DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY
(Effective from the academic year 2021-2022) SEMESTER – IV

Subject Code	21CSL45	**CIE Marks**	50
Teaching Hours/Week(L:T:P)	0:0:2	**SEE Marks**	50
Total Number of Lab Contact Hours	20	**Total Marks**	100
Credits	01	**Exam Hours**	03 Hrs.

Course Learning Objectives: This course will enable students to:

- Design and implement various algorithms in JAVA
- Employ various design strategies for problem solving.
- Measure and compare the performance of different algorithms.

Descriptions (if any):

- Design, develop, and implement the specified algorithms for the following problems using Java language under LINUX /Windows environment. Net beans / Eclipse or IntellijIdea Community Edition IDE tool can be used for development and demonstration.

Programs List:

1.	
a.	Create a Java class called ***Student*** with the following details as variables within it. (i) USN (ii) Name (iii) Programme (iv) Phone Write a Java program to create n Student objects and print the USN, Name, Programme and Phone of these objects with suitable headings.
b.	Write a Java program to implement the **Stack** using arrays. Write Push (), Pop (), and Display () methods to demonstrate its working.
2.	
a.	Design a super class called ***Staff*** with details as Staff Id, Name, Phone, Salary. Extend this class by writing three subclasses namely ***Teaching*** (domain, publications), ***Technical*** (skills), and ***Contract*** (period). Write a Java program to read and display at least 3 *staff* objects of all three categories.
b.	Write a Java class called ***Customer*** to store their name and date_of_birth. The date_of_birth format should be dd/mm/yyyy. Write methods to read customer data as <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using String Tokenizer class considering the delimiter character as "/".
3.	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
4.	Sort a given set of N integer element using Selection Sort technique and Compute it's taken. Run the program for different values of N and record the time taken to sort.
5.	Sort a given set of N integer element using Insertion Sort technique and Compute it's taken
6.	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 Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case. Compare the priori and posteriori analysis of an algorithm.
7.	Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Compare the priori and posteriori analysis of an algorithm.
8.	Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm
9.	Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.
10.	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.
11.	Implement All-Pairs Shortest Paths problem using Floyd's algorithm and find transitive closure of a given graph using Warshalls Algorithm.
12.	Implement in Java, the 0/1 Knapsack problem using Dynamic Programming method.
13.	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 does not have a solution.
14.	Implement N-Queens problem using Backtracking.
15.	Implement Topological Sorting using DFS based method.

AIM:

1 a. Create a Java class called ***Student*** with the following details as variables within it.

(i) USN
(ii) Name
(iii) Branch
(iv) Phone

Write a Java program to create *nStudent* objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.

ALGORITHM:

1. Create a student class with arguments to the constructor is USN, Name, Branch, Phone

2. Read the number of student objects to be created.

3. Read each student object details (USN, Name, Branch, Phone)

4. Display the USN, Name, Branch, and Phone number of each Student.

PROGRAM:

```
import java.util.Scanner;
class Student
{
    String name,usn,branch,ph_no;
    void readdata() // To read the Data
    {
      Scanner sobj= new Scanner(System.in);
      System.out.print("Enter Name of Student:");
      name = sobj.next();
      System.out.print("Enter USN of Student:");
      usn = sobj.next();
      System.out.print("Enter the Branch of Student:");
      branch = sobj.next();
      System.out.print("Enter the Phone Number of Student:");
      ph_no= sobj.next();
      System.out.println("\n");
    }
    void displaydata() // To Display The Information
   {
      System.out.println ("Name= "+name);
      System.out.println ("USN = "+usn);
      System.out.println ("Branch= "+branch);
      System.out.println ("Phone Number= "+ph_no);
      System.out.println("\n");
   }
}
```

```
public class StudentDemo
{
    public static void main(String args[])
    {
      int n;

      System.out.println("Enter the Number of Students:");
      Scanner sobj=new Scanner(System.in);
      n=sobj.nextInt();
     // To Create Array of Object
     Student[] stobj = new Student[n];
     for(int i=0;i<n;i++)
     {
       stobj[i]=new Student(); // initialize it zero by default
constructor
     }
     for(int i=0;i<n;i++)
     {
       stobj[i].readdata();
     }
     System.out.println("Information about Students is:\n");

     for(int i=0;i<n;i++)
     {
       stobj[i].displaydata();
     }
  }
}
```

_____Output_____

```
Enter the Number of Students:
1
Enter Name of Student:Rahul
Enter USN of Student:1212
Enter the Branch of Student:cse
Enter the Phone Number of Student:212

Information about Students is:

Name= Rahul
USN = 1212
Branch= cse
Phone Number= 212
```

AIM:

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

ALGORITHM:

```
Stack(Max_Size)
        N = Max_Size
        Define a stack using array stack_array[0..N-1] of size N
        top = -1

        function push(data)
                if(isFull())
                        Stack is full
                else
                        top = top + 1
                stack_array[top] = data

        function pop()
                if(isEmpty())
                        Stack is empty
                else
                        stacktop_data = stack_array[top]
                        top = top - 1
                        return stacktop_data

        function display()
                if(isEmpty())
                        Stack is empty
                else
                        for i=top down to 0
                        print array[i]

        function Boolean isEmpty()
                return top == -1

        function Boolean isfull()
                return top = N-1
```

PROGRAM:

```
import java.util.Scanner; //Scanner Method means to read the
values from keyboard
class Stack
{
```

```
      int size=10;
      int arr[] = new int[size];
      int top = -1;
      void push(int item)
      {
            if(top==size-1)
            {
                  System.out.println("Error !Stack Overflow ");
            }
            else
            {
                  top++;
                  arr[top]=item;
                  System.out.println("The Item\t " +item + "\t
              is pushed on to the stack");
            }
      }

  // Method to Delete(pop) the Elements from the stack
      void pop()
      {
            if(top==-1)
            {
                  System.out.println("ERROR!!! stack
underflow");
            }
            else
            {
                  int item;
                  item =arr[top];
                  System.out.println("The Item\t" + arr[top] +
"\tis
                  poped out of the stack");
                  top--;
            }
      }

 // Method to Print/Display the Elements from the stack
      void display()
      {
            if(top==-1)
            {
                  System.out.println("Stack Empty ");
            }
 else
            {
                  System.out.println("Elements in stack ");
```

```
                    for(int i=0;i<=top;i++)
                    {
                          System.out.println(arr[i]);
                    }
              }
       }
}
// Main Method class
public class StackDemo
{
       public static void main(String args[])
       {
              Stack stk= new Stack();
              int x;
              Scanner s =new Scanner(System.in);
              int ch;
              System.out.println("Enter 1: to push element");
              System.out.println("Enter 2: to pop element");
 System.out.println("Enter 3: to display elements");
              System.out.println("Enter 4: to Exit ");
              do
              {
                    System.out.println("Enter your choice: ");
                    ch=s.nextInt();
                    switch(ch)
                    {
                          case 1:  System.out.println("Enter
     element:”);
                          x=s.nextInt();
                          stk.push(x);
                          break;

                          case 2:  stk.pop();
                          break;

                          case 3:  stk.display();
                          break;

                          case 4: System.exit(0);
                          default: System.out.println("Please
Enter
                          correct choice");
                    }
              }
              while (ch!=4);
       }
}
```

_____Output_____
_____

```
Enter 1: to push element
Enter 2: to pop element
Enter 3: to display elements
Enter 4: to Exit
Enter your choice:
1
Enter element:
3
The Item      3      is pushed on to the stack
Enter your choice:
3
Elements in stack
3
Enter your choice:
1
Enter element:
4
The Item      4      is pushed on to the stack
Enter your choice:
3
Elements in stack
3
4
Enter your choice:
2
The Item     4      is poped out of the stack
Enter your choice:
3
Elements in stack
3
Enter your choice:
4
```

AIM:

2 a. Design a superclass called ***Staff*** with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely ***Teaching*** (domain, publications), ***Technical*** (skills), and ***Contract*** (period). Write a Java program to read and display at least 3 *staff* objects of all three categories.

PROGRAM:

1. Create a Super class with the name staff and define required parameter

2. Create subclass named Teaching and extend the super class staff facilities

3. Create subclass named Technical and extend the super class staff facilities

4. Create the subclass Contract and extend the super class facilities staff facilities

PROGRAM:

```
import java.util.Scanner;

//Super class
class Staff
{
     String sid,name,ph;
     float sal;

     void getdetails()
     {
               Scanner s=new Scanner(System.in);
               System.out.println("Enter Id");
               sid=s.next();
               System.out.println("Enter Name");
               name=s.next();
               System.out.println("Enter phone number");
               ph=s.next();
               System.out.println("Enter Salary");
               sal=s.nextFloat();
     }

     void putdetails()
     {
          System.out.println("-----Staff Details are-----");
          System.out.println("Staff Id is:"+sid);
          System.out.println("Staff Name is:"+name);
          System.out.println("Staff Phone number is:"+ph);
          System.out.println("Staff Salary is:"+sal);}
     }

//class 1
class Teaching extends Staff
{
     String dom,pub;
```

```
     Teaching()
     {
          getdetails();
          Scanner s=new Scanner(System.in);
          System.out.println("Enter domian");
               dom=s.next();
          System.out.println("Enter Publications");
          pub=s.next();
     }

     void dispTeach()
     {
               putdetails();
               System.out.println("Staff Domain is:"+dom);
               System.out.println("Staff has Published:"+pub);
     }
}

//class 2
class Technical extends Staff
{
     String skills;
     Technical()
     {
          getdetails();
          Scanner s=new Scanner(System.in);
          System.out.println("Enter the Skills");
          skills=s.next();
     }
     void dispTech()
     {
          putdetails();
System.out.println("Skills of the Staff:"+skills);
     }
}

//Class 3
class Contract extends Staff
{
     int period;
     Contract()
     {
          getdetails();
          Scanner s=new Scanner(System.in);
          System.out.println("Enter Contract period in yrs");
          period=s.nextInt();
```

```
     }

     void dispContract()
     {
          putdetails();
          System.out.println("Staff Contract period
          is:"+period+"yrs");
     }
}

//main class
public class Demo
{
     public static void main(String arg[])
     {
          System.out.println("Enter the details of Teaching
          Staff");
          Teaching t1=new Teaching();  // Teaching Staff
          System.out.println("Enter the details of Technical
Staff");
          Technical t2= new Technical();//Technical Staff
          System.out.println("Enter the details of Contract
          Based Staff");
          Contract c=new Contract(); //Contract based Staff
          System.out.println("Teaching Staff");
          t1.dispTeach();
System.out.println("Technical Staff");
          t2.dispTech();
          System.out.println("Contract Based Staff");
          c.dispContract();
     }
}
```

_____Output_____

_____

```
Enter the details of Teaching Staff
Enter Id 1
Enter Name ss
Enter phone number 34343
Enter Salary 345555
Enter domain cs
Enter Publications intrn
Enter the details of Technical Staff
Enter Id 2
Enter Name rr
Enter phone number 32442
Enter Salary 324444
```

```
Enter the Skills lang
Enter the details of Contract Based Staff
Enter Id 3
Enter Name tt
Enter phone number 345353
Enter Salary 55656
Enter Contract period in yrs 3
Teaching Staff
-----Staff Details are-----
Staff Id is:1
Staff Name is:ss
Staff Phone number is:34343
Staff Salary is:345555.0
Staff Domain is:cs
Staff has Published:intrn
Technical Staff
-----Staff Details are-----
Staff Id is:2
Staff Name is:rr
Staff Phone number is:32442
Staff Salary is:324444.0
Skills of the Staff:lang
Contract Based Staff
-----Staff Details are-----
Staff Id is:3
Staff Name is:tt
Staff Phone number is:345353
Staff Salary is:55656.0
Staff Contract period is:3yrs
```

AIM:

2 b. Write a Java class called ***Customer*** to store their name and date_of_birth. The date_of_birth format should be dd/mm/yyyy. Write methods to read customer data as <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using StringTokenizer class considering the delimiter character as "/".

ALGORITHM:

1. Create a Java class Customer and define the required features

2. Read the date of birth in the prescribed format

3. Create a method to read the date string

4. Use StringTokenizer java class to tokenize the date string and print

PROGRAM:

```
import java.util.Scanner;
import java.util.StringTokenizer;
class Customer
{
String cname,dob;
Scanner sobj=new Scanner(System.in);
void read()
{
System.out.println("Enter Customer name:");
cname=sobj.next();
System.out.println("Enter Customer DOB in the format
dd/mm/yyy");
dob=sobj.next();
}
void display()
{
StringTokenizer st = new StringTokenizer(dob, "/");
System.out.print(cname+",");
while(st.hasMoreTokens())
{
String val = st.nextToken();
System.out.print(val);
if(st.countTokens()!=0)
System.out.print(","+" ");

}
}
```

```
}
public class sttoken {
     public static void main(String[] args) {
                                        Customer cobj=new Customer();
                cobj.read();
                System.out.println("Customer Details");
                System.out.println("---------------------");
                cobj.display();
                }
}
```

_____Output_____

```
Enter Customer name:
subash
Enter Customer DOB in the format dd/mm/yyy
01/05/2020
Customer Details
---------------------
subash,01, 05, 2020
```

AIM:

3 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.

ALGORITHM:

1. Read two integers a and b

2. Compute division a/b

3. If b is not zero print the result without exception

4. If b = 0 print the exception by using Java maths exceptions

PROGRAM:

```
import java.util.Scanner;
public class DivideException
{
     public static void main(String[] args)
     {
          Scanner s = new Scanner(System.in);
          System.out.print("Enter first number(numerator): ");
          int a = s.nextInt();
```

```
            System.out.print("Enter second number(denominator):
");
            int b = s.nextInt();
            try
            {
                  if(b!=0)
                  {
                        int res=a/b;
                        System.out.println("result="+res);
                  }
                  else
                  throw new ArithmeticException();
            }
            catch (ArithmeticException e)
            {
                  System.out.println("Divide by Zero Error");
            }
      }
}
```

_____Output_____
_____

```
Enter first number (numerator): 10
Enter second number (denominator): 5
result = 2

Enter first number (numerator): 10
Enter second number(denominator): 0
Divide by Zero Error
```

AIM:

4. Sort a given set of N integer element using Selection Sort technique and Compute it's taken. Run the program for different values of N and record the time taken to sort.

ALGORITHM:

1. For i=0 to n-1
2. min=i

3. for j=i+1 to n
4. if(a[j]<a[m])
5. m=j
6. if(min!= i)
7. swap(a, min, i)

8. function printarray()
9. for i = 1 to n
10. print A[i]

PROGRAM:

```
class Selectionsort
 {
        void sort(int arr[])
         {
             int n = arr.length;

             // One by one move boundary of unsorted subarray
             for (int i = 0; i < n-1; i++)
             {
                 // Find the minimum element in unsorted array
                 int min_idx = i;
                 for (int j = i+1; j < n; j++){
                     if (arr[j] < arr[min_idx])
                         min_idx = j;
                 }

                 // Swap the minimum element found with the first
                 // element
                 int temp = arr[min_idx];
                 arr[min_idx] = arr[i];
                 arr[i] = temp;
             }
         }

         // Prints the array
         void printArray(int arr[])
         {
             int n = arr.length;
             for (int i=0; i<n; ++i)
                 System.out.print(arr[i]+" ");
             System.out.println();
         }
 }
 public class ssort{
```

```
public static void main(String args[])
     {
 Selectionsort ob = new Selectionsort();
        int arr[] = {64,25,12,22,11};
        long start_time=System.currentTimeMillis();
        ob.sort(arr);
        long end_time=System.currentTimeMillis();
        System.out.println("Sorted array");
        ob.printArray(arr);
  System.out.println("\nTime taken="+(end_time-start_time)+"  Milli seconds");
     }
 }
```

_____Output_____

Sorted array
11 12 22 25 64

```
Time taken=0  Milli seconds
```

Aim:

5. Sort a given set of N integer element using Insertion Sort technique and Compute it's taken.

ALGORITHM:

1. for i=1 to i+1
2. temp=a[i]
3. j=i-1
4. while(j>=0 & a[j]>temp)
5. a[j+1]=a[i]
6. j—
7. a[j+1]=temp

8. function printarray()
9. for i = 1 to n
10. print A[i]

PROGRAM:

```java
public class insertionsort {
      //public class InsertionSort {
          /*Function to sort array using insertion sort*/
          void sort(int arr[])
          {
              int n = arr.length;
              for (int i = 1; i < n; ++i) {
                  int key = arr[i];
                  int j = i - 1;

                  /* Move elements of arr[0..i-1], that are
                     greater than key, to one position ahead
                     of their current position */
                  while (j >= 0 && arr[j] > key) {
                      arr[j + 1] = arr[j];
                      j = j - 1;
                  }
                  arr[j + 1] = key;
              }
          }

          /* A utility function to print array of size n*/
          static void printArray(int arr[])
          {
              int n = arr.length;
               System.out.println ("sorted Array :");
              for (int i = 0; i < n; ++i)
                  System.out.print(arr[i] + " ");

              System.out.println();
          }

          // Driver method
          public static void main(String args[])
          {
              int arr[] = { 12, 11, 13, 5, 6 };
             long start_time=System.currentTimeMillis();
                    insertionsort ob = new insertionsort();
             ob.sort(arr);
             long end_time=System.currentTimeMillis();
             printArray(arr);
        System.out.println("\nTime taken="+(end_time-start_time)+"  Milli seconds");
          }
      }
```

_____Output_____

```
sorted Array :
5 6 11 12 13

Time taken=0  Milli seconds
```

Aim:

6. 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 ***n***on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case. Compare the priori and posteriori analysis of an algorithm.

ALGORITHM:

Quicksort

1. function QuickSort(A[1..n], p, r)
2. if (p < r)
3. q = Parition(A[1..n], p, r)
4. QuickSort(A[1..n], p, q-1)
5. QuickSort(A[1..n], q+1, r)

6. function display()
7. for i = 1 to n
8. print A[i]
9. function Partition(A[1..n], p, r)
10. x = A[r]
11. i = p-1
12. for j = p to r - 1
13. if(A[j] <= x

14 i = i + 1

15 exchange A[i] and A[j]

16. exchange A[i+1] and A[r]
17. return i+1

PROGRAM:

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

```
import java.util.Random;

class QuickSort_class
{
	int n;
	int a[];
	QuickSort_class(int x)
	{
		n=x;
		a=new int[n];
	}
	void generate()
	{
		System.out.println("Generating n random
numbers....");
		Random r=new Random();
		for(int i=0;i<n;i++)
		{
			a[i]=r.nextInt(100);
			System.out.print(a[i]+" ");
		}
		System.out.println();
	}
	int partition(int a[],int lb,int ub)
	{

		Scanner s=new Scanner(System.in);
		int pi,down,temp,up;
		pi=a[lb];
		down=lb;
		up=ub;
		while(down<up)
		{
			while((a[down]<=pi)&&(down<up))
			down++;

			while(a[up]>pi)

up--;
			if(down<up)
			{
				temp=a[down];
				a[down]=a[up];
				a[up]=temp;
			}
		}
		a[lb]=a[up];
```

```
            a[up]=pi;
            return up;
        }
        void qsort(int a[],int lb,int ub)
      {

            if(lb<ub)
            {
int j=partition(a,lb,ub);
                  qsort(a,lb,j-1);
                  qsort(a,j+1,ub);
            }
      }

      void display()
      {
            for(int i=0;i<n;i++)
            System.out.print(a[i]+" ");
      }
}
public class QuickSort
{
      public static void main(String args[])
      {
            Scanner s=new Scanner(System.in);
            System.out.println("Enter the number of elements");
            int x=s.nextInt();
            QuickSort_class q=new QuickSort_class(x);
            q.generate();
            long start_time=System.currentTimeMillis();
            q.qsort(q.a,0,q.n-1);
            long end_time=System.currentTimeMillis();
            double time_taken = (end_time-start_time);
            System.out.println("\nThe Sorted array is:");
            q.display();
            System.out.println("\nTime taken="+time_taken+"
            Milli seconds");
      }
}
```

_____Output_____

```
Enter the number of elements
20
Generating n random numbers....
44 75 53 35 80 81 79 74 2 73 4 46 38 88 99 2 18 73 19 82

The Sorted array is:
```

```
2 2 4 18 19 35 38 44 46 53 73 73 74 75 79 80 81 82 88 99
Time taken=2.0 Milli seconds
```

AIM:

7. Sort a given set of ***N*** integer elements using **Merge Sort** method and compute its time complexity. Run the program for different values of *N* and record the time taken to sort. Compare the priori and posteriori analysis of an algorithm.

ALGORITHM:

Merge sort

1. function MergeSort(A[1..n], p, r)
2. if(p < r)
3. q = ((p + r)/2)
4. MergeSort(A[], p, q)
5. MergeSort(A[], q+1, r)
6. Merge(A[], p, q, r)

7. function display()
8. for i = 1 to n
9. print A[i]
10. function Merge(A[], p, q, r)
11. n1 = q - p + 1
12. n2 = r - q
13. Define two arrays L[1..(n1 + 1)] and R[1...(n2 + 1)]
14. for i = 1 to n1
15. L[i] = A[p + i - 1]
16. L[n1+1] = 100000000
17. for j = 1 to n2
18. R[j] = A[q + j]
19. R[n2+1] = 100000000
20 x=1
21 y=1
22. for k = p to r
23. if(L[x] <= R[y])
24. A[k] = L[x]
25. x = x + 1
26. else
27. A[k] = R[y]

28. y = y + 1

PROGRAM:

```
import java.util.*;
class MergeSort
{
     int n;
     int a[];
     MergeSort(int x)
     {
          n=x;
          a=new int[n];
     }
     void generate()
     {
          System.out.println("Generating n random numbers....");
          Random r=new Random();
          for(int i=0;i<n;i++)
          {
               a[i]=r.nextInt(20);
               System.out.print(a[i]+" ");
          }
}
     void Merge(int low,int mid,int high)
     {
          int h,i,j,k;
          int [] b=new int[n];
          h=low; i=low; j=mid+1;
          while ((h<=mid)&&(j<=high))
          {
               if(a[h]<=a[j])
               {
                    b[i]=a[h];
                    h=h+1;
               }
               else
               {
                    b[i]=a[j];
                    j=j+1;
               }
               i=i+1;
          }
          if(h>mid)
          {
               for(k=j;k<=high;k++)
               {
                    b[i]=a[k];
                    i=i+1;
```

```
                }
            }
            else
            {
                for(k=h;k<=mid;k++)
                {
                    b[i]=a[k];
                    i=i+1;
                }
            }
            for(k=low;k<=high;k++)
            a[k]=b[k];
        }
        void mSort(int low,int high)
        {
            if(low<high)
            {
                int mid=(low+high)/2;
                mSort(low,mid);
                mSort(mid+1,high);
Merge(low,mid,high);
            }
        }
        void display()
        {
            for(int i=0;i<n;i++)
            System.out.print(a[i]+" ");
        }
}
public class MergeSortDemo
{
        public static void main(String[] args)
        {
            Scanner s=new Scanner(System.in);
            System.out.println("Enter the number of elements");
            int x=s.nextInt();
            MergeSort m=new MergeSort(x);
            m.generate();
            long start_time=System.currentTimeMillis();
            m.mSort(0,m.n-1);
            long end_time=System.currentTimeMillis();
            System.out.println("\nThe Sorted array is:");
            m.display();
            System.out.println("\nTime taken="+(end_time
            start_time)+"milli seconds");
        }
}
```

_____Output_____

_____

```
Enter the number of elements
20
Generating n random numbers....
4 2 5 7 7 0 7 17 5 2 4 11 2 17 4 3 7 1 10 10
The Sorted array is:
0 1 2 2 2 3 4 4 4 5 5 7 7 7 7 10 10 11 17 17
Time taken=0milli seconds
```

AIM:

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

ALGORITHM:

MST-PRIM(G, w r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = Q.V
while Q = φ
    u = EXTRACT-MIN(Q) //minimum priority queue
for each v ∈ G.Adj(u)
    v ∈ Q and w(u, v) < v.key
    v.π = u
    v.key = w(u, v)

PROGRAM:

```
import java.util.Scanner;
class Prims
{
    int n,c[][],st[][];
    void read()
    {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter number of vertices");
```

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

}
void primsAlg()
{
        int i,j,w,u=0,nr[],min,min_cost=0;
        st=new int[n+1][3];
        nr=new int[n+1];
        for(i=1;i<=n;i++)
            nr[i]=1;
            nr[1]=0;
        for(i=1;i<n;i++)
        {
            min=999;
            for(j=1;j<=n;j++)
            {
                if(nr[j]!=0&&c[j][nr[j]]<min)
{
                    min=c[j][nr[j]];
                    u=j;
                }
            }
            st[i][1]=u;
            st[i][2]=nr[u];
            min_cost=min_cost+c[u][nr[u]];
            nr[u]=0;
            for(w=1;w<=n;w++)
            {
                if(nr[w]!=0&&c[w][nr[w]]>c[w][u])
                nr[w]=u;
            }
        }
        System.out.println("the minimum spanning tree is:");
        for(i=1;i<=n-1;i++)
        System.out.println(st[i][1]+"<->"+st[i][2]);
        System.out.println("minimum cost="+min_cost);
```

```
    }
}
public class PrimsDemo
{
    public static void main(String[] args)
    {
        Prims p=new Prims();
        p.read();
        p.primsAlg();
    }
}
```

_____Output_____

```
Enter number of vertices
4

Enter the cost adjacency matrix
0 10 999 40
10 0 20 999
999 20 0 30
40 999 30 0

the minimum spanning tree is:
2<->1
3<->2
4<->3

minimum cost=60
```

AIM:

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

ALGORITHM:

KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3 MAKE-SET(v)
4 foreach (u, v) in G.E ordered by weight (u, v), increasing:

5 if FIND-SET (u) ≠ FIND-SET (v):
6 A = A ∪ {(u, v)}
7 UNION (u, v)
8 return A

PROGRAM:

```
import java.util.*;
class Kruskal
{
     int n,c[][],st[][],par[];
     void read()
     {
          Scanner scr=new Scanner(System.in);
          System.out.println("enter the no of vertices");
          n=scr.nextInt();
          c=new int[n+1][n+1];
          par=new int[n+1];
          System.out.println("enter the cost adjacency matrix");
for(int i=1;i<=n;i++)
               for(int j=1;j<=n;j++)
                    c[i][j]=scr.nextInt();
               for(int i=1;i<=n;i++)
                    par[i]=i;
     }
     int find(int i)
     {
          i=par[i];
          return i;
     }
     void algo()
     {
          int mincost=0,e=0,min,u=0,v=0,a,b;
          st=new int[n+1][n+1];
          for(int i=1;i<=n;i++)
               for(int j=1;j<=n;j++)
                    st[i][j]=c[i][j];
                    System.out.println("min cost spanning tree
is:");
                    while(e!=n-1)
                    {
                         min=999;
```

```
                    for(int i=1;i<=n;i++)
                        for(int j=1;j<=n;j++)
                            if(min>st[i][j])
                            {
                                min=st[i][j];
                                u=i;
                                v=j;
                            }
                            System.out.println(u+"->"+v);
                            st[u][v]=999;
                            a=find(u);
                            b=find(v);
                        if(a!=b)
{
                            e++;
                            System.out.print(e+":");
                            System.out.println(u+"
                            >"+v+"cost:"+min);
                            unions(a,b);
                            mincost=mincost+min;
                        }
                        else
                        System.out.println(u+"-
                        >"+v+"rejected:forms a cycle");
                }
                System.out.println("cost of spanning
                tree"+mincost);
        }
        void unions(int i,int j)
        {
            par[j]=i;
        }
}
public class KruskalDemo
{
        public static void main(String[] args)
        {
            Kruskal k=new Kruskal();
            k.read();
            k.algo();
```

```
    }
}
```

_____Output_____

```
enter the no of vertices
4
enter the cost adjacency matrix
0 10 999 40
10 0 20 999
999 20 0 30
40 999 30 0
min cost spanning tree is:
1->1
1->1rejected:forms a cycle
2->2
2->2rejected:forms a cycle
3->3
3->3rejected:forms a cycle
4->4
4->4rejected:forms a cycle
1->2
1:1->2cost:10
2->1
2->1rejected:forms a cycle
2->3
2:2->3cost:20
3->2
3->2rejected:forms a cycle
3->4
3:3->4cost:30
cost of spanning tree60
```

AIM:

10. From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's algorithm**. Write the program in Java.

ALGORITHM:

function Dijkstra(Graph, source):

1 create vertex set Q
2 for each vertex v in Graph: // Initialization
3 dist[v] ← INFINITY // Unknown distance from source to v
4 prev[v] ← UNDEFINED //previous node in optimal path from source
5 add v to Q // All nodes initially in Q (unvisited nodes)
6 dist[source] ← 0 // Distance from source to source
7 while Q is not empty:
8 u ← vertex in Q with min dist[u]//Node with the least distance
9 // will be selected first
10 remove u from Q
12 for each neighbor v of u: // where v is still in Q.
13 alt ← dist[u] + length(u, v)
14 if alt < dist[v]: // A shorter path to v has been found
15 dist[v] ← alt
16 prev[v] ← u
17 return dist[], prev[]

PROGRAM:

```
import java.util.*;
class Dijikstra
{
	int n,s,v[],c[][],dist[],p[];
	void read()
	{
		Scanner scr=new Scanner(System.in);
		System.out.println("Enter the number of
vertices.....");
		n=scr.nextInt();  // 6
		c=new int[n+1][n+1]; // c[6][6]
		System.out.println("Enter the adjacency
matrix.....");
		for(int i=1;i<=n;i++)
			for(int j=1;j<=n;j++)
				c[i][j]=scr.nextInt();
		System.out.println("Enter the source vertex.....");
s=scr.nextInt();
	}
	void dijAlg()
	{
		v=new int[n+1];
		dist=new int[n+1];
		p=new int[n+1];
		for(int i=1;i<=n;i++)
```

```
		{
			 v[i]=0;
			dist[i]=c[s][i];
			p[i]=s;
		}
		v[s]=1;
		dist[s]=0;
		p[s]=0;
		int u=0;
		for(int i=2;i<=n;i++)
		{
			int min=999;
			for(int j=1;j<=n;j++)
				if(v[j]==0 && dist[j]<min)  // 0 &&
999<999
				{
					min=dist[j];
					 u=j;
				}
				v[u]=1;
			for(int w=1;w<=n;w++)
				if(v[w]==0 &&
dist[w]>(dist[u]+c[u][w]))
				{
					dist[w]=dist[u]+c[u][w];
					p[w]=u;
				}
		}
	}
	void path(int s,int i)
	{
		if(p[i]!=s)
		path(s,p[i]);
		System.out.print("-->"+i);
	}
}
public class DijDemo
{
	public static void main(String[] args)
	{
		int i,j;
		Dijikstra d=new Dijikstra();
		d.read();
		d.dijAlg();
		System.out.println("The shortest path");
		for(i=1;i<=d.n;i++)
		{
```

```
                    if(i!=d.s)
                    {
                           System.out.print(d.s);
                           d.path(d.s,i);
                           System.out.println("with
                          distance="+d.dist[i]);
                    }

                    System.out.println();
              }
       }
}
```

Output

```
Enter the number of vertices.....
6
Enter the adjacency matrix.....
0 15 10 999 45 999
999 0 15 999 20 999
20 999 0 20 999 999
999 10 999 0 35 999
999 999 999 30 0 999
999 999 999 4 999 0
Enter the source vertex.....
1
The shortest path

1-->2with distance=15

1-->3with distance=10

1-->3-->4with distance=30

1-->2-->5with distance=35

1-->6with distance=999
```

AIM:

11. Write Java programs to Implement All-Pairs Shortest Paths problem using **Floyd's**

algorithm.

ALGORITHM:

Floyds()
1. for i=1 to n
2. for j=1 to n
3. d[0][i][j]=a[i][j]
4. for k=1 to n
5. for i=1 to n
6. for j=1to n
7. d[k][i][j]=min(d[k-1][i][j] (d[k-1][i][k]+d[k-1][k][j]))
8. write ()
9 for k=1 to n
5. for i=1 to n
6. for j=1to n
7.print d[k][i][j].

PROGRAM:

```
import java.util.*;
class floyd
{
     int a[][]=new int[10][10];
     int d[][][]=new int[10][10][10];
     int n;
     void read()
     {
          int i,j;
          Scanner scan=new Scanner(System.in);
          System.out.println("Enter the number of vertices\n");
          n=scan.nextInt();
          System.out.println("Enter the adjacency matrix\n");
          for(i=1;i<=n;i++)
               for(j=1;j<=n;j++)
                    a[i][j]=scan.nextInt();
     }
     void write()
     {
          int i,j;
          System.out.println("Computing all pairs shortest
path\n");
          for(int k=0;k<=n;k++)
          {
               System.out.println("d["+k+"]<<=<<\n");
               for(i=1;i<=n;i++)
```

```
                {
                        for(j=1;j<=n;j++)
                        {
System.out.print(d[k][i][j]+"\t");
                        }
                        System.out.print("\n");
                }
        }
    }
    void floyds()
    {
        int i,j,k;
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                d[0][i][j]=a[i][j];
            }
        }
        for(k=1;k<=n;k++)
        {
            for(i=1;i<=n;i++)
            {
                for(j=1;j<=n;j++)
                {
                    d[k][i][j]=min(d[k-1][i][j],(d[k-
                    1][i][k]+d[k-1][k][j]));
                }
            }
        }
    }
    int min(int a,int b)
    {
        if(a<b)
            return a;
        else
            return b;
    }
}
public class FloydDemo
{
    public static void main(String arg[])
    {
        floyd f=new floyd();;
        f.read();
        f.floyds();
        f.write();
```

```
    }
}
```

_____Output_____

```
Enter the number of vertices

4
Enter the adjacency matrix

0 1 2 999
1 0 999 3
2 999 0 4
999 3 4 0
Computing all pairs shortest path

d[0]<<=<<

0    1    2    999
1    0    999  3
2    999  0    4
999  3    4    0
d[1]<<=<<

0    1    2    999
1    0    3    3
2    3    0    4
999  3    4    0
d[2]<<=<<

0    1    2    4
1    0    3    3
2    3    0    4
4    3    4    0
d[3]<<=<<

0    1    2    4
1    0    3    3
2    3    0    4
4    3    4    0
d[4]<<=<<

0    1    2    4
1    0    3    3
2    3    0    4
4    3    4    0
```

AIM:

12. Implement in Java, the **0/1 Knapsack** problem using (a) Dynamic Programming method (b) Greedy method.

ALGORITHM:

Knapsack(m,n,w,p,v)

for i←0 to n do

    for j←0 to m do

        if(i=0 or j=0)

            v[i,j]=0;

        else if(j<w[i])
            v[i,j]=v[i-1,j];
        else
            v[i,j]=max(v[i-1,j],v[i-1,j-w[i]]+p[i]);
        end if

    end for

end for

return

PROGRAM:

```
import java.util.Scanner;
class knaptest
{
     int n,m;
     int v[][]=new int[10][10];
     int p[]=new int[10];
     int w[]=new int[10];

     void input()
     {
          int i,j;
          Scanner s=new Scanner(System.in);
          System.out.println("Enter the value of n(number of
          objects):");
          n=s.nextInt();
          for(i=1;i<=n;i++)
```

```
          {
               System.out.println("Enter the weight & profit of-
               -"+i+"---object");
               w[i]=s.nextInt();
               p[i]= s.nextInt();
          }
          System.out.println("Enter the capacity of knapsack:");
          m= s.nextInt();
     }
     int max(int a,int b)
     {
          return a>b?a:b;
     }
     void opt_sol()
     {
          int i,j;
          for(i=0;i<=n;i++)
          {
               for(j=0;j<=m;j++)
               {
                    if(i==0||j==0)
                         v[i][j]=0;
                    else if(j<w[i])
                         v[i][j]=v[i-1][j];
                    else
                         v[i][j]=max(v[i-1][j],v[i-1][j-
w[i]]+p[i]);
}
          }
     }
     void output()
     {
          int i,j,x[];
          x=new int[10];
          System.out.println("The optimal solution matrix is:
");
          for(i=0;i<=n;i++)
          {
               for(j=0;j<=m;j++)
               {
                    System.out.print(v[i][j]+" ");
               }
               System.out.println();

          }
          System.out.println("The optimal solution
is:"+v[n][m]);
```

```
          System.out.println(" ");
          for(i=0;i<=n;i++)
               x[i]=0;
               i=n;
               j=m;
          while(i!=0&&j!=0)
          {
               if(v[i][j]!=v[i-1][j])
               {
                    x[i]=1;
                    j=j-w[i];
               }
               i=i-1;
          }
          System.out.println("objects selected are:");
          for(i=1;i<=n;i++)
               if(x[i]==1)
               System.out.println(i);
     }
}
class Knap
{
     public static void main(String arg[])
     {
          knaptest kt=new knaptest();
kt.input();
          kt.opt_sol();
          kt.output();
     }
}
```

_____Output_____

_____

```
Enter the value of n(number of objects):
4
Enter the weight & profit of---1---object
2
12
Enter the weight & profit of---2---object
1
10
Enter the weight & profit of---3---object
3
20
Enter the weight & profit of---4---object
2
15
```

```
Enter the capacity of knapsack:
5
The optimal solution matrix is:
0 0 0 0 0 0
0 0 12 12 12 12
0 10 12 22 22 22
0 10 12 22 30 32
0 10 15 25 30 37
The optimal solution is: 37 objects selected are: 1 2 4
```

AIM:

13. 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.

1.

PROGRAM:

```
import java.util.Scanner;
public class subsets {
	static int c=0;
	static int w[]=new int[10];
	static int x[]=new int[10];
	static int n,d,i,sum=0;
	public static void main(String[] args)
	{
	Scanner in=new Scanner(System.in);
	System.out.println("Enter number of elements:");
	n=in.nextInt();
	System.out.println("Enter the elements in increasing
order:");
	for(i=0;i<n;i++)
	w[i]=in.nextInt();
	System.out.println("Enter the value of d:");
	d=in.nextInt();
	for(i=0;i<n;i++)
	sum=sum+w[i];
	System.out.println("SUM="+sum);
	if(sum<d || w[0]>d)
```

```
{
System.out.println("Subset is not possible!");
System.exit(0);
}
subset(0,0,sum);
if(c==0)
System.out.println("Subset is not possible!");
}
static void subset(int wsf,int k,int trw )
{
      int i;
      x[k]=1;
      if(wsf+w[k]==d)
      {
      System.out.println("Subset solution="+(++c));
      for(i=0;i<=k;i++)
      {
      if(x[i]==1)
      System.out.println(w[i]);
      }
      return;
      }
      if(wsf+w[k]+w[k+1]<=d)
      subset(wsf+w[k],k+1,trw-w[k]);
      if((wsf+trw-w[k]>=d) && (wsf+w[k+1]<=d))
      {
      x[k]=0;
      subset(wsf,k+1,trw-w[k]);
      }
      }
      }
```

_____Output_____

```
Enter number of elements:
5
Enter the elements in increasing order:
3 6 7 8 9
Enter the value of d:
15
SUM=33
Subset solution=1
6
9
Subset solution=2
7
8
```

AIM:

14. Implement N-Queens problem using Backtracking

PROGRAM:

```
public class nquens {
    static final int N = 4;

   // print the final solution matrix
    static void printSolution(int board[][])
    {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                System.out.print(" " + board[i][j]
                        + " ");
            System.out.println();
        }
    }

    // function to check whether the position is safe or not
    static boolean isSafe(int board[][], int row, int col)
    {
        int i, j;
        for (i = 0; i < col; i++)
            if (board[row][i] == 1)
                return false;


        for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
            if (board[i][j] == 1)
                return false;

        for (i = row, j = col; j >= 0 && i < N; i++, j--)
            if (board[i][j] == 1)
                return false;

        return true;
    }

    // The function that solves the problem using backtracking
    public static boolean solveNQueen(int board[][], int col)
    {
        if (col >= N)
            return true;
```

```
        for (int i = 0; i < N; i++) {
            //if it is safe to place the queen at position i,col ->
place it
            if (isSafe(board, i, col)) {
                board[i][col] = 1;

                if (solveNQueen(board, col + 1))
                    return true;

                //backtrack if the above condition is false
                board[i][col] = 0;
            }
        }
        return false;
    }

    public static void main(String args[])
    {
        int board[][] = { { 0, 0, 0, 0 },
                { 0, 0, 0, 0 },
                { 0, 0, 0, 0 },
                { 0, 0, 0, 0 } };

        if (!solveNQueen(board, 0)) {
            System.out.print("Solution does not exist");
            return;
        }

        printSolution(board);

    }
```

_____Output_____

```
 0   0   1   0
 1   0   0   0
 0   0   0   1
 0   1   0   0
```

AIM:

15. Implement Topological Sorting using DFS based method.

ALGORITHM:

- Create a **stack** to store the nodes.
- Initialize **visited** array of size **N** to keep the record of visited nodes.
- Run a loop from **0** till **N**

- If the node is not marked **True** in **visited** array
- Call the recursive function for topological sort and perform the following steps.
- Mark the current node as **True** in the **visited** array.
- Run a loop on all the nodes which has a directed edge to the current node
- if the node is not marked **True** in the **visited** array:
- Recursively call the topological sort function on the node
- Push the current node in the stack.
- Print all the elements in the stack.

PROGRAM:

```
 import java.io.*;
import java.util.*;

//This class represents a directed graph
//using adjacency list representation
class graph {
 // No. of vertices
 private int V;

 // Adjacency List as ArrayList of ArrayList's
 private ArrayList<ArrayList<Integer> > adj;

 // Constructor
 graph(int v)
 {
     V = v;
     adj = new ArrayList<ArrayList<Integer> >(v);
     for (int i = 0; i < v; ++i)
         adj.add(new ArrayList<Integer>());
 }

 // Function to add an edge into the graph
 void addEdge(int v, int w) { adj.get(v).add(w); }

 // A recursive function used by topologicalSort
 void topologicalSortUtil(int v, boolean visited[],
                          Stack<Integer> stack)
 {
     // Mark the current node as visited.
     visited[v] = true;
     Integer i;

     // Recur for all the vertices adjacent
     // to thisvertex
     Iterator<Integer> it = adj.get(v).iterator();
     while (it.hasNext()) {
         i = it.next();
         if (!visited[i])
             topologicalSortUtil(i, visited, stack);
```

```
        }

        // Push current vertex to stack
        // which stores result
        stack.push(new Integer(v));
    }

    // The function to do Topological Sort.
    // It uses recursive topologicalSortUtil()
    void topologicalSort()
    {
        Stack<Integer> stack = new Stack<Integer>();

        // Mark all the vertices as not visited
        boolean visited[] = new boolean[V];
        for (int i = 0; i < V; i++)
            visited[i] = false;

        // Call the recursive helper
        // function to store
        // Topological Sort starting
        // from all vertices one by one
        for (int i = 0; i < V; i++)
            if (visited[i] == false)
                topologicalSortUtil(i, visited, stack);

        // Print contents of stack
        while (stack.empty() == false)
            System.out.print(stack.pop() + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        // Create a graph given in the above diagram
        graph g = new graph(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);

        System.out.println("Following is a Topological "
                           + "sort of the given graph");
        // Function Call
        g.topologicalSort();
    }
}
```

_____Output_____

```
Following is a Topological sort of the given graph
5 4 2 3 1 0
```

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY
(Effective from the academic year 2021-2022)
SEMESTER – IV
(21CSL45)

LAB MANUAL
(2022-2023)