SORTING:-

```java
package AOA;
import java.util.*;

public class sorting {

    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    public static void insertion(int[] arr, int n) {
        for (int i = 1; i < n; i++) {
            int j = i - 1;
            int x = arr[i];
            while (j > -1 && arr[j] > x) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = x;
        }
    }

    public static void selection(int[] arr,int n){
        for(int i=0;i<n-1;i++){
            int min = i;
            for (int j = i+1; j < arr.length; j++) {
                if (arr[j]<arr[min]) {
                    min = j;
                }
            }
            if (arr[i]>arr[min]) {
                swap(arr, i, min);
            }
        }
    }

    public static int partition(int[] arr , int l , int h){
        int pivot = arr[l],i=l,j=h;
        while (i<j) {
            do {
                i++;
            } while (arr[i]<=pivot);
```

```
            do {
                j--;
            } while (arr[j]>pivot);
            if (i<j) {
                swap(arr, i, j);
            }
        }
        swap(arr, l, j);
        return j ;
    }
    public static void quicksort(int[]arr,int beg,int end){
        if (beg<end) {
            int j = partition(arr, beg, end);
            quicksort(arr, beg, j-1);
            quicksort(arr, j+1,end);
        }
    }

     public static void merge(int[] arr , int beg , int mid ,int
end){
        int n1 = mid-beg+1;
        int n2 = end-mid;
        int[] arr1 = new int[n1];
        int[] arr2 = new int[n2];

        for (int i = 0; i < n2; i++) {
            arr2[i]=arr[mid+1+i];
        }
        for (int i = 0; i < n1; i++) {
            arr1[i]=arr[beg+i];
        }

        int i =0,j=0,k=beg;
        while (i<n1 && j<n2) {
            if (arr1[i]<arr2[j]) {
                arr[k] = arr1[i];
                i++;
                k++;
            }
            else{
                arr[k] = arr2[j];
                j++;
                k++;
```

```
            }
        }
        while (i<n1) {
            arr[k] = arr1[i];
            i++;
            k++;
        }
        while (j<n2) {
            arr[k] = arr2[j];
            j++;
            k++;
        }
    }

    public static void mergesort(int[] arr , int beg , int end){
        if(beg<end){
            int mid = (beg+end)/2;
            mergesort(arr, beg, mid);
            mergesort(arr, mid+1,end);
            merge(arr, beg, mid, end);
        }

    }

    public static void main(String[] args) {
        int[] arr = {3, 6, 7, 2, 8, 9};
        int n = arr.length; // Number of elements in the array
        insertion(arr, n);
        selection(arr, n);
        mergesort(arr, 0, n-1);
        quicksort(arr, 0, n-1);
        System.out.print("Sorted array: "+Arrays.toString(arr));

    }
}
```

BINARY : -

```java
package AOA;
import java.util.*;

public class binarysearch {
    public static int Bsearch (int[] arr , int n , int key) {
        int i = 0 , j = n;
        while (i<j) {
            int mid = (i+j)/2;
            if (arr[mid]==key) {
                return key ;
            }

            if (key>arr[mid]) {
                i = mid+1;
            } else {
                j = mid - 1 ;
            }

        }
        return 0 ;

    }
    public static void main(String[] args) {
        int[] arr = {2,6,8,22,34,56,78};
        int n =arr.length;
        int flag = Bsearch(arr, n-1, 22);
        if (flag==0) {
            System.out.println("Element not found");
        } else {
            System.out.println("Element found");
        }
        flag = Bsearch(arr, n-1, 33);
        if (flag==0) {
            System.out.println("Element not found");
        } else {
            System.out.println("Element found");
        }

    }
}
```

DICKSTRA : -

```java
package AOA;

import java.util.*;

public class dickstra {
    static int infinity = 9999;

    static void dickstraa(int[][] graph, int start, int v) {
        int[][] cost = new int[v][v];
        boolean[] visited = new boolean[v];
        int[] distance = new int[v];
        int[] pred = new int[v];
        int u = 0;

        // Initialize cost matrix with infinity for non-existing edges
        for (int i = 0; i < v; i++) {
            for (int j = 0; j < v; j++) {
                if (graph[i][j] == 0) {
                    cost[i][j] = infinity;
                } else {
                    cost[i][j] = graph[i][j];
                }
            }
        }

        // Initialize distance and visited arrays
        for (int i = 0; i < v; i++) {
            distance[i] = cost[start][i];
            pred[i] = start;
            visited[i] = false;
        }

        distance[start] = 0;
        visited[start] = true;
        int count = 1;

        while (count < v) { // Change to < v instead of v-1
            int mindist = infinity;
            for (int i = 0; i < v; i++) {
                if (!visited[i] && distance[i] < mindist) {
                    mindist = distance[i];
                    u = i;
```

```
                }
            }

            visited[u] = true;

            for (int i = 0; i < v; i++) {
                 if (!visited[i] && mindist + cost[u][i] < distance[i])
{
                    distance[i] = mindist + cost[u][i];
                    pred[i] = u;
                }
            }
            count++;
        }

        // Print distances
        for (int i = 0; i < v; i++) {
             System.out.println("Distance " + start + " -> " + i + " : "
+ (distance[i] == infinity ? "INF" : distance[i]));
        }
    }

    public static void main(String[] args) {
        int[][] graph = {
            {0, 0, 1, 2, 0, 0, 0},
            {0, 0, 2, 0, 0, 3, 0},
            {1, 2, 0, 1, 3, 0, 0},
            {2, 0, 1, 0, 0, 0, 1},
            {0, 0, 3, 0, 0, 2, 0},
            {0, 3, 0, 0, 2, 0, 1},
            {0, 0, 0, 1, 0, 1, 0}
        };
        dickstraa(graph, 0, 7);
    }
}
```

PIMS : -

```
package AOA;

import java.util.Arrays;

public class pim {
    static final int MAX_VERTICES = 100;

    // Function to find the vertex with the minimum key value
    static int minKey(int key[], boolean mstSet[], int V) {
        int min = Integer.MAX_VALUE, min_index = -1;
        for (int v = 0; v < V; v++)
            if (!mstSet[v] && key[v] < min){
                min = key[v];
                min_index = v;
            }
        return min_index;
    }

    // Function to print the minimum spanning tree
     static void printMST(int parent[],int key[], int graph[][], int V)
{
        int sum = 0 ;
        System.out.println("Edges in the minimum spanning tree:");
        for (int i = 0; i < V; i++){
             sum = sum + key[i];
                  System.out.println((parent[i]+1) + " -- " + (i+1) +
"\tWeight: " + key[i]);
        }
        System.out.println("Cost : "+sum);
}

    // Function to construct and print the minimum spanning tree
    static void primMST(int graph[][], int V) {
        int parent[] = new int[V];
        int key[] = new int[V];
        boolean mstSet[] = new boolean[V];

        // Initialize all keys as infinite, mstSet as false
        Arrays.fill(key, Integer.MAX_VALUE);
        Arrays.fill(mstSet, false);

        // Set the key of the first vertex to 0
```

```
        key[0] = 0;
        parent[0] = -1;

        // Construct the MST with V-1 vertices
        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet, V);
            mstSet[u] = true;

              // Update key and parent indices of the adjacent vertices
of the picked vertex
            for (int v = 0; v < V; v++) {
                    if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] <
key[v]) {
                     parent[v] = u;
                     key[v] = graph[u][v];
                }
            }
        }

        // Print the constructed MST
        printMST(parent,key, graph, V);
    }

    public static void main(String[] args) {
        int V = 7; // Number of vertices
        int graph[][] = {
            {0, 28, 0, 0, 0, 10, 0},
            {28, 0, 16, 0, 0, 0, 14},
            {0, 16, 0, 12, 0, 0, 0},
            {0, 0, 12, 0, 22, 0, 18},
            {0, 0, 0, 22, 0, 25, 24},
            {10, 0, 0, 0, 25, 0, 0},
            {0, 14, 0, 18, 24, 0, 0}
        };

        primMST(graph, V);
    }
}
```

JOB SCHEDULING : -

```
package AOA;
/**
 * job
 */
import java.util.*;
class job {
    int id ;
    int pr;
    int dl;
}
public class jobseq {
    public static void sequence(job[] arr,int n){
        boolean[] check= new boolean[n];
                    Arrays.sort(arr,Comparator.comparingInt((job  j)  ->
j.pr).reversed());
        int t = 0 ;
        for (int i = 0; i < n; i++) {
            t = Math.max(t, arr[i].dl);
        }

        job[] seq = new job[t];
        for (int i = 0; i < n; i++) {
            for (int k = Math.min(t,arr[i].dl)-1; k>=0; k--) {
                if (check[k]==false) {
                    check[k]=true;
                    seq[k] = arr[i];
                    break;
                }
            }
        }
        System.out.print("OPTIMAL SEQUENCE : ");
        int sum = 0 ;
        for (int i = 0; i < seq.length; i++) {
            sum = sum + seq[i].pr;
            System.out.print(seq[i].id + " ");
        }
        System.out.print("\nTotal PROFIT : "+sum);

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        int n ;
        System.out.print("Enter no.of Jobs : ");
        n = sc.nextInt();
        job[] arr = new job[n];

        for (int i = 0; i < arr.length; i++) {
            arr[i] = new job();
            arr[i].id = i+1 ;
            System.out.print("Job "+(i+1)+" :\nProfit : ");
            arr[i].pr = sc.nextInt();
            System.out.print("Deadline : ");
            arr[i].dl = sc.nextInt();
            System.out.println("");
        }

        sequence(arr,n);
    }


}
```

FLOYYD :-

```
package AOA;

public class floyd {
    static final int V = 4;
    static final int INF = 99999;

    static void printSolution(int[][] dist) {
        // System.out.println("The following matrix shows the shortest
distances between every pair of vertices");
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][j] == INF)
                    System.out.printf("\tINF");
                else
                    System.out.printf("\t"+dist[i][j]);
            }
            System.out.println();
        }
    }

    static void floydWarshall(int[][] graph) {


        for (int k = 0; k < V; k++) {
            for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                    if (i==k||j==k||graph[i][j]==0) {
                        graph[i][j] = graph[i][j];
                    }
                    else if (graph[i][k] + graph[k][j] < graph[i][j]) {
                        graph[i][j] = graph[i][k] + graph[k][j];
                    }
                }
            }
        }
        System.out.println("\nSolution :");
        printSolution(graph);
    }

    public static void main(String[] args) {
        int[][] graph = {
                {0, 3, INF, 7},
```

```
                {8, 0, 2, INF},
                {5, INF, 0, 1},
                {2, INF, INF, 0}
        };
        System.out.println("The given matrix :");
        printSolution(graph);
        floydWarshall(graph);
    }
}
```

SUBSETS : -

```java
package AOA;

// A Dynamic Programming solution for subset
// sum problem

public class subsex {

    // Flag to check if there exists a subset with the given
    // sum
    static boolean flag = false;

    // Print all subsets if there is at least one subset of
    // set[] with the sum equal to the given sum
    static void printSubsetSum(int i, int n, int[] set,
                               int targetSum,
                               String subset)
    {
        // If targetSum is zero, then there exists a subset.
        if (targetSum == 0) {
            // Prints a valid subset
            flag = true;
            System.out.println("[" + subset + "]");
            return;
        }

        if (i == n) {
            // Return if we have reached the end of the
            // array
            return;
        }

        // Not considering the current element
        printSubsetSum(i + 1, n, set, targetSum, subset);

        // Consider the current element if it is less than
        // or equal to the targetSum
        if (set[i] <= targetSum) {
            // Include the current element in the subset
            String newSubset;
            if (subset.equals("")) {
                 newSubset = Integer.toString(set[i]);
            }
```

```
            else {
                newSubset = subset + ", " + set[i];
            }


            // Recursive call for considering the current
            // element
            printSubsetSum(i + 1, n, set,
                           targetSum - set[i], newSubset);
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        // Test case 1
        int[] set1 = { 1, 8, 2, 3, 4, 5 };
        int sum1 = 10;
        int n1 = set1.length;
        System.out.println("Output 1:");
        printSubsetSum(0, n1, set1, sum1, "");
        System.out.println();

    }

}
```

TSP : -

```
package AOA;
import java.util.*;

public class tsp {
    static int v = 5;

     public static int tspD(int[][] graph, int start, int current, int
sum, boolean[] visited, int count,ArrayList<Integer> stack) {
        if (count == v && graph[current][start] != 0) {
            //stack.add(0);
            return sum + graph[current][start];
        }

        int min = Integer.MAX_VALUE;
        for (int i = 0; i < v; i++) {
            if (!visited[i] && graph[current][i] != 0) {
                visited[i] = true;
                stack.add(i); // Add the current vertex to the stack
                 System.out.println("Added to stack: " + i + ", Stack: "
+ stack);
                            int newCost = tspD(graph, start, i, sum +
graph[current][i], visited, count + 1, stack);
                min = Math.min(min, newCost);
                visited[i] = false;
                    stack.remove(stack.size() - 1); // Remove the last
added vertex from the stack
                    System.out.println("Removed from stack, Stack: " +
stack);
            }
        }

        return min;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        //System.out.print("Enter no. of cities: ");
        //v = sc.nextInt();
        v = 5;

        int[][] graph = {
            {0,20,30,10,11},
```

```
            {15,0,16,4,2},
            {3,5,0,2,4},
            {19,6,18,0,3},
            {16,4,7,16,0}
        };

        boolean[] visited = new boolean[v];
        visited[0] = true;

        ArrayList<Integer> stack = new ArrayList<>(v) ;
        stack.add(0);
         System.out.println("Minimum cost of traversal: " + tspD(graph,
0, 0, 0, visited, 1,stack));
        stack.add(0);
        System.out.println("Correct Order : "+stack);
        sc.close();
    }
}
```

NQUUEENS : -

```java
package AOA;

import java.util.Scanner;

public class Nqueens {

    static void printSolution(int[][] board, int N) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                if (board[i][j] == 1)
                    System.out.print("Q ");
                else
                    System.out.print(". ");
            System.out.println();
        }
    }

    static boolean isSafe(int[][] board, int row, int col, int N) {
        int i, j;

        // Check if there's a queen in the same row
        for (i = 0; i < col; i++)
            if (board[row][i] == 1)
                return false;

        // Check upper diagonal on left side
        for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
            if (board[i][j] == 1)
                return false;

        // Check lower diagonal on left side
        for (i = row, j = col; j < N && i < N; i++, j++)
            if (board[i][j] == 1)
                return false;

        return true;
    }

    static boolean solveNQUtil(int[][] board, int col, int N) {
        if (col >= N)
            return true;
```

```
        for (int i = 0; i < N; i++) {
            if (isSafe(board, i, col, N)) {
                board[i][col] = 1;
                if (solveNQUtil(board, col + 1, N))
                    return true;
                board[i][col] = 0;
            }
        }

        return false;
    }



    public static void main(String[] args) {
        int N;
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the value of N: ");
        N = scanner.nextInt();
        scanner.close();
        int[][] board = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                board[i][j] = 0;
            }
        }

        if (!solveNQUtil(board, 0, N)) {
            System.out.println("Solution does not exist");
        }
        else{
            printSolution(board, N);
        }


    }
}
```

KNAPSACK :-

```
package AOA;

import java.util.*;;
public class knapsack {
    public static void knap(int[] pr , int[] wt , int n ,int max) {
        int[][] K = new int[n+1][max+1];
        int[] knap = new int[n];
        for (int i = 0; i <= n ; i++) {
            for (int w = 0; w <= max ; w++) {
                if (i==0||w==0) {
                    K[i][w] = 0;
                }
                else if (w>=wt[i]) {
                                K[i][w] = Math.max(K[i-1][w], pr[i] +
K[i-1][w-wt[i]]);
                }
                else{
                    K[i][w]=K[i-1][w];
                }
            }
        }
        System.out.println("Max : "+K[n][max]);

        int i = n,j=max,pos=n-1;
        while (i>0 && j>0) {
            if (K[i][j]==K[i-1][j]) {
                knap[pos] = 0;
                pos--;
                i--;
            }
            else{
                knap[pos]=1;
                pos--;
                j = j - wt[i];
                i--;
            }
        }
                            System.out.println("Selected    numbers    :
"+Arrays.toString(knap));


    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n , max ;
        System.out.print("Enter no. of elements : ");
        n = sc.nextInt();
        System.out.print("Enter max capacity : ");
        max = sc.nextInt();

        int[] pr = new int[n+1];
        int[] wt =new int[n+1];
        pr[0]=0;
        wt[0]=0;
        System.out.print("Enter weigths : ");
        for (int i = 1; i < wt.length; i++) {
            wt[i]=sc.nextInt();
        }
        System.out.print("Enter Profits : ");
        for (int i = 1; i < pr.length; i++) {
            pr[i]=sc.nextInt();
        }

        knap(pr, wt, n, max);


    }
}
```

KSP : -

```
package AOA;
import java.util.*;

public class ksp {

    static void KMPSearch(String pat, String txt) {
        int M = pat.length();
        int N = txt.length();

        int[] lps = new int[M];
        computeLPSArray(pat, M, lps);

        int i = 0;
        int j = 0;
        while (i<N && j<M) {
            if (pat.charAt(j) == txt.charAt(i)) {
                j++;
                i++;
            }


            if (j == M) {
                    System.out.println("Found pattern at index " + (i -
j));

            }
            else if (j<M && i < N && pat.charAt(j) != txt.charAt(i)) {
                if (j != 0)
                    j = lps[j - 1];
                else
                    i = i + 1;
            }
        }
    }

    static void computeLPSArray(String pat, int M, int[] lps) {
        int j = 0;
        lps[0] = 0;

        int i = 1;
        while (i < M) {
            System.out.println(pat.charAt(i)+" <-> "+pat.charAt(j));
```

```
            if (pat.charAt(i) == pat.charAt(j)) {

                j++;
                lps[i] = j;
                i++;
            } else {
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
        }
        System.out.println(Arrays.toString(lps));
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the text: ");
        String txt = scanner.nextLine();
        System.out.print("Enter the pattern: ");
        String pat = scanner.nextLine();
        KMPSearch(pat, txt);
    }
}
```