CIS 575. Introduction to Algorithm Analysis
Material for March 1, 2024

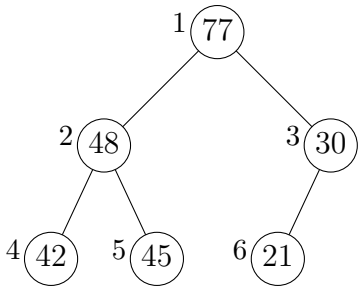
Representing Binary Heaps

The topic of this note is covered in *Cormen*'s Section 6.1.

1 Binary Heaps

From now one, we shall restrict our attention to **binary** heaps, that is trees where each node has at most two children, and that are **balanced** except that some rightmost leaves may be missing. Such a tree can be represented as an array $A[1..n]$ where $A[1]$ is the root, $A[2]$ ($A[3]$) the left (right) child of the root, $A[4..7]$ the grandchildren of the root, etc.

For example, we may have the heap (where for each node, its key is depicted inside and its array index depicted outside)



Observe that for a node with index x, its left child (if exists) has index $2x$ and its right child (if exists) has index $2x + 1$; if the node is not the root, its parent has index $\lfloor \frac{x}{2} \rfloor$. It is easy to see that a binary tree of height h fits up to $2^{h+1} - 1$ nodes; hence we infer that:

> a binary heap with n nodes has height h given by $h = \lfloor \lg(n) \rfloor$.

2 Priority Queues By Binary Heaps

In the previous note, we showed how to represent a priority queue using a heap, but the description was very loosely specified leaving a lot of freedom to the implementer. But when we use a **binary** heap to represent a priority queue, the implementation is fully deterministic:

- for Insert, we insert the new node at the first free array position (in the above example that would be at index 7) and then percolate up that leaf;
- for DeleteMax, we move the rightmost bottom leaf (in the above example that would be the leaf at index 6) to the root and then sift it down.

Since INSERT and DELETEMAX have worst case running time proportional to the height of the tree, we conclude:

> a priority queue with n records can be implemented such that all operations have worst case running time in $O(\lg(n))$.