Recursion

1.

```
import java.util.*;


public class PossibleStrings {
	public static void printAllStrings(char[] set, int k) {
		int n = set.length;
		printAllStringsHelper(set, "", n, k);
	}

	public static void printAllStringsHelper(char[] set, String prefix, int n, int k) {
		if (k == 0) {
			System.out.println(prefix);
			return;
		}

		for (int i = 0; i < n; ++i) {
			String newPrefix = prefix + set[i];
			printAllStringsHelper(set, newPrefix, n, k - 1);
		}
	}

	public static void main(String[] args) {
		 Scanner sc=new Scanner(System.in);
		 System.out.println(“Enter no. of inputs:”);
		 int n=sc.nextInt();
		char[] set1 = new char[n]
		int k = sc.nextInt();
```

```
            printAllStrings(set1, k1);



        }
}
```

……..

2.

```
Import java.util.*;
public class UniquePaths {

        public static void main(String[] args) {
                Scanner sc=new Scanner(System.in);
                int m = sc.nextInt();
                int n =sc.nextInt();

                int uniquePaths = findUniquePaths(m, n);
                System.out.println("Number of unique paths: " + uniquePaths);
        }

        public static int findUniquePaths(int m, int n) {
                int[][] paths = new int[n][m];
                for (int i = 0; i < n; i++) {
                        paths[i][0] = 1;
                }
                for (int j = 0; j < m; j++) {
                        paths[0][j] = 1;
                }

                for (int i = 1; i < n; i++) {
```

```
                for (int j = 1; j < m; j++) {
                        paths[i][j] = paths[i-1][j] + paths[i][j-1];
                }
        }

        return paths[n-1][m-1];
    }
}
```

……

3.

```
Import java.util.*;
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        next = null;
    }
}

public class Solution {
    public void reorderList(ListNode head) {
        if (head == null || head.next == null || head.next.next == null) {
            return;
        }

        ListNode prev = null;
        ListNode slow = head;
```

```
        ListNode fast = head;

        while (fast != null && fast.next != null) {
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }

        prev.next = null;
        ListNode l1 = head;
        ListNode l2 = reverse(slow);
        merge(l1, l2);
    }

    private ListNode reverse(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode prev = null;
        ListNode curr = head;

        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
```

```
        return prev;
    }

    private void merge(ListNode l1, ListNode l2) {
        while (l1 != null && l2 != null) {
            ListNode l1Next = l1.next;
            ListNode l2Next = l2.next;
            l1.next = l2;
            l2.next = l1Next;
            l1 = l1Next;
            l2 = l2Next;
        }
    }
}
```

……

4.

```
import java.util.*;
public class TowersOfHanoi {
    public static void main(String[] args) {
       Scanner sc= new Scanner(System.in);
        int n = sc.nextInt();
        towerOfHanoi(n, 1, 3, 2);
    }

    public static void towerOfHanoi(int n, int from, int to, int aux) {
        if (n == 1) {
            System.out.println("Move disk from rod " + from + " to rod " + to);
            return;
        }
```

```
        towerOfHanoi(n - 1, from, aux, to);
        System.out.println("Move disk from rod " + from + " to rod " + to);
        towerOfHanoi(n - 1, aux, to, from);
    }
}
```

……

5.

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public class Solution {
    public List<TreeNode> allPossibleFBT(int n) {
        List<TreeNode> res = new ArrayList<>();
        if (n == 1) {
            res.add(new TreeNode(0));
            return res;
        }
        for (int i = 1; i < n; i += 2) {
            List<TreeNode> left = allPossibleFBT(i);
            List<TreeNode> right = allPossibleFBT(n - i - 1);
            for (TreeNode l : left) {
                for (TreeNode r : right) {
                    TreeNode root = new TreeNode(0);
```

```
                        root.left = l;
                        root.right = r;
                        res.add(root);
                    }
                }
            }
            return res;
        }
}
```

……..

6.

```
Import java.util.*;
public class SubsetSum {
        public static void main(String[] args) {
             Scanner sc = new Scanner(System.in);
             Int n=sc.nextInt();
            int[] arr =new int[n];
             for(int i=0;i<n;i++){
                  arr[i]=sc.nextInt();
                  }
            printSubsetSum(arr, arr.length, 0);
        }

        public static void printSubsetSum(int[] arr, int n, int sum) {
            if (n == 0) {
                System.out.print(sum + " ");
                return;
            }
```

```
            printSubsetSum(arr, n - 1, sum + arr[n - 1]);



            printSubsetSum(arr, n - 1, sum);
        }
}
```

……..

7.

```
Import java.util*.;
public class Knapsack {
        public static int knapsack(int[] profits, int[] weights, int capacity, int n) {
                if (n == 0 || capacity == 0) {
                        return 0;
                }

                if (weights[n - 1] > capacity) {
                        return knapsack(profits, weights, capacity, n - 1);
                }



                return Math.max(profits[n - 1] + knapsack(profits, weights, capacity - weights[n - 1], n - 1),
                                        knapsack(profits, weights, capacity, n - 1));
        }

        public static void main(String[] args) {
                 int n = sc.nextInt();
                int[] profits=new int[];
                int[] weights = new int[];
               for(int i=0;i<n;i++)
```

```
            {
              profits[i]=sc.nextInt();
            }
for(int i=0;i<n;i++)
            {
              weights[i]=sc.nextInt();
            }

              int capacity = sc.nextInt();



              int maxProfit = knapsack(profits, weights, capacity, n);
              System.out.println("Maximum profit: " + maxProfit);
        }
}
```

……..

8.

```
import java.util.*;
class Solution {
        public List<List<String>> partition(String s) {
                List<List<String>> result = new ArrayList<>();
                if (s == null || s.length() == 0) {
                        return result;
                }
                helper(s, new ArrayList<>(), result);
                return result;
        }

        private void helper(String s, List<String> currentList, List<List<String>> result) {
```

```
        if (s.length() == 0) {
            result.add(new ArrayList<>(currentList));
            return;
        }
        for (int i = 1; i <= s.length(); i++) {
            String currentString = s.substring(0, i);
            if (isPalindrome(currentString)) {
                currentList.add(currentString);
                helper(s.substring(i), currentList, result);
                currentList.remove(currentList.size() - 1);
            }
        }
    }

    private boolean isPalindrome(String s) {
        int left = 0, right = s.length() - 1;
        while (left < right) {
            if (s.charAt(left++) != s.charAt(right--)) {
                return false;
            }
        }
        return true;
    }
}
```