**Steps for DP Framework**

* Check whether greedy works or not (if greedy exists).
* If greedy fails / if greedy doesn't exist, be exhaustive (try all possibilities - choose and no choose).
* Identify repeated sub problems.
* If there are repeated sub problems optimize it with DP.
* Decide on the DP Matrix / DP Array based on the number of decision making parameters.
* Decide on the range of values in the DP Matrix / DP Array by looking at the possible values in the exhaustive tree.

Binary Tree, BST, Red Black Tree

Binary Tree: Types:

1. Full Binary Tree: Every node has 0 or 2 children
2. Complete Binary Tree: All levels are completely filled except possibly the last level, which should be filled from left to right
3. Perfect Binary Tree: All internal nodes have two children, and all leaf nodes are at the same level.
4. Balanced Binary tree: The height of the left and right subtrees of any node differs by at most one.

Binary Search Tree: A binary tree where the nodes are organized to allow efficient searching, insertion, and deletion operations.

* Left subtree node <= Parent node
* Right subtree node > parent node
* Left and right subtrees are also BSTs
* TC: O(h) for search, insert and delete – h is the height of the tree.

Red – Black tree: A type of self-balancing BST that ensures the tree remains balanced after insertions, deletions.

Prop: Maintains O(logn ) height.

* Node coloring: Each node is either Red or Black
* Root Property: Root is always Black
* Red Rule: Red nodes can’t have red children
* Black height rule: Every path from a node to its descendent null pointers has the same number of black nodes.
* Balance property: Ensures the tree remains approximately balanced, preventing extreme skewness.

Operations: for Insertions or Deletions: may require recoloring and/or rotations to maintain properties

Advantages: Search, insert and Delete complexity ( O(logN)).  
Widely used in Databases and file systems. ( Java Treemap and TreeSet ).

HASHMAP and HASHTABLE: Both use key-value pairs but differ in imp aspects

1. Synchronization and thread-safe:

* HashMap: Not synchronized, meaning not thread-safe. Multiple threads can access and modify a HashMap concurrently, leading to inconsistent behavior if proper synchronization is not applied externally.
* Suitable for single thread applications or when used with external synchronization like Collections.synchronizedMap() or Concurrent HashMap.
* So Faster compared to HashTable
* HashTable: Synchronized, thread-safe. Methods like put() and get() are synchronized, ensuring safe access by multiple threads

1. Null Keys and values:

* HashMap: Allows one **null** key and multiple **null** values.
* HashTable: Doesn’t allow null key or null values

LinkedHashMap: LinkedHashMap is a Hash table and Linked list implementation of the Map Interface, with predictable iteration order.

It maintains the insertion order in LinkedHashMap.

TreeMap: It’s a red-black tree based NavigableMap implementation. It is sorted according to the natural ordering of the keys.