```
public class StriverA2ZDsa {
    //**************************Linked
List****************************************************************** */
    class Solution {
        // Function to insert a node at the beginning of the linked list.
        Node insertAtBeginning(Node head, int x) {
            // code here
            Node temp = new Node(x);
            temp.next = head;
            return temp;
        }

        // Function to insert a node at the end of the linked list.
        Node insertAtEnd(Node head, int x) {
            // code here
            if (head == null) {
                return new Node(x);
            }
            Node curr = head;
            while (curr != null && curr.next != null) {
                curr = curr.next;
            }
            Node temp = new Node(x);
            curr.next = temp;
            temp.next = null;
            return head;

        }
    }

    class Solution {
        // Function to count nodes of a linked list.
        public static int getCount(Node head) {

            // Code here
            int len = 0;
            Node curr = head;
            while (curr != null) {
                len++;
                curr = curr.next;
            }
            return len;
        }
    }

    class GfG {
        Node deleteNode(Node head, int x) {
            // Your code here
            int len = 0;
```

```
            Node prev = head;
            Node curr = head;
            if (x == 1) {
                return head.next;
            }
            while (curr.next != null && len != (x - 1)) {
                prev = curr;
                len++;
                curr = curr.next;
            }
            prev.next = curr.next;
            return head;
        }
    }
    //
**************************Recursion*****************************************************************
******
    // */
    // Generate all binary strings
    class Solution {
        public static ArrayList<String> generateString(int k) {
            // Write your code here.
            ArrayList<String> ls = new ArrayList<>();
            String str = "";
            binary(0, -1, k, ls, str);
            return ls;
        }

        public static void binary(int ind, int prev, int n, ArrayList<String> ls, String temp) {
            if (ind == n) {
                ls.add(temp);
                return;
            }
            binary(ind + 1, 0, n, ls, temp + "0");
            if (prev == -1 || prev == 0) {
                binary(ind + 1, 1, n, ls, temp + "1");
            }
        }
    }

    // Generate Paranthesis
    class Solution {

        public List<String> AllParenthesis(int n) {
            // Write your code here
            List<String> ls = new ArrayList<>();
            f(0, 0, n, ls, "");
            return ls;
```

```
        }

        public void f(int open, int close, int n, List<String> ls, String path) {
            if (close == n) {
                ls.add(path);
                return;
            }
            if (open > close) {
                f(open, close + 1, n, ls, path + ")");
            }
            if (open < n) {
                f(open + 1, close, n, ls, path + "(");
            }
        }
    }

    // Print all subsequences/Power Set
    // Learn All Patterns of Subsequences …
    // Count all subsequences with sum K
    // Check if there exists a subsequence…
    // Combination Sum
    class Solution {
        public void findCombinations(int ind, int arr[], int target, List<List<Integer>> ans,
List<Integer> ds) {
            if (ind == arr.length) {
                if (target == 0) {
                    ans.add(new ArrayList<>(ds));
                }
                return;
            }
            if (arr[ind] <= target) {
                ds.add(arr[ind]);
                findCombinations(ind, arr, target - arr[ind], ans, ds);
                ds.remove(ds.size() - 1);
            }
            findCombinations(ind + 1, arr, target, ans, ds);
        }

        public List<List<Integer>> combinationSum(int[] candidates, int target) {
            List<List<Integer>> ans = new ArrayList<>();
            findCombinations(0, candidates, target, ans, new ArrayList<>());
            return ans;
        }
    }

    // Combination Sum-II
    class Solution {
        public List<List<Integer>> combinationSum2(int[] candidates, int target) {
            List<List<Integer>> ans = new ArrayList<>();
```

```
        Arrays.sort(candidates);
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }

    static void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans,
List<Integer> ds) {
        if (target == 0) {
            ans.add(new ArrayList<>(ds));
            return;
        }
        for (int i = ind; i < arr.length; i++) {
            if (i > ind && arr[i] == arr[i - 1]) {
                continue;
            }
            if (arr[i] > target)
                break;
            ds.add(arr[i]);
            findCombinations(i + 1, arr, target - arr[i], ans, ds);
            ds.remove(ds.size() - 1);
        }
    }
}

// Subset Sum-I
class Solution {
    ArrayList<Integer> subsetSums(ArrayList<Integer> arr, int N) {
        // code here
        ArrayList<Integer> res = new ArrayList<Integer>();
        printSubsetSums(arr, res, 0, N, 0);
        return res;
    }

    void printSubsetSums(ArrayList<Integer> arr, ArrayList<Integer> res, int ind, int N,
int sum) {
        if (ind >= N) {
            res.add(sum);
            return;
        }
        printSubsetSums(arr, res, ind + 1, N, sum + arr.get(ind));
        printSubsetSums(arr, res, ind + 1, N, sum);
    }
}

// Subset Sum-II
class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ansList = new ArrayList<>();
```

```
        findSubsets(0, nums, new ArrayList<>(), ansList);
        return ansList;
    }

    public void findSubsets(int ind, int[] nums, List<Integer> ds, List<List<Integer>>
ansList) {
        ansList.add(new ArrayList<>(ds));
        for (int i = ind; i < nums.length; i++) {
            if (i != ind && nums[i] == nums[i - 1]) {
                continue;
            }
            ds.add(nums[i]);
            findSubsets(i + 1, nums, ds, ansList);
            ds.remove(ds.size() - 1);
        }
    }
}

// Combination Sum - III
// Letter Combinations of a Phone numb…
class Solution {
    // Function to find list of all words possible by pressing given numbers.
    static ArrayList<String> possibleWords(int a[], int N) {
        // your code here
        String keyboard[] = new String[] { "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "
wxyz" };
        ArrayList<String> ls = new ArrayList<>();
        String temp = "";
        getCombinations(0, N, keyboard, ls, temp, a);
        return ls;
    }

    static void getCombinations(int ind, int n, String map[], ArrayList<String> ls, String
temp, int a[]) {
        if (ind == n) {
            ls.add(temp);
            return;
        }
        for (int i = 0; i < map[a[ind] - 2].length(); i++) {
            getCombinations(ind + 1, n, map, ls, temp + map[a[ind] - 2].charAt(i), a);
        }
    }
}
Introduction to Trees

Binary Tree Representation in C++
Binary Tree Representation in Java
Binary Tree Traversals in Binary Tr…
Preorder Traversal of Binary Tree
```

Inorder Traversal of Binary Tree
Post-order Traversal of Binary Tree

Level order Traversal / Level order…

Iterative Preorder Traversal of Bin…

Iterative Inorder Traversal of Bina…

Post-order Traversal of Binary Tree…

Post-order Traversal of Binary Tree…

Preorder, Inorder, and Postorder Tr…

Height of a Binary Tree

Check if the Binary tree is height-…

Diameter of Binary Tree

Maximum path sum

Check if two trees are identical or…

Zig Zag Traversal of Binary Tree

Boundary Traversal of Binary Tree

Vertical Order Traversal of Binary …

Top View of Binary Tree

Bottom View of Binary Tree

Right/Left View of Binary Tree

Symmetric Binary Tree

Root to Node Path in Binary Tree

LCA in Binary Tree

Maximum width of a Binary Tree

Check for Children Sum Property

Print all the Nodes at a distance o…

Minimum time taken to BURN the Bina…

Count total Nodes in a COMPLETE Bin…

Requirements needed to construct a …

Construct Binary Tree from inorder …

Construct the Binary Tree from Post…

Serialize and deserialize Binary Tr…

Morris Preorder Traversal of a Bina…

Morris Inorder Traversal of a Binar…

Flatten Binary Tree to LinkedList

}