Data Structure Related important points to remember:

String

Arrays

Hash table

Linked List

Stack

Queue

Tree (Binary Search Tree, Segment Tree)

Graph

Heap

Trie

Searching (Binary Search, DFS, BFS)

Sorting

Bit Manipulation

Patterns:

Two pointers

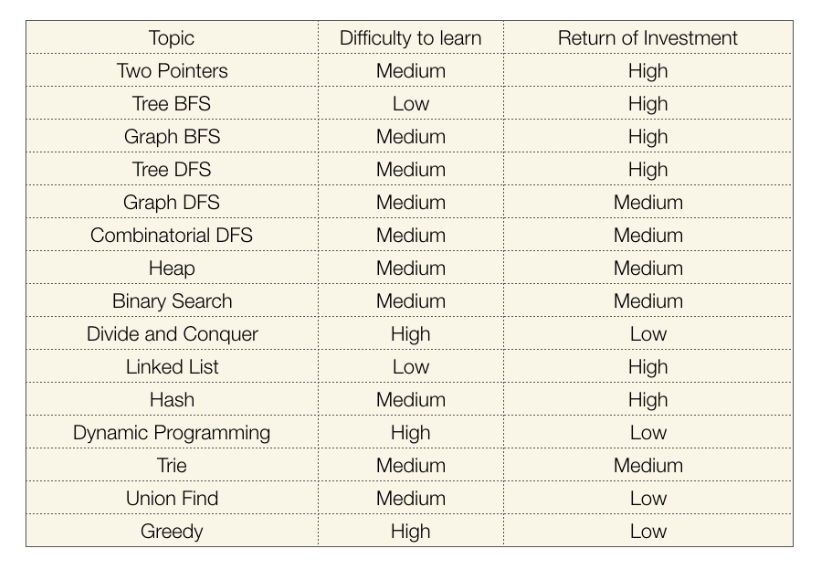
Sliding Window

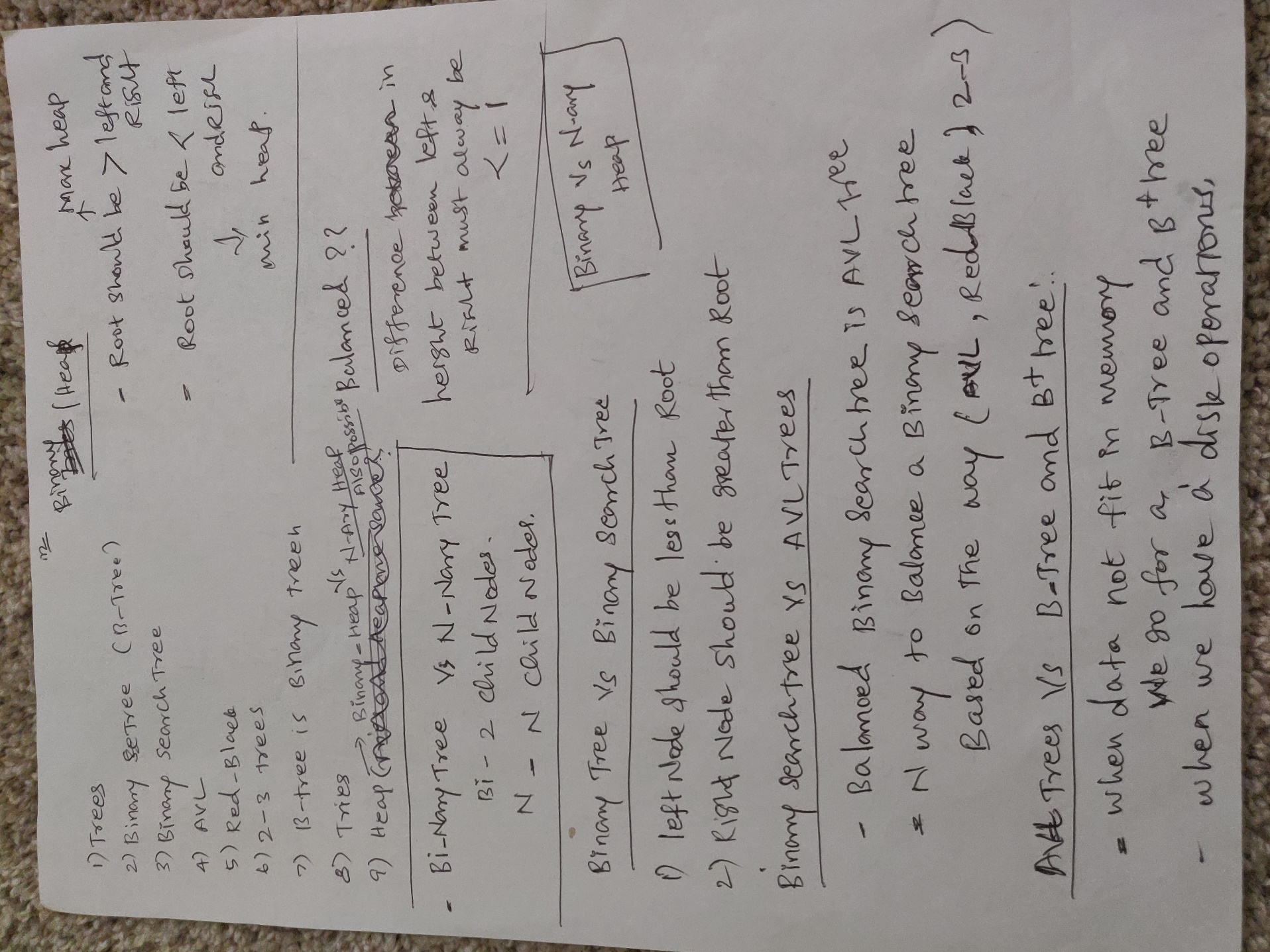
Divide and Conquer

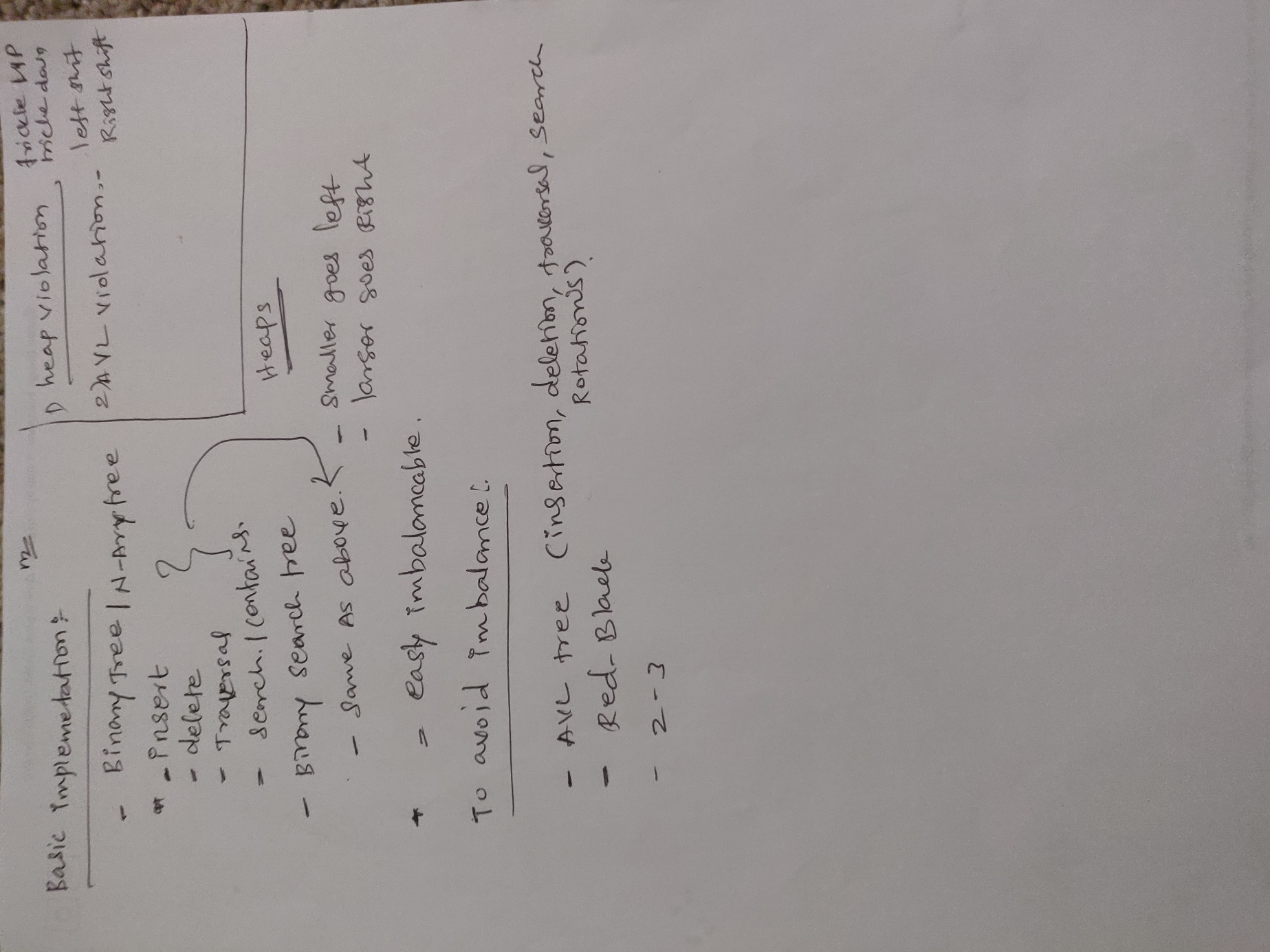
Dynamic Programming

Backtracking

Greedy







Red Black Tree is a type of 2-3 Tree.

**AVL vs Red-Black tree:**

1. AVL trees provide **faster lookups** than Red Black Trees because they are more strictly balanced.
2. Red Black Trees provide **faster insertion and removal** operations than AVL trees as fewer rotations are done due to relatively relaxed balancing.
3. AVL trees store **balance factors or heights** with each node, thus requires storage for an integer per node whereas Red Black Tree requires only 1 bit of information per node.
4. Red Black Trees are used in most of the language libraries like [**map**](http://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/), [**multimap**](https://www.geeksforgeeks.org/multimap-associative-containers-the-c-standard-template-library-stl/), [**multiset**](http://www.geeksforgeeks.org/multiset-in-cpp-stl/) in C++ whereas AVL trees are used in **databases** where faster retrievals are required.

Reference:

<https://www.geeksforgeeks.org/red-black-tree-vs-avl-tree/>

Tree:

Removing a Node:

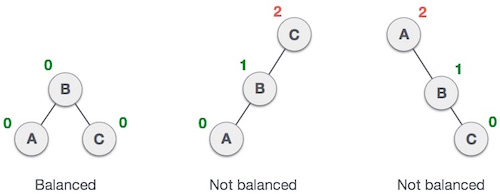
1. If we are removing leaf node set it’s parent to NULL.
2. If we are removing a parent with one child, set it’s parent to point to it’s child.
3. If we are removing a parent node with two child, SWAP them with In-order Successor or Predecessor and remove the leaf node.

In-order successor 🡪 go to left and all the way to right.

In-order predecessor 🡪 Go left and all the way to left.

Rotation:

Balanced vs Not Balanced:



***BalanceFactor*** = height(left-sutree) − height(right-sutree)

***BalanceFactor*** <= 1 and >= -1

* Left rotation
* Right rotation
* Left-Right rotation
* Right-Left rotation

Left Rotation

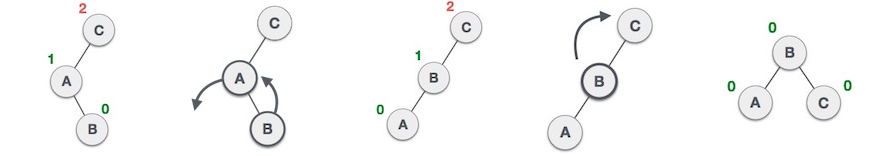


Right Rotation



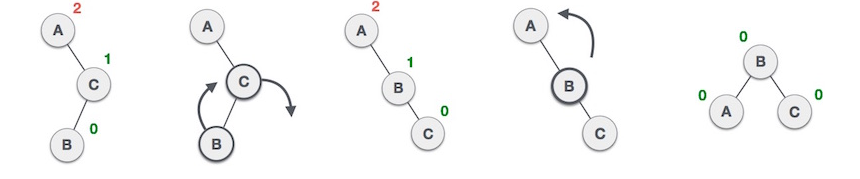
Left – Right Rotation

* Left rotate the parent
* Right rotate the grand parent

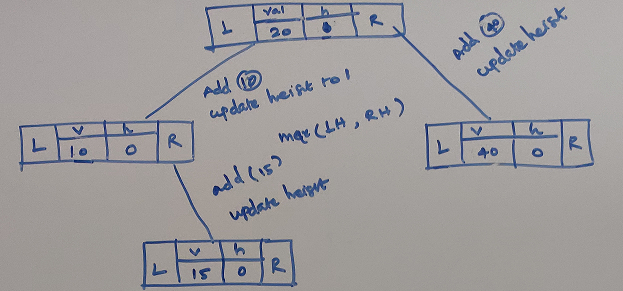


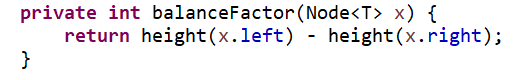
Right – Left Rotation

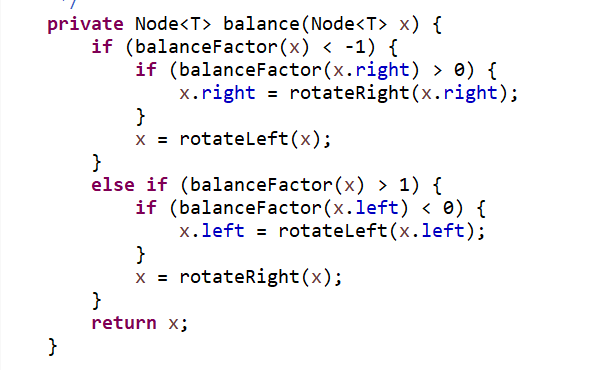
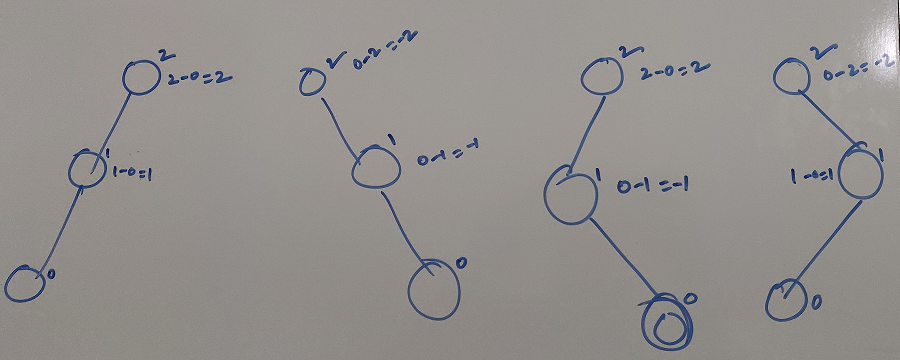
* Right rotate the parent
* Left rotate the grand parent



Checking Binary Tree Balance:







Graph:

