1. TSP

```
import java.util.*;
public class TSPDynamicProgramming {
        static int[][] distance;
    static int[][] memo;
    static int n;

    public static int tsp(int mask, int pos) {
        if (mask == (1 << n) - 1) {
            return distance[pos][0]; // Return to the starting city
        }

        if (memo[mask][pos] != -1) {
            return memo[mask][pos];
        }

        int minCost = Integer.MAX_VALUE;

        for (int city = 0; city < n; city++) {
            if ((mask & (1 << city)) == 0) { // If city not visited
                int newCost = distance[pos][city] + tsp(mask | (1 << city), city);
                minCost = Math.min(minCost, newCost);
            }
        }

        return memo[mask][pos] = minCost;
    }

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

System.out.print("Enter the number of cities: ");
        n=sc.nextInt();
        distance = new int[n][n] ;
System.out.print("Enter the distance between cities: \n");
for(int i=0;i<n;i++){
for(int j=0;j<n;j++){
distance[i][j]=sc.nextInt();
}
}

        memo = new int[1 << n][n];
        for (int[] row : memo) {
            Arrays.fill(row, -1);
        }
```

```
        int minCost = tsp(1, 0); // Start from city 0
        System.out.println("Minimum cost to visit all cities: " + minCost);
            }
}
```

2. Nqueens

```
public class NQueens {

    private int[] result;
    private boolean[] column;
    private boolean[] leftDiagonal;
    private boolean[] rightDiagonal;
    private int n;

    public NQueens(int n) {
        this.n = n;
        result = new int[n];
        column = new boolean[n];
        leftDiagonal = new boolean[2 * n - 1];
        rightDiagonal = new boolean[2 * n - 1];
    }

    public boolean solve() {
        return solveNQueens(0);
    }

    private boolean solveNQueens(int row) {
        if (row == n) {
            printSolution();
            return true;
        }
        boolean res = false;
        for (int col = 0; col < n; col++) {
            if (isSafe(row, col)) {
                placeQueen(row, col);
                res = solveNQueens(row + 1) || res; // Note: This allows finding all solutions
                removeQueen(row, col); // Backtrack
            }
        }
        return res;
    }

    private boolean isSafe(int row, int col) {
        return !column[col] && !leftDiagonal[row - col + n - 1] && !rightDiagonal[row
+ col];
    }
```

```
    private void placeQueen(int row, int col) {
        result[row] = col;
        column[col] = true;
        leftDiagonal[row - col + n - 1] = true;
        rightDiagonal[row + col] = true;
    }

    private void removeQueen(int row, int col) {
        column[col] = false;
        leftDiagonal[row - col + n - 1] = false;
        rightDiagonal[row + col] = false;
    }

    private void printSolution() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (result[i] == j) {
                    System.out.print("Q ");
                } else {
                    System.out.print(". ");
                }
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n = 6; // You can change the value of n to solve for different sizes of the board
        NQueens queens = new NQueens(n);
        if (!queens.solve()) {
            System.out.println("No solution exists");
        }
    }
}
```

3. KnapSack

```
import java.util.Scanner;

public class knapsack
{
    static int Knapsack(int[] weights, int[] values, int capacity) {
        return branchAndBound(weights, values, capacity, 0, 0, 0);
    }
```

```
    static int branchAndBound(int[] weights, int[] values, int capacity, int index, int
currentWeight, int currentValue) {
        if (currentWeight > capacity) {
            return 0;
        }

        if (index == weights.length) {
            return currentValue;
        }

        int withItem = 0;
        if (currentWeight + weights[index] <= capacity) {
            withItem = branchAndBound(weights, values, capacity, index + 1,
currentWeight + weights[index], currentValue + values[index]);
        }
        int withoutItem = branchAndBound(weights, values, capacity, index + 1,
currentWeight, currentValue);
        return Math.max(withItem, withoutItem);
    }

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

        System.out.print("No of items: ");
        int n = sc.nextInt();
        int[] weights = new int[n];
        int[] values = new int[n];

        System.out.println("Weights of items:");
        for (int i = 0; i < n; i++) {
            weights[i] = sc.nextInt();
        }

        System.out.println("Values of items:");
        for (int i = 0; i < n; i++) {
            values[i] = sc.nextInt();
        }

        System.out.print("Capacity of knapsack: ");
        int capacity = sc.nextInt();

        int maxValue = Knapsack(weights, values, capacity);
        System.out.println("Maximum value: " + maxValue);
        }
}
```

4. Sum-subset

```
import java.util.Scanner;

public class SumOfSubsets {
   static int count = 0;

   static void findSubsets(int currentSum, int k, int remainingSum, int[] included, int[]
weights, int target) {
      int n = weights.length;
      if (currentSum == target) {
         count++;
         System.out.print("Solution " + count + ": {");
         for (int i = 0; i < n; i++) {
            if (included[i] == 1) {
               System.out.print(weights[i] + " ");
            }
         }
         System.out.println("}");
      } else if (k < n) {
         // Include weights[k] in the subset
         included[k] = 1;
         if (currentSum + weights[k] <= target) {
            findSubsets(currentSum + weights[k], k + 1, remainingSum - weights[k],
included, weights, target);
         }

         // Exclude weights[k] from the subset
         included[k] = 0;
         if (currentSum + remainingSum - weights[k] >= target) {
            findSubsets(currentSum, k + 1, remainingSum - weights[k], included,
weights, target);
         }
      }
   }

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

      System.out.print("Enter the number of elements in the set: ");
      int n = sc.nextInt();
      int[] weights = new int[n];
      int[] included = new int[n];
      int totalSum = 0;

      System.out.println("Enter the elements: ");
      for (int i = 0; i < n; i++) {
```

```
            weights[i] = sc.nextInt();
            totalSum += weights[i];
        }

        System.out.print("Enter the desired sum: ");
        int target = sc.nextInt();

        System.out.println("Total sum of elements: " + totalSum);
        findSubsets(0, 0, totalSum, included, weights, target);
    }
}
```

5. Dijkstras

```
import java.util.Scanner;

public class DijkstraAlgorithm {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter the number of nodes: ");
        int n = in.nextInt();

        int[][] cost = new int[n + 1][n + 1];
        System.out.println("Enter the cost matrix:");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                cost[i][j] = in.nextInt();
            }
        }

        System.out.print("Enter the source vertex: ");
        int src = in.nextInt();

        int[] dist = new int[n + 1];
        int[] path = new int[n + 1];
        boolean[] visited = new boolean[n + 1];
        dijkstra(cost, dist, src, n, path, visited);
        printPath(src, n, dist, path, visited);

    }

    static void dijkstra(int[][] cost, int[] dist, int src, int n, int[] path, boolean[] visited)
{
        for (int i = 1; i <= n; i++) {
            dist[i] = cost[src][i];
            path[i] = cost[src][i] == 999 ? 0 : src;
```

```
            visited[i] = false;
        }
        dist[src] = 0;
        visited[src] = true;

        for (int count = 2; count <= n; count++) {
            int min = 999, v = -1;
            for (int w = 1; w <= n; w++) {
                if (!visited[w] && dist[w] < min) {
                    min = dist[w];
                    v = w;
                }
            }
            if (v == -1) return;  // All remaining nodes are unreachable

            visited[v] = true;

            for (int w = 1; w <= n; w++) {
                if (!visited[w] && dist[w] > dist[v] + cost[v][w]) {
                    dist[w] = dist[v] + cost[v][w];
                    path[w] = v;
                }
            }
        }
    }

    static void printPath(int src, int n, int[] dist, int[] path, boolean[] visited) {
        for (int w = 1; w <= n; w++) {
            if (visited[w] && w != src) {
                System.out.println("The shortest distance between " + src + " and " + w + "
is: " + dist[w]);
                System.out.print("Path: " + w);
                int t = path[w];
                while (t != src && t != 0) {
                    System.out.print(" --> " + t);
                    t = path[t];
                }
                System.out.println(" <-- " + src);
            }
        }
    }
}
```

6. QuickSort

```
import java.util.*;
import java.io.*;
class QuickSort{
  static int max=5000;
  void quick(int arr[],int l,int h)
  {
    int s;
    if(l<h){
      s = partition(arr,l,h);
      quick(arr,l,s-1);
      quick(arr,s+1,h);
    }
  }
  int partition(int arr[],int l,int h){
    int p,i,j,temp;
    p = arr[l];
    i = l+1;
    j = h;
    while(l<h){
      while(arr[i]<p && i<h){
        i++;
      }
      while(arr[j]>p){
        j--;
      }
      if(i<j){
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
      else{
        temp = arr[l];
        arr[l] = arr[j];
        arr[j] = temp;
        return j;
      }
    }
    return j;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter number of ele: ");
    int n = sc.nextInt();
    Random gen = new Random();
    int arr[] = new int[max];
    for(int i=0;i<n;i++){
```

```
    arr[i] = gen.nextInt(1000);
   }
   System.out.println("Random ele: ");
   for(int i=0;i<n;i++){
    System.out.print(arr[i]+" ");
   }
   System.out.println();
   long start = System.nanoTime();
   QuickSort qs = new QuickSort();
   qs.quick(arr,0,n-1);
   long stop = System.nanoTime();
   System.out.println("array after sorting: ");
   for(int i=0;i<n;i++){
      System.out.print(arr[i]+" ");
   }
   System.out.println("Time taken: "+(stop-start));

  }

}
```

7. Floyds

```
import java.util.*;
class flyods{
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of vertices: ");
    int n =sc.nextInt();
    System.out.println("Enter the adj matrix:(enter 999 for infinity) ");
    int adj[][] = new int[10][10];
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        adj[i][j] = sc.nextInt();
      }
    }
    flyod(adj,n);
    System.out.println("the all pair shoretst path is: ");
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        System.out.print(adj[i][j]+" ");
      }
      System.out.println();
    }
  }
  static void flyod(int arr[][],int n){
    for(int k=1;k<=n;k++){
```

```
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        arr[i][j] = min(arr[i][j],(arr[i][k]+arr[k][j]));
      }
    }
  }
 }
 static int min(int a,int b){
  if(a<b){
   return a;
  }
  return b;
 }
}
```

8. Bellman Ford

```
import java.util.*;
class Graph {
   static class Edge {
   int src, dest, weight;

   Edge(int s, int d, int w) {
     src = s;
     dest = d;
     weight = w;
   }
  };

  int V, E;
  Edge edge[];

   Graph(int v, int e) {
   V = v;
   E = e;
   edge = new Edge[e];
  }


  void BellmanFord(Graph graph, int src) {
   int V = graph.V, E = graph.E;
   int dist[] = new int[V];

     for (int i = 0; i < V; ++i)
    dist[i] = Integer.MAX_VALUE;
   dist[src] = 0;
```

```
    for (int i = 1; i < V; ++i) {
   for (int j = 0; j < E; ++j) {
    int u = graph.edge[j].src;
    int v = graph.edge[j].dest;
    int weight = graph.edge[j].weight;
    if (dist[u] != Integer.MAX_VALUE
        && dist[u] + weight < dist[v])
      dist[v] = dist[u] + weight;
   }
  }

   for (int j = 0; j < E; ++j) {
   int u = graph.edge[j].src;
   int v = graph.edge[j].dest;
   int weight = graph.edge[j].weight;
   if (dist[u] != Integer.MAX_VALUE
      && dist[u] + weight < dist[v]) {
    System.out.println(
       "Graph contains negative weight cycle");
    return;
   }
  }
  printArr(dist, V);
}

  void printArr(int dist[], int V) {
  System.out.println("Vertex Distance from Source");
  for (int i = 0; i < V; ++i)
    System.out.println(i + "\t\t" + dist[i]);
}

  public static void main(String[] args) {
  Scanner in = new Scanner(System.in);
  System.out.print("Enter no. of vertices: ");
  int V = in.nextInt();
  System.out.print("Enter no. of edges: ");
  int E = in.nextInt();
  Graph graph = new Graph(V, E);
  for (int i = 0; i < E; i++) {
    System.out.print("Enter src, dest and weight for edge " + (i + 1) + " : ");
    int src = in.nextInt();
    int dest = in.nextInt();
    int weight = in.nextInt();
    graph.edge[i] = new Edge(src, dest, weight);
  }
  graph.BellmanFord(graph, 0);
```

```
  }
}
```

9. Prims

```
import java.util.Scanner;
public class PrimsClass
{
final static int MAX = 20;
static int n;                          // No. of vertices of G
static int cost[][];                   // Cost matrix
static Scanner scan = new Scanner(System.in);
public static void main(String[] args)
{
ReadMatrix();
Prims();
}

static void ReadMatrix()
{
int i, j;
cost = new int[MAX][MAX];
System.out.println("\n Enter the number of nodes:");
n = scan.nextInt();
System.out.println("\n Enter the adjacency matrix:\n");
for (i = 1; i <= n; i++)
 for (j = 1; j <= n; j++)
{
  cost[i][j] = scan.nextInt();
  if (cost[i][j] == 0)
  cost[i][j] = 999;

}
}
static void Prims()
{
int visited[] = new int[10];
int ne = 1, i, j, min, a = 0, b = 0, u = 0, v = 0;
int mincost = 0;
visited[1] = 1;
while (ne < n)
{
for (i = 1, min = 999; i <= n; i++)
for (j = 1; j <= n; j++)
if (cost[i][j] < min)
if (visited[i] != 0)
{
```

```
min = cost[i][j];
a = u = i;
b = v = j;
}
        if (visited[u] == 0 || visited[v] == 0)
{
                System.out.println("Edge" + ne++ + ":(" + a + "," + b + ")" + "cost :" +
min);
mincost += min;
visited[b] = 1;
        }
cost[a][b] = cost[b][a] = 999;
}
System.out.println("\n Minimun cost" + mincost);
   }
}
```

10. Kruskals

```
import java.util.Scanner;
public class KruskalsClass
{
final static int MAX = 20;
static int n; // No. of vertices of G
static int cost[][]; // Cost matrix
static Scanner scan = new Scanner(System.in);
public static void main(String[] args)
{
ReadMatrix();
Kruskals();
}
static void ReadMatrix()

{
int i, j;
cost = new int[MAX][MAX];
System.out.println("Implementation of Kruskal's algorithm");
System.out.println("Enter the no. of vertices");
n = scan.nextInt();
System.out.println("Enter the cost adjacency matrix");
for (i = 1; i <= n; i++)
 {
for (j = 1; j <= n; j++)
{
cost[i][j] = scan.nextInt();
if (cost[i][j] == 0)
cost[i][j] = 999;
```

```
}
}
}
static void Kruskals()
{
int a = 0, b = 0, u = 0, v = 0, i, j, ne = 1, min, mincost = 0;
int parent[] = new int[9];
for (i = 1; i <= n; i++)
{
parent[i]=0; //making Set
}
System.out.println("The edges of Minimum Cost Spanning Tree are");
while (ne < n)
{
min = 999;
for (i = 1; i <= n; i++)
{
for (j = 1; j <= n; j++)
      {
if (cost[i][j] < min)
{
min = cost[i][j];
a = u = i;
b = v = j;
}
}

}

while(parent[u]!=0)

        u=parent[u];

 while(parent[v]!=0) //finding Set
      v=parent[v];
 if (u != v) // can union be done?
 {
 System.out.println(ne++ + "edge (" + a + "," + b + ") =" + min);
 mincost += min;
 parent[v]=u; //union
 }
 cost[a][b] = cost[b][a] = 999;

 }
 System.out.println("Minimum cost :" + mincost);
 }
 }
```

11. Selection Sort

```
import java.util.Scanner;
public class SelectionSort {
public static void main(String args[]){
Scanner sc=new Scanner(System.in);
int n;
System.out.print("Enter the number of elements in the array: ");
n=sc.nextInt();
int a[]=new int[n];
System.out.print("Enter the elements of the array:");

for(int i=0;i<n;i++)
a[i]=sc.nextInt();
int min=0;
System.out.print("Array before sorting:");
for(int i=0;i<n;i++)
{
System.out.print(a[i]+" ");
}
System.out.println();

for(int i=0;i<n;i++){
min=i;
for(int j=i+1;j<n;j++){
if(a[min]>a[j])
min=j;
}
int temp=a[min];
a[min]=a[i];
a[i]=temp;
}
System.out.print("Array after sorting:");
for(int i=0;i<n;i++)
{
System.out.print(a[i]+" ");
}
System.out.println();
}
```

12. Fibonacci using recursion

```
import java.util.Scanner;

public class Fib

{

    static int fib(int x)
```

```
    {
        if(x==1)
            return 15;
        if(x==2)
            return 23;
        else
            return fib(x-1)+fib(x-2);
    }

    public static void main (String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("The next 3 terms of the series 15,23,38,61 is: ");
        for(int i=1;i<=7;i++)
            System.out.print(fib(i)+" ");
    }
}
```

13. Binary using time complexity

```
import java.util.*;
public class knapsack
{
static void search(int a[], int key)
{
int n=a.length;
int start=0, end=n-1,mid=-1;
long startTime=System.nanoTime();
while(start<=end)
{
mid=(start+end)/2;

if(a[mid]==key)
{
long endTime=System.nanoTime();
long totalTime=endTime-startTime;
System.out.println("Total time taken="+totalTime+"\n Element found at index:"+mid);
return ;
}
```

```
else if(a[mid]>key)
   end=mid-1;
else if(a[mid]<key)
   start=mid+1;
}
long endTime=System.nanoTime();
long totalTime=endTime-startTime;
System.out.println("Total time taken="+totalTime+"\n Element found at index:-1");
return ;
}

  public static void main(String args [])
{
Scanner sc=new Scanner(System.in);
int n;
System.out.println("Enter the size of the array:");
n=sc.nextInt();
int a[]= new int[n];
System.out.println("Enter the elements in sorted ascending order:");
for(int i=0;i<n;i++)
   a[i]=sc.nextInt();
int key;

System.out.println("Enter the search element:");
key=sc.nextInt();
search(a,key);
}
}
```

14. NCR

```
import java.util.Scanner
public class NCR
{
        static int fact(int x)
        {
                if(x==0||x==1)
                        return 1;
                else
                        return (x*fact(x-1));
        }
        public static void main(String args[])
        {
                int n,r,res;
                Scanner sc=new Scanner(System.in);
                System.out.print("Enter number of items to choose from: ");
                n=sc.nextInt();
```

```
                System.out.print("Enter number of items to be chosen: ");
                r=sc.nextInt();
                res=fact(n)/(fact(n-r)*fact(r));
                System.out.print("No of ways: "+res);
        }
    }
```

15. Linear

```
import java.util.Scanner;

public class LinearSearch
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int arr[]=new int [10];
        int i,n,key;
        boolean found=false;
        System.out.print("Enter Number of Elements: ");
        n=sc.nextInt();
        System.out.println("Enter the Elements:");
        for(i=0;i<n;i++)
        {
            arr[i]=sc.nextInt();
        }
        System.out.println();
        System.out.print("Enter the search Element: ");
        key=sc.nextInt();

        for(i=0;i<n;i++)
        {
            if(key==arr[i])
            {
                System.out.println(key+" found at position "+(i+1));
                found=true;
            }
        }
        if(!found)
        {
            System.out.println(key+" not found!");
        }

    }
}
```