Heaps

Heap: brief review. Here are the running times of priority queues operations for an implementation with a simple linked list, vs. an implementation with heaps.

abstract data structure	*data structure*	
Priority queue	Linked list	Min-heap
finding the minimum	$O(n)$	$O(1)$
extracting the minimum	$O(n)$	$O(\log n)$
insertion	$O(1)$	$O(\log n)$

What are min-heaps? Viewed abstractly, min-heaps are nearly-complete binary trees (binary trees such that every level is filled completely, except possibly for the bottom level, which is filled in left-to-right order up to a point) which satisfy the *min-heap property:* every node has a key which is greater than or equal to the key of its parent.

- To find the minimum, just look at the root.
- To remove the minimum, replace it with the last key x (the one in the rightmost node of the bottom level; that node is removed), and once x is at the root, repeatedly swap x with the smaller of its two children until the heap property is satisfied.
- To insert a new value y, just add a node to the tree, at the first available spot (the leftmost non-existent node at the bottom level), put y in it, and repeatedly swap y with its parent until the heap property is satisfied.

The stated time complexities follow from the fact that a nearly complete binary tree has height at most $\log_2 n$.

Implementation details: Concretely, nearly-complete binary trees are implemented with an array, where the children of the node associated with $a[i]$ are located at $a[2i]$ and $a[2i+1]$ for the left and right child respectively. The array implementation makes heap operations very fast in practice.

Max-heaps are analogous to min-heaps, and is suitable when the target operations are finding and removing the maximum instead of the minimum. The main structural difference is that the min-heap property is replaced by the max-heap property: every node has a key which is less than or equal to the key of its parent.