GLOBAL ACADEMY OF TECHNOLOGY

(An Autonomous Institute under VTU)

Department of Computer Science & Engineering(AI&ML)

Affiliated to VTU, Accredited by NAAC with 'A' grade

RR Nagar, Bengaluru – 560 098

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY MANUAL(INTEGRATED)

IV Semester

Course Code:22CML43

Academic Year 2023-24

[As per the Choice Based Credit System Scheme]

Scheme: 2022

Version1.0

w.e.f 13th May 2024

Editorial Committee

HEMALATHA.M, Dept. of CSE(AI&ML)

Approved by

H.O.D, Department. of CSE(AI&ML)

Document Log

Name of the document	Design and Analysis of Algorithms Laboratory Manual
Scheme	2022
Current version number and date	V 1.0 / 13.05.2024
Subject code	22CML43
Editorial Committee	Prof. Hemalatha.M Assistant professor, Dept. of CSE(AI&ML)
Approved by	HOD, Dept. of CSE(AI&ML)

Table of Contents

Vision of the Institute

Become a premier institution imparting quality education in engineering and management to meet the changing needs of society.

Mission of the Institute

M1: Create environment conductive for continuous learning through quality teaching and learning processes supported by modern infrastructure.

M2: Promote research and innovation through collaboration with industries.

M3: Inculcate ethical values and environmental conscious through holistic education programs.

Vision of the Department

To Become a leading hub of excellence in education, research in the field of Computer Science and Engineering providing AI-driven solutions with holistic development for the needs of the Society.

Mission of the Department

- To Provide aspiring engineers for industry and academia by offering excellent education in emerging AI techniques.
- To equip value-added technical and research-oriented education by satisfying societal needs
- To educate with professional integrity values and ethics for the environment awareness.

PROGRAM EDUCATIONAL OBJECTIVES(PEOs)

- **PEO1:** Design and Develop innovative intelligent systems for the welfare of the Society.
- **PEO2:** Engage in lifelong learning process through higher education and to inculcate innovative ideas in research field.
- **PEO3:** Lead the IT Industry with management and Entrepreneurship skills.

PROGRAM SPECIFIC OUTCOMES(PSOs)

- **PSO1:** To Produce graduates with industry-ready skills with the knowledge of Computer Science & Machine Learning technology based to solve real world problems.
- **PSO2:** Ability to develop many successful applications and designing efficient algorithms for intelligent systems within the realm of artificial intelligence incorporating in data analytics, Natural language processing and Internet of Things.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‘s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Details

Course Name : Design and Analysis of Algorithms Laboratory
Course Code : 22CML43
Course prerequisite : Basics of Programming in C, Data Structures.

Course Objectives

CLO1: Apply mathematical concepts and notations to define a problem.

CLO2: Understand and apply algorithms design techniques.

CLO3: Gain ability to solve real life problems using algorithms techniques.

CLO4: Understand the limitations of Algorithmic power.

COURSE OUTCOMES

Upon successful completion of this course, student will be able to:

Cos	COURSE OUTCOMES
22CML43.1	Explain the basic techniques of analyzing the algorithms using time & space complexity and asymptotic notations.
22CML43.2	Devise algorithms using brute force and Divide and Conquer techniques for a given problem.
22CML43.3	Demonstrate Graph Algorithms using greedy method, Transform and Conquer Approach to model Engineering Problems.
22CML43.4	Employ Dynamic Programming and Decrease & Conquer strategies to solve a given problem.
22CML43.5	Use Back Tracking, Branch and Bound design techniques for solving Computationally hard problems.

Syllabus

Course Code	**22CML43**	**CIE Marks**	**50**
Hours/Week (L: T: P)	**3:0:2**	**SEE Marks**	**50**
No. of Credits	**4**	**Examination Hours**	**03**

Laboratory Experiments

Implement the specified algorithms for the following problems and compute its time complexity under LINUX/Windows environment using Java Programming language.

1. Sort a given set of n integer elements using Quick Sort method and compute its time complexity

2. Sort a given set of n integer elements using Merge Sort method and compute its time complexity.

3. Find Minimum Cost Spanning Tree of a given connected undirected graph using:
i) Kruskal's algorithm.
ii) Prim's algorithm.

4. Write a program to find shortest path using Dijkstra's algorithm.

5. Write a program to implement All-Pairs Shortest Paths problem using Floyd's algorithm.

6. Write a program to implement the 0/1 Knapsack problem using Dynamic Programming.

7. Write a program to Check whether a given graph is connected or not using BFS method.

8. Write a program to print the topological ordering using DFS method.

9. Implement N-Queen's problem using Back Tracking.

10. Design and implement a program to find subset of a given set S = {Sl, S2,.....,Sn} of n positive integers Whose SUM is equal to a given positive integer d. 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.

Conduction of Practical Examination:

1. All laboratory programs are to be included for practical examination.
2. Students are allowed to pick one program from lot.
3. Strictly follow the instructions as printed on the cover page of answer script.
4. Marks distribution:
   Procedure + Conduction + Viva: **15 + 70 +15 = 100 Marks(Scale downed to10)**
5. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

CO-PO-PSO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
22CML43.1	3	3	3	-	3	-	-	2	-	-	-	2	2	3
22CML43.2	3	3	3	-	3	-	-	2	-	-	-	2	2	3
22CML43.3	3	3	3	-	3	-	-	2	-	-	-	2	2	3
22CML43.4	3	3	3	-	3	-	-	2	-	-	-	2	2	3
22CML43.5	3	3	3	-	3	-	-	2	-	-	-	2	2	3
Average	**3**	**3**	**3**		**3**	**-**	**-**	**2**	**-**	**-**	**-**	**2**	**2**	**3**

LAB EVALUATION PROCESS WEEK WISE

EVALUATION OF EACH PROGRAM

S. No	ACTIVITY	MARKS
1	Record +observation +Viva	**10**

INTERNAL ASSESSMENT EVALUATION
(End of Semester)

S. No	ACTIVITY	MARKS
1	Write-up	15
2	Conduction	70
3	Viva	15
	TOTAL	100(Scaled down to 10)

FINAL INTERNAL ASSESSMENT CALCULATION		
S. No	**ACTIVITY**	**MARKS**
1	Average of weekly entries	10
2	Internal Assessment	10
	TOTAL	**20**

LAB RUBRICS

Rubrics for Evaluation of observation:

Attribute	Max Marks	Good	Satisfactory	Poor
		2	1	0
Write Up (W)	2	• Written complete program without any errors with proper indentation. • Written input and expected output for all the test cases.	• Written Program with few logical errors with moderate indentation. • Written input and expected output for few testcases.	• Not submitted • Input and expected output is not written
	Max Marks	2	1	0
Conduction of experiment	2	• All possible cases of input and output demonstrated	• The code is not tested with all possible cases.	• Unable to execute within the allotted
	Max Marks	1	0.5	0
Viva	1	• Able to explain the design of algorithm and logic. • Able to answer all question related to the concept.	• Partially understood the logic of the experiment. • Able to answer 1 out of 3 questions	• Not understoodthe logic of the experiment. Not answering any questions

Rubrics for Evaluation of Record:

Attribute	Max Marks	Good	Satisfactory	Poor
		3-5	1-3	0
Write Up and Time Management	5	• Written complete program without any errors with proper indentation • Written input and expected output for all the test cases. • Index entry promptly done and submits on time	• Written Program with few logical errors with moderate indentation. • Written input and expected output for few testcases.	• Not submitted • Input and expected output is not written

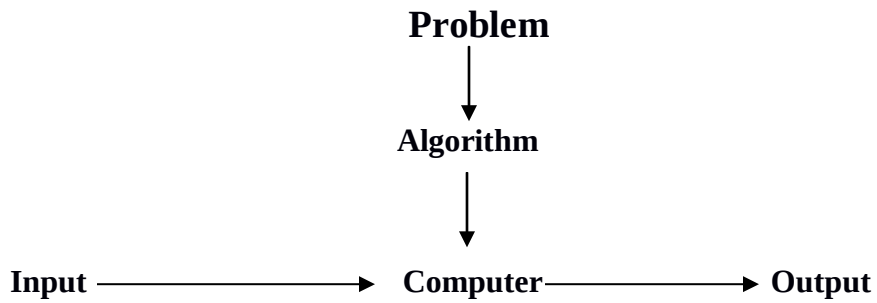
Rubrics for Evaluation of Internal Test:

Attribute	Max Marks	Excellent	Good	Satisfactory	Poor
		11-15	6-10	1-5	0
Write-up	15	• Completes source code. • No syntax and logical errors • All possible inputs listed with expected output	• Completes source code. • Syntax and logical errors exist.	• Incomplete source code. • All possible inputs listed with expected output	• Change of program
	Max Marks	60-70	20-59	11-19	0-10
Execution	70	• Able to debug the program and get the rightset of outputs • Able to modify the Program as per the examiner's Direction	• Able to execute program for few possible inputs without Errors	• Able to clear the errors. • Not able to execute program	• Not able to clear the errors and execute the program
	Max Marks	11-15	6-10	1-5	0
Viva Voce	15	• Able to answer all questions correctly	• Able to answer 3 questions out of 5 questions correctly	• Able to answer2 - 5 questions	• Not able to answer any questions.

CHAPTER 1

BASICS OF ALGORITHMS

An algorithm is a sequence of finite number of unambiguous instructions for solving a problem, i.e for obtaining a required output for any legitimate input in a finite amount of time.



- The non-ambiguity requirement for each step of an algorithm.
- The range of inputs has to be specified carefully.
- The same algorithm can be represented in several different ways.
- Several algorithms for solving the same problem may exist.
- Algorithm for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

Properties Of Algorithms

1. **Input** – an algorithm has zero or more inputs.
2. **Output** – an algorithm has one or more outputs.
3. **Finiteness** – an algorithm must terminate after a finite number of steps.
4. **Definiteness** – each step must be precisely defined.
5. **Effectiveness** – all operations can be carried out exactly in finite time (lesser amount of time).

Algorithm Paradigms

Often there are large collections of problems that can be solved using the same general techniques or paradigms. A few of the most common are described below:

Brute Force: A straightforward approach to solving a problem based on the problem statement and concepts involved. Brute force algorithms are rarely efficient.

Examples:

- Bubble sort.
- Computing the sum of n numbers by direct addition.
- Standard matrix multiplication.
- Linear search.

Divide And Conquer: Perhaps the most famous algorithm paradigm, divide and conquer is based on partitioning the problem into two or more smaller sub-problems, solving them and combining the sub-problem solutions into a solution for the original problem.

Examples:

- Binary Search, Merge sort and quick sort.
- The Fast Fourier Transform (FFT).
- Strassen's matrix multiplication.

Greedy Algorithms: Greedy algorithms always make the choice that seems best at the moment. This is locally optimal choice is made with the hope that it leads to a globally optimal solution. Some greedy algorithms may not be guaranteed to always produce an optimal solution. Greedy algorithms are often applied to combinatorial optimization problems. Given an instance of the problems, there is a set of candidates or feasible solutions that satisfy the constraints of the problem. For each feasible solution there is a value determined by an objective function. An optimal solution minimizes (or maximizes) the value of objective function.

Examples:

- Kruskal's and Prim's minimal spanning tree algorithms.
- Dijkstra‟s single source shortest path algorithm.
- Huffman coding.

Dynamic Programming: A nutshell definition of dynamic programming is difficult, but to summarize, problems which lend themselves to a dynamic programming attack have the following characteristics: We have to search over a large space for an optimal solution. The optimal solution can be expressed in terms of optimal solution to sub-problem. The number of sub-problems that must be solved is small. Dynamic programming algorithms have the following features:

- A recurrence that is implemented iteratively.
- A table, built to support the iteration.
- Tracing through the table to find the optimal solution.
- Examples - Fibonacci number computation, Knapsack problems, Floyd's and Warshall's algorithm.

CHAPTER 2

LAB SYLLABUS PROGRAMS

Program 1

Sort a given set of n integer elements using Quick Sort method and compute its time complexity using java.

ABOUT THE EXPERIMENT:

Quick sort is one of the sorting algorithms working on the Divide and Conquer strategy. In Quick sort the given array elements is divided into two parts based on pivot (a key element) . Pivot element partitions the array into three sub arrays such that

- The left sub array contains all elements which are less than or equal to the pivot
- The middle sub arrays contains pivot
- The right sub array contains all elements which are greater than or equal to the pivot. The innermost loop is so efficient that it runs faster than mergesort and heapsort.

Time Complexity Analysis: Quick Sort
Algorithm Best case performance is **Ω(n log)**
Average case performance is **Θ(1.38*nlog n)**
Worst case Performance is **$O(n^2)$**

APPLICATIONS:

- The sorting algorithm is used for **information searching** and as Quicksort is the fastest algorithm so it is widely used as a better way of searching.
- It is used everywhere where a **stable sort** is not needed.
- Quicksort is a cache friendly algorithm as it has a good locality of reference when used for arrays.
- It is **tail -recursive** and hence all the call optimization can be done.
- It is an in-place sort that does not require any extra storage memory.
- It is used in operational research and event-driven simulation.
- Numerical computations and in scientific research, for accuracy in calculations most of the efficiently developed algorithm uses priority queue and quick sort is used for sorting.
- Variants of Quicksort are used to separate the **Kth smallest** or **largest elements**.

Source Code:

```
package javaprogram;
import java.util.Random;
import java.util.Scanner;
public class Quicksort
{
        static int max=30000;
        public static void main(String[] args)
        {

                int a[]=new int[max];
                long start,end;
                Scanner sobj=new Scanner (System.in);
                System.out.println("Enter the no. of elements to be sorted :");
                int n=sobj.nextInt();
                Random rand=new Random();
                for(int i=0;i<n;i++)
                {
                 a[i]=rand.nextInt(30000);
                }
                System.out.println("Array elements to be sorted are :");
               for(int i=0;i<n;i++)
               {
                 System.out.print(a[i]+" ");
               }
               start=System.nanoTime();
               qsort(a,0,n-1);
               end=System.nanoTime();
               System.out.println("\nThe sorted elements are :");
               for(int i=0;i<n;i++)
               {
                System.out.print(a[i]+" ");
               }
               System.out.println("\nThe time taken to sort is "+(end-start)+"ns");
         }//end of main



       static void qsort(int a[],int low,int high)
                {
                   int s;
                   if(low<high)
                   {
                       s=partition(a,low,high);//s is the final position of pivot element in qsort(a,low,s-1);
```

```
                qsort(a,low,s-1);
                qsort(a,s+1,high);
            }
        }

    static int partition(int a[],int low,int high)
        {
            int pivot,i,j,temp;
            pivot=a[low];
            i=low;
            j=high;
            while(i<=j)
            {
                while(a[i]<=pivot)
                    i++;
                while(a[j]>pivot)
                    j--;
                if(i<j)
                {
                    temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
            }
        a[low]=a[j];
        a[j]=pivot;
        return j;
        }

}
```

OUTPUT: (Plot a Graph for time complexity verses input size)

RUN1	RUN2	RUN3
INPUT:	**INPUT:**	**INPUT:**

Program 2

Sort a given set of n integer elements using Merge Sort method and compute its time complexity using java.

ABOUT THE EXPERIMENT:

Merge sort is a perfect example of a successful application of the divide-and-conquer technique. The steps followed in merge sort are:

- Merge sort works by dividing the given array into two sub arrays of equal size
- The sub arrays are sorted independently using recursion
- The sorted sub array are then merged to get the solution for the original array
- In merge sort given set of elements (n) split into sets a[1]……a[n/2] and a[n/2+1]……a[n].
- Each set is individually sorted and resulting sorted sequence merged to produce a single sorted sequence of n elements

Time Complexity Analysis:

Best/Average/Worst case performance is Θ (n log n)

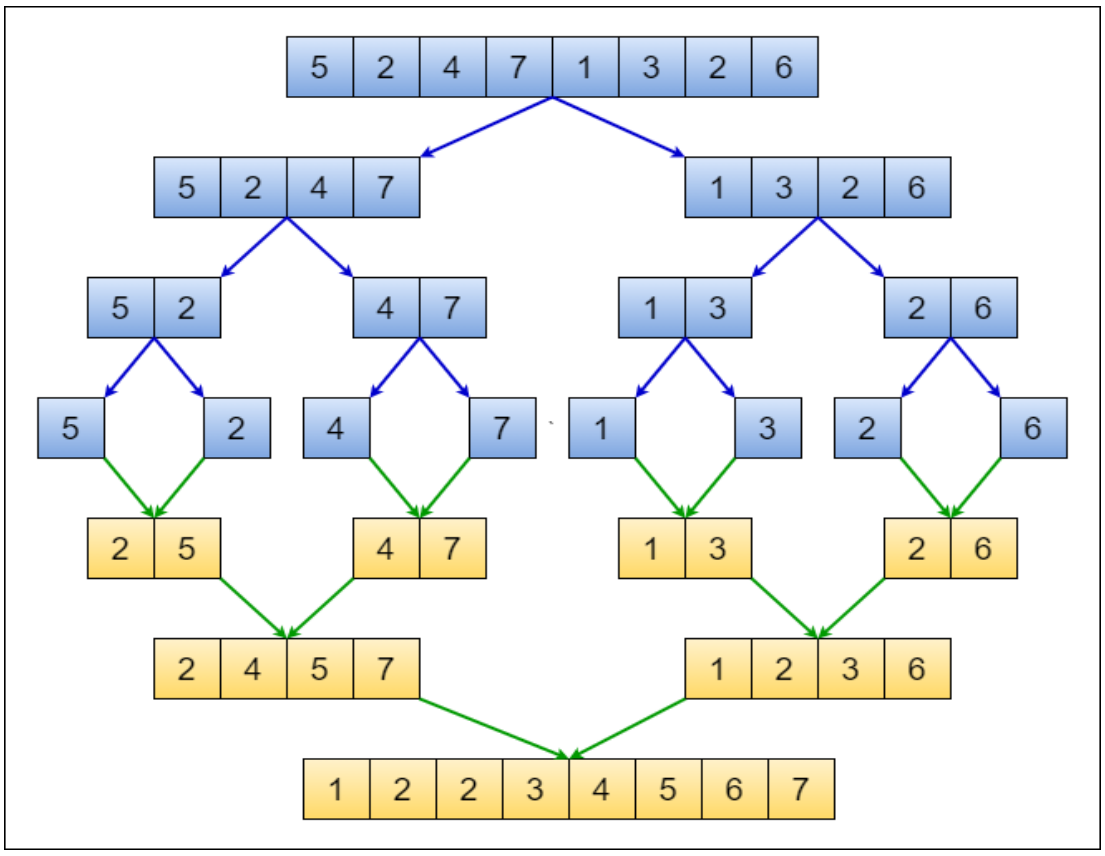


Fig 1:Tree Construction for dividing the array and Tree construction for Merging sorted array.

APPLICATIONS:

- **M**erge sort is the best algorithm to sort a linked list with the complexity of O(n logn).

- Merge sort helps to solve this problem by telling the number of inversion pairs in an unsorted array. The inversion count problem tells how many pairs need to be swapped in order to get a sorted array. Merge sort works best for solving this problem.
- **External sorting:** Merge sort is an external sorting technique. Thus, if we have data of 1GB but the available RAM size is 500MB, then we will use merge sort.

Source Code:

```
import java.util.Random;
import java.util.Scanner;
public class merge_sort
  {
    static int max=30000;
     public static void main(String[] args)
     {
       int a[]=new int[max];
       long start,end;
       Scanner sobj=new Scanner (System.in);
       System.out.println("Enter the no. of elements to be sorted :");
       int n=sobj.nextInt();
       Random rand=new Random();
       for(int i=0;i<n;i++)
       {
           a[i]=rand.nextInt(30000);
       }
       System.out.println("Array elements to be sorted are :");
       for(int i=0;i<n;i++)
       {
            System.out.print(a[i]+" ");
       }
      start=System.nanoTime();
      mergesort(a,0,n-1);
      end=System.nanoTime();
      System.out.println("\nThe sorted elements are :");
      for(int i=0;i<n;i++)
      {
         System.out.print(a[i]+" ");
       }
      System.out.println("\nThe time taken to sort is "+(end-start)+"ns");
  }//end of main
static void mergesort(int a[],int low,int high)
{
   int mid;
   if(low<high)
   {
      mid=(low+high)/2;
      mergesort(a,low,mid);//recursively sort left part
```

```
            mergesort(a,mid+1,high);//recursively sort right part
            merge(a,low,mid,high);
          }
        }
        static void merge(int a[],int low,int mid,int high)
        {
           int i,j,h,k;
           int b[]=new int[max];
           h=low;//h points to first element in first half a[low:mid]
           i=low;
           j=mid+1;//j points to first element in second half a[mid+1:high]
           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];
        }//end of merge
}//end of class merge
```

OUTPUT:

RUN1	RUN2	RUN3
INPUT:	**INPUT:**	**INPUT:**

Program 3

Find Minimum Cost Spanning Tree of a given connected undirected graph using

i) Kruskal's algorithm using java.

APPLICATIONS:

- Landing cables
- TV Network
- Tour Operations
- LAN Networks
- A network of pipes for drinking water or natural gas.
- An electric grid
- Single-link Cluster

Source Code :

```
package javaprogram;
import java.util.Scanner;
public class kruskal
{
        int parent[]=new int[10];
        int find(int m)
        {
            int p=m;
            while(parent[p]!=0)
                    p=parent[p];
           return p;
        }

        void union(int i,int j)
        {
           if(i<j)
               parent[i]=j;
         else
             parent[j]=i;
        }

        void krkl(int[][]a, int n)
        {
          int u=0,v=0,min,k=0,i,j,sum=0;
          while(k<n-1)
          {
              min=999;
              for(i=1;i<=n;i++)
              for(j=1;j<=n;j++)
           if(a[i][j]<min&&i!=j)
```

```
        {
            min=a[i][j];
            u=i;
            v=j;
        }
      i=find(u);
      j=find(v);
      if(i!=j)
      {
          union(i,j);
          System.out.println("("+u+","+v+")"+"="+a[u][v]);
          sum=sum+a[u][v];
          k++;
      }
     a[u][v]=a[v][u]=999;
   }
System.out.println("The cost of minimum spanning tree = "+sum);
}

public static void main(String[] args)
{
    int a[][]=new int[10][10];
    int i,j;
    System.out.println("Enter the number of vertices of the graph");
    Scanner sc=new Scanner(System.in);
    int n;
    n=sc.nextInt();
    System.out.println("Enter the wieghted matrix");
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
            a[i][j]=sc.nextInt();
   kruskal k=new kruskal();
   k.krkl(a,n);
   sc.close();
}

}
```

OUTPUT

RUN 1	RUN 2
INPUT:	**INPUT:**

ii) using Prim's algorithm.

APPLICATIONS:

- All the applications stated in the Kruskal's algorithm's applications can be resolved using Prim's algorithm (use in case of a dense graph).
- Network for roads and Rail tracks connecting all the cities.
- Irrigation channels and placing microwave towers
- Designing a fiber-optic grid or ICs.
- Cluster analysis.
- Pathfinding algorithms used in AI(Artificial Intelligence).
- Game Development

Source Code:

```
package javaprogram;
import java.util.Scanner;
public class prims
{
        public static void main(String[] args)
        {
            int w[][]=new
            int[10][10];
            int n,i,j,s,k=0;
            int min; int sum=0;
            int u=0,v=0; int flag=0;
            int sol[]=new int[10];
            System.out.println("Enter the number of vertices");
            Scanner sc=new Scanner(System.in);
            n=sc.nextInt();
            for(i=1;i<=n;i++)
                sol[i]=0;
            System.out.println("Enter the weighted graph");
            for(i=1;i<=n;i++)
              for(j=1;j<=n;j++)
                 w[i][j]=sc.nextInt();
           System.out.println("Enter the source vertex");
           s=sc.nextInt();
           sol[s]=1; k=1;
          while (k<=n-1)
          {
              min=999;
              for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                     if(sol[i]==1&&sol[j]==0)
                         if(i!=j&&min>w[i][j])
```

```
                    {
                        min=w[i][j]; u=i;
                         v=j;
                    }
            sol[v]=1;
            sum=sum+min;
            k++;
            System.out.println(u+"->"+v+"="+min);
          }
      for(i=1;i<=n;i++)
        if(sol[i]==0)
             flag=1;
      if(flag==1)
        System.out.println("No spanning tree");
      else
        System.out.println("The cost of minimum spanning tree is"+sum);
      sc.close();
  }
  }
```

OUTPUT:

RUN 1	RUN 2
INPUT:	

Program 4

Write a java program to find shortest path using Dijkstra's algorithm.

APPLICATIONS:

- Digital Mapping Services in Google Maps.
- Social Networking Applications.
- Telephone Network.
- IP routing to find Open shortest Path First.
- Fighting Agenda: For example, If a person needs software for making an agenda of flights for customers.
- Designate file server
- Robotic Path

Source Code:

```
package javaprogram;
import java.util.Scanner;
public class dijkstras
{
   int d[]=new int[10];
   int p[]=new int[10];
   int visited[]=new int[10];
   public void dijk(int[][]a, int s, int n)
    {
        int u=-1,v,i,j,min;
        for(v=1;v<=n;v++)
        {
           d[v]=999;
           p[v]=-1;
        }
       d[s]=0;
       for(i=1;i<=n;i++)
       {
          min=999;
          for(j=1;j<=n;j++)
           {
              if(d[j]<min&& visited[j]==0)
               {
                  min=d[j]; u=j;
               }
           }
          visited[u]=1;
          for(v=1;v<=n;v++)
          {
             if((d[u]+a[u][v]<d[v])&&(u!=v)&&visited[v]==0)
```

```
            {
               d[v]=d[u]+a[u][v];
               p[v]=u;
            }
         }
      }
   }
  void path(int v,int s)
  {
    if(p[v]!=-1)
    path(p[v],s);
    if(v!=s)
       System.out.print("->"+v+" ");
  }

  void display(int s,int n)
  {
   int i;
   for(i=1;i<=n;i++)
    {
      if(i!=s)
      {
        System.out.print(s+" ");
        path(i,s);
      }
      if(i!=s)
      System.out.print("="+d[i]+" ");
      System.out.println();
     }
   }

 public static void main(String[] args)
 {
     int a[][]=new int[10][10];
     int i,j,n,s;
     System.out.println("enter the number of vertices");
     Scanner sc = new Scanner(System.in);
     n=sc.nextInt();
     System.out.println("enter the weighted matrix");
     for(i=1;i<=n;i++)
       for(j=1;j<=n;j++)
          a[i][j]=sc.nextInt();
     System.out.println("enter the source vertex");
     s=sc.nextInt();
     dijkstras tr=new dijkstras();
```

```
    tr.dijk(a,s,n);
    System.out.println("the shortest path between source"+s+"to remaining vertices are");
    tr.display(s,n);
    sc.close();
 }
}
```

OUTPUT:

RUN 1	RUN 2
INPUT:	**INPUT:**

Program 5

Write a java program to implement All-Pairs Shortest Paths problem using Floyd's algorithm.

APPLICATIONS for Floyd's algorithm.

- Google Maps
- Social Network Analysis
- To find a maximum bandwidth path in network systems.
- To compare two graphs to find the similarities between them.

Source Code :

```
package javaprogram;
import java.util.Scanner;
public class floyds
{
        int min(int a,int b)
        {
           if(a < b)
             return a;
           else
             return b;
        }

  void flyd(int[][] d,int n)
   {
      int i,j,k;
      for(k=1;k<=n;k++)
          for(i=1;i<=n;i++)
              for(j=1;j<=n;j++)
              {
                  d[i][j]= min(d[i][j], d[i][k] + d[k][j]);
               }
   }

public static void main(String[] args)
{
      int a[][]=new int[10][10];
      int n,i,j;
      System.out.println("enter the number of vertices");
      Scanner sc=new Scanner(System.in);
      n=sc.nextInt();
      System.out.println("Enter the weighted matrix");
      for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
              a[i][j]=sc.nextInt();
```

```
        floyds f=new floyds();
        f.flyd(a, n);
        System.out.println("The shortest path matrix is");
        for(i=1;i<=n;i++)
         {
              for(j=1;j<=n;j++)
              {
                  System.out.print(a[i][j]+" ");
              }
             System.out.println();
         }
      sc.close();
    }
  }
```

OUTPUT:

RUN 1	RUN 2
INPUT:	**INPUT:**

Program 6

Write a java Program to implement the 0/1 Knapsack problem using Dynamic Programming.

Knapsack Problem statement:

Given n objects, their weights wi and profits pi, where i=1 to n , and the capacity of the knapsack is W, the objective is select the objects for the knapsack such that the maximum profit is earned and the weight of the objects chosen shouldn't exceed the capacity of the given knapsack **.**

APPLICATIONS:

- Used as Internet Download Manager.
- Used in preparation for exam paper just a night before exam.
- Think about the same optimizing: cargo ships or trucks load, containers contents- packing pallets, pieces cut from a single sheet of wood, fabric, metal

Source Code:

```
package javaprogram;
import java.util.Scanner;
public class knapsack
{
 public void solve(int[] wt, int[] val, int W, int N)
 {
  int i,j;
  int sol[][] = new int[N + 1][W + 1];
  int selected[] = new int[N+1];
  for ( i = 0; i <= N; i++)
   {
      for ( j = 0; j <= W; j++)
       {
          if(i==0||j==0)
             sol[i][j]=0;
          else if(wt[i]>j)
                sol[i][j]=sol[i-1][j];

          else
                    sol[i][j]=Math.max(sol[i-1][j],(sol[i - 1][j - wt[i]] + val[i]));
       }
   }
   System.out.println("The profit table is:: ");
  for(i=0;i<=N;i++)
  {
   for(j=0;j<=W;j++)
      System.out.print(sol[i][j]+" ");
   System.out.println();
  }
```

```
    System.out.println("The optimal profit obtained is "+sol[N][W]);
    i=N; j=W;
   while (i>0&&j>0)
   {
      if (sol[i][j] !=sol[i-1][j])
       {
           selected[i]=1;
           j = j - wt[i];
       }
     i--;
    }
   System.out.println("\nItems selected : ");
   for(i=1;i<=N;i++)
     if (selected[i] == 1)
        System.out.print(i +" ");
        System.out.println();
   }

  public static void main(String[] args)
   {
      Scanner scan = new Scanner(System.in);
      knapsack ks=new knapsack();
      System.out.println("***** KNAPSACK PROBLEM - DYNAMIC PROGRAMMING *****");
      System.out.println("Enter number of elements ");
      int n = scan.nextInt();
      int wt[] = new int[n + 1];
      int val[] = new int[n + 1];
      System.out.println("\nEnter weight for "+ n +" elements");
      for (int i=1;i<=n;i++)
           wt[i] = scan.nextInt();
      System.out.println("\nEnter profit value for "+ n +" elements");
      for (int i=1;i<=n;i++)
            val[i] = scan.nextInt();
      System.out.println("\nEnter knapsack weight ");
      int W = scan.nextInt();
      ks.solve(wt, val, W, n);
   }
}
```

OUTPUT:

RUN 1	RUN 2
INPUT	**INPUT**

Program 7

Write a java program to check whether a graph is connected or not using BFS method

APPLICATIONS:

- Shortest Path and Minimum Spanning Tree for unweighted graph
- Peer to Peer Networks
- Crawlers in Search Engines
- Social Networking Websites
- GPS Navigation systems.
- Broadcasting in Network.
- Garbage Collection
- Cycle detection in undirected graph

Source Code :

```
package javaprogram;
import java.util.Scanner;
public class bfs
{
    public static void main(String[] args)
        {
                int n,source,i,j,count;
                int a[][]=new int [10][10];
                int v[]=new int[10];
                 System.out.println("Enter no of nodes: ");
                 Scanner s=new Scanner (System.in);
                 n=s.nextInt();
                 System.out.println("\n Read Adjacency matrix \n");
                 for(i=1;i<=n;i++)
                   for(j=1;j<=n;j++)
                            a[i][j]=s.nextInt();
                 for(i=1;i<=n;i++)
                     v[i] = 0;
                 count = 0;
                 for(i=1;i<=n;i++)
                 {
                     if(v[i] == 0)
                      {
                         bfs1(a,n,v,i);
                         count++;
                      }
                 }
                 if(count == 1)
                  System.out.println("Graph is Connected");
                 else
                    System.out.println("Graph is NOT Connected with %d Components\n"+count);
        }
```

```
 public static void bfs1(int a[][], int n, int v[], int source)
 {
  int q[]=new int[10];
  int front=0, rear=-1;
  int node, i;
  v[source] = 1;
  q[++rear] = source;
  while(front <= rear)
  {
     node = q[front++];
     for(i=1;i<=n;i++)
      {
        if(a[node][i] == 1 && v[i] == 0)
          {
              v[i] = 1;
              q[++rear] = i;
          }
      }
  }
 }
}
```

Output:

RUN 1	RUN 2
INPUT:	**INPUT:**

Program 8

Write a java program to perform topological ordering using DFS method.

APPLICATIONS:

- For an unweighted graph, DFS traversal of the graph produces the minimum spanning tree.
- Detecting cycle in a graph
- Path Finding
- Topological Sorting
- To test if a graph is bipartite
- Finding Strongly Connected Components of a graph
- Solving puzzles with only one solution, such as mazes.

Source Code:

```
package javaprogram;
import java.util.Scanner;
public class dfs
 {
      public static int topo[]= new int [10];
      public static int k;
      public static void main(String[] args)
      {
              int n,i,j;
              int a[][]=new int [10][10];
              int v[]=new int [10];
              System.out.println("\n Enter the no of Vertices : ");
              Scanner sobj=new Scanner (System.in);
              n=sobj.nextInt();
              System.out.println("\n Enter the Adjacency matrix\n");
              for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        a[i][j]=sobj.nextInt();

              for(i=1;i<=n;i++)
                   v[i]=0;
              for(i=1;i<=n;i++)
              {
                 if(v[i]==0)
                       dfs(a,n,v,i);
               }
              System.out.println("\n The topological ordering is\n");
              for(i=k;i>=1;i--)
              System.out.println(topo[i]);
      }
```

```
    public static void dfs(int a[][],int n,int v[],int source)
      {
           int i;
           v[source]=1;
           for(i=1;i<=n;i++)
           {
              if(v[i]==0 && a[source][i]==1)
                   dfs(a,n,v,i);
           }
         topo[++k]=source;
      }
    }
```

OUTPUT:

RUN 1	RUN 2
INPUT:	**INPUT:**

Program 9

Write a java program to implement N-Queen's problem using Back Tracking.

APPLICATIONS:

- To Find All possible/feasible solution in computationally hard problem using backtracking.

Source Code :

```
package javaprogram;
public class nqueens
 {
        final int N = 4;

        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();
                }
        }

     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;
        }

        boolean solveNQUtil(int board[][], int col)
        {
                if (col >= N)
                        return true;

                for (int i = 0; i < N; i++)
           {
```

```
            if (isSafe(board, i, col))
             {
                        board[i][col] = 1;
                        if (solveNQUtil(board, col + 1) == true)
                          return true;
                    board[i][col] = 0; // BACKTRACK
                   }
              }

            return false;
        }

        boolean solveNQ()
        {
            int board[][] = { { 0, 0, 0, 0 },
                                        { 0, 0, 0, 0 },
                                        { 0, 0, 0, 0 },
                                        { 0, 0, 0, 0 } };

            if (solveNQUtil(board, 0) == false) {
                System.out.print("Solution does not exist");
                return false;
            }

            printSolution(board);
            return true;
        }

        public static void main(String args[])
        {
            nqueens Queen = new nqueens();
            Queen.solveNQ();
        }
}
```

OUTPUT:

RUN 1	RUN 2
INPUT:	**INPUT:**

Program 10

Design and implement a java program to find subset of a given set S = {Sl, S2,.....,Sn} of n positive integers Whose SUM is equal to a given positive integer d. 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.

APPLICATIONS:

- Computer passwords
- Message verification

Source Code :

```
package javaprogram;
import java.util.Scanner;
import static java.lang.Math.pow;
public class subset
{
        public static void main(String[] args)
        {
                int a[]=new int[10];
                int x[]=new int[10];
                int n,d,sum,present=0;
                int j;
                System.out.println("enter the number of elements of set");
                Scanner sc=new Scanner(System.in);
                n=sc.nextInt();
                System.out.println("enter the elements of set");
                for(int i=1;i<=n;i++)
                    a[i]=sc.nextInt();
                System.out.println("enter the positive integer sum");
                d=sc.nextInt();
                if(d>0)
                 {
                   for(int i=1;i<=Math.pow(2,n)-1;i++)
                    {
                      subset s=new subset();
                      s.subset1(i,n,x);
                      sum=0;
                      for(j=1;j<=n;j++)
                          if(x[j]==1)
                                sum=sum+a[j];
                     if(d==sum)
                      {
                        System.out.print("Subset={");
                        present=1;
```

```
                    for(j=1;j<=n;j++)
                       if(x[j]==1)
                           System.out.print(a[j]+",");
                    System.out.print("}="+d);
                    System.out.println();
                  }
                 }
                }
                if(present==0)
                  System.out.println("Solution does not exists");
               }

    void subset1(int num,int n, int x[])
    {
      int i;
      for(i=1;i<=n;i++)
          x[i]=0;
      for(i=n;num!=0;i--)
       {
         x[i]=num%2;
         num=num/2;
        }
     }
   }
```

Output:

RUN 1	RUN 2	RUN 3
INPUT:	**INPUT:**	**INPUT:**

CHAPTER 4

VIVA QUESTIONS AND ANSWERS

a. What is an algorithm?

An algorithm is a sequence of finite number of unambiguous instructions for solving a problem. i.e., for obtaining a required output for any legitimate input in a finite amount of time.

b. Name some basic Efficiency classes.

Constant, Logarithmic, Linear, nlogn, Quadratic, Cubic, Exponential, Factorial

c. What are algorithm design techniques?

Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing. General design techniques are:

i. Brute force (ii) divide and conquer (iii) Decrease and conquer (iv) transform and conquer (v) Greedy technique (vi) dynamic programming (vii) Backtracking (viii) branch and bound

d. How is the efficiency of the algorithm defined?

The efficiency of an algorithm is defined with the components.

- Time efficiency -indicates how fast the algorithm runs
- Space efficiency -indicates how much extra memory the algorithm needs

e. Write the Analysis for the Quicksort.

O(nlogn) in average and best cases $O(n^2)$ in worst case

f. **Is insertion sort better than the mergesort?**

Insertionsortworksexceedinglyfastonarraysoflessthen16elements,though for large,, n‟ its computing time is $O(n^2)$.

g. **Specify the algorithms used for constructing Minimum cost spanning tree**.

Prims Algorithm, Kruskals Algorithm

h. Write the difference between the Greedy method and Dynamic programming.

Greedy method	Dynamic programming
1. Only one sequence of decision is generated.	1. Many number of decisions are generated.
2. It does not guarantee to give an optimal solution always.	2. It definitely gives an optimal solution always.

i. State the time efficiency of floyd's algorithm

$O(n^3)$.It is cubic.

j. Give the time complexity and space complexity of traveling salesperson problem.

Time complexity is O (n^2 2^n).

Space complexity is O (n 2^n).

k. What is the average case complexity for merge sort algorithm?

Time complexity is O(nlogn)

l. Quick sort uses _

Divide and Conquer Technique

m. What is the best case complexity for insertion sort

Time complexity is O(n)

n. **Which of the following standard algorithms is not Dynamic Programming based**.

- Bellman–Ford Algorithm for single source shortest path
- Floyd Warshall Algorithm for all pairs shortest paths
- Knapsack problem
- Prim‟s Minimum Spanning Tree
- Prim‟s Minimum Spanning Tree is a Greedy Algorithm. All other are dynamic programming based.

o. **The recurrence relation capturing the optimal time of the Tower of Hanoi problem with n discs is**.

(a) t(n) = 2t(n – 2) +2

(b) t(n) = 2t(n – 1) +n

(c) t(n) = 2t(n/2) + 1

(d) t(n) = 2t(n – 1) + 1

p. Define spanning tree

A spanning tree of a connected graph is its connected acyclic subgraph (i.e a tree) that contains all vertices of a graph.

q. Define Minimum Spanning tree.

A Minimum Spanning tree of a weighted connected graph is its spanning tree of the smallest weight, where weight of the tree is defined sum of the weights on all its edges.

r. Define Kruskal's Algorithm

Kruskal's algorithm looks at a minimum spanning tree for a weighted connected graph G = (V,E) as an acyclic subgraph with | V| - 1 edges for which the sum of the edge weights is the smallest

s. Define Prim's Algorithm

Prim's algorithm is a greedy algorithm for constructing a minimum spanning tree of a weighted connected graph. It works by attaching to a previously constructed subtree a vertex to the vertices already in the tree.

t. Explain Warshalls algorithm

Warshall's algorithm constructs the transitive closure of a given digraph with n vertices through a series of n by n Boolean matrices R(0), ……, R(k-l)R(k),…….., R(n) Each of these matrices provides certain information about directed paths in the digraph.

u. Explain All-pair shortest-paths problem

Given a weighted connected graph (undirected or directed), the all pairs shortest paths problem asks to find the distances(the lengths of the shortest path) from each vertex to all other vertices.

v. Explain Knapsack problem

Given n items of known weights w1,w2.....wn and values v1,v2.....vn and aknapsackofcapacityW,findthemostvaluablesubsetoftheitemsthatfitintotheknapsack.(Assuming all the weights and the knapsack's capacity are positive integers the itemvalues do not have to be integers.)

w. Explain ²Traveling salesman problem"?

A salesman has to travel n cities starting from any one of the cities and visit the remaining cities exactly once and come back to the city where he started his journey in such a manner that either the distance is minimum or cost is minimum. This is known as traveling salesman problem.

x. Explain Backtracking

The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows.

- If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component.
- If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option

y. Explain promising and non-promising node.

A node in a state space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise it is called non- promising.

z. Explain N-Queens problem.

The problem is to place N queens on an N by N chessboard so that no two queens attack each other by being in the same row or same column or on the same diagonal.

aa. Explain Subset-Sum Problem.

We consider the subset-sum problem: Find a subset of a given set S={S1,S2,.....Sn}of N positive integers whose sum is equal to a given positive integer d.

bb. Explain Knapsack Problem.

Find the most valuable subset of n items of given positive integer weights and values that fit into a knapsack of a given positive integer capacity.

cc. Define order of growth.

The efficiency analysis framework concentration the order of growth of analgorithm's basic operation count as the principal indicator of the algorithms efficiency. To compare and rank such orders of growth we use three notations

- (Big oh)notation
- Ω (Big Omega) notation&
- Θ (Big Theta)notation

dd. What is the use of Asymptotic Notations?

The notations O, Ω and Θ and are used to indicate and compare the asymptotic orders of growth of functions expressing algorithm efficiencies.

CHAPTER 5

Additional Programs

1) Design and implement the presence of Hamiltonian Cycle in an undirected Graph G of n vertices using java.

```
import java.util.*;
 class Hamiltoniancycle
    {
  private int adj[][],x[],n;
  public Hamiltoniancycle()
  {
      Scanner src = new Scanner(System.in);
      System.out.println("Enter the number of nodes");
      n=src.nextInt();
      x=new int[n];
      x[0]=0;
      for (int i=1;i<n; i++)
            x[i]=-1;
      adj=new int[n][n];
      System.out.println("Enter the adjacency matrix");
      for (int i=0;i<n; i++)
      for (int j=0; j<n; j++)
          adj[i][j]=src.nextInt();
 }
 public void nextValue (int k)
{    int i=0;
     while(true)
    {    x[k]=x[k]+1;
         if (x[k]==n)
             x[k]=-1;
         if (x[k]==-1)
                return;
         if (adj[x[k-1]][x[k]]==1)
          for (i=0; i<k; i++)
                if (x[i]==x[k])
                    break;
          if (i==k)
               if (k<n-1 || k==n-1 && adj[x[n-1]][0]==1)
                    return;
       }
 }
```

```
   public void getHCycle(int k)
    {
         while(true)
         {
              nextValue(k);
              if (x[k]==-1)
                    return;
              if (k==n-1)
             {
                  System.out.println("\nSolution : ");
                  for (int i=0; i<n; i++)
                         System.out.print((x[i]+1)+" ");
                   System.out.println(1);
             }
            else
                   getHCycle(k+1);
         }
    }

 public static void main(String args[])
  {
         Hamiltoniancycle obj=new Hamiltoniancycle();
         obj.getHCycle(1);
   }
}
```

2. Implement in Java, the 0/1 Knapsack problem using Greedy method. .

```
import java.util.Scanner;
public class KnapsackGT
{
 double weight[];   double profit[];  double ratio[];
 double cap;
 int nItems;
 KnapsackGT()
{
   Scanner scan = new Scanner(System.in);
   System.out.println("********* KNAPSACK PROBLEM-GREEDY METHOD*********");
   System.out.println("Enter the number of items in the store: ");
   nItems = scan.nextInt();
   System.out.println("Enter the (weight and profit) of items: ");
   weight = new double[nItems];
   profit = new double[nItems];
   ratio = new double[nItems];
   for (int i = 0; i <nItems; ++i)
    {
      weight[i] = scan.nextDouble();
      profit[i] = scan.nextDouble();
      ratio[i] = profit[i] / weight[i];
    }
   System.out.println("Enter the capacity of the knapsack: ");
   cap = scan.nextDouble();
}
int getNext()
{
  double max = 0;
  int index = -1;
  for (int i = 0; i <profit.length; i++)
   {
     if (ratio[i] > max)
       {
         max = ratio[i];
         index = i;
       }
   }
return index;
}
```

```
void fill()
{
  double cW = 0; //current weight
  double cP = 0; //current profit
  double select[]=new double[nItems];//marking item selection
  while (cW< cap)
  {
     int item = getNext();        //next max ratio
    if (item == -1)         //No items left
     {
       break;
     }
    if (cW + weight[item] <= cap)
     {
       cW += weight[item];
       cP += profit[item];
       ratio[item] = 0; //mark as used for the getNext() (ratio)function
       select[item]=1;
     }
    else
     {
      select[item]=(cap-cW)/weight[item];
      cP += (ratio[item] * (cap - cW));
      cW += (cap - cW);
      break; //the knapsack is full
     }
   }
System.out.println("\nItems Selected Fraction Selected(0/1/Partial) ");
System.out.println("************************************************");
for(int i=0;i<nItems;i++)
System.out.println("\t"+(i+1)+"\t\t"+select[i]);
System.out.println("\nMax Profit = " + cP + ", Max Weight = " + cW);
}
public static void main(String[] args)
{
  KnapsackGT ks = new KnapsackGT();
  ks.fill();
}
}
```

APPENDIX

Time Complexity of Sorting Algorithms

Algorithm Name	Average	Best	Worst	Space	Stability	Remarks
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Constant	Stable	Always use a modified bubble sort
Modified Bubble sort	$O(n^2)$	O(n)	$O(n^2)$	Constant	Stable	Stops after reaching a sorted array
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Constant	InStable	Even a perfectly sorted input requires scanning the entire array
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$	Constant	Stable	In the best case (already sorted), every insert requires constant time
Heap Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Constant	Instable	By using input array as storage for the heap, it is possible to achieve constant space
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Depends	Stable	On arrays, merge sort requires O(n) space; on linked lists, merge sort requires constant space
Quicksort	Ω (n*log(n))	Θ(1.38n*log(n))	$O(n^2)$	Constant	InStable	Randomly picking a pivot value (or shuffling the array prior to sorting) can help avoid worst case scenarios such as a perfectly sorted Array.