JGi JAIN DEEMED-TO-BE UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Design and Analysis of Algorithms Lab

4th Semester

Submitted By

Name: Soayeb Ahamed Nayon

USN:20BTRCO043

FACULTY OF ENGINEERING AND TECHNOLOGY

JAIN DEEMED-TO-BE UNIVERSITY

Laboratory Certificate

This is to certify That Mr /Ms ....has satisfactorily completed the course of experiments in practical **Design and Analysis of Algorithms Lab** (18CS1401L) presented by the Jain (Deemed-to-be University) six semester course in laboratory of this college in the year 2021-22.

Date:

Name of the Candidate:..

USN:...................................

Date of Practical Examination..............................

Valued	
Examiner-1	
Examiner-2	

Signature of the Teacher
Incharge of the Batch

Contents

TO CREATE n STUDENTS OBJECTS

Experiment - 1A

1. **To create nStudent objects and to implement the Stack using arrays**

   a. **Create a Java class called Student with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone no. Write a Java program to create nStudent objects and print the USN, Name, Branch, and Phone no. of these objects with suitable headings.**

Aim : The aim of this program is to create a class and make objects of that class to print the

details of a student.

Algorithm :

- **//Input: Values for USN, name, branch and phone no**
- **//Output: Displaying the details of n student objects**
- **//Steps:**
  - **class "student" is created**
  - **Declare variables USN, Name, Branch, and Phone no.**
  - **a constructor of Student class is created to initialize these variables**
  - **a function "display_details" is created that prints these details like usn, name, branch and phone no Multiple objects of "student" class calls the function "display_details" to print the details contained in student class.**

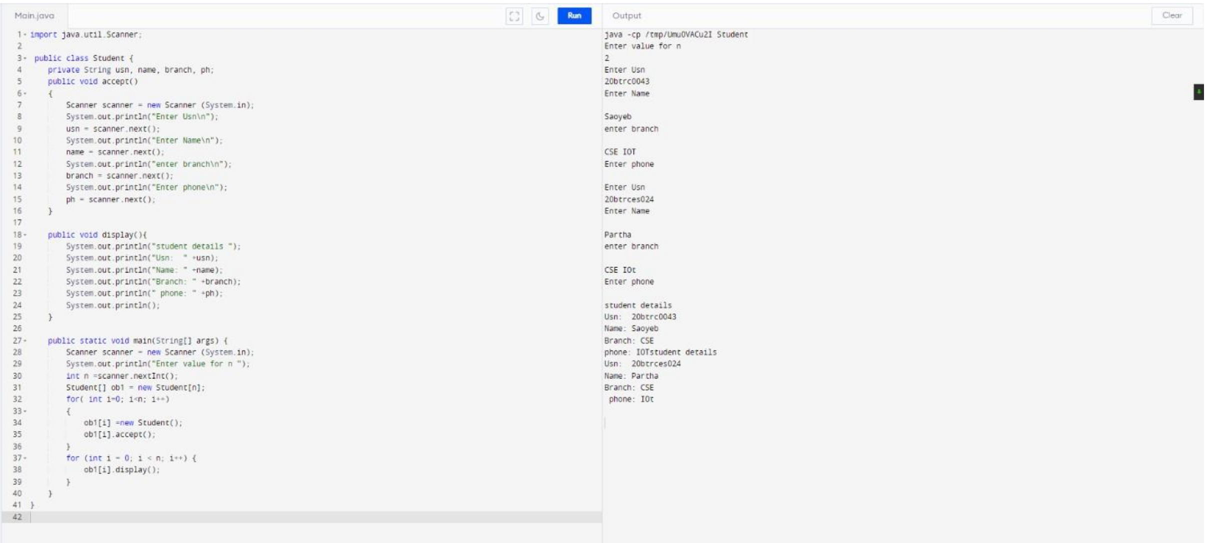
Program :-

```
import java.util.Scanner;

public class student {

 String USN;
```

```
 String Name;

 String branch;

 String phone;

              void insertRecord(String reg, String name, String brnch, String
ph) {

          USN = reg;

          Name = name;

          branch = brnch;

          phone = ph; }

     void displayRecord() {

          System.out.println(USN + " " + Name + " " + branch + " " +
phone); }

      public static void main(String args[]) {

          student s[] = new student[100];

          Scanner sc = new Scanner(System.in);

          System.out.println("enter the number of students");

          int n = sc.nextInt();

          for (int i = 0; i < n; i++)

                   s[i] = new student();

          for (int j = 0; j < n; j++) {

                   System.out.println("enter the usn,name,branch,phone");
```

```
            String USN = sc.next();
            String Name = sc.next();
            String branch = sc.next();
            String phone = sc.next();
            s[j].insertRecord(USN, Name, branch, phone); }
    for (int m = 0; m < n; m++) {
            s[m].displayRecord();
    }}}
```

OUTPUT :-



To Implement The Stack Using Arrays

1 (b) Write a Java program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working.

Aim :

The aim of this program is to create stack using arrays and perform all the stack related functions like pushing elements, popping elements and displaying the contents of stack. Stack is abstract data type which demonstrates Last in first out (LIFO) behavior. We will implement same behavior using Array.

Algorithm:

```
push(int pushedElemnet)  {
          if(stack is not full)   {
                   Top++;
                   Array[top]=pushedElement;     }
         else {
                  Stack is full  }}
```

- **A function created for popping the elements from stack :**

```
Pop(){
if(stack is not empty) {
                 A=top;
                 Top--; }
      else {
        Stack is empty  }}
```

- **A function is created for displaying the elements in the stack:**

```
printElemnts()  {
       if(top>=0)   {
       for(i=0;i<=top;i++)  {
```

Print all elements of array } }

- **A boolen function is created to cheak whether stack is empty or full :**

```
Boolen isFull () {
    return (size-1==top)  }
Boolen isEmpty()  {
return (top==-1)
}
```

Program :-

```
import java.util.Scanner;
public class Program1b {
    static int[] integerStack;
    static int top = -1;
    public static void main(String[] args) {
    System.out.println("Enter stack size:");
    Scanner scanner = new Scanner(System.in);
    int size = scanner.nextInt();
    integerStack = new int[size];
    System.out.println("Stack operations:");
    System.out.println("1. Push");
    System.out.println("2. Pop");
    System.out.println("3. Display");
```

```
System.out.println("4. Exit");
System.out.println("Enter your choice.");
int choice = scanner.nextInt();
while (choice != 4) {
    if (choice == 1) {
        System.out.println("Enter element to push:");
        int element = scanner.nextInt();
        if (top == size - 1)
            System.out.println("stack is full"); else {
            top = top + 1;
            integerStack[top] = element;}
    } else if (choice == 2) {
        if (top == -1) {
            System.out.println("stack is empty."); } else {
            System.out.println("Popped element is :" +
integerStack[top]);
            top = top - 1; }
    } else if (choice == 3) {
if (top == -1)
          System.out.println("stack is empty");
    else {
```

```
System.out.println("stack elementa are :");

for (int i = top; i >= 0; i--)

System.out.println(integerStack[i]);

        }

        }

else

        System.out.println("Enter correct choice.");

        System.out.println("Stack operations:");

        System.out.println("1. Push");

        System.out.println("2. Pop");

        System.out.println("3. Display");

        System.out.println("4. Exit");

        System.out.println("Enter your choice.");

        choice = scanner.nextInt();}}

        }
```

OUTPUT :-

```
lab2 [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (Jun 17, 202
Enter correct choice.
Stack operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice.
2
stack is empty.
Stack operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice.
1
Enter element to push:
4
Stack operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice.
3
stack elementa are :
4
Stack operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice.
```

Multithreading function

EXPERIMENT - 2

2(a) Implements a multi-thread application

a. Write a Java program to read two integers a and b. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.

Aim :

Understanding the concepts of exception handling in java

Algorithm :

- **// Input: values of two operand i.e a and b**
- **// Output: a) answer displayed when b != 0**

Arithmetic exception raised and error message displayed when b = 0.

Steps :

- **A class is created containing the main method**
- **Two variables are declared i.e. a and b**
- **Input is obtained from console**

Scanner sc = new Scanner(System.in);

a = sc.nextInt();

b = sc.nextInt();

1) **The code to calculate division is kept under**

try block try

```
{
        System.out.println(a/b);
}
```

2) **The arethemetic exception raised when**

b=0 is handled in catck block that follows try block

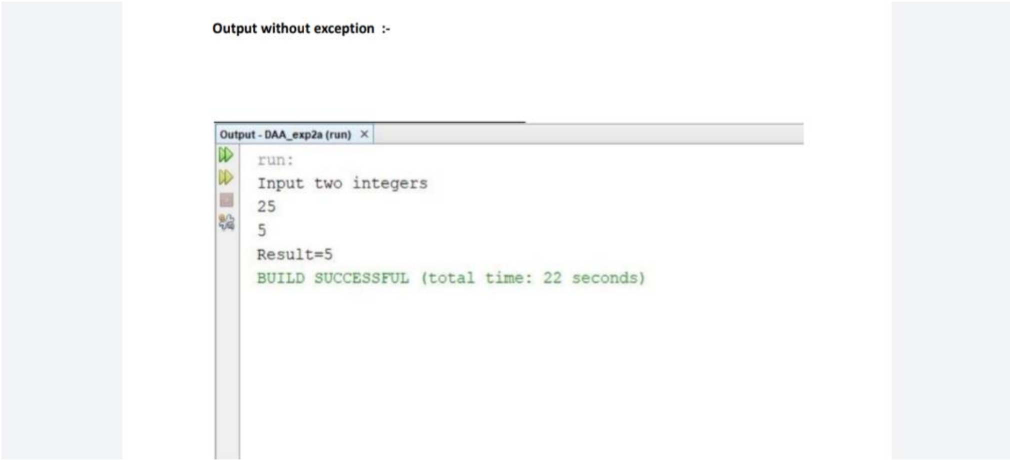
```
catch(ArithmeticException e)
   {
            e.printStackTrace();
    }
```

Program :-

```
import java.util.Scanner;
class division
{
public static void main(String[] args)
```

```
{
int a,b,result;
Scanner input =new Scanner(System.in);
System.out.println("Input two integers");
a=input.nextInt();
b=input.nextInt();
try
{
result=a/b;
System.out.println("Result="+result);
}
catch(ArithmeticException e)
{
System.out.println("exception caught: Divide by zero error"+e);
}
}
}
```

OUTPUT :-



OUTPUT WITH EXECUTION :-

a. 2(b). Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.

Aim :

To understand the concepts of multithreading by creating three threads that perform

different tasks when one thread is suspended for some time duration.

Algorithm :

- **// Input: Random number**
- **//Output: square and cube of the number**

Steps: Three threads are created.

- **Three classes RandomNumber, SquareGenerator and CubeGenerator are created**
- **Class RandomNumber generates an integer using random number generator and prints the integer**

  with thread t1

- **Next class SquareGenerator is called to generate square of the number and print it with thread t2.**
- **At last class CubeGenerator is called to generate cube of the number and print it with thread t3.**

Program :

```
import java.util.Random;
class Square extends Thread
{
 int x;
 Square(int n) {
 x = n; }
 public void run() {
 int sqr = x * x;
 System.out.println("Square of " + x + " = " + sqr ); }}
class Cube extends Thread {
 int x;
 Cube(int n)
```

```
{x = n; }
public void run() {
int cub = x * x * x;
System.out.println("Cube of " + x + " = " + cub );  } }
class Number extends Thread{
public void run(){
Random random = new Random();
for(int i =0; i<5; i++){
int randomInteger = random.nextInt(100);
System.out.println("Random Integer generated : " + randomInteger);
Square s = new Square(randomInteger);
s.start();
Cube c = new Cube(randomInteger);
c.start();
try {
Thread.sleep(1000);
}catch (InterruptedException ex) {
System.out.println(ex);} }}}
public class Thr {
public static void main(String args[])
{
Number n = new Number();
n.start();
}}
```

OUTPUT :-

```
<terminated> Threads [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (Jun 17, 2022, 8:50:45 PM – 8:50:52
Random Integer generated : 21
Square of 21 = 441
Cube of 21 = 9261
Random Integer generated : 65
Square of 65 = 4225
Cube of 65 = 274625
Random Integer generated : 92
Square of 92 = 8464
Cube of 92 = 778688
Random Integer generated : 79
Square of 79 = 6241
Cube of 79 = 493039
Random Integer generated : 98
Square of 98 = 9604
Cube of 98 = 941192
Random Integer generated : 88
Square of 88 = 7744
Cube of 88 = 681472
```

Implementation of BFS and DFS

EXPERIMENT - 3

3A. **Implements of BFS and DFS method**

a. Print all the nodes reachable from a given starting node in a digraph using BFS an DFS method.

Aim:

Write a java program to find the Bredth First Search traversal of the graph.

Algorithm :

Step-1: Start.
Step-2: Create a empty queue to push root node to it.
Step-3: Choose 30 as root note to start traversing.
Step-4: Visit its neighbour nodes and need to find unvisit nodes until the queue is empty.
Step-5: Insert the visited node into queue.
Step-6: Stop.

Source Code:

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
public class bfs
{
private Queue<Node> queue;
static ArrayList<Node> nodes=new ArrayList<Node>();
static class Node
{
int data;
boolean visited;
List<Node> neighbours;
Node(int data)
{
this.data=data;
this.neighbours=new ArrayList<>();
```

```
}
public void addneighbours(Node
neighbourNode) {
this.neighbours.add(neighbourNode);
}
public List<Node> getNeighbours() {
return neighbours;
}
public void setNeighbours(List<Node> neighbours)
{ this.neighbours = neighbours;
}
}
public bfs()
{
queue = new LinkedList<Node>();
}
public void bfs(Node node)
{
queue.add(node);
node.visited=true;
while (!queue.isEmpty())
{
Node element=queue.remove();
System.out.print(element.data + "\t");
List<Node>
neighbours=element.getNeighbours(); for (int i =
0; i < neighbours.size(); i++) { Node
n=neighbours.get(i);
if(n!=null && !n.visited)
{
queue.add(n);
n.visited=true;
}
}
}
```

```java
}
public static void main(String arg[])
{
Node node40 =new Node(40);
Node node10 =new Node(10);
Node node20 =new Node(20);
Node node30 =new Node(30);
Node node60 =new Node(60);
Node node50 =new Node(50);
Node node70 =new Node(70);
node40.addneighbours(node10);
node40.addneighbours(node20);
node10.addneighbours(node30);
node20.addneighbours(node10);
node20.addneighbours(node30);
node20.addneighbours(node60);
node20.addneighbours(node50);
node30.addneighbours(node60);
node60.addneighbours(node70);
node50.addneighbours(node70);
System.out.println("The BFS traversal of the graph is ");
bfs bfsExample = new bfs();
bfsExample.bfs(node30);
}
}
```

Output:

3.B

Aim: Write a Java program to find the Depth First Search traversal of the graph.

Algorithm :

Step-1: Start.
Step-2: Initialize a stack.
Step-3: Take 40 as a root node and push into stack.
Step-4: Find the neighbour nodes, which hasn’t been explored in the stack yet and continue this process until a node is encountered which has no neighbour node.
Step-5: Push unvisited nodes in the stack.
Step-5: Stop.

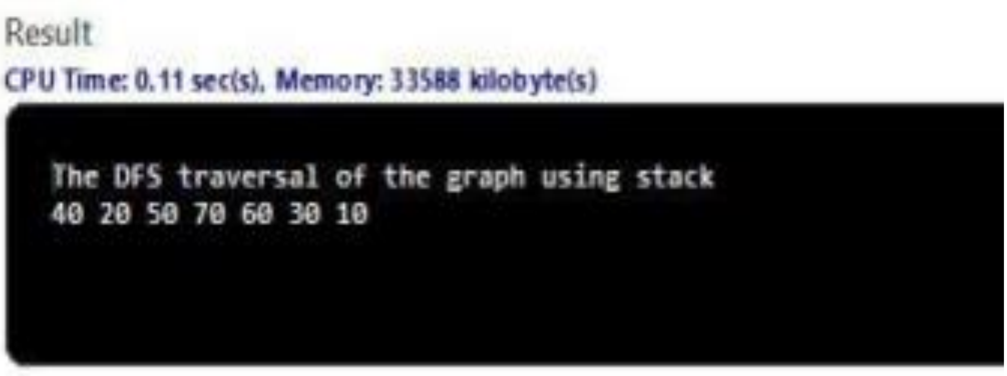
Source Code:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
public class DFS
{
static class Node
{
int data;
boolean visited;
List<Node> neighbours;
Node(int data)
{
this.data=data;
this.neighbours=new ArrayList<>();
}
public void addneighbours(Node
neighbourNode) {
this.neighbours.add(neighbourNode);
}
```

```
public List<Node> getNeighbours() {
return neighbours;
}
public void setNeighbours(List<Node> neighbours)
{ this.neighbours = neighbours;
}
}
// Iterative DFS using stack
public void dfsUsingStack(Node node)
{
Stack<Node> stack=new
Stack<Node>(); stack.add(node);
while (!stack.isEmpty())
{
Node element=stack.pop();
if(!element.visited)
{
System.out.print(element.data + " ");
element.visited=true;
}
List<Node> neighbours=element.getNeighbours();
for (int i = 0; i < neighbours.size(); i++) {
Node n=neighbours.get(i);
if(n!=null && !n.visited)
{
stack.add(n);
}
}
}
}
public static void main(String
arg[]) {
Node node40 =new Node(40);
Node node10 =new Node(10);
Node node20 =new Node(20);
```

```
Node node30 =new Node(30);
Node node60 =new Node(60);
Node node50 =new Node(50);
Node node70 =new Node(70);
node40.addneighbours(node10)
;
node40.addneighbours(node20);
node10.addneighbours(node30);
node20.addneighbours(node10);
node20.addneighbours(node30);
node20.addneighbours(node60);
node20.addneighbours(node50);
node30.addneighbours(node60);
node60.addneighbours(node70);
node50.addneighbours(node70);
DFS dfsExample = new DFS();
System.out.println("The DFS traversal of the graph using stack
");
dfsExample.dfsUsingStack(node40);
System.out.println();
}
}
```

Output:



Implement Of Quick Sort Method

EXPERIMENT – 4

a. **Sort a given set of elements using the quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the 1st to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

Aim :

The aim of this program is to sort "n" randomly generated elements using Quick sort and plotting the graph of the time taken to sort n elements versus.

Algorithm:

Quick sort (A[l….r])

Steps:

if l < r

s = Partition (A[l..r])

//s is a split position Quick sort (A[l …s-1])

Quick sort (A[s+1…r])

ALGORITHM

Steps:

p=A[l] i=l;

j=r+1;

repeat

repeat i= i+1 until A[i] >= p

repeat j=j-1 until A[J] <= p

Swap (A[i],A[j]) until

i >=j

Swap (A[i],A[j]) // Undo last

Swap when i>= j Swap (A[l],A[j])

return j

Program :-

```
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; //Sentinel value
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;
/* times for n = 0, 10, ..., 100, 200, ..., 5000 */
System.out.println ( "\n\nN\tRepetitions\tTime\n" );
for ( n = 5000; n < 50000; n += step )
{
a = new int[n + 1];
qSort = new QuickSort(a);
/*get time for size n */
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
qSort.qSort(0, n - 1);
} while ( System.nanoTime() - start < 1000000000 );
/* repeat until enough time has elapsed */
duration = ( ( double ) ( System.nanoTime() - start ) ) / 1000000000;
duration /= repetitions;
System.out.println ( n + "\t" + repetitions + "\t\t" + duration );
}}}
```

OUTPUT:-

```
Enter the Size of an Array:
4
System automatically generates numbers
Before Sort:
2       0       2       0

After Sort:
0       0       2       2

N       Repetitions     Time

5000    1439            6.951701876302988E-4
7000    1015            9.8590039408867E-4
9000    786             0.0012738365139949108
11000   631             0.0015862681458003170
13000   534             0.0018740376404494381
D15000  447             0.0022394340044742727
```

Implements Merge Sort Algorithms

EXPERIMENT – 5

Implement merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Aim:

The aim of this program is to sort "n" randomly generated elements using Merge Sort and plotting the graph of the time taken to sort n elements versus n.

ALGORITHM Merge sort (A[0…n-1]

If n > 1

Copy A[0…(n/2)-1] to B[0…(n/2)-1] Copy

A[n/2…n-1] to C[0…(n/2)-1] Mergesort

(B[0…(n/2)-1])

Mergesort (C[0…(n/2)-1])

Merge(B,C,A)

ALGORITHM Merge (B[0…p-1], C[0…q-1],A[0….p+q-1])

j = 0;

k= 0;

while i < p and j < q do

```
if B[i] <= C[j] A[k]= B[i];
i = i+1;
else
A[k] = C[j];
j = j+1;
k=k+1;
if i = = p Copy C[ j..q-1] to A[k….p+q-1]
else
Copy B[i … p-1] to A[k …p+q-1]
```

Program :

```
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 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] = input.nextInt();
```

```
}
a[i] = 100000; //Sentinel value
MergeSort mSort = new MergeSort(a);
System.out.println("Before Sort: ");
for ( i = 0; i < n; ++ i )
{
System.out.print(a[i] + "\t");
}
int 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;
/* times for n = 0, 10, ..., 100, 200, ..., 5000 */
System.out.println ( "\n\nN\tRepetitions\tTime\n" );
for ( n = 5000; n < 50000; n += step )
{
a = new int[n + 1];
mSort = new MergeSort(a);
/*get time for size n */
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 );
/* repeat until enough time has elapsed */
duration = ( ( double ) ( System.nanoTime() - start ) ) / 1000000000;
duration /= repetitions;
System.out.println ( n + "\t" + repetitions + "\t\t" + duration );
}}}
```

OUTPUT:-

```
Number of elements : 5
 Before Sorting:
   8    7    6    5    8
After Sorting:
   5    6    7    8    8
Number of elements : 7
Before Sorting:
   2    2    1    1    13    11    6
After Sorting:
   1    1    2    2    6    11    13
Number of elements : 4
Before Sorting:
   1    3    6    1
After Sorting:
   1    1    3    6
Number of elements : 3
Before Sorting:
   5    1    4
After Sorting:
```

1 4 5

Number of elements : 2

Before Sorting:

3 3

After Sorting:

3 3

Number of elements : 1

Before Sorting:

0

After Sorting:

0

Number of elements : 6

Before Sorting:

4 6 0 7 4 7

After Sorting:

0 4 4 6 7 7

Number of elements : 7

Before Sorting:

11 8 3 11 1 11 1

After Sorting:

1 1 3 8 11 11 11

Number of elements : 8

Before Sorting:

14 15 2 5 11 15 5 5

After Sorting:

2 5 5 5 11 14 15 15

Number of elements : 9

Before Sorting:

2 1 8 4 13 8 9 16 13

After Sorting:

1 2 4 8 8 9 13 13 16

Process finished with exit code 0

Implements of Kruskal's algorithm and Prim's algorithm

EXPERIMENT- 6

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

Aim :

Kruskal's Algorithm for computing the minimum spanning tree is directly based on the generic MST algorithm. It builds the MST in forest. Prim's algorithm is based on a generic MST algorithm. It uses greedy approach.

(A). Kruskal's Algorithm

```
KRUSKAL(G):

A = ∅

For each vertex v ∈ G.V:

   MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):

   if FIND-SET(u) ≠ FIND-SET(v):

   A = A ∪ {(u, v)}

   UNION(u, v)

return A
```

(B). Prim’s Algorithm

```
T = ∅;
```

```
U = { 1 };

while (U ≠ V)

    let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;

    T = T ∪ {(u, v)}

    U = U ∪ {v}
```

Program:

(a) Kruskal's algorithm

```
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);
}}
```

OUTPUT:-

```
<terminated> KruskalsChulindra [Java Application] C:\Program Files\Java\jdk-11.0.12\
Enter the number of vertices

3
Enter the cost matrix

0
3
44
55
66
77
2
34
5
1edge(3,1)=2
2edge(1,2)=3
The minimum cost of spanning tree is 5
```

(b) Prim's algorithm.

```
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);
}}
```

OUTPUT:-

```
@ Javadoc  Console
<terminated> Prims [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (J
Enter the number of vertices

3
Enter the cost matrix

1
0
34
5
5
6
6
72
33
1edge(1,3)=34
2edge(3,2)=72
The minimum cost of spanning tree is 106
```

Implements of Dijkstra's algorithm

EXPERIMENT -7

a. **From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstras algorithm.**

Aim:

Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. Algorithm starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on. Following implementation assumes that graph G is represented by adjacency lists.

Algorithm:

- **Read number of vertices of graph G**
- **Read weighted graph G**
- **Print weighted graph**
- **Initialize distance from source for all vertices as weight between source node and other vertices, i,**

  and none in tree

- **For all other vertices,**

  dv[i] = wt_graph[s,i], TV[i]=0, prev[i]=0 dv[s] = 0,

  prev[s] = s // source vertex

  • Repeat for y = 1 to n

v = next vertex with minimum dv value, by calling FindNextNear() Add v to tree.

For all the adjacent u of v and u is not added to the tree,

if dv[u]> dv[v] + wt_graph[v,u]

then dv[u]= dv[v] + wt_graph[v,u] and prev[u]=v.

- **findNextNear**

For k =1 to n

if k vertex is not selected in tree and

if dv[k] < minm

{

minm = dv [k] j=k }

Program:

```
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]);
}}
```

OUTPUT:-

```
<terminated> Dijkstrachulin [Java Application] C:\Program Files\Java\j
Enter the number of vertices
3
Enter the matrix
1
2
3
4
5
6
7
8
9
Enter the source vertex
1
Shortest path from 1 to all other vertices
To 0 is 4
To 1 is 0
To 2 is 6
```

Knapsack problem using Greedy method and Dynamic Programming method

EXPERIMENT - 8

a. Implement in Java, the 0/1 Knapsack problem using Greedy method and Dynamic Programming method

Aim:

We are given a set of n items from which we are to select some number of items to be carried in a knapsack(BAG). Each item has both a weight and a profit.the objective is to choose the set of items that fits in the knapsack and maximizes the profit.

Given a knapsack with maximum capacity, and a set S consisiting of n items,Each item i has some weight wi and benefit value bi(all wi,bi and W are integer values).

Problem:How to pack the knapsack to achieve maximum total value of packed items?

Algorithm:

Greedy Method Algorithms:

- Assume knapsack holds weight W and items have value vi and weight wi
- Rank items by value/weight ratio:

  vi / wi Thus: vi / wi ≥ vj / wj, for all i ≤ j

- Consider items in order of decreasing ratio
- Take as much of each item as possible based on knapsack‟s capacity

USING : Dynamic programming

It gives us a way to design custom algorithms which systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency).

ALGORITHM

• Repeat for i = 0 to n

set V(i,0) = 0

• Repeat for j = 0 to W Set V(0,j) = 0

• Repeat for i = 1 to n

repeat for j = 1 to W

if (wi <= j) V(i,j)) = max{ V(i-1,j), V(i-1,j-wi) + vi }

if (wi > j) V(i,j) = V(i-1,j)

• Print V(n,W)

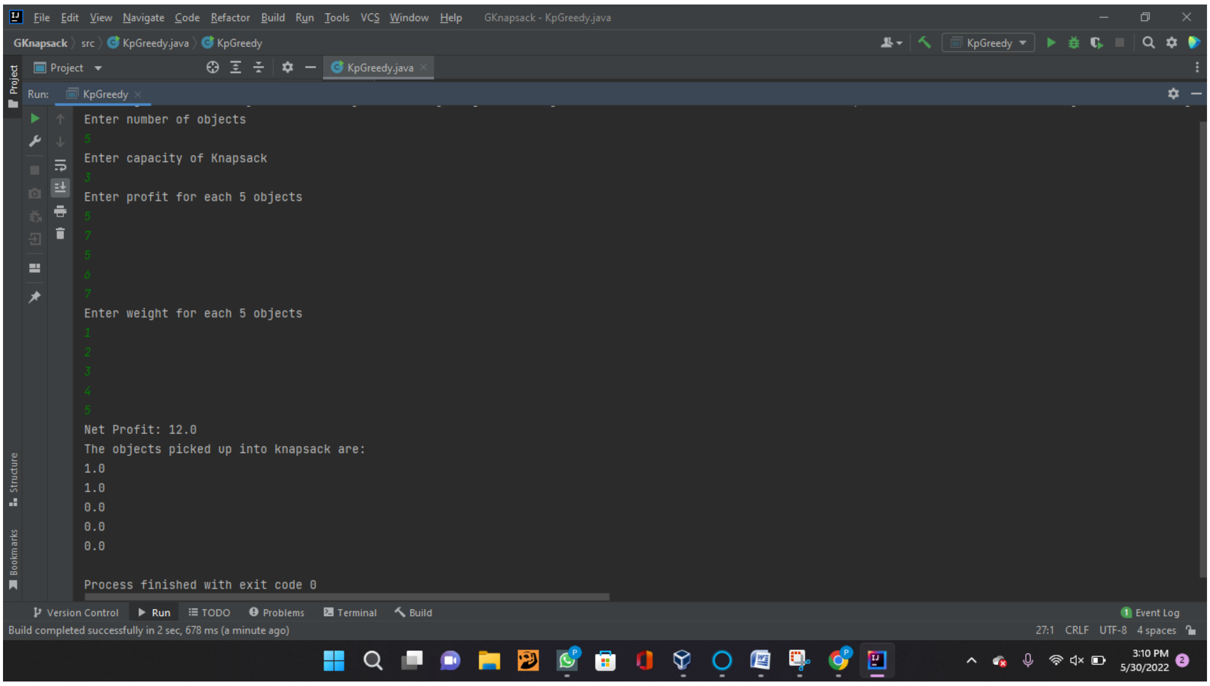
Program :

A) Greedy method

```
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();}}
```

OUTPUT:-



(B) Dynamic Programming method

Program:

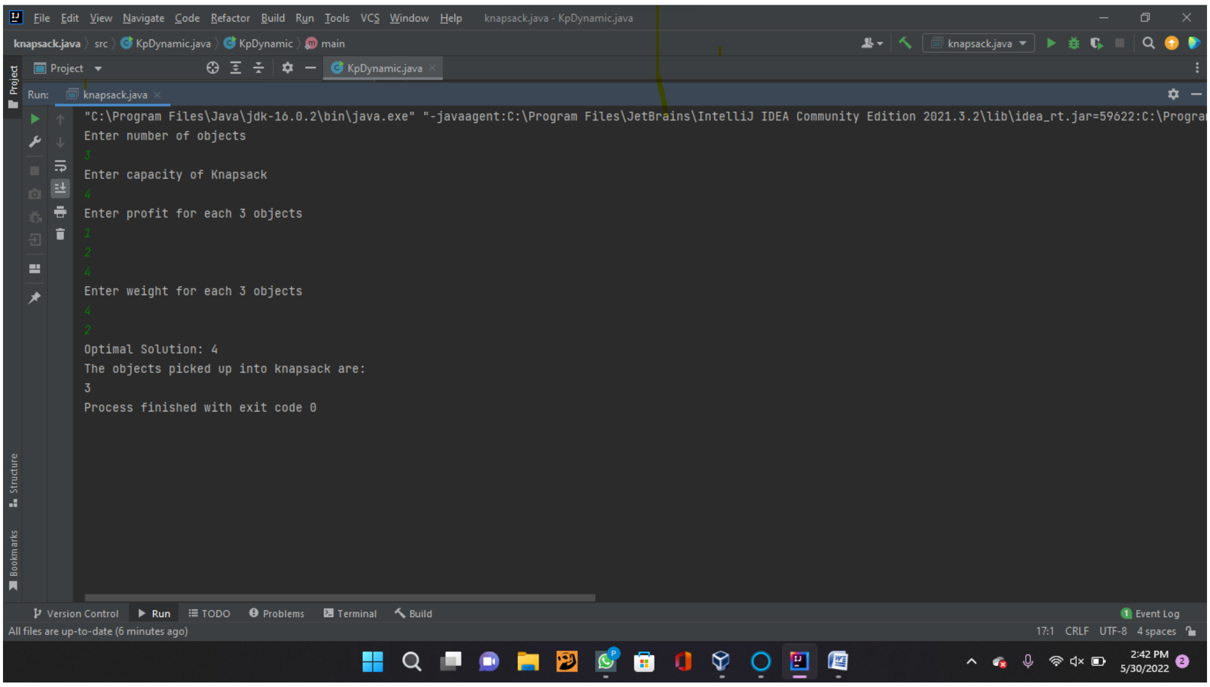
```
import java.util.Scanner;
class DKnapsack
{
int n;
int c;
int p[];
int w[];
int v[][];
public DKnapsack(int n, int c, int[] p, int[] w)
{
super();
```

```
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();
}
}
```

OUTPUT :



TSP problem using Dynamic programming

EXPERIMENT - 9

a. **Write Java programs to Implement Travelling Sales Person problem using Dynamic programming**

Aim:

To Implement Travelling Salesperson problem using Dynamic programming

Algorithm:

C ({1}, 1) = 0

for s = 2 to n do

for all subsets S Є {1, 2, 3, … , n} of size s and containing 1

C (S, 1) = ∞

for all j Є S and j ≠ 1

C (S, j) = min {C (S – {j}, i) + d(i, j) for i Є S and i ≠ j}

Return minj C ({1, 2, 3, …, n}, j) + d(j, i)

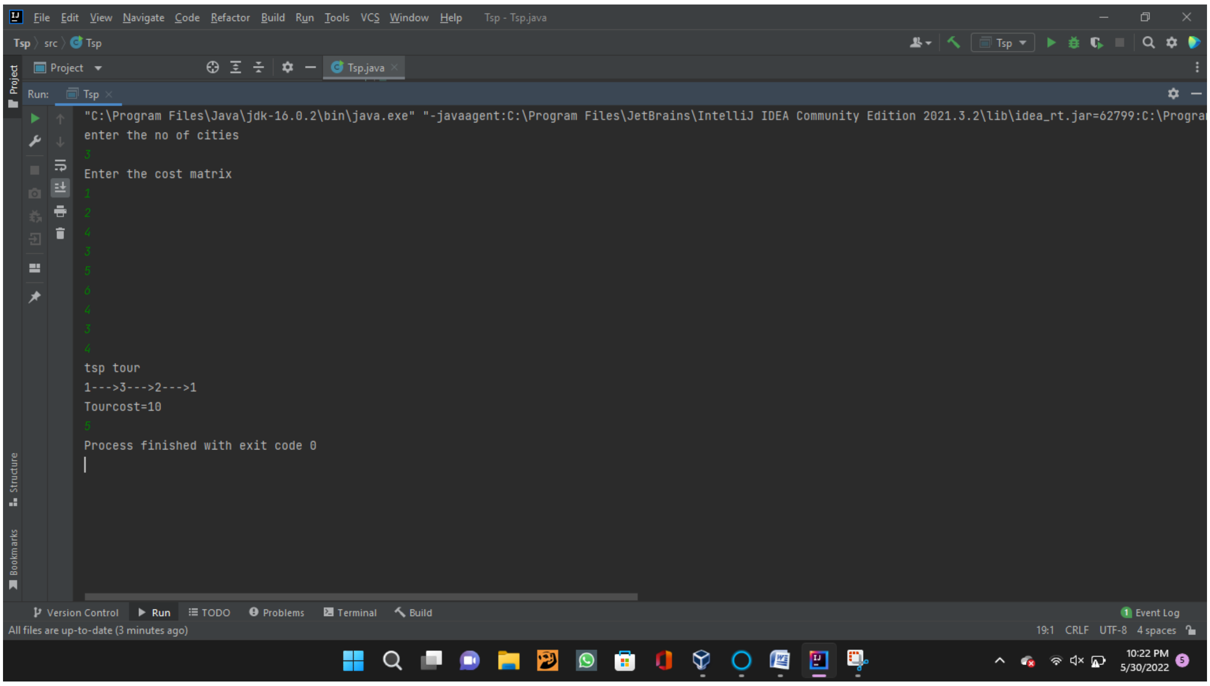
Program:

```
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);
}}
```

OUTPUT:-



Floyd's algorithm

EXPERIMENT - 10

Write a java program to implement all-parts problem using Floyd's algorithm.

Aim:

The Floyd–Warshall algorithm (sometimes known as the WFI Algorithm or Roy–Floyd algorithm) is a graph analysis algorithm for finding shortest paths in a weighted graph

(with positive or negative edge weights). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices though it does not return

details of the paths themselves. The algorithm is an example of dynamic programming.

Algorithm:

Floyd's Algorithm

- **Accept no .of vertices**
- **Call graph function to read weighted graph // w(i,j)**
- **Set D[ ] <- weighted graph matrix // get D {d(i,j)} for k=0**
- **// If there is a cycle in graph, abort. How to find?**
- **Repeat for k = 1 to n**
  - **Repeat for i = 1 to n**
  - **Repeat for j = 1 to n D[i,j] = min {D[i,j], D[i,k] + D[k,j]}**
- **Print D**

Program:

```
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);
}
```

OUTPUT:-

```
Eneter the no of vertices

3
Enter the cost matrix:
1
2
23
4
5
6

7
8
5
all pair shortest paths matrix

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

Find a subset problem

EXPERIMENT - 11

a. **Design and implement in Java to find 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.**

Aim:

An instance of the Subset Sum problem is a pair (S, t), where S = {x1, x2, ..., xn} is a set of positive integers and t (the target) is a positive integer. The decision problem asks for a subset of S whose sum is as large as possible, but not larger than t.

Algorithm:

- **accept n: no of items in set**
- **accept their values, sk in increasing order**
- **accept d: sum of subset desired**
- **initialise x[i] = -1 for all i**
- **check if solution possible or not**
- **if possible then call SumOfSub(0,1,sum of all elements)**
- **SumOfSub (s, k, r)**
- * **Set x[k] = 1**
- **If (s + s[k] = d) then subset found, print solution**
- **If (s + s[k] + s[k+1] <=d)**

then SumOfSum (s + s[k], k+1, r – s[k])

//Generate right child i.e. element k absent

- **If (s + r - s[k] >=d) AND (s + s[k+1])<=d**

THEN

{

x[k]=0;

SumOfSub(s, k+1, r – s[k])

}

OUTPUT :

```
<terminated> Subset [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (Jun 17, 2022, 10:49:33 PM
Enter the number of elements
3
Enter the elements
2
3
2
Enter the sum
23
subset is not possible
```

Hamiltonian Cycles

EXPERIMENT - 12

a. **Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.**

Aim:

Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

Algorithm:

hamiltonian(p,index)

if index>n then

path is complete display the values in p

else

for each node v in G

if v can be added to the path then add v to path p and call hamiltonian(p,index+1)

end Hamiltonian.

Program :

```
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);
}}
```

OUTPUT:-

```
Enter the number of vertices :
4
Enter the matrix :
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
1
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
```