Linear

```
import java.util.Scanner;


class Ll {
   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
      boolean found = false;
      System.out.print("Enter the number of elements: ");
      int n = sc.nextInt();
      System.out.print("Enter the Elements: ");
      int arr[] = new int[n];
      for (int i = 0; i < n; i++) {
         arr[i] = sc.nextInt();
      }
      System.out.print("Enter the key element to be searched: ");
      int key = sc.nextInt();
      for (int i = 0; i < n; i++) {
         if (key == arr[i]) {
            System.out.println("The element is found at index " + i);
            found = true;
            break;
         }
      }
      if (!found) {
         System.out.println("The element is not found");
      }
   }
}
```

NCR

```
import java.util.Scanner;

class ncr {
   public static int fact(int x) {
      int fac = 1;
      for (int i = 1; i <= x; i++) {
         fac *= i;
      }
      return fac;
   }

   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the total number of items to be selected:");
      int n = sc.nextInt();
      System.out.println("Enter the chosen items:");
      int r = sc.nextInt();
      int res = fact(n) / (fact(r) * fact(n - r));
      System.out.println(res);
   }
}
```

FIB

```
import java.util.Scanner;

public class fib {
   static int fibo(int x) {
      if (x == 1)
         return 15;
      if (x == 2)
         return 23;
```

```
        else
            return fibo(x - 1) + fibo(x - 2);
    }

    public static void main(String args[]) {
        System.out.print("The next series of Fibonacci is: ");
        for (int i = 1; i <= 7; i++)
            System.out.print(fibo(i) + " ");
    }
}
```

Selection

```
import java.util.Scanner;
public class ss
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the number of elements");
        int n=sc.nextInt();
        int i;
        int min=0;
        int arr[]=new int[20];
        System.out.println("Enters the elements");
        for( i=0;i<n;i++)
        {
            arr[i]=sc.nextInt();
        }
        System.out.println("The elements before sorting is:");
        for( i=0;i<n;i++)
        {
            System.out.println(arr[i]+" ");
```

```
        }
        for(i=0;i<n-1;i++){
          min=i;
        for(int j=i+1;j<n;j++)

           if(arr[min]>arr[j]){
               min=j;}
        int temp=arr[min];
        arr[min]=arr[i];
        arr[i]=temp;}
        System.out.println("The elemnts after sorting are");
        for( i=0;i<n;i++){
    System.out.println(arr[i]+" ");
}}}
```

Binary search

```
import java.util.Scanner;

public class BS {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements:");
        int n = sc.nextInt();
        int arr[] = new int[n];
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
           arr[i] = sc.nextInt();
        }
        System.out.println("Enter the element to search:");
        int key = sc.nextInt();
```

```
        int l = 0;
        int h = n - 1;
        int mid = -1;
        boolean found = false;
        long startTime = System.nanoTime();

        while (l <= h) {
            mid = (l + h) / 2;
            if (arr[mid] == key) {
                long endTime = System.nanoTime();
                System.out.println("The element is found at position " + mid + ". Time taken: " + (endTime - startTime) + " ns");
                found = true;
                break;
            } else if (arr[mid] > key) {
                h = mid - 1;
            } else {
                l = mid + 1;
            }
        }
        if (!found) {
            long endTime = System.nanoTime();
            System.out.println("Element not found. Total time taken: " + (endTime - startTime) + " ns");
        }
    }
}
```

QuickSort

```
import java.util.Scanner;
import java.util.Random;

public class quick {
   void quickk(int arr[], int l, int h) {
      int s;
      if (l < h) {
         s = partition(arr, l, h);
         quickk(arr, l, s - 1);
         quickk(arr, s + 1, h);
      }
   }

   public int partition(int arr[], int l, int h) {
      int p = arr[l];
      int temp, i, j;
      i = l + 1;
      j = h;
      while (i <= j) {
         while (i <= h && arr[i] < p) {
            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 the number of elements");
       int n = sc.nextInt();
       Random gen = new Random();
       int arr[] = new int[n];
       int i;
       for (i = 0; i < n; i++) {
          arr[i] = gen.nextInt(1000);
       }
       long start = System.nanoTime();
       quick qs = new quick();
       qs.quickk(arr, 0, n - 1);
       System.out.println("Sorted elements:");
       for (i = 0; i < n; i++) {
          System.out.print(arr[i] + " ");
       }
       long end = System.nanoTime();
       System.out.println("\nTotal time taken: " + (end - start) + " ns");
       sc.close();
    }
}
```

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;
```

```
            removeQueen(row, col);
        }
    }
    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;
        NQueens queens = new NQueens(n);
        if (!queens.solve()) {
            System.out.println("No solution exists");
        }
    }
}
```

SumofSumbets

```
package daa_sea;
import java.util.Scanner;

public class LP17_SumOfSubSubset
{
    static int count = 0;

    static void subset(int cs, int k, int r, int x[], int w[], int d)
    {
        x[k] = 1;
        int n = w.length;

        if (cs + w[k] == d)
        {
            count++;
            System.out.print("Solution " + count + ": is {");
```

```
        for (int i = 0; i < n; i++)
        {
            if (x[i] == 1)
            {
                System.out.print(w[i] + " ");
            }
        }
        System.out.println("}");
    }
    else if ((cs + w[k + 1]) <= d)
    {
        subset(cs + w[k], k + 1, r - w[k], x, w, d);
    }
    if ((cs + r - w[k]) >= d && (cs + w[k + 1]) <= d)
    {
        x[k] = 0;
        subset(cs, k + 1, r - w[k], x, w, d);
    }
}

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

    int n, d, sum = 0;
    System.out.println("Enter the number of elements in the set: ");
    n = sc.nextInt();
    int w[] = new int[n];
    int x[] = new int[n];
    System.out.println("Enter the elements: ");
    for (int i = 0; i < n; i++)
```

```
        {
            w[i] = sc.nextInt();
        }
        System.out.println("Enter the desired sum: ");
        d = sc.nextInt();
        for (int i = 0; i < n; i++)
        {
            x[i] = 0;
            sum += w[i];
        }
        System.out.println("Sum is: " + sum);
        subset(0, 0, sum, x, w, d);
    }
}
```

Knapsack IO

```
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);
    }
}
```

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;
  }
}
```

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);
  }
}
```

TSP

```
package q151;
import java.util.Arrays;
import java.util.Scanner;
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);
             }
}
```

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);
   }
}
```

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)       //finding Set

                        {
                        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);
}
}
```

DJ

```
import java.util.Scanner;

public class LP12_Dijkstra
{
	public static void main(String[] args)
	{
	int i, j;
	int dist[]=new int[10], visited[]=new int[10];
	int cost[][]=new int[10][10], path[]=new int[10];
	Scanner in = new Scanner(System.in);
	System.out.println("**** DIJKSTRA'S ALGORITHM ******");
	System.out.println("Enter the number of nodes: ");
	int n = in.nextInt();
	System.out.println("Enter the cost matrix");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	cost[i][j] = in.nextInt();
	System.out.println("The entered cost matrix is");
	for(i=1;i<=n;i++)
	{
```

```
for(j=1;j<=n;j++)
{
System.out.print(cost[i][j]+"\t");
}
System.out.println();
}
System.out.println("Enter the source vertex: ");
int sv = in.nextInt();
dij(cost,dist,sv,n,path,visited);
printpath(sv,n,dist,path,visited );
System.out.println("\n********* ************** *********");
}

static void dij(int cost[][],int dist[],int src,int n, int path[],int visited[])
{
	int count=2,min,v=0;

	for(int i=1;i<=n;i++)
	{
		visited[i]=0;
		dist[i]=cost[src][i];
		if(cost[src][i]==999)
			path[i]=0;
		else
			path[i]=src;
	}
	visited[src]=1;

	while(count<=n)
	{
		min=999;
```

```
                    for(int w=1;w<=n;w++)
                    {
                        if(dist[w]<min && visited[w]==0)
                        {
                            min=dist[w];
                            v=w;
                        }
                    }
                    visited[v]=1;
                    count++;
                    for(int w=1;w<=n;w++)
                    {
                        if(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[],int visited[])
            {

                for(int w=1;w<=n;w++)
                {
                    if(visited[w]==1 && w!=src)
                    {
                        System.out.println("The short distance between: "+src+"-->"+w+" is:
"+dist[w]);
                        System.out.print("Path is: "+w+"-->");
```

```
                                int t=path[w];
                                while(t!=src)
                                {
                                        System.out.print(t+"-->");
                                        t=path[t];
                                }
                                System.out.print(src+"\n");
                        }
                }
        }
}
```