

The Binary Heap

A binary heap looks similar to a binary search tree, but has a different property/invariant that each node in the tree satisfies. In a binary heap, all the values stored in the subtree rooted at a node are greater than or equal to the value stored at the node.

We can use a binary heap for a couple of things: maintaining a priority queue, and performing a heapsort.

To maintain a priority queue, we need efficient findMin and deleteMin operations.

When we maintain a binary heap, we will do so as a balanced binary tree. Interestingly enough, we can even store a binary heap in an array instead of an actual binary tree. Basically, the children of node i are nodes $2i$ and $2i+1$. Consider the picture below:

Heap Operations: Insert, deleteMin, Heap Construction

To do an insertion, consider an existing heap. Since we want to keep the heap balanced, we must insert into the following spot in the heap. (This would be the next array location if you are storing the heap in an array.)

However, the problem is in all likelihood, if the insertion is done in this location, the heap property will not be maintained. Thus, you must do the following "bubble up" procedure:

If the parent of the newly inserted node is greater than the inserted value, swap the two of them. This is a single "bubble up" step. Now, continue this process until the inserted node 's parent stores a number lower than it.

Since the height of the tree is $O(\lg n)$, this is an $O(\lg n)$ operation.

Consider the picture below:

The first part of a deleteMin operation is quite easy. All you have to do is return the value stored in the root. BUT, after you find this value, you must also, fix up the heap. This means deleting the "last" node of the heap and finding a new spot for it in the tree.

Temporarily place this "last" node in the vacated root of the tree. But, this is almost definitely not going to be the correct location for this value. Chances are one of it's two children will be storing a value lower than it. If so, swap this value with the minimum of the two child values. This is a single "bubble down" step. Continue these steps until this "last" node has children both with larger values than it.

Here's how I like to look at this operation: Imagine that the CEO of the company as retired. So some hotshot upstart thinks they can take their place. But, quickly, the two most senior officers realize what has happened and try to rectify the situation. The higher ranking of the two takes the CEO position, relegating the hotshot upstart to their position. But soon again, someone realizes what has happened, once again demoting the hotshot upstart. This continues until the upstart has found a position in the company that he rightfully deserves.

Bottom-Up Heap Construction

We will show how to construct a heap out of unsorted elements.

The basic idea is as follows:

- 1) Keep all leaf nodes in the binary tree where they are. (Leave "ghost" nodes for the rest of the tree.)**
- 2) Now, introduce/"unghost" the parents of each of these leaf nodes and add them into the tree. Each of these parents will be the root of some subtree. Run the "bubble down" procedure on each of these nodes.**
- 3) Now, add the parents of the nodes just added and "bubble down" each of these nodes. Continue this process until the root has been added and "bubbled down."**

Heap Sort

Now that we have determined how to execute several operations on a heap, we can use these to sort values using a heap. Here is the idea:

- 1) Insert all items into a heap
- 2) Extract the minimum item n times in a row, storing the values sequentially in an array.

Since each inserting and extraction take $O(\lg n)$ time, this sort works in $O(n \lg n)$ time. Let's trace through an example: