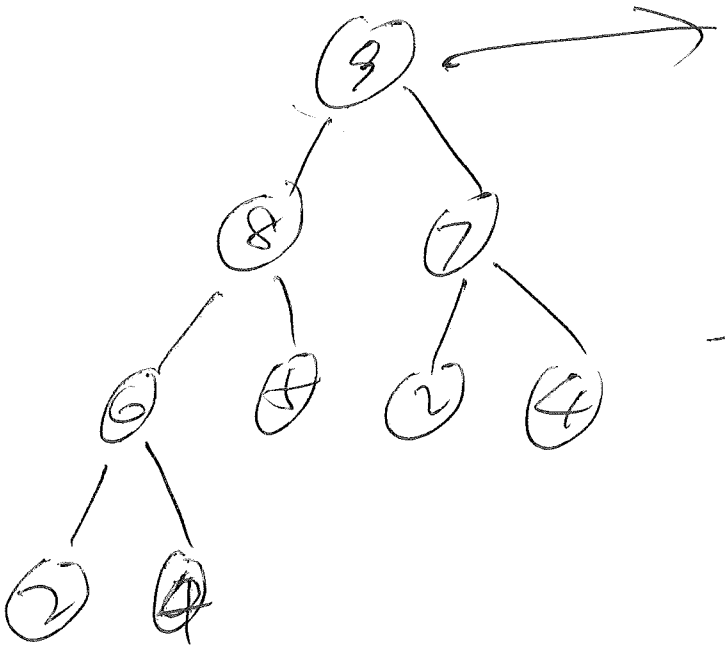
For j = 1 to n

    A[j] = extract_max(H)

heap_insert





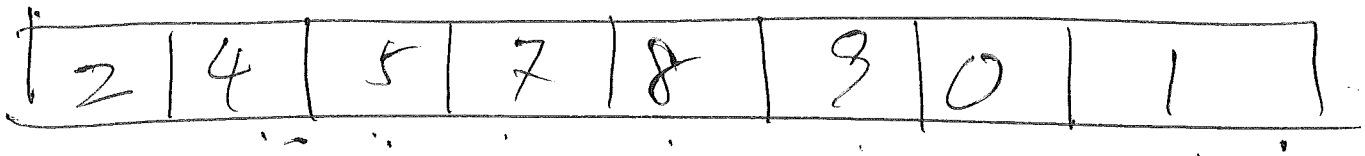


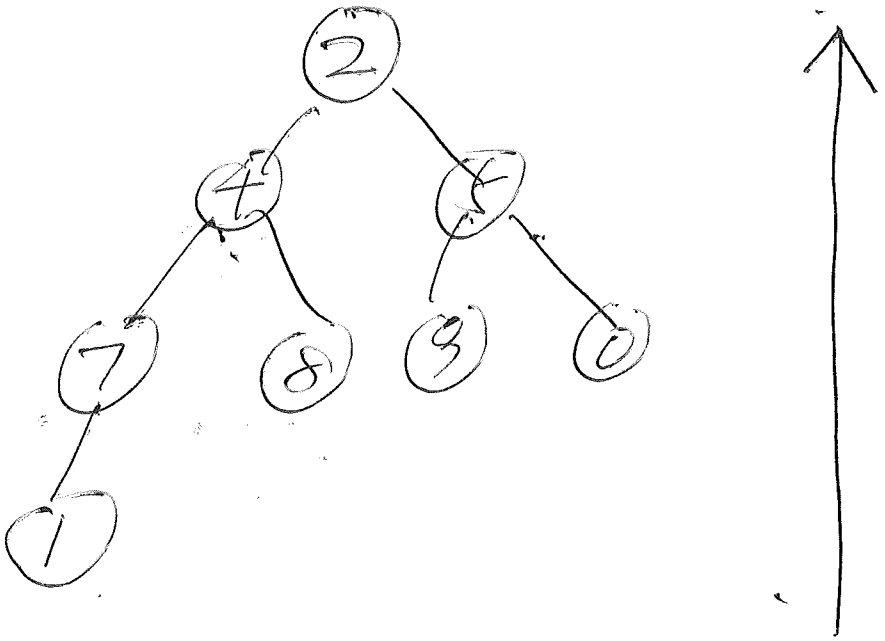
Time: $2n \log n$

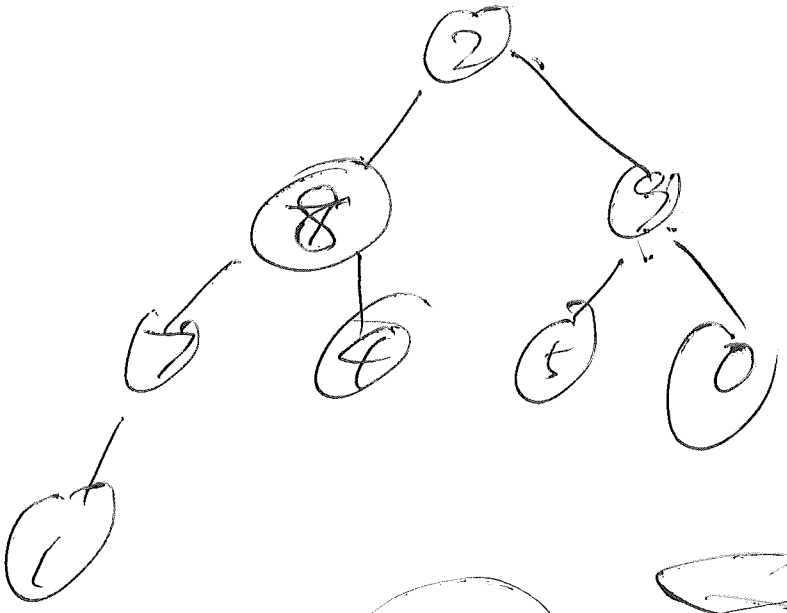
$= \Theta(n \log n)$

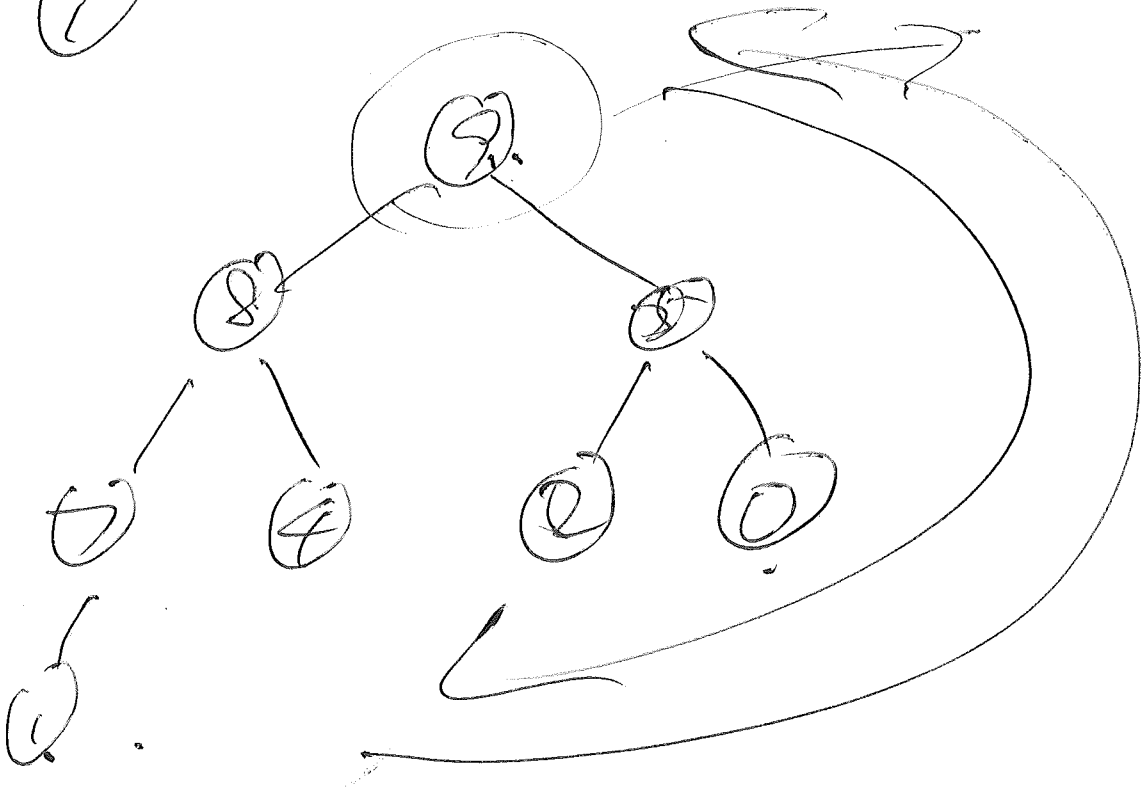
Space $2n$

We need to reduce the storage to n.









- Heap Sort.

input $A[1..n]$

output A sorted.

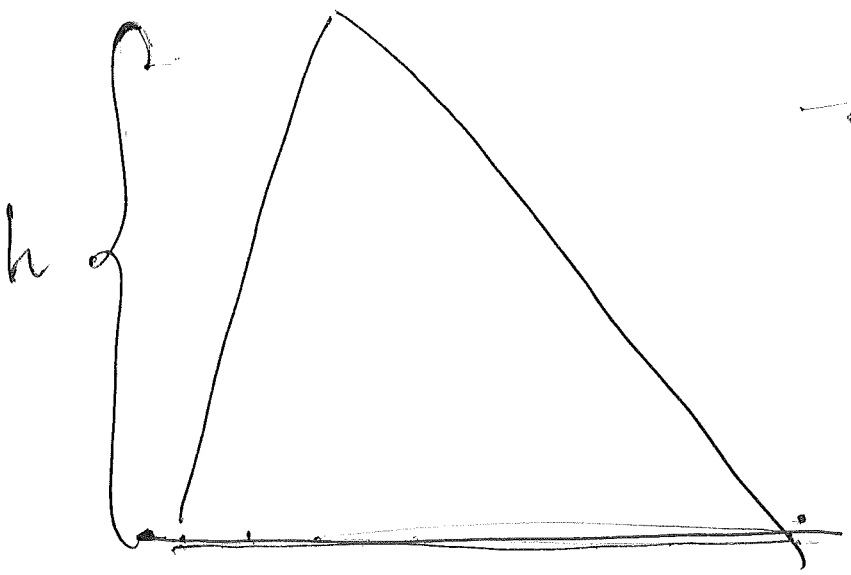
```
# Build a heap in A
    For j = n to 1
        heapify(j, A)
```

}?

```
# Sort.
    For j = n to 1
        Swap A[1] and A[j]
        heapify(1, A[1..j-1])
```

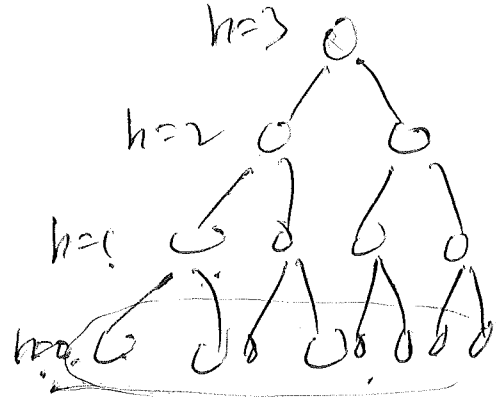
} $O(n \log n)$

Analysis of Building the Heap



$\approx 2^{h+1}$

$n = 15$



$$\sum_{h=0}^{\log n} \frac{n}{2^{h+1}} h$$

$$\frac{n}{2} \sum_{h=0}^{\log n} \frac{h}{2^h}$$

Focus on $\sum_{h=0}^{\log n} \frac{h}{2^h}$

(15)

$$S = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \frac{\log n}{n} \cdots$$

$$\frac{1}{2}S = \frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \cdots$$

$$\frac{1}{2}S = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots$$

$$\frac{1}{2}S = 1$$

$$S \leq 2$$

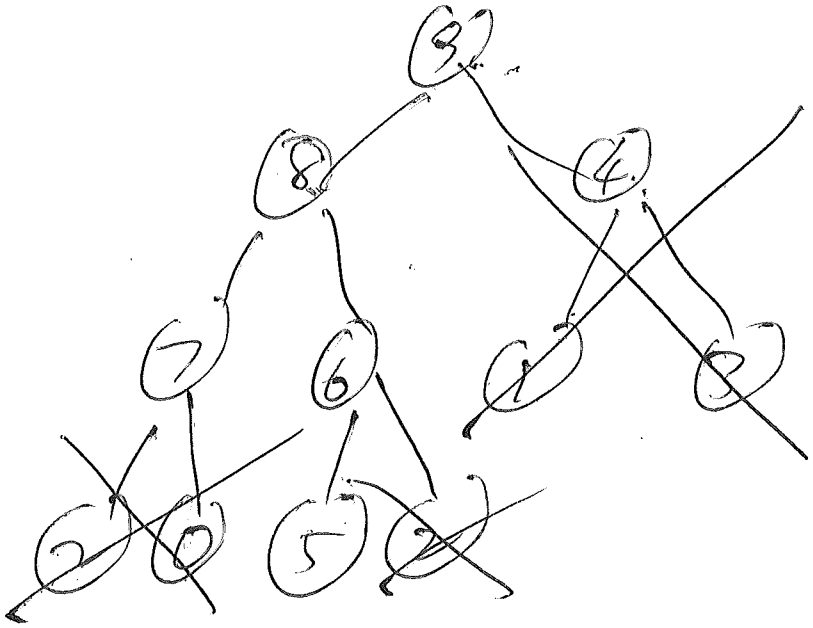
Thus building the heap takes linear time

Observe that the heap is good for implement dynamic queries of the form $[a, +\infty)$

Suppose you are given n keys.

output all keys ~~satisfy~~ satisfying $[a, +\infty)$

the data structure must be dynamic allow insertion, and has to be quick for repeated queries.



$[5, +\infty)$

$O(k)$ time

k is the number of keys in the range