**Trees**

**Dom** – Document Object Model is an example of a tree structure.

* <head> and <body> are children of the <html> tag
* <div>, <p>, <button> all have parent – child relationship

**Abstract Syntax Tree** – this is how code is being run

Diagram

Description automatically generated

**Binary Tree**

lookup O(log N)

insert O(log N)

delete O(log N)

O(log N) – think of it as a dictionary. When we want to lookup a word like ‘restful’, we would open the dictionary to where we think the letter ‘r’ is at then divide and conquer by going forward or backwards in the dictionary until we get to the word ‘restful’.

Divide and Conquer = O(log n)

Each parent (node) can have either 0, 1, or 2 children (node)

Each child (node) can only have 1 parent (node)

This is a perfect binary tree. This means all leaf nodes are full and there are no nodes that only have 1 child. The number of nodes doubles as we go down the tree. 1 -> 2, 3 -> 4, 5, 6, 7

The number of nodes at the lowest level is equal to the sum of all the nodes of all levels above plus 1.

1, 2, 3 + 1 = 4, 5, 6, 7

4 = 4

Perfect Binary Tree Levels:

Level 0: 2^0 = 1

Level 1: 2^1 = 2

Level 2: 2^2 = 4

Level 3: 2^3 = 8

Diagram

Description automatically generated

This is a full binary tree. This means that a node either has 0 or 2 children. Never only 1 child.

Diagram

Description automatically generated

**Binary Search Tree**

* Better than hash tables because there is a relationship between nodes. It is a parent/child relationship.
* Preserves relationships – just like the relationship that our folders have on our computers. Folders allows us to group things together.
* All child nodes to the right must be greater than the current node
* All child nodes to the left must be less than the current node
* Each node can have only 2 children

lookup O(log N)

insert O(log N)

delete O(log N)

**Advantages**

* Better than O(n) – assuming that the tree is balanced
* Ordered
* Flexible Size – can place node anywhere in memory

**Disadvanages**

* No O(1) operations

**Unbalanced Binary Search Tree**

O(n) – this may occur when all the nodes are to either left or right causing the tree to look exactly like a linked list.

**Binary Heap**

Used in any algorithms where order is important.

Commonly used in priority queues.

Used for comparing data such as getting all values that are greater than 20.

Used in sorting algorithms.

Left to right insertion of elements making the tree auto balanced.

Unlike binary search tree, we DO NOT need to rebalance a binary heap.

Arrays can be used in a binary heap.

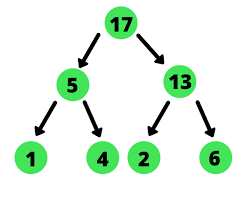
Parent node is always greater than the children.

lookup O(n)

insert O(log N)

delete O(log N)

Max Heap



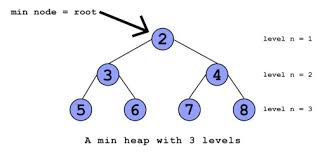
Max heap is when the root node is higher than the child nodes.

17 > 5, 13

5 > 1, 4

13 > 2, 6

Min Heap



Min heap is when the root node is lower than the child nodes.

Advantages of Binary Heaps:

* Better than O(n)
* Priority
* Flexible Size
* Fast Insert

Disadvantages of Binary Heaps:

* Slow lookup

**Priority Queue**

Each element has a priority.

Elements with a higher priority gets precedence over elements with a lower priority.

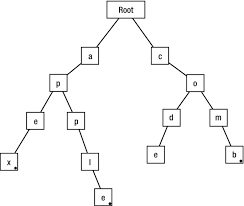
Think of it like a long line at a night club. We let some people in and the line gets shorter. Now there are people beginning to line up in the VIP line. The people in the VIP line will get a higher priority than the people in the regular line.

Another way to look at it is an emergency room. A patient comes into the emergency room with a stomach pain and is told to have a seat and wait. An hour later, a baby comes in that has a high fever. Although the baby came in later, he/she will be admitted first because it has a higher priority.

**Trie**

Also known as a prefix tree.

Special tree used in searching usually for texts.



* Usually has an empty root which is the start.
* Kind of like auto-completion like when we search for something on Google, and it auto fills in what it thinks we are looking for.
* Speed and space are very fast
* Big O is O(length of the word)