Heaps and some uses…

Another Tree: The Heap

When discussing heaps, we can discuss either MinHeaps or MaxHeaps.

All heaps are complete binary trees.

In a MaxHeap, for any node everything in its subtrees is smaller than what is in the node itself.

Questions:

- What is the height of a heap?
- How long does it take to find a value?
- How long to insert/delete a value?

Heap::Insert(val)

Due to the structure of the heap, you always know *where* the next new node will be placed.

The challenge is to maintain [in O(log*n*) time] the heap's overall property of subtrees having contents less than (or for a MinHeap greater than) the local root node.

Let's look at how a MaxHeap would be built using the values:

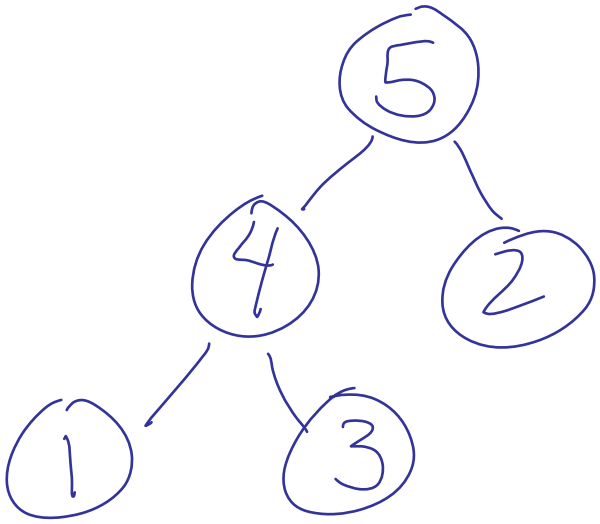
1, 2, 3, 4, 5, 6

Heap::Delete()

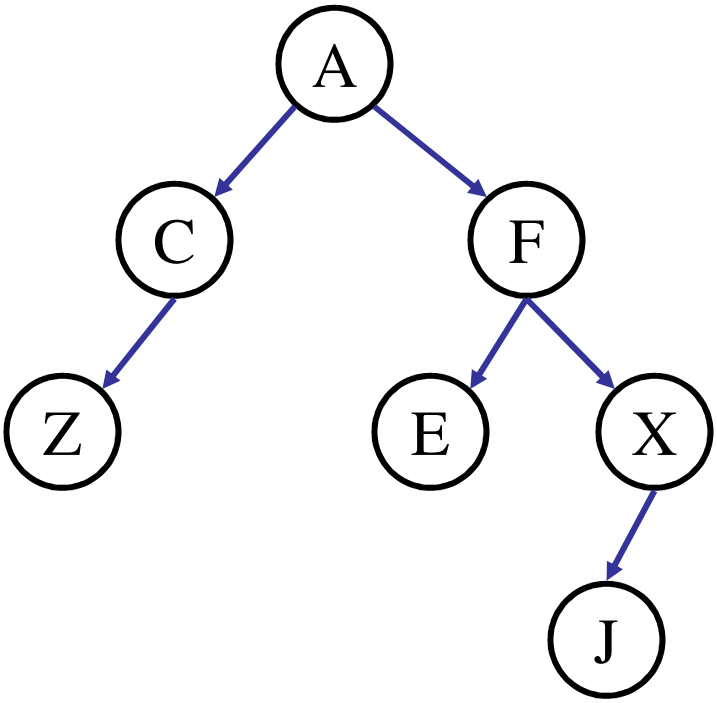
Due to the structure of the heap, you **also** always know *where* the next new node is going to be dropped.

Once again, the challenge is to maintain [in O(log*n*) time] the heap's overall data property.

Let's look at what it would be like to delete the root of a MaxHeap:



Array Storage Example



A	C	F	Z	-1	E	X	-1	-1	-1	-1	-1	-1	J	-1	....

Note: When used when complete trees such as heaps, there are no gaps within the tree - only empty spaces on the extreme right.

Heap as Priority Queue

One use of a heap is that of Priority Queue. It has good run-times for insertion and deletion, and provides *instant* access to the next item in the queue.

HeapSort

Since the heap provides an easy way to extract values in their relative order, it is trivial to create an O($n$log$n$) sorting algorithm using a heap.

- Build the heap.
- Harvest the heap.

Some questions to consider:

- Can we sort an array of values without using a linear amount of extra memory?
- Is the worst-case runtime actually better than $n$log$n$?

Heapify

Let's assume a MaxHeap. If you have a complete tree where the root is the only thing that is violating the heap property, you can just "bubble" that value down to a valid position.

- If the value in the root is smaller than either child, swap it with the larger of the children and then repeat this until it is no longer smaller than either of its children.

What's the runtime of this?

Could this be used to ***build*** a heap an array in-place?

Run-time of Build-Heap()

The run-time of each of the mutliple calls to heapify is based on the current height of the heap.

Assuming that the lowest level of the heap is full the heap could contain a total of $n=2^{h+1}-1$ values.

How much work would be done if we took an unordered array of this size and turned it into a heap this way?

Harvesting the heap…

As a thought question, what would the run-time be to harvest the heap in order?