BHARATI VIDYAPEETH'S

INSTITUTE OF COMPUTER APPLICATIONS & MANAGEMENT

(Affiliated to Guru Gobind Singh Indraprastha University, Approved by AICTE, New Delhi)

DESIGN AND ANALYSIS OF ALGORITHMS LAB

(MCA-261)

Practical File

Submitted To:
Dr Vinay Kumar
(Assistant Professor)

Submitted By:
Abhijeet Rana (01811604422)
MCA 3rd Sem, Sec 1

S.No	Problem Description
1	Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n>5000 and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using any programming language, you are comfortable with, how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
2	Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n>5000, and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
3	Write a program to implement the 0/1 Knapsack and fractional Knapsack problem using (a) Dynamic Programming method (b) Greedy method. respectively.
4	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write a program to implement it.
5	Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.
6	Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
7	Write a program to (a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm. (b) Implement Travelling Sales Person problem using Dynamic programming.
8	Write a program to implement Subset sum problem. A subset of a given set S = {Sl,S2,.... ,Sn} of n positive integers whose SUM is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.
9	Implement the algorithm to determine all possible Hamiltonian Cycles in a connected undirected Graph G. of n vertices using backtracking principle. You may use any programming language. Show the output of your program
10	Write a program to implement backtracking algorithm for the N-queens problem.
11	Write a program to implement greedy algorithm for job sequencing with deadlines.
12	Write a program to implement Dynamic Programming algorithm for the Optimal

	Binary Search Tree Problem.
13	Write a program to fill magic square of size n = 3, 5, 7 and 9. Compute the magic number as well.
14	**Algorithms for Extracting Square Roots:** Babylonians Method. Write a program to implement the following algorithm to compute square root of a positive integer using Babylonians method. Start your calculation with 1 (any number is just as good). Call this Old. The new improved calculation is New = (Old + N/Old)/2. Then call this new value Old and repeat. You should print out all the calculated approximations starting with 1 and ending with the final value. The data types for Old and New should be float. For example, the first few lines of output for N=2 are: 1 1.5 1.416666 You should stop computing new guesses when you have two guesses in a row that agree up through the first four places after the decimal point
15	Write a program to compute product of any two large integers. A variable of type integer may not be able to hold a large integer. Use linked list to store such numbers to overcome the size limitation of an integer variable.

Q1 Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n>5000 and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using any programming language, you are comfortable with, how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

Sol

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

class QuickSort
{
   private int a[];

   public QuickSort(int[] a)
   {
          this.a = a;
   }

   public int partition ( int a[], int m, int p )
   {
          int v = a[m];
          int i = m;
          int j = p;
          do
          {
         while ( a[++ i] < v );
         while ( a[-- j] > v );
                    if ( i < j )
                              interchange ( a, i, j );
          } while ( i <= j );
          a[m] = a[j]; a[j] = v;
          return j;
   }

   public void qSort ( int p, int q )
   {
          int j;
          if ( p < q )
          {
                    j = partition ( a, p, q + 1 );
                    qSort ( p, j - 1 );
                    qSort ( j + 1, q );
          }
   }
```

```
    public void interchange ( int a[], int i, int j )
    {
        int t;
        t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}

public class QuickSortDemo
{
    public static void main(String[] args)
    {
        int n, a[], i;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the Size of an Array: ");
        n = input.nextInt();
        a = new int[n + 1];
        Random rn = new Random();
        System.out.println("System automatically generates numbers ");
        for ( i = 0; i < n; ++ i )
        {
            a[i] = rn.nextInt(n);
        }
        a[i] = 100000;
        QuickSort qSort = new QuickSort(a);

        System.out.println("Before Sort: ");
        for ( i = 0; i < n; ++ i )
        {
            System.out.print(a[i] + "\t");
        }

        int p = 0;
        int q = n - 1;

        qSort.qSort(p, q);
        System.out.println("\n\nAfter Sort: ");
        for ( i = 0; i < n; ++ i )
        {
            System.out.print(a[i] + "\t");
        }

        int step = 2000;
        double duration;
        System.out.println ( "\n\nN\tRepetitions\tTime\n" );
        for ( n = 5000; n < 50000; n += step )
        {
            a = new int[n + 1];
            qSort = new QuickSort(a);
            long repetitions = 0;
            long start = System.nanoTime();
```

```
                do
                {
                        repetitions ++;
                        for ( i = 0; i < n; ++ i )
                                a[i] = rn.nextInt(n);
                        a[i] = 100000;
                        qSort.qSort(0, n - 1);
                } while ( System.nanoTime() - start < 1000000000 );
               duration = ( ( double ) ( System.nanoTime() - start ) ) / 1000000000;
               duration /= repetitions;
               System.out.println ( n + "\t" + repetitions + "\t\t" + duration );
            }
        }
}
```

```
Enter the Size of an Array:
5
System automatically generates numbers
Before Sort:
4       4       4       2       2

After Sort:
2       2       4       4       4

N       Repetitions     Time

5000    2009            4.981173718267795E-4
7000    1491            6.711923541247485E-4
9000    1050            9.525220952380952E-4
11000   861             0.001162322531939605
13000   759             0.0013190814229249012
15000   590             0.0016955225423728815
17000   542             0.0018463118081180814
19000   454             0.002204237444933921
21000   436             0.0022963013761467892
23000   330             0.0030306309090909088
25000   266             0.003986977443609022
27000   190             0.005279111578947368
29000   170             0.005882624705882354
31000   142             0.0070856140845070425
33000   141             0.00712929219858156
35000   169             0.005923463905325444
37000   144             0.0069512736111111105
39000   135             0.007449207407407407
41000   119             0.008500223529411766
43000   106             0.009468248113207548
45000   119             0.008479005882352942
47000   111             0.009013182882882883
49000   110             0.009130097272727273
```

Q2 Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n>5000, and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate how the divide-and-conquer method works along with its time complexity

analysis: worst case, average case and best case.

Sol

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

class MergeSort
{
   private int a[];

   public MergeSort(int[] a)
   {
         this.a = a;
   }

   void merge ( int low, int mid, int high )
   {
         int b[] = new int[high + 1];
         int h = low;
         int i = low;
         int j = mid + 1;
         int k;

         while ( ( h <= mid ) && ( j <= high ) )
         {
                  if ( a[h] <= a[j] ) b[i ++] = a[h ++];
                  else b[i ++] = a[j ++];
         }
         if ( h > mid )
         {
                  for ( k = j; k <= high; ++ k ) b[i ++] = a[k];
         }
         else
         {

                  for ( k = h; k <= mid; ++ k ) b[i ++] = a[k];
         }

         for ( k = low; k <= high; ++ k ) a[k] = b[k];
   }

   void mergeSort ( int low, int high )
   {
         int mid;

         if ( low < high )
         {
                  mid = ( low + high ) / 2;

                  mergeSort ( low, mid );
```

```
mergeSort ( mid + 1, high );
```

```
                merge ( low, mid, high );
            }
    }
}

public class MergeSortDemo
{
    public static void main(String[] args)
    {
            int n, a[], i;
            Scanner input = new Scanner(System.in);
            System.out.println("Enter the Size of an Array: ");

            n = input.nextInt();
            a = new int[n + 1];

            Random rn = new Random();
            System.out.println("System automatically generates numbers ");
            for ( i = 0; i < n; ++ i )
            {
                a[i] = rn.nextInt(n);
            }
            a[i] = 100000;
            MergeSort mSort = new MergeSort(a);

            System.out.println("Before Sort: ");
            for ( i = 0; i < n; ++ i )
            {
                System.out.print(a[i] + "\t");
            }


            low = 0;
            int high = n - 1;

            mSort.mergeSort(low, high);

            System.out.println("\n\nAfter Sort: ");
            for ( i = 0; i < n; ++ i )
            {
                System.out.print(a[i] + "\t");
            }

            int step = 2000;
            double duration;
            System.out.println ( "\n\nN\tRepetitions\tTime\n" );
            for ( n = 5000; n < 50000; n += step )
            {
                a = new int[n + 1];
                mSort = new MergeSort(a);
                long repetitions = 0;
                long start = System.nanoTime();
```

```
                do
                {
                        repetitions ++;
                        for ( i = 0; i < n; ++ i )
                                a[i] = rn.nextInt(n);
                        a[i] = 100000; //Sentinel value
                        mSort.mergeSort(0, n - 1);
                } while ( System.nanoTime() - start < 1000000000 );
               duration = ( ( double ) ( System.nanoTime() - start ) ) / 1000000000;
                duration /= repetitions;
                System.out.println ( n + "\t" + repetitions + "\t\t" + duration );

            }
        }
}
```

```
Enter the Size of an Array:
5
System automatically generates numbers
Before Sort:
3       4       3       0       0

After Sort:
0       0       3       3       4

N       Repetitions     Time

5000    67              0.014959305970149253
7000    66              0.015226736363636365
9000    43              0.023410646511627907
11000   32              0.03137559375
13000   25              0.04082402
15000   20              0.051511885
17000   16              0.06448805625
19000   13              0.08025479230769231
21000   11              0.09635564545454546
23000   9               0.1112634
25000   8               0.1286064625
27000   7               0.14653311428571428
29000   6               0.1710956166666664
31000   5               0.20473210000000003
33000   5               0.21543972
35000   5               0.23813866
37000   4               0.26532595
39000   4               0.293747525
41000   4               0.3224128
43000   3               0.3619551666666666
45000   3               0.39963566666666667
47000   3               0.4432737
49000   3               0.46206243333333336
```

Q3 Write a program to implement the 0/1 Knapsack and fractional Knapsack problem using
(a) Dynamic Programming method

Sol

```
import java.util.Scanner;
class DKnapsack
{
int n;
int c;
int p[];
int w[];
int v[][];

public DKnapsack(int n, int c, int[] p, int[] w)
{
super();
this.n = n;
this.c = c;
this.p = p;
this.w = w;
this.v = new int[n + 1][c + 1];
    }

        void compute()
        {
            for ( int i = 0; i <= n; ++ i)
            {
                for ( int j = 0; j <= c; ++ j)
                {
                    if ( i == 0 || j == 0 )
                    {
                        v[i][j] = 0;
                    }
                    else if ( j - w[i] >= 0 )
                    {
    v[i][j] = max ( v[i - 1][j], p[i] + v[i - 1][j - w[i]]);
                    }
                    else if ( j - w[i] < 0 )
                    {
                        v[i][j] = v[i - 1][j];
                    }
                }
            }


            System.out.println("Optimal Solution: " + v[n][c]);
            traceback();
        }
```

```
  void traceback()
  {
        System.out.println("The objects picked up into knapsack are:");

        int i = n;
        int j = c;

        while( i > 0)
        {
                if(v[i][j] != v[i-1][j])
                {
                        System.out.print(i + " ");
                        j = j - w[i];
                        i--;
                }
                else
                {
                        i--;
                }
        }
  }


  private int max(int i, int j)
  {
        if ( i > j ) return i;
        else return j;
  }
}
public class KpDynamic
{
  public static void main(String[] args)
  {
        int n;
        int c;

        Scanner input = new Scanner(System.in);
        System.out.println("Enter number of objects");
        n = input.nextInt();

        int[] p = new int[n+1];
        int[] w = new int[n+1];
        int i;



        System.out.println("Enter capacity of Knapsack");
        c = input.nextInt();
```

```
System.out.println("Enter profit for each " + n + " objects");
```

```
        for ( i = 1; i <= n; i ++)
                p[i] = input.nextInt();

        System.out.println("Enter weight for each " + n + " objects");

        for ( i = 1; i <= n; i ++)
                w[i] = input.nextInt();

        DKnapsack dk = new DKnapsack(n, c, p, w);
        dk.compute();
    }
}
```

```
C:\Users\HP\Desktop\daa>java KpDynamic
Enter number of objects
5
Enter capacity of Knapsack
20
Enter profit for each 5 objects
3
4
5
8
10
Enter weight for each 5 objects
2
3
4
5
9
Optimal Solution: 26
The objects picked up into knapsack are:
5 4 3 1
```

(b) Greedy method.
Sol

```
import java.util.Scanner;

class GKnapsack
{
  int n;
  double c;
  double p[];
  double w[];

  public GKnapsack(int n, double c, double[] p, double[] w)
  {
        super();
        this.n = n;
        this.c = c;
        this.p = p;
        this.w = w;
```

```
}

void compute()
{
	int i;
	double[] x= new double[n+1];

	for (i=0; i<n; i++)
	{
		x[i] = 0.0;
	}

	double rc = c;

	for(i=0; i<n; i++)
	{
		if(w[i] > rc) break;
		x[i] = 1;
		rc = rc - w[i];
	}

	if(i<=n)
	{
		x[i] = rc/w[i];
	}

	double netProfit = 0.0;



	for ( i = 0; i < n; ++ i)
	{
		if ( x[i] > 0.0)
		{
			netProfit = netProfit + x[i] * p[i];

		}

	}

	System.out.println("Net Profit: " + netProfit);
	System.out.println("The objects picked up into knapsack are:");

	for ( i = 0; i < n; ++ i)
	{
		System.out.println(x[i] + " ");

	}
}
```

```
}

public class KpGreedy
{
  public static void main(String[] args)
  {
        int n;
        double c;

        Scanner input = new Scanner(System.in);
        System.out.println("Enter number of objects");
        n = input.nextInt();

        double[] p = new double[n+1];
        double[] w = new double[n+1];
        int i;

        System.out.println("Enter capacity of Knapsack");
        c = input.nextDouble();

        System.out.println("Enter profit for each " + n + " objects");

        for ( i = 0; i < n; i ++)
                p[i] = input.nextDouble();

        System.out.println("Enter weight for each " + n + " objects");



        for ( i = 0; i < n; i ++)
                w[i] = input.nextDouble();

        GKnapsack gk = new GKnapsack(n, c, p, w);
        gk.compute();
  }
}
```

```
C:\Users\HP\Desktop\daa>java KpGreedy
Enter number of objects
7
Enter capacity of Knapsack
15
Enter profit for each 7 objects
6
10
18
15
3
5
7
Enter weight for each 7 objects
1
2
4
5
1
3
7
Net Profit: 55.333333333333336
The objects picked up into knapsack are:
1.0
1.0
1.0
1.0
1.0
0.6666666666666666
0.0
```

Q4 From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write a program to implement it.

Sol

```
import java.util.Arrays;
import java.util.Scanner;

public class Dijkstra
{

   static int n,cost[][],i,j,u,dist[],src;
   void dij(int src,int cost[][],int dist[],int n)
   {
         int visited[],min;
         visited=new int[n];

         for(i=0;i<n;i++)
         {
               visited[i]=0;
               dist[i]=cost[src][i];
```

```
        }

        visited[src]=1;
        dist[src]=0;

        for(i=0;i<n;i++)
        {
                if(i==src) continue;
                min=999;

                for(j=0;j<n;j++)
                        if((visited[j]==0)&&(min>dist[j]))
                        {
                                min=dist[j];
                                u=j;
                        }
                        visited[u]=1;

                for(j=0;j<n;j++)
                        if(visited[j]==0)
                        {
                                if(dist[j]>dist[u]+cost[u][j])
          dist[j]=dist[u]+cost[u][j];
                        }
        }
}


Public static void main(String[] args)
{
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        n=sc.nextInt();

        System.out.println("Enter the matrix");
        cost=new int[n][n];
        dist=new int[n];

        Arrays.fill(dist,0);

        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        cost[i][j]=sc.nextInt();

        System.out.println("Enter the source vertex");
        src=sc.nextInt();

        new Dijkstra().dij(src, cost, dist, n);
```

```
                System.out.println("Shortest path from "+src+" to all other vertices");
                for(i=0;i<n;i++)
                        System.out.println("To " +i+" is "+dist[i]);
        }
}
```

```
Enter the number of vertices
4
Enter the matrix
0 15 10 9999
9999 0 15 9999
20 9999 0 20
9999 10 9999 0
Enter the source vertex
2
Shortest path from 2 to all other vertices
To 0 is 20
To 1 is 30
To 2 is 0
To 3 is 20
```

Q5 Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.

Sol

```
import java.util.Scanner;

public class Kruskals
{
  static int parent[],cost[][], mincost,n,i,j,ne,a,b,min,u,v;


  public void kruskal(int n,int[][] cost)
  {
        ne=1;
        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;
                                }
                }

                u=find(u);
```

```
                v=find(v);

                if(v!=u)
                {
                                System.out.println(
              ne+"edge("+a+","+b+")="+min);
                        ne=ne+1;
                        mincost=mincost+min;
                        uni(u,v);
                }
                cost[a][b]=cost[b][a]=999;
        }
        System.out.println("The minimum cost of spanning tree is "+mincost);
}

public int find (int i)
{


        while (parent[i] != 0)
                i=parent[i];
        return i;
}

        public void uni(int i,int j)
        {
parent[j]=i;
        }


        public static void main(String[] args)
{

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number of vertices\n");
        n=sc.nextInt();

        int cost[][]= new int [n+1][n+1];

        parent=new int[n+1];

        System.out.println("Enter the cost matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        cost[i][j]=sc.nextInt();
                        if(cost[i][j]==0)
```

```
                        cost[i][j]=999;
                }
            }

            Kruskals k = new Kruskals();
            k.kruskal(n,cost);


        }
}
```

```
C:\Users\HP\Desktop\daa>java Kruskals
Enter the number of vertices

7
Enter the cost matrix

0 28 999 999 999 10 999
28 0 16 999 999 999 14
999 16 0 12 999 999 999
999 999 12 0 22 999 18
999 999 999 22 0 25 24
10 999 999 999 25 999 999
999 14 999 18 24 999 999
1edge(1,6)=10
2edge(3,4)=12
3edge(2,7)=14
4edge(2,3)=16
5edge(4,5)=22
6edge(5,6)=25
The minimum cost of spanning tree is 99
```

Q6 Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

Sol

```
import java.util.Scanner;

public class Prims
{
  static int mincost=0,n,i,j,ne,a=0,b=0,min,u = 0,v=0;

  public void prim(int n,int[][] cost)
  {

        int[] visited = new int[n+1];

        for(i=2;i<=n;i++)
                visited[i]=0;

        visited[1]=1;
        ne=1;
        while(ne<n)
```

```
        {
                min=999;
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if(cost[i][j]<min)
                                {
                                        if(visited[i]==0)
                  continue;
                                        else
                                        {
                   min=cost[i][j];
                                 a=u=i;
                                 b=v=j;
                                        }
                                }

                        }
                }
                if(visited[u]==0||visited[v]==0)
                {

                         System.out.println((ne)+"
                       edge("+a+","+b+")="+min);
                         ne=ne+1;

                         mincost=mincost+min;
                        visited[v]=1;

                }
                cost[a][b]=cost[b][a]=999;
        }
        System.out.println("The minimum cost of spanning tree is "+mincost);

}

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

  System.out.println("Enter the number of vertices\n");
  n=sc.nextInt();

  int cost[][]= new int [n+1][n+1];
  System.out.println("Enter the cost matrix\n");
  for(i=1;i<=n;i++)
  {
                for(j=1;j<=n;j++)
                {
                        cost[i][j]=sc.nextInt();
                        if(cost[i][j]==0)
```

```
cost[i][j]=999;
```

```
                    }
      }

      Prims p = new Prims();
      p.prim(n,cost);
    }

      }
```

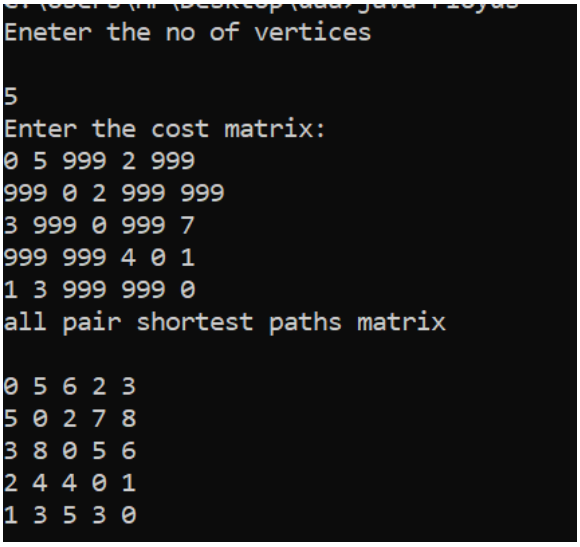
```
C:\Users\HP\Desktop\daa>java q6
Enter the number of vertices

7
Enter the cost matrix

0 28 999 999 999 10 999
28 0 16 999 999 999 14
999 16 0 12 999 999 999
999 999 12 0 22 999 18
999 999 999 22 0 25 24
10 999 999 999 25 999 999
999 14 999 18 24 999 999
1edge(1,6)=10
2edge(6,5)=25
3edge(5,4)=22
4edge(4,3)=12
5edge(3,2)=16
6edge(2,7)=14
The minimum cost of spanning tree is 99
```

Q7 Write a program to

a) **Implement All-Pairs Shortest Paths problem using Floyd's algorithm. Sol**

```
import java.util.*;

public class Floyds
{
  static int n,i,j,k;

  public void floyd(int n , int[][] cost)
  {

        for(k=1;k<=n;k++)
        {
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                        cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]);

                        }
                }
        }
```

```
        System.out.println("all pair shortest paths matrix \n");

                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                System.out.print(cost[i][j]+" ");
                        }
                        System.out.println();
                }

          }

          public int min(int i,int j)
                {
                 if(i<j)
                        return i;
                 else
                         return j;
        }




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

          System.out.println("Eneter the no of vertices\n");
          n=sc.nextInt();

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

          Floyds f = new Floyds();
          f.floyd(n,cost);

        }
}
```



```
Eneter the no of vertices

5
Enter the cost matrix:
0 5 999 2 999
999 0 2 999 999
3 999 0 999 7
999 999 4 0 1
1 3 999 999 0
all pair shortest paths matrix

0 5 6 2 3
5 0 2 7 8
3 8 0 5 6
2 4 4 0 1
1 3 5 3 0
```

(b) Implement Travelling Sales Person problem using Dynamic programming.

Sol

```
import java.util.Scanner;

public class Tsp
{
  static int cost[][];

  public int tsp(int[] path,int start,int n)
  {
        int i,j,k,ccost;

        int[] mintour=new int[n+1];
        int[] temp=new int[n+1];


        if(start==n-1)
         return cost[path[n-1]][path[n]]+cost[path[n]][1];

        int mincost=999;

        for(i=start+1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                 temp[j]=path[j];

                        temp[start+1]=path[i];
                         temp[i]=path[start+1];

if(cost[path[start]][path[i]]+(ccost=tsp(temp,start+1,n))<mincost)
```

```
                                {
 mincost=cost[path[start]][path[i]]+ccost;

                                           for(k=1;k<=n;k++)
                                                      mintour[k]=temp[k];

                                }
          }

          for(i=1;i<=n;i++)
                     path[i]=mintour[i];

          return mincost;

          }



public static void main(String[] args)
 {
  int mincost,n,i,j;
  Scanner s = new Scanner(System.in);

  System.out.println("enter the no of cities");
  n=s.nextInt();

  int path[] =new int[n+1];
  cost = new int[n+1][n+1];


  System.out.println("Enter the cost matrix");
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
          cost[i][j]=s.nextInt();
          for(i=1;i<=n;i++)
                     path[i]=i;

  Tsp obj = new Tsp();

  mincost=obj.tsp(path,1,n);

  System.out.println("tsp tour");
  for(i=1;i<=n;i++)
          System.out.print(path[i] + "--->");

  System.out.println("1");
  System.out.println("Tourcost=" + mincost);
 }
```

```
}
```

```
C:\Users\HP\Desktop\daa>java Tsp
enter the no of cities
4
Enter the cost matrix
999 1 3 6
1 999 2 3
3 2 999 1
6 3 1 999
tsp tour
1--->2--->4--->3--->1
Tourcost=8
```

Q8 Write a program to implement Subset sum problem. A subset of a given set S = {Sl,S2,. ,Sn} of n positive integers whose SUM is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

Sol

```
import java.util.Scanner;
public class Subset
{
 static int w[],x[],flag,sum,n,total,i,s,k,r;
  public void sumOfSubset(int s,int k,int r)
  {
   x[k]=1;
   if(s+w[k]==sum)
   {
        System.out.println("The subset: ");
        for(i=1;i<=k;i++)
        {
         flag=1;
         if(x[i]==1)
         {
                System.out.println(w[i]);
         }
        }
   }

   else if(s+w[k]+w[k+1]<=sum)
   {
        sumOfSubset(s+w[k],k+1,r-w[k]);
   }
   if(s+r-w[k]>=sum && s+w[k+1]<=sum)
```

```
  {
        x[k]=0;
        sumOfSubset(s,k+1,r-w[k]);
  }
}
public static void main(String args[])
{
  Scanner s=new Scanner(System.in);
  System.out.println("Enter the number of elements");
  n=s.nextInt();
  w=new int[n+1];
  x=new int[n+1];
  System.out.println("Enter the elements");
  for(int i=1;i<=n;i++)
  {
        w[i]=s.nextInt();
        total=total+w[i];
  }
  System.out.println("Enter the sum");
  sum=s.nextInt();
  if(total<sum)
  {
        System.out.println("subset is not possible");
        System.exit(0);
  }
  Subset ss = new Subset();
  ss.sumOfSubset(0,1,total);
  if(flag==0)
  {
        System.out.println("Subset not possible");
  }
}
```

```
C:\Users\HP\Desktop\daa>java Subset
Enter the number of elements
7
Enter the elements
1 2 3 4 5 6 7
Enter the sum
8
The subset:
1
2
5
The subset:
1
3
4
The subset:
1
7
The subset:
2
6
The subset:
3
5
```

Q9 Implement the algorithm to determine all possible Hamiltonian Cycles in a connected undirected Graph G. of n vertices using backtracking principle. You may use any programming language. Show the output of your program

Sol

```
import java.util.Scanner;

class HamiltonianCycles
{
  int n,g[][],x[],i,j,k;

  public HamiltonianCycles(int n,int[][] g)
  {
        this.n=n;
        this.g=g;
        this.x = new int[n+1];
        x[1]=1;

  }


  public void hamiltonian(int k)
  {

        while(true)
```

```
	{
		nextValue(k);

		if(x[k] == 0)
		{
			return;

		}

		if(k==n)
		{
			System.out.println("Solution :");
			for(int i=1;i<=n;i++)
			{
				System.out.print(x[i] + "\t");

			}

			System.out.println(1);
		}
		else
		{
			hamiltonian(k+1);
		}


	}
}


public void nextValue(int k)
{

	while(true)
	{
		x[k] = (x[k]+1)%(n+1);

		if(x[k]==0)
		{
			return;
		}

		if(g[x[k-1]][x[k]] != 0)
		{
			for(j=1;j<=k-1;j++)
			{
				if(x[j] == x[k])
				{
					break;
```

```
}
```

```
                    }

                    if(j==k)
                    {
if((k<n) || ((k==n) && (g[x[n]][x[1]] != 0 )))
                         {
                              return;
                         }
                    }

               }


          }
     }


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

          System.out.println("Enter the number of vertices :");
          n=s.nextInt();

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

          HamiltonianCycles ham = new HamiltonianCycles(n,g);

          ham.hamiltonian(2);


     }
   }
```

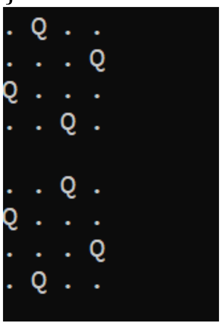
```
Enter the number of vertices :
4
Enter the matrix :
0 4 1 3
4 0 2 1
1 2 0 5
3 1 5 0
Solution :
1       2       3       4       1
Solution :
1       2       4       3       1
Solution :
1       3       2       4       1
Solution :
1       3       4       2       1
Solution :
1       4       2       3       1
Solution :
1       4       3       2       1
```

Q10 Write a program to implement backtracking algorithm for the N-queens problem.

Sol

```
public class NQueens {
    public static void solveNQueens(int n) {
        int[] queens = new int[n];
        placeQueens(queens, 0, n);
    }

    public static void placeQueens(int[] queens, int row, int n) {
        if (row == n) {
            printQueens(queens);
            return;
        }

        for (int col = 0; col < n; col++)
                        {
            if (isSafe(queens, row, col)) {
                queens[row] = col;
                placeQueens(queens, row + 1, n);
            }
        }
    }

    public static boolean isSafe(int[] queens, int row, int col) {
        for (int i = 0; i < row; i++) {
            if (queens[i] == col || Math.abs(queens[i] - col) == Math.abs(i - row)) {
                return false;
            }
        }
        return true;
    }
```

```
    public static void printQueens(int[] queens) {
        int n = queens.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (queens[i] == j) {
                    System.out.print("Q ");
                } else {
                    System.out.print(". ");
                }
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n = 4;
        solveNQueens(n);
    }
}
```



Q11 Write a program to implement greedy algorithm for job sequencing with deadlines.

Sol

```
import java.util.Arrays;

class Job {
    char id;
    int deadline;
    int profit;

    public Job(char id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
}

public class JobSequencingGreedy {

    public static void jobSequence(Job[] jobs) {
```

```
    // Sort the jobs in decreasing order of profit
    Arrays.sort(jobs, (a, b) -> Integer.compare(b.profit, a.profit));

    int maxDeadline = 0;
    for (Job job : jobs) {
        if (job.deadline > maxDeadline) {
            maxDeadline = job.deadline;
        }
    }

    char[] result = new char[maxDeadline];
    boolean[] slot = new boolean[maxDeadline];

    // Initialize slots to be empty
    Arrays.fill(slot, false);

    for (int i = 0; i < jobs.length; i++) {
        for (int j = Math.min(maxDeadline - 1, jobs[i].deadline - 1); j >= 0; j--) {
            if (!slot[j]) {
                result[j] = jobs[i].id;
                slot[j] = true;
                break;
            }
        }
    }

    // Print the job sequence and total profit
    int totalProfit = 0;
    System.out.print("Job Sequence: ");
    for (char c : result) {
        if (c != '\0') {
            System.out.print(c + " ");
            totalProfit += findJobById(jobs, c).profit;
        }
    }
    System.out.println("\nTotal Profit: " + totalProfit);
}

public static Job findJobById(Job[] jobs, char id) {
    for (Job job : jobs) {
        if (job.id == id) {
            return job;
        }
    }
    return null;
}

public static void main(String[] args) {
    Job[] jobs = {
        new Job('A', 2, 100),
```

```
            new Job('B', 1, 19),
            new Job('C', 2, 27),
            new Job('D', 1, 25),
            new Job('E', 3, 15)
        };

        System.out.println("Given Jobs:");
        for (Job job : jobs) {
            System.out.println("Job " + job.id + " (Deadline: " + job.deadline + ", Profit: " +
job.profit + ")");
        }

        System.out.println("\nJob Sequence and Total Profit:");
        jobSequence(jobs);
    }
}
```

```
Given Jobs:
Job A (Deadline: 2, Profit: 100)
Job B (Deadline: 1, Profit: 19)
Job C (Deadline: 2, Profit: 27)
Job D (Deadline: 1, Profit: 25)
Job E (Deadline: 3, Profit: 15)

Job Sequence and Total Profit:
Job Sequence: C A E
Total Profit: 142
```

Q12 Write a program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.

Sol

```
public class OptimalBinarySearchTree {

    public static int optimalBST(int[] keys, double[] probabilities) {
        int n = keys.length;
        double[][] cost = new double[n][n];

        // Initialize cost matrix
        for (int i = 0; i < n; i++) {
            cost[i][i] = probabilities[i];
        }

        // Build the cost matrix
        for (int length = 2; length <= n; length++) {
            for (int i = 0; i <= n - length; i++) {
                int j = i + length - 1;
                cost[i][j] = Double.MAX_VALUE;

                double sum = 0;
                for (int k = i; k <= j; k++) {
                    sum += probabilities[k];
                }
```

```
                for (int k = i; k <= j; k++) {
                    double currentCost = (k > i ? cost[i][k - 1] : 0) +
                            (k < j ? cost[k + 1][j] : 0) + sum;
                    cost[i][j] = Math.min(cost[i][j], currentCost);
                }
            }
        }

        return (int) cost[0][n - 1];
    }

    public static void main(String[] args) {
        int[] keys = {10, 12, 20, 25};
        double[] probabilities = {0.34, 0.33, 0.18, 0.15};

        int minCost = optimalBST(keys, probabilities);
        System.out.println("Minimum Average Search Time: " + minCost);
    }
}
```

```
C:\Users\HP\Desktop>java OptimalBinarySearchTree
Minimum Average Search Time: 1
```

Q13 Write a program to fill magic square of size n = 3, 5, 7 and 9. Compute the magic number as well.

Sol

```
public class MagicSquare {
    public static void generateOddMagicSquare(int n) {
        int[][] magicSquare = new int[n][n];

        int row = 0;
        int col = n / 2;
        int num = 1;

        while (num <= n * n) {
            magicSquare[row][col] = num;

            num++;

            int nextRow = (row - 1 + n) % n;
            int nextCol = (col + 1) % n;

            if (magicSquare[nextRow][nextCol] == 0) {
                row = nextRow;
                col = nextCol;
            } else {
                row = (row + 1) % n;
            }
        }
```

```
        printMagicSquare(magicSquare);
    }

    public static void printMagicSquare(int[][] square) {
        int n = square.length;

        System.out.println("Magic Square of size " + n + " with magic number " + n * (n * n +
1) / 2);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(square[i][j] + "\t");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[] sizes = {3, 5, 7, 9};

        for (int n : sizes) {
            generateOddMagicSquare(n);
            System.out.println();
        }
    }
}
```

```
C:\Users\HP\Desktop\daa>java MagicSquare
Magic Square of size 3 with magic number 15
8       1       6
3       5       7
4       9       2

Magic Square of size 5 with magic number 65
17      24      1       8       15
23      5       7       14      16
4       6       13      20      22
10      12      19      21      3
11      18      25      2       9

Magic Square of size 7 with magic number 175
30      39      48      1       10      19      28
38      47      7       9       18      27      29
46      6       8       17      26      35      37
5       14      16      25      34      36      45
13      15      24      33      42      44      4
21      23      32      41      43      3       12
22      31      40      49      2       11      20

Magic Square of size 9 with magic number 369
47      58      69      80      1       12      23      34      45
57      68      79      9       11      22      33      44      46
67      78      8       10      21      32      43      54      56
77      7       18      20      31      42      53      55      66
6       17      19      30      41      52      63      65      76
16      27      29      40      51      62      64      75      5
26      28      39      50      61      72      74      4       15
36      38      49      60      71      73      3       14      25
37      48      59      70      81      2       13      24      35
```

Q14 Algorithms for Extracting Square Roots: Babylonians Method.
Write a program to implement the following algorithm to compute square root of a positive integer using Babylonians method.
Start your calculation with 1 (any number is just as good). Call this Old. The new improved calculation is New = (Old + N/Old)/2. Then call this newvalue Old and repeat. You should print out all the calculated approximations starting with 1 and ending with the final value. The data types for Old and New should be float. For example, the first few lines of output for N=2 are:
1
1.5
1.416666

You should stop computing new guesses when you have two guesses in a row that agree up through the first four places after the decimal point
Sol

```
public class BabylonianSquareRoot {
    public static void main(String[] args) {
        int N = 2;
```

```
        float Old = 1.0f;
        float New;
        float epsilon = 0.0001f;

        System.out.println(Old);

        do {
            New = (Old + N / Old) / 2;
            System.out.println(New);
            if (Math.abs(New - Old) < epsilon) {

                break;
            }
            Old = New;
        } while (true);
    }
}
```

```
C:\Users\HP\Desktop\daa>javac BabylonianSquareRoot.java

C:\Users\HP\Desktop\daa>java BabylonianSquareRoot
1.0
1.5
1.4166667
1.4142157
1.4142135
```

Q15 Write a program to compute product of any two large integers. A variable of type integer may not be able to hold a large integer. Use linked list to store such numbers to overcome the size limitation of an integer variable.

Sol

```
import java.util.LinkedList;

public class LargeIntegerMultiplication {
    public static LinkedList<Integer> multiply(LinkedList<Integer> num1,
LinkedList<Integer> num2) {
        int m = num1.size();
        int n = num2.size();
        int[] result = new int[m + n];

        for (int i = m - 1; i >= 0; i--) {
            int carry = 0;
            int digit1 = num1.get(i);
            int k = m - i - 1;

            for (int j = n - 1; j >= 0; j--) {
                int digit2 = num2.get(j);
                int product = digit1 * digit2 + carry + result[k];
                carry = product / 10;
                result[k] = product % 10;
```

```
                k++;
            }

            if (carry > 0) {
                result[k] += carry;
            }
        }

        LinkedList<Integer> resultLinkedList = new LinkedList<>();
        for (int digit : result) {
            resultLinkedList.add(digit);
        }

        return resultLinkedList;
    }

    public static void printLargeInteger(LinkedList<Integer> num) {
        for (int digit : num) {
            System.out.print(digit);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedList<Integer> num1 = new LinkedList<>();
        LinkedList<Integer> num2 = new LinkedList<>();

        // Initialize num1 and num2 with the large integers
        num1.add(9);
        num1.add(8);
        num1.add(7);
        num1.add(6);

        num2.add(5);
        num2.add(4);
        num2.add(3);

        LinkedList<Integer> result = multiply(num1, num2);

        System.out.print("Product: ");
        printLargeInteger(result);
    }
}
```