6.006 Recitation #9 10/9/2009

Topics

* Priority Queues
* Heaps
* Heap operations

Priority Queues

A priority queue is an abstract data structure supporting the operations:

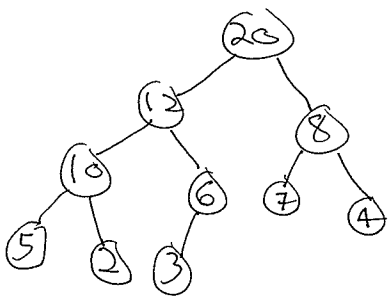
- Insert
- FindMin (or FindMax)
- DeleteMin (or DeleteMax)

Different concrete data structures implementing the priority queue abstraction will have different time complexities, but our goal for today is to support them in $O(\log n)$ time, where n is the number of elements in the data structure.

Heaps

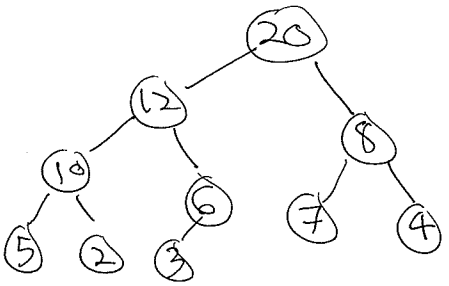
A heap is a particular data structure which we will be able to use as a priority queue.

A heap is a binary tree with the property that every node has exactly two children, except for the nodes at the very bottom of the tree:

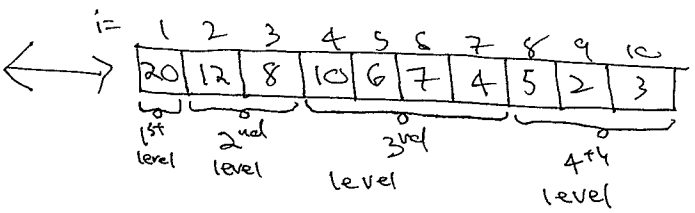


Additionally, every node satisfies the heap property of being greater than or equal to its children (or $\leq$ its children, for a min-heap).

Because heaps are almost complete binary trees with very specific structure, it's easy to store the elements of the tree in an array.



1st level
2nd level
3rd level
4th level



The root is stored in position 1. For ~~and~~ any node i, its children are stored in positions 2i and 2i+1, if these positions are $\leq n$.

Node i: ⟶ Children 2i & 2i+1
⟶ Parent $\lfloor i/2 \rfloor$

Heap operations

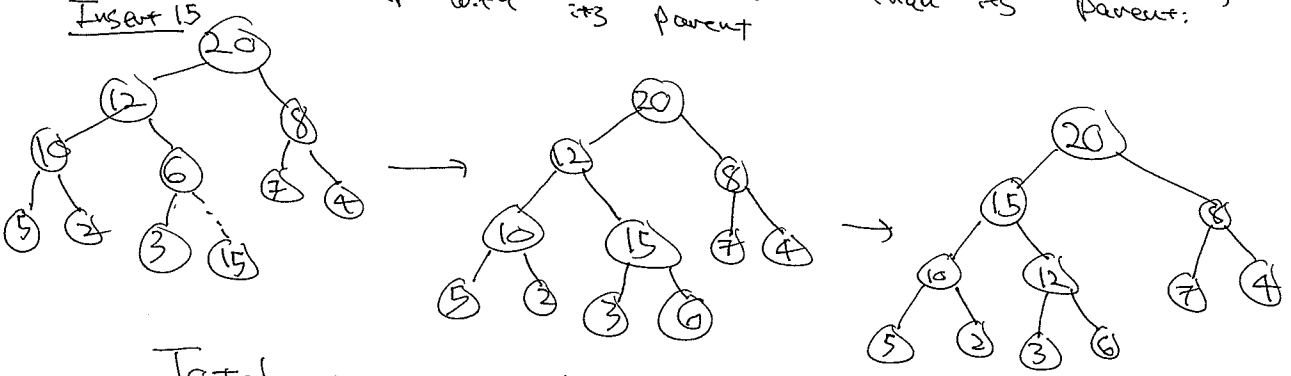
FindMax is easy: the max is just the root of the tree.

Insert

To insert:

* Add new node at end of array (end of heap)
* while the new node is bigger than its parent:
  - Swap with its parent

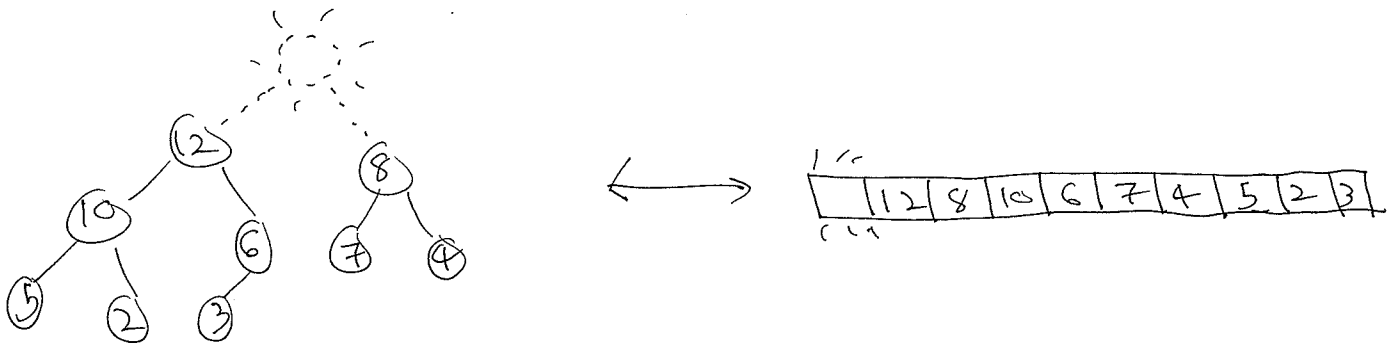
Insert 15



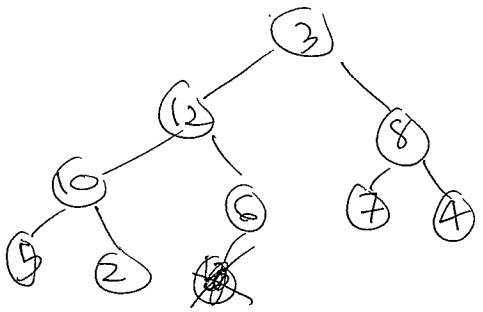
Total time is O(height) = O(log n)

Delete Max

If we delete the max, at the root of the tree, we'll have a hole in our nice heap structure:



We need to fill this hole, so we move the last element of the heap to position 1 (the root):



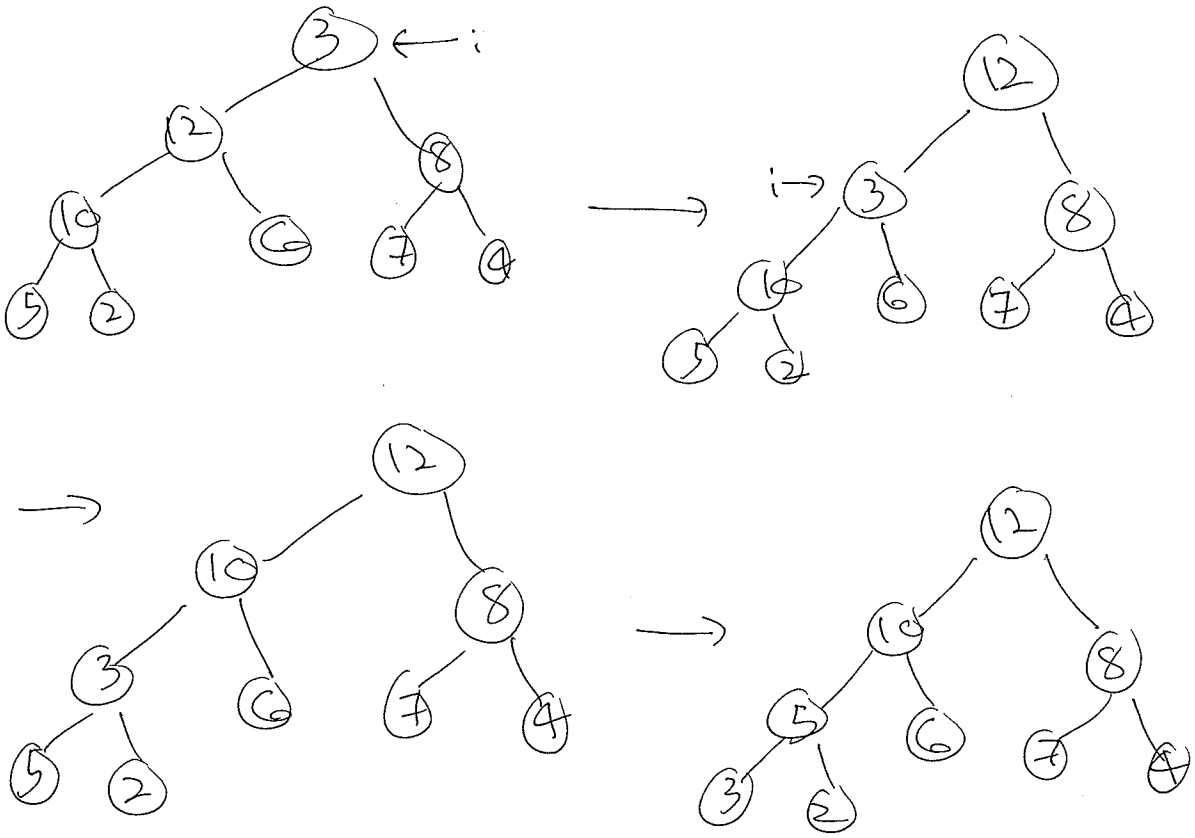
Now the heap property is violated at the root. We need a function to fix this. That function is:

Max Heapify

This function takes a position i and assumes that the subtrees at each of i's children are valid heaps, but that the heap property is violated at i.

~~start~~

* while node i is smaller than one of its children
  - Swap with the larger child
  - Continue with i = the position swapped with.



Given ~~a~~ Max Heapify we can write ~~a~~ DeleteMax as:
* Replace the root with the last element
* Call MaxHeapify at the root.

Build Heap

Given an unsorted ~~list~~ list of n keys, we can make them into a heap in $O(n \log n)$ -time by starting w/ an empty heap and inserting each one. ~~Here~~ Here's another way:

```
for i = n/2 ... 1:
    max_heapify(i)
```

This algorithm starts at the bottom and works up, pushing each new node down the tree until its subtree is a heap.
The loop invariant is that at each iteration the heap property is satisfied at all nodes from $i+1$ through n. This means that when adding a new node, both children are subheaps and we can use max_heapify to fix the heap property.

In this algorithm nodes $\frac{n}{2}+1$ to n don't need to move at all. Nodes $\frac{n}{4}+1$ to $\frac{n}{2}$ only move 1 at most. Nodes $\frac{n}{8}+1$ to $\frac{n}{4}$ move at most 2. So the total runtime is:

$$0 \cdot \frac{n}{2} + 1 \cdot \frac{n}{4} + 2 \cdot \frac{n}{8} + 3 \cdot \frac{n}{16} + \cdots + (\log n - 1) \cdot 1$$

$$= \frac{n}{2}\left(\frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \cdots\right)$$
$$+ \left(\frac{n}{8} + \frac{n}{16} + \frac{n}{32} + \cdots\right)$$
$$+ \left(\frac{n}{16} + \frac{n}{32} + \frac{n}{64} + \cdots\right)$$
$$+ \cdots$$

$$= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \cdots$$

$$= n$$

So building a heap in this way is $O(n)$.

Despite the fact that we can build a heap in $O(n)$ time and we can use a heap for sorting, we can't use this to sort in $O(n)$ time because to sort we still have to remove all n elements from the heap, one at a time, which takes $O(n \log n)$ time.