ANALYSIS AND DESIGN OF ALGORITHMS

Lab Manual

Department of Computer Science and Engineering
The NorthCap University, Gurugram

ANALYSIS AND DESIGN OF ALGORITHMS

Lab Manual

CSL 230

Ms. Monika Lamba

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2022-23

Published by:

School of Engineering and Technology

Department of Computer Science & Engineering

The NorthCap University Gurugram

• **Laboratory Manual is for Internal Circulation only**

Copying or facilitating copying of lab work comes under cheating and is considered as use of unfair means. Students indulging in copying or facilitating copying shall be awarded zero marks for that particular experiment. Frequent cases of copying may lead to disciplinary action. Attendance in lab classes is mandatory.

Labs are open up to 7 PM upon request. Students are encouraged to make full use of labs beyond normal lab hours.

PREFACE

Analysis and Design of Algorithms Lab Manual is designed to meet the course and program requirements of NCU curriculum for B.Tech 2nd year students of CSE branch. The concept of the lab work is to give brief practical experience for basic lab skills to students. It provides the space and scope for self-study so that students can come up with new and creative ideas.

The Lab manual is written on the basis of "teach yourself pattern" and expected that students who come with proper preparation should be able to perform the experiments without any difficulty. Brief introduction to each experiment with information about self-study material is provided. The laboratory exercises will familiarize the students with basic concepts of algorithm development and programming skills. The exercises include designing and analysis of basic algorithms like sorting and searching and will gradually develop programs for advanced techniques such as dynamic programming and greedy algorithms. The lab exercises will also enable the students to gain insights to advanced graph algorithms such as minimum spanning trees and shortest paths, NP-completeness theory. Students are expected to come thoroughly prepared for the lab. General disciplines, safety guidelines and report writing are also discussed.

The lab manual is a part of curriculum for The NorthCap University, Gurugram. Teacher's copy of the experimental results and answer for the questions are available as sample guidelines.

We hope that lab manual would be useful to students of CSE branch and author requests the readers to kindly forward their suggestions / constructive criticism for further improvement of the work book.

Author expresses deep gratitude to Members, Governing Body-NCU for encouragement and motivation.

Authors
The NorthCap University
Gurugram, India

CONTENTS

SYLLABUS

1. Department:	**Department of Computer Science and Engineering**

2. Course Name: Analysis and Design of Algorithms	**3. Course Code**	**4. L-T-P**	**5. Credits**
	CSL230	3-0-2	4

6. Type of Course (Check one):	Programme Core	✓	Programme Elective		Open Elective	

7. Pre-requisite(s), if any: Algebra, Programming Language

8. Frequency of offering (check one):

Odd		Even	✓	Either semester		Every semester	

9. Brief Syllabus:

This course is an introduction to analysis of algorithms. The course will start with designing and analysis of basic algorithms like sorting and searching and will gradually cover advanced techniques such as dynamic programming and greedy algorithms. Throughout the course, you will gain insights to advanced graph algorithms such as minimum spanning trees and shortest paths, NP-completeness theory. At the end of this course, students will be able to design algorithms for various computing problems and analyze the time and space complexity of algorithms. Students will be able to Critically analyze the different algorithm design techniques for a given problem as well as modify existing algorithms to improve efficiency

Total lecture, Tutorial and Practical Hours for this course
(Take 15 teaching weeks per semester): 75 hours

The class size is maximum 30 learners.

	Practice	
Lectures: 45 hours	**Tutorials : 1**0 hours	**Lab Work: 2**0 hours

10. Course Outcomes (COs)

Possible usefulness of this course after its completion i.e., how this course will be practically useful to him once it is completed.

CO 1	Design and analysis of algorithms for a given problem.

CO 2	Analyse various complexity measures and the performance of algorithms.
CO 3	Apply and analyse the complexity of certain divide and conquer, greedy, and dynamic programming algorithms.
CO 4	Explain and apply backtracking algorithms.
CO 5	Ability to design and analyse branch and bound techniques to deal with some hard problems.
CO 6	Understand the classes P, NP, and NP-Complete and be able to prove that a certain problem is NP-Complete.

11. UNIT WISE DETAILS **No. of Units: 6**

Unit Number: 1	**Title: Introduction to algorithms**	**No. of hours: 7**
Content Summary: Role of algorithms in computing, Algorithms as technology, analyzing and designing algorithms, Growth of Functions, Asymptotic notations, Recurrences, Substitution method, Recursion tree method, Master method.		
Unit Number: 2	**Title: Divide and Conquer**	**No. of hours: 7**
Content Summary: General method, binary search, merge sort, quick sort, selection sort, heap sort, insertion sort, Stassen's matrix multiplication algorithms and analysis of algorithms for these problems.		
Unit Number: 3	**Title: Greedy and Dynamic Programming**	**No. of hours: 15**
Content Summary: General method, knapsack problem, job sequencing with deadlines, minimum spanning trees (Kruskal's Algorithm, Prim's Algorithm), Shortest path algorithm (Dijkstra's Algorithm) and analysis of these problems. BFS, DFS, Activity selection problem. Dynamic Programming: General method, Principle of optimality, 0/1- knapsack, the traveling salesperson problem, Optimal binary search tree.		
Unit Number: 4	**Title: Backtracking**	**No. of hours: 8**
Content Summary: General method, 8-queen's problem, subset sum problem, Graph Coloring, Hamiltonian cycles, analysis of these problems.		
Unit Number: 5	**Title: Branch and Bound**	**No. of hours: 5**
Content Summary: Introduction to Branch and Bound, LC search and FIFO search, 0/1- knapsack and traveling salesperson problem, efficiency considerations.		
Unit Number: 6	**Title: NP and NP complete**	**No. of hours: 3**
Content Summary: Basic concepts, Cook's theorem, NP hard graph and NP scheduling problems some simplified NP hard problems.		

12. Brief Description of Self-learning components by students (through books/resource material etc.):

Supplementary MOOC Courses
https://onlinecourses.nptel.ac.in/noc21_cs22/preview
Design and Analysis of algorithms - Course (swayam2.ac.in)

GATE/NET/other PSU Exams
Algorithms | Subject Wise Questions – AcademyEra
Gate CSE Question Bank – AcademyEra
https://www.geeksforgeeks.org/gate-corner-2-gq/
Algorithm (gradeup.co)

13. Books Recommended :

Textbooks:

1. Ellis Horowitz, Sartaj Sahani, Sanguthevar Rajashekaran, "Fundamentals of Computer Algorithms", Orient Black Swan, 2nd Edition, 2008.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms", MIT Press, 3rd Edition, 2009.

Reference Books:

1. J. Kleinberg and E. Tardos, "Algorithm Design", Pearson,1st Edition, 2013.

Reference Websites: (nptel, swayam, coursera, edx, udemy, lms, official documentation weblink)

https://swayam.gov.in/nd1_noc20_cs10/preview
The Design and Analysis of Algorithm Masterclass [2019] | Udemy
Design and Analysis of Algorithms | Udemy

eBooks:

- Horowitz and Sahani, Fundamentals of Computer Algorithms, 2ND Edition - PDF Drive
- Introduction to Algorithms, Third Edition - PDF Drive
- Algorithms illuminated Part 2 Graph Algorithms and Data Structures by Tim Roughgarden - PDF Drive
- Python Algorithms: Mastering Basic Algorithms by Magnus Lie Hetland - PDF Drive

Interview/Placement related Commonly asked Questions:

- Top 8 Algorithm Interview Questions And Answer {Updated For 2020} (educba.com)

- Top 25 Algorithm Interview Questions - javatpoint
- 19 Essential Algorithm Interview Questions and Answers (toptal.com)
- Commonly Asked Algorithm Interview Questions | Set 1 - GeeksforGeeks

1. INTRODUCTION

That 'learning is a continuous process' cannot be over emphasized. The theoretical knowledge gained during lecture sessions need to be strengthened through practical experimentation. Thus, practical makes an integral part of a learning process. The algorithms learnt during the theory classes will be implemented in the lab sessions which will help the student to build strong and deeper understanding of the concepts.

COURSE OBJECTIVES:

1. Design and analysis of algorithms for a given problem.
2. Analyse various complexity measures and the performance of algorithms.
3. Apply and analyse the complexity of certain divide and conquer, greedy, and dynamic programming algorithms.
4. Explain and apply backtracking algorithms.
5. Ability to design and analyse branch and bound techniques to deal with some hard problems.
6. Understand the classes P, NP, and NP-Complete and be able to prove that a certain problem is NP-Complete.

2. LAB REQUIREMENTS

Requirements	Details
Software Requirements	C/C++/Java
Operating System	Any Operating System
Hardware Requirements	
Required Bandwidth	NA

3. GENERAL INSTRUCTIONS

3.1 General discipline in the lab

- Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
- Students will not be allowed to enter late in the lab.
- Students will not leave the class till the period is over.
- Students should come prepared for their experiment.
- Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
- Students must get the connection of the hardware setup verified before switching on the power supply.
- Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
- Violating the above code of conduct may attract disciplinary action.
- Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.

3.2 Attendance

- Attendance in the lab class is compulsory.
- Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
- On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case.

3.3 Preparation and Performance

- Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self-study reference is provided on LMS.
- Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
- Each student is required to write a complete report of the experiment he has performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
- Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
- Refer **Annexure 1** for Lab Report Format.

4. LIST OF EXPERIMENTS

Sr. No.	Title of the Experiment	Software /Hardware based	Unit covered	Time Required
1.	Implement and Calculate the time and space complexity of following programs: i. Factorial of a number ii. Fibonacci series	C/C++/Java	1	1 hour
2.	Implement Insertion/Selection sort algorithm and compute its time and space complexities.	C/C++/Java	2	2 hours
3.	Implement Merge sort/Quick sort algorithm. Compute its time and space complexities	C/C++/Java	2	2 hours
4.	Implement Strassen's matrix algorithm. Compute its time complexity.	C/C++/Java	2	1 hour
5.	Implement fractional knapsack algorithm.	C/C++/Java	3	2 hours
6.	Implement Prim's and Kruskal's algorithms. Compute and compare their space and time complexities.	C/C++/Java	3	2 hours
7.	Implement 0/1 Knapsack algorithm.	C/C++/Java	3	2 hours
8.	Implement travelling salesman algorithm.	C/C++/Java	4	2 hours
	Value Added Experiments			
1.	Implement 8 Queen's Problem	C/C++/Java	4	2 hours
2.	Implement Hamiltonian and Graph coloring algorithm.	C/C++/Java	4	2 hours
3.	Implement 0/1 Knapsack problem using Branch and Bound	C/C++/Java	5	2 hours

5. FLIP EXPERIMENTS

6. LIST OF PROJECTS: NA

7. RUBRICS

Marks Distribution	
Total	70
Online course(NPTEL)	10
Regular evaluation/ file/ mid term	30
Create GUI for Demonstration of any graph based algorithm/ Hackerank problem	10
End term viva	20

Annexure 1

ANALYSIS AND DESIGN OF ALGORITHMS

(CSL 230)

Lab Practical Report

Faculty Name: Dr. Anshul Bhatia

Student Name: Piyush Gambhir

Roll No.: 21CSU349.

Semester: IV

Group: AL-3 (CS-IV-AIML-B)

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2022-2023

THE NORTHCAP UNIVERSITY
Workbook

Analysis and Design of Algorithms (CSL 230) Lab
2022-23

Analysis and Design of Algorithms

(CSL 230)

Lab Workbook

THE NORTHCAP UNIVERSITY

Faculty Name: Dr. Anshul Bhatia

Student Name: Piyush Gambhir

Roll No.: 21CSU349

Semester: IV

Group: AL-3 (AIML-B)

A. Anshul
28/4/23

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122017, India

Session 2022-23

THE NORTHCAP UNIVERSITY
Workbook

Analysis and Design of Algorithms (CSL 230) Lab
2022-23

Student Name: Piyush Gambhir

Student Roll No.: 21CSU349

INDEX

S. No	Experiment Title	Date of Experiment	Date of Submission	Sign student	CO Covered	Sign Faculty	MARKS
1.	Implement and Calculate the time and space complexity of following programs: I)Factorial of a number II)Fibonacci series	06.01.2023	20.01.2023	Piyush	CO-1	20/1/23	8
2.	Implement Insertion/Selection sort algorithm and compute its time and space complexities.	20.01.2023 03.02.2023	20.01.2023 03.02.2023	Piyush	CO-2	3/2/23	4
3.	Implement Merge sort/Quick sort algorithm. Compute its time and space complexities	03.02.2023 10.02.2023	10.02.2023 10.02.2023	Piyush	CO-2	10/2/23	4
4.	Implement Strassen's matrix algorithm. Compute its time complexity.	24.02.2023	24.02.2023	2 Piyush	CO-2	24/2/23	2
5.	Implement fractional knapsack algorithm.	03.03.2023	03.03.2023	2 Piyush	CO-3	3/3/23	2
6.	Implement Prim's and Kruskal's algorithms. Compute and compare their space and time complexities.	03.03.2023 17.03.2023	03.03.2023 17.03.2023	Piyush Piyush	CO-3	3/3/23 17/3/23	2+2 =4
7.	Implement 0/1 Knapsack algorithm using Dynamic programming	14.04.2023	14.04.2023	Piyush	CO-3	14/4/23	2
8.	Implement travelling salesman algorithm using Dynamic Programming	14.04.2023	14.04.2023	Piyush	CO-3	14/4/23	2
9.	Implement 8- queens problem using backtracking	21.04.2023	28.04.2023	Piyush	CO-4	28/4/23	2
10.	Implement 0/1 Knapsack problem using Branch and bound	21.04.2023	28.04.2023	Piyush	CO-5	28/4/23	2

EXPERIMENT NO. 1

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 06.01.2023	
Faculty Signature:	
Marks:	

Objective(s):

The students will be able to design and perform analysis of algorithms for factorial of a number and Fibonacci number by applying different notations.

Outcome:

After completion of lab, students will be able to compare complexities of iterative versus recursive algorithm.

Problem Statement:

1.1 Write a program for factorial of a number (Iterative/ Recursive) and calculate the time and space complexity of the program.
1.2 Write a program for Fibonacci series (Recursive) and calculate the time and space complexity of the program

Background Study:

Analysis of Fibonacci algorithm

The factorial algorithm can be expressed using simple recursion:

fun fact 0 = 1

fact n = n * fact (n - 1);

Looking at the computation that has to be done, we might identify three things that will consume time: integer multiplication, integer subtraction and recursive function calls. Let us ignore the cost of making recursive calls, and suppose that the cost of a multiplication is M and that that of a subtraction is S. We can then define a function T(*n*), meaning “the time required by the algorithm to compute *n*!”, in a very similar form to that of the actual algorithm:

$T(0) = 0, T(n) = M + S + T(n - 1)$ for $n > 0$

From this,

$T(n) = (M + S) + T(n - 1) = 2(M + S) + T(n - 2) = \ldots = n(M + S) + T(n - n) = n(M + S)$

If we regard M and S as being constants, this expression indicates that the time to compute *n*! is proportional to *n* so, for example, computing (2*n*)! will take twice as long as computing *n*! does. A more accurate expression for T(n) would be:

$T(n) = (n - 1)\lg(n - 1)M + S + T(n - 1)$

Analysis of Fibonacci algorithm:

In Fibonacci algorithm, there are two conspicuous things that consume time: integer addition and recursive function calls. Letting *A* be a constant representing the time required for a simple addition, we can write down a function *T*(*n*) meaning "the time required by the algorithm to compute the *n*-th Fibonacci number":

$$\begin{aligned} T(0) &= 0 \\ T(1) &= 0 \\ T(n) &= A + T(n-2) + T(n-1) \quad \text{for } n > 1 \end{aligned}$$

Using appropriate solution techniques, we discover (to our slight horror) that *T*(*n*) is roughly proportional to 2^n. This is a very fast rate of growth, and helps to explain why our SML implementations run very slowly, even for modest values of *n*. The problem is the pair of recursive calls, which duplicate much work. A little thought leads to the alternative fastfib algorithm that eliminates one of the recursions, and so has:

$$T(n) = A + T(n-1)$$

which is of similar style to the earlier factorial time analysis. Thus, the time requirement is proportional to *n* — a dramatic improvement. Again, as with the factorial algorithm, we might worry about whether the addition operations can be done simply.

Question Bank:

1. Which of the following problems can't be solved using recursion?

a) Factorial of a number
b) Nth Fibonacci number
c) Length of a string
d) Problems without base case

2. Recursion is a method in which the solution of a problem depends on ____________

a) Larger instances of different problems
b) Larger instances of the same problem
c) Smaller instances of the same problem
d) Smaller instances of different problems

3. In recursion, the condition for which the function will stop calling itself is ____________

a) Best case
b) Worst case
c) Base case
d) There is no such condition

4. Which of the following statements is true?

a) Recursion is always better than iteration
b) Recursion uses more memory compared to iteration

c) Recursion uses less memory compared to iteration
d) Iteration is always better and simpler than recursion

5. How is time complexity measured?

a) By counting the number of statements in an algorithm
b) By counting the number of primitive operations performed by the algorithm on a given input size
c) By counting the size of data input to the algorithm
d) None of the above

6. Which of the following does NOT belong to the family of notations?

a) Big (O)
b) Big (Ω)
c) Big (θ)
d) Big (⋈)

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement 1.1

Code:

```
/*
1.1 Write a program for factorial of a number (Iterative/ Recursive)
and calculate the time and space complexity of the program.
*/

import java.util.Scanner;

/**
 * experiment_1_problem_statement_1
 */
public class experiment_1_problem_statement_1 {

    // Recursive function to calculate the factorial of a number

    /*
     * Time Complexity: O(n) - Linear Time - Number of recursive calls
     * Space Complexity: O(n) - Recusive Stack Space to store the
recursive calls
     */
    public static int recursiveFactorial(int n) {
        if (n == 0) {
            return 1;
        } else {
            return n * recursiveFactorial(n - 1);
        }
    }

    // Iterative function to calculate the factorial of a number

    /*
     * Time Complexity: O(n) - Linear Time - Number of iterations
     * Space Complexity: O(1) - Constant Space - No extra space is
used
     */
    public static int iterativeFactorial(int n) {
        int fact = 1;
        for (int i = 1; i <= n; i++) {
            fact = fact * i;
```

```java
        }
        return fact;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        System.out.println("Factorial(Recursive) of " + n + " is " +
recursiveFactorial(n));
        System.out.println("Factorial(Iterative) of " + n + " is " +
iterativeFactorial(n));
    }
}
```

Output:

```
PowerShell

PowerShell 7.3.2

 A new PowerShell stable release is available: v7.3.3
 Upgrade now, or check out the release page at:
   https://aka.ms/PowerShell-Release?tag=v7.3.3

PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230
- Analysis and Design
-CSL230-ADA-Lab_Manual\Experiment 1> java experiment_1_problem_statement_1
Enter a number: 10
Factorial(Recursive) of 10 is 3628800
Factorial(Iterative) of 10 is 3628800
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230
- Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 1>
```

Problem Statement 1.2

Code:

```
/*
1.2 Write a program for Fibonacci series (Recursive) and calculate the
time and space complexity of the program
*/

import java.util.Scanner;

/**
 * experiment_1_problem_statement_1
 */
public class experiment_1_problem_statement_2 {

    // Iterative function to calculate the Fibonacci of a number

    /*
     * Time Complexity: O(n) - Linear Time - Number of iterations
     * Space Complexity: O(1) - Constant Space - No extra space is
used
     */

    public static int iterativeFibonacci(int n) {
        int a = 0, b = 1, c = 0;
        if (n == 0) {
            return a;
        } else if (n == 1) {
            return b;
        } else {
            for (int i = 2; i <= n; i++) {
                c = a + b;
                a = b;
                b = c;
            }
```

```
            return c;
        }
    }

    // Recursive function to calculate the Fibonacci of a number

    /*
     * Time Complexity: O(2^n) - Exponential Time - Number of
recursive calls
     * Space Complexity: O(n) - Recusive Stack Space to store the
recursive calls
     */
    public static int recursiveFibonacci(int n) {
        if (n == 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            return recursiveFibonacci(n - 1) + recursiveFibonacci(n -
2);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        System.out.println("Fibonacci(Recursive) of " + n + " is " +
recursiveFibonacci(n) + ".");
        System.out.println("Fibonacci(Iterative) of " + n + " is " +
iterativeFibonacci(n) + ".");
    }
}
```

Output:

```
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semeste
r)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Exper
iment 1> javac .\experiment_1_problem_statement_2.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semeste
r)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Exper
iment 1> java experiment_1_problem_statement_2
Enter a number: 10
Fibonacci(Recursive) of 10 is 55.
Fibonacci(Iterative) of 10 is 55.
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semeste
r)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Exper
iment 1>
```

EXPERIMENT NO. 2

Student Name and Roll Number: Piyush Gambhir – 21CSU349
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)
Date: 20-01-2023 – 03-02-2023
Faculty Signature:
Marks:

Objective:

Perform insertion sort algorithm and compute its time and space complexities.

Outcome:

Students will be able to calculate execution time and space complexity of the exchange sort algorithms.

Problem Statement:

2.1 Write a program to implement insertion sort algorithm.
2.2 Write a program to implement selection sort algorithm.

Background Study:

Analysis of the Insertion Sort:

To insert the last element, we need at most N-1 comparisons and N-1 movements. To insert the N-1st element we need N-2 comparisons and N-2 movements. To insert the 2nd element, we need 1 comparison and one movement.
To sum up:
2* (1 + 2 + 3 +… N - 1) = 2 * (N - 1) * N / 2 = (N-1) * N = Θ (N2)
If the greater part of the array is sorted, the complexity is almost O(N). The average complexity is proved to be = Θ (N^2).
Analysis of the Selection Sort:

- 1st iteration of outer loop: inner executes N - 1 times
- 2nd iteration of outer loop: inner executes N - 2 times
- ...
- Nth iteration of outer loop: inner executes 0 times

This is our old favorite sum:
N-1 + N-2 + ... + 3 + 2 + 1 + 0
Which we know is O (N^2).

Question Bank:

1. Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).
   (A) Quick Sort
   (B) Heap Sort
   (C) Merge Sort
   (D) Insertion Sort

2. Given an unsorted array. The array has this property that every element in array is at most k distance from its position in sorted array where k is a positive integer smaller than size of array. Which sorting algorithm can be easily modified for sorting this array and what is the obtainable time complexity?

   (A) Insertion Sort with time complexity O(kn)
   (B) Heap Sort with time complexity O(nLogk)
   (C) Quick Sort with time complexity O(kLogk)
   (D) Merge Sort with time complexity O(kLogk)

3. What is the time complexity of fun()?

```
int fun(int n)
{
  int count = 0;
  for (int i = 0; i < n; i++)
     for (int j = i; j > 0; j--)
        count = count + 1;
  return count;
}
```

   a) Theta (n)
   b) Theta (n^2)
   c) Theta (nlogn)
   d) Theta (n logn logn)

4. If the number of records to be sorted is small, then …… sorting can be efficient.

   a) Merge
   b) Heap
   c) Selection
   d) Bubble

5. The complexity of sorting algorithm measures the …… as a function of the number n of items to be sorter.
   a) average time
   b) running time
   c) average-case complexity

d) case-complexity

6. What is an external sorting algorithm?
   a) Algorithm that uses tape or disk during the sort
   b) Algorithm that uses main memory during the sort
   c) Algorithm that involves swapping
   d) Algorithm that are considered ‘in place’

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement 2.1

Code:

```
/*
2.1 Write a program to implement insertion sort algorithm.
*/

import java.util.Scanner;

/**
 * experiment_2_problem_statement_1
 */
public class experiment_2_problem_statement_1 {

    // Function to sort array using insertion sort

    /*
     * Time Complexity: O(n^2) - Number of iterations
     * Space Complexity: O(1) - Constant Space - No extra space is
used
     */
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }
```

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

        System.out.print("Enter the size of Array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.print("Enter the Elements of Array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.print("Unsorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        insertionSort(arr);

        System.out.print("Sorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        sc.close();
    }
}
```

Output:

```
PowerShell

PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
javac .\experiment_2_problem_statement_1.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 2> java experiment_2_problem_statement_1
Enter the size of Array: 5
Enter the Elements of Array: 1
2
3
4
5
Unsorted Array is: 1 2 3 4 5
Sorted Array is: 1 2 3 4 5
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 2>
```

Problem Statement 2.2

Code:

```
/*
2.2 Write a program to implement selection sort algorithm.
*/

import java.util.Scanner;

/**
 * pracatical_2_problem_statement_2
 */
public class experiment_2_problem_statement_2 {

    // Function to sort array using selection sort

    /*
     * Time Complexity: O(n^2) - Number of iterations
     * Space Complexity: O(1) - Constant Space - No extra space is
used
     */
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
```

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

        System.out.print("Enter the size of Array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.print("Enter the Elements of Array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.print("Unsorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        selectionSort(arr);

        System.out.print("Sorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        sc.close();
    }
}
```

Output:

```
PowerShell
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 2> javac .\experiment_2_problem_statement_2.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 2> java experiment_2_problem_statement_2
Enter the size of Array: 5
Enter the Elements of Array: 1 2 3 4 5
Unsorted Array is: 1 2 3 4 5
Sorted Array is: 1 2 3 4 5
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 2>
```

EXPERIMENT NO. 3

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 03-02-2023 – 10.02-2023	
Faculty Signature:	
Marks:	

Objective(s):

Perform problems based on divide and conquer technique.

Outcome:

Students will be able to understand the concept of divide and conquer algorithmic technique.

Problem Statement:

3.1 Write a program to implement merge sort algorithm.
3.2 Write a program to implement quick sort algorithm.

Background Study:

Complexity Analysis:

The straightforward version of function merge requires at most 2n steps (n steps for copying the sequence to the intermediate array *b*, and at most n steps for copying it back to array *a*). The time complexity of merge sort is therefore

$T(n) \leq 2n + 2\,T(n/2)$ and

$T(1) = 0$

The solution of this recursion yields

$T(n) \leq 2n\log(n) \in O(n\log(n))$

Thus, the mergesort algorithm is optimal, since the lower bound for the sorting problem of $\Omega(n \log(n))$ is attained. In the more efficient variant, function merge requires at most 1.5n steps (n/2 steps for copying the first half of the sequence to the intermediate array b, n/2 steps for copying it back to array *a*, and at most *n*/2 steps for processing the second half). This yields a running time of mergesort of at most 1.5 *n* log (*n*) steps. Algorithm mergesort has a time complexity of $\Theta(n \log(n))$ which is optimal. A drawback of mergesort is that it needs an additional space of $\Theta(n)$ for the temporary array *b*. There are different possibilities to implement function merge. The most efficient of these is variant *b*. It requires only half as much additional space, it is faster than the other variants, and it is stable.

Question Bank:

1. In quick sort, the number of partitions into which the file of size n is divided by a selected record is

(A) n

(B) n-1

(C) 2

(D) n/2

2. The worst-case time complexity of Quick Sort and Merge Sort is________.

(A) $O(n^2)$, O(logn)

(B) $O(n^2)$, O(nlogn)

(C) $O(n^2)$, O(n2)

(D) O(nlogn), O(nlogn)

3. The algorithm like Quick sort does not require extra memory for carrying out the sorting procedure. This technique is called __________.

a) In-place

b) Stable

c) Unstable

d) In-partition

4. A list of n strings, each of length n, is sorted into lexicographic order using the merge-sort algorithm. The worst-case running time of this computation is

A. O (n log n)

B. O (n2 log n)

C. O (n2 + log n)

D. O (n2)

5. Quick sort efficiency can be improved by adopting

A. non-recursive method

B. insertion method

C. tree search method

D. None of the above

6. Suppose we need to sort a list of employee records in ascending order, using the social security number (a 9-digit number) as the key (i.e., sort the records by social security number). If we need to guarantee that the running time will be no worse than n log n, which sorting methods could we use?

A. merge sort

B. quicksort

C. insertion sort

D. Either merge sort or quicksort

E. None of these sorting algorithms guarantee a worst-case performance of n log n or better

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
3.1 Write a program to implement merge sort algorithm.
*/

import java.util.Scanner;

/**
 * experiment_3_problem_statement_1
 */
public class experiment_3_problem_statement_1 {

    // Recursive merge sort function to sort the array

    /*
     * Time Complexity: O(nlogn)
     * Space Complexity: O(n) - Recusrive Stack Space
     */
    private static void recursiveMergeSort(int[] arr, int start, int
end) {
        if (start < end) {
            int mid = (start + end) / 2;
            recursiveMergeSort(arr, start, mid);
            recursiveMergeSort(arr, mid + 1, end);
            merge(arr, start, mid, end);
        }
    }

    public static void recursiveMegreSort(int[] arr) {
        recursiveMergeSort(arr, 0, arr.length - 1);
    }
```

```
    // merge function to merge two subarrays
    private static void merge(int[] arr, int start, int mid, int end)
{
        int n1 = mid - start + 1;
        int n2 = end - mid;

        int[] left = new int[n1];
        int[] right = new int[n2];

        for (int i = 0; i < n1; i++) {
            left[i] = arr[start + i];
        }
        for (int i = 0; i < n2; i++) {
            right[i] = arr[mid + 1 + i];
        }

        int i = 0, j = 0, k = start;
        while (i < n1 && j < n2) {
            if (left[i] <= right[j]) {
                arr[k] = left[i];
                i++;
            } else {
                arr[k] = right[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = left[i];
            i++;
            k++;
        }

        while (j < n2) {
```

```
            arr[k] = right[j];
            j++;
            k++;
        }
    }

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

        System.out.print("Enter the size of Array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.print("Enter the Elements of Array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println();

        System.out.print("Unsorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        recursiveMegreSort(arr);

        System.out.print("Sorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        sc.close();
    }
}
```

Output:

```
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 3> javac .\experiment_3_problem_statement_1.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 3> java experiment_3_problem_statement_1
Enter the size of Array: 5
Enter the Elements of Array: 5
6
4
2
1

Unsorted Array is: 5 6 4 2 1
Sorted Array is: 1 2 4 5 6
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 3>
```

Code:

```
/*
3.2 Write a program to implement quick sort algorithm.
*/

import java.util.Scanner;

/**
 * experiment_3_problem_statement_2
 */
public class experiment_3_problem_statement_2 {

    // Recursive quick sort function to sort the array

    /*
     * Time Complexity: O(n^2)
     * Space Complexity: O(n) - Recusrive Stack Space
     */
    private static void recursiveQuickSort(int[] arr, int start, int
end) {
        if (start < end) {
            int pivot = partition(arr, start, end);
            recursiveQuickSort(arr, start, pivot - 1);
            recursiveQuickSort(arr, pivot + 1, end);
        }
    }

    public static void recursiveQuickSort(int[] arr) {
        recursiveQuickSort(arr, 0, arr.length - 1);
    }

    // partition function to partition the array
    private static int partition(int[] arr, int start, int end) {
```

```
        int pivot = arr[end];
        int i = start - 1;
        for (int j = start; j < end; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[end];
        arr[end] = temp;
        return i + 1;
    }

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

        System.out.print("Enter the size of Array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.print("Enter the Elements of Array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println();

        System.out.print("Unsorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
```

```java
        recursiveQuickSort(arr);

        System.out.print("Sorted Array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        sc.close();
    }
}
```

Output:

```
PowerShell

PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 3> javac .\experiment_3_problem_statement_2.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 3> java experiment_3_problem_statement_2
Enter the size of Array: 5
Enter the Elements of Array: 6
4
3
2
2

Unsorted Array is: 6 4 3 2 2
Sorted Array is: 2 2 3 4 6
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 3>
```

EXPERIMENT NO. 4

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: [Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)]	
Date: 24-02-2023	
Faculty Signature:	
Marks:	

Objective:

Perform problems based on divide and conquer technique.

Outcome:

Students will be able to understand the concept of divide and conquer algorithmic technique and apply on real world problems.

Problem Statement:

Write a program to implement Strassen's matrix algorithm. Compute its time complexity.

Background Study:

Complexity Analysis

$$T(N) = \begin{cases} 1 & if\ N = 1 \\ 7T\left(\frac{N}{2}\right) + cN^2 & otherwise \end{cases}$$

Applying Master's method, we get

$$T(N) = O\left(N^{ln7}\right)$$
$$T(N) = O(\mathrm{N}^{2.81})$$

Question Bank:

1. Steps of Divide and Conquer approach
a. Divide, Conquer and Combine
b. Combine, Conquer and Divide
c. Combine, Divide and Conquer
d. Divide, Combine and Conquer

2. The number of operations in Matrix multiplications M1, M2, M3, M4 and M5 of sizes 5X10, 10X100, 100X2, 2X20 and 20X50
a. 5830
b. 4600
c. 6900
d. 12890

3. Division Pattern of Problems in Divide and Conquer approach
a. Iterative
b. Recursive

c. Parallel
d. Random

4. Which of the following sorting algorithms does not have a worst-case running time of O(n2)?
a. Quick sort
b. Merge sort
c. Insertion sort
d. Bubble sort

5. Time complexity of matrix chain multiplication Select one:
a. O(n2)
b. O(n)
c. O(nlogn)
d. O(n3)

6. RANDOMIZE-IN-PLACE(A)

```
n=A.length
For i=1 to n
    Swap A[i] with A[RANDOM(I,n.]
```

The above procedure RANDOMIZE-IN-PLACE(A) computes

a. a uniform deliberate permutation
b. a different random permutation
c. a different deliberate permutation
d. a uniform random permutation

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Problem Statement:
Write a program to implement Strassen’s matrix algorithm. Compute its
time complexity.

Strassen’s matrix algorithm is a divide and conquer algorithm for
matrix multiplication. It is faster than the standard matrix
multiplication algorithm. It is based on the observation that if we
break a matrix into four submatrices, then the product of the matrices
can be computed using only 7 multiplications instead of 8. The
algorithm is as follows:
*/

/**
 * experiment_4_problem_statement
 */
public class experiment_4_problem_statement {

    // Function to print a matrix
    private static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    // Function to multiply two matrices
    // Time Complexity : O(n^3)
```

```
    public static void multiplyMatrix(int[][] matrix1, int[][]
matrix2) {
        if (matrix1[0].length != matrix2.length) {
            System.out.println("Invalid Input");
            return;
        }

        for (int i = 0; i < matrix1.length; i++) {
            for (int j = 0; j < matrix2[0].length; j++) {
                int sum = 0;
                for (int k = 0; k < matrix1[0].length; k++) {
                    sum += matrix1[i][k] * matrix2[k][j];
                }
                System.out.print(sum + " ");
            }
            System.out.println();
        }
    }

    // Function to subtract two matrices
    // Time Complexity : O(n^2)
    public static int[][] subtractMatrix(int[][] matrix1, int[][]
matrix2) {
        int[][] result = new int[matrix1.length][matrix1[0].length];

        for (int i = 0; i < matrix1.length; i++) {
            for (int j = 0; j < matrix1[0].length; j++) {
                result[i][j] = matrix1[i][j] - matrix2[i][j];
            }
        }

        return result;
    }

    // Function to add two matrices
```

```
    // Time Complexity : O(n^2)
    public static int[][] addMatrix(int[][] matrix1, int[][] matrix2)
{
        int[][] result = new int[matrix1.length][matrix1[0].length];

        for (int i = 0; i < matrix1.length; i++) {
            for (int j = 0; j < matrix1[0].length; j++) {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }

        return result;
    }

    // Function to multiply two matrices using Strassen's Algorithm
    // Time Complexity : O(n^2.8074)
    public static int[][] StrassenMatrixAlgorithm(int[][] matrix1,
int[][] matrix2) {
        if (matrix1[0].length != matrix2.length) {
            System.out.println("Invalid Input");
            return null;
        }

        int[][] result = new int[matrix1.length][matrix2[0].length];

        // Base Case
        if (matrix1.length == 1) {
            result[0][0] = matrix1[0][0] * matrix2[0][0];
            return result;
        }

        // Dividing the matrices into sub-matrices
        int[][] a11 = new int[matrix1.length / 2][matrix1.length / 2];
        int[][] a12 = new int[matrix1.length / 2][matrix1.length / 2];
        int[][] a21 = new int[matrix1.length / 2][matrix1.length / 2];
```

```
        int[][] a22 = new int[matrix1.length / 2][matrix1.length / 2];

        int[][] b11 = new int[matrix2.length / 2][matrix2.length / 2];
        int[][] b12 = new int[matrix2.length / 2][matrix2.length / 2];
        int[][] b21 = new int[matrix2.length / 2][matrix2.length / 2];
        int[][] b22 = new int[matrix2.length / 2][matrix2.length / 2];

        // Dividing the matrix a into 4 halves
        for (int i = 0; i < matrix1.length / 2; i++) {
            for (int j = 0; j < matrix1.length / 2; j++) {
                a11[i][j] = matrix1[i][j];
                a12[i][j] = matrix1[i][j + matrix1.length / 2];
                a21[i][j] = matrix1[i + matrix1.length / 2][j];
                a22[i][j] = matrix1[i + matrix1.length / 2][j +
matrix1.length / 2];
            }
        }

        // Dividing the matrix b into 4 halves
        for (int i = 0; i < matrix2.length / 2; i++) {
            for (int j = 0; j < matrix2.length / 2; j++) {
                b11[i][j] = matrix2[i][j];
                b12[i][j] = matrix2[i][j + matrix2.length / 2];
                b21[i][j] = matrix2[i + matrix2.length / 2][j];
                b22[i][j] = matrix2[i + matrix2.length / 2][j +
matrix2.length / 2];
            }
        }

        // Calculating p1 to p7
        int[][] p1 = StrassenMatrixAlgorithm(a11, subtractMatrix(b12,
b22));
        int[][] p2 = StrassenMatrixAlgorithm(addMatrix(a11, a12),
b22);
```

```
        int[][] p3 = StrassenMatrixAlgorithm(addMatrix(a21, a22),
b11);
        int[][] p4 = StrassenMatrixAlgorithm(a22, subtractMatrix(b21,
b11));
        int[][] p5 = StrassenMatrixAlgorithm(addMatrix(a11, a22),
addMatrix(b11, b22));
        int[][] p6 = StrassenMatrixAlgorithm(subtractMatrix(a12, a22),
addMatrix(b21, b22));
        int[][] p7 = StrassenMatrixAlgorithm(subtractMatrix(a11, a21),
addMatrix(b11, b12));

        // Calculating c11, c12, c21, c22
        int[][] c11 = addMatrix(subtractMatrix(addMatrix(p5, p4), p2),
p6);
        int[][] c12 = addMatrix(p1, p2);
        int[][] c21 = addMatrix(p3, p4);
        int[][] c22 = subtractMatrix(subtractMatrix(addMatrix(p5, p1),
p3), p7);

        // Joining the 4 halves into a single result matrix
        for (int i = 0; i < result.length / 2; i++) {
            for (int j = 0; j < result.length / 2; j++) {
                result[i][j] = c11[i][j];
                result[i][j + result.length / 2] = c12[i][j];
                result[i + result.length / 2][j] = c21[i][j];
                result[i + result.length / 2][j + result.length / 2] =
c22[i][j];
            }
        }
        return result;
    }

    public static void main(String[] args) {
```

```
        int[][] matrix1 = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10,
11, 12 }, { 13, 14, 15, 16 } };
        int[][] matrix2 = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10,
11, 12 }, { 13, 14, 15, 16 } };

        System.out.println("Matrix 1 : ");
        printMatrix(matrix1);
        System.out.println();

        System.out.println("Matrix 2 : ");
        printMatrix(matrix2);
        System.out.println();

        System.out.println("Matrix Multiplication : ");
        multiplyMatrix(matrix1, matrix2);
        System.out.println();

        System.out.println("Strassen's Matrix Algorithm : ");
        printMatrix(StrassenMatrixAlgorithm(matrix1, matrix2));
    }
}
```

Output:

```
PowerShell
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 4> java experiment_4_problem_statement
Matrix 1 :
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Matrix 2 :
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Matrix Multiplication :
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```

```
PowerShell

Matrix 2 :
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Matrix Multiplication :
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600

Strassen's Matrix Algorithm :
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 4>
```

EXPERIMENT NO. 5

Student Name and Roll Number: Piyush Gambhir – 21CSU349
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)
Date: 03.03.2023
Faculty Signature:
Marks:

Objective(s):

Perform problems based on Greedy algorithmic technique.

Outcome:

Students will be able to understand the concept of greedy algorithm.

Problem Statement:

Implement Fractional Knapsack Algorithm.

Background Study:

Complexity Analysis

If the items are already sorted into decreasing order of vi / wi, then the while-loop takes a time in *O(n)*.

Therefore, the total time including the sort is in *O(n log n)*. If we keep the items in heap with largest vi/wi at the root. Then

- creating the heap takes *O(n)* time
- while-loop now takes *O(log n)* time

Question Bank:

1. Fractional knapsack problem is also known as __________

a) 0/1 knapsack problem

b) Continuous knapsack problem

c) Divisible knapsack problem

d) Non continuous knapsack problem

2. Fractional knapsack problem is solved most efficiently by which of the following algorithm?

a) Divide and conquer

b) Dynamic programming

c) Greedy algorithm

d) Backtracking

3. What is the objective of the knapsack problem?

a) To get maximum total value in the knapsack

b) To get minimum total value in the knapsack
c) To get maximum weight in the knapsack
d) To get minimum weight in the knapsack

4 Which of the following statement about 0/1 knapsack and fractional knapsack problem is correct?

a) In 0/1 knapsack problem items are divisible and in fractional knapsack items are indivisible
b) Both are the same
c) 0/1 knapsack is solved using a greedy algorithm and fractional knapsack is solved using dynamic programming
d) In 0/1 knapsack problem items are indivisible and in fractional knapsack items are divisible

5. Time complexity of fractional knapsack problem is ____________
a) O(n log n)
b) O(n)
c) $O(n^2)$
d) O(nW)

6. The main time taking step in fractional knapsack problem is ___________

a) Breaking items into fraction
b) Adding items into knapsack
c) Sorting
d) Looping through sorted items

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Implement Fractional Knapsack Algorithm
*/

/**
 * experiment_5_problem_statement
 */
public class experiment_5_problem_statement {

    public static void fractionalKnapsackAlgorithm(int[] weight, int[]
profit, int maxWeight) {
        int n = weight.length;
        double[] ratio = new double[n];
        for (int i = 0; i < n; i++) {
            ratio[i] = (double) profit[i] / weight[i];
        }
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (ratio[i] < ratio[j]) {
                    double temp = ratio[j];
                    ratio[j] = ratio[i];
                    ratio[i] = temp;

                    int temp2 = weight[j];
                    weight[j] = weight[i];
                    weight[i] = temp2;

                    temp2 = profit[j];
                    profit[j] = profit[i];
                    profit[i] = temp2;
                }
```

```
            }
        }

        double totalProfit = 0;
        for (int i = 0; i < n; i++) {
            if (weight[i] <= maxWeight) {
                maxWeight -= weight[i];
                totalProfit += profit[i];
            } else {
                totalProfit += (ratio[i] * maxWeight);
                break;
            }
        }
        System.out.println("The maximum profit that can be obtained is
" + totalProfit);

    }

    public static void main(String[] args) {
        int[] weight = { 10, 40, 20, 30 };
        int[] profit = { 60, 40, 100, 120 };
        int maxWeight = 50;

        fractionalKnapsackAlgorithm(weight, profit, maxWeight);
    }
}
```

Output:

```
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230
- Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 5> javac .\exp
eriment_5_problem_statement.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230
- Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 5> java experi
ment_5_problem_statement
The maximum profit that can be obtained is 240.0
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Semester)\CSL230
- Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 5>
```

EXPERIMENT NO. 6

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 03.03.2023 – 17.03.2023	
Faculty Signature:	
Marks:	

Objective:

Perform problems based on Greedy algorithmic technique.

Outcome:

Students will be able to understand the concept of greedy algorithm.

Problem Statement:

Implement Prim's and Kruskal's algorithms for finding minimum spanning tree of a given graph.

Background Study:

Complexity Analysis

Prim's Algorithm

Running Time = O(m + n log n) (m = edges, n = nodes)

If a heap is not used, the run time will be O(n^2) instead of O(m + n log n). However, using a heap complicates the code since you're complicating the data structure. A Fibonacci heap is the best kind of heap to use, but again, it complicates the code. Unlike Kruskal's, it doesn't need to see all of the graph at once. It can deal with it one piece at a time. It also doesn't need to worry if adding an edge will create a cycle since this algorithm deals primarily with the nodes, and not the edges. For this algorithm the number of nodes needs to be kept to a minimum in addition to the number of edges. For small graphs, the edges matter more, while for large graphs the number of nodes matters more.

Kruskal's Algorithm

- Initialization *O*(*V*) time
- Sorting the edges Q(*E* lg *E*) = Q(*E* lg *V*) (why?)
- *O*(*E*) calls to FindSet
- Union costs
  - Let *t(v)* – the number of times *v* is moved to a new cluster
  - Each time a vertex is moved to a new cluster the size of the cluster containing the vertex at least doubles: *t(v)* £ log *V*
  - Total time spent doing Union $\sum_{v \in V} t(v) \leq |V| \log |V|$

Total time: O(E lg V)

Question Bank:

1. Which of the following is false in the case of a spanning tree of a graph G?
a) It is tree that spans G
b) It is a subgraph of the G
c) It includes every vertex of the G
d) It can be either cyclic or acyclic

2. Every graph has only one minimum spanning tree.
a) True
b) False

3. Consider a complete graph G with 4 vertices. The graph G has _____ spanning trees.
a) 15
b) 8
c) 16
d) 13

4 Consider a undirected graph G with vertices { A, B, C, D, E}. In graph G, every edge has distinct weight. Edge CD is edge with minimum weight and edge AB is edge with maximum weight. Then, which of the following is false?
a) Every minimum spanning tree of G must contain CD
b) If AB is in a minimum spanning tree, then its removal must disconnect G
c) No minimum spanning tree contains AB
d) G has a unique minimum spanning tree

5. Which of the following is false?
a) The spanning trees do not have any cycles
b) MST have n – 1 edges if the graph has n edges
c) Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph
d) Removing one edge from the spanning tree will not make the graph disconnected

6. If all the weights of the graph are positive, then the minimum spanning tree of the graph is a minimum cost subgraph.
a) True
b) False

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Implement Prim’s and Kruskal’s algorithms for finding minimum spanning
tree of a given graph.
*/

import java.util.*;

/**
 * experiment_6_problem_statement_kruskel
 */
public class experiment_6_problem_statement_kruskel {
    static class Edge {
        int src, dest, weight;

        public Edge(int src, int dest, int weight) {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
    }

    static class Graph {
        int V, E;
        List<Edge> edges;

        public Graph(int V, int E) {
            this.V = V;
            this.E = E;
            edges = new ArrayList<>();
        }
```

```
        public void addEdge(Edge e) {
            edges.add(e);
        }
    }

    static class Subset {
        int parent, rank;
    }

    private static int find(Subset[] subsets, int i) {
        if (subsets[i].parent != i) {
            subsets[i].parent = find(subsets, subsets[i].parent);
        }
        return subsets[i].parent;
    }

    private static void union(Subset[] subsets, int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        if (subsets[xroot].rank < subsets[yroot].rank) {
            subsets[xroot].parent = yroot;
        } else if (subsets[xroot].rank > subsets[yroot].rank) {
            subsets[yroot].parent = xroot;
        } else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    public static Graph kruskalMinimumSpanningTree(Graph graph) {
        Graph MST = new Graph(graph.V, graph.V - 1);
        Edge[] edges = new Edge[graph.E];
        for (int i = 0; i < graph.E; i++) {
            edges[i] = graph.edges.get(i);
```

```
        }

        Arrays.sort(edges, new Comparator<Edge>() {
            @Override
            public int compare(Edge e1, Edge e2) {
                return e1.weight - e2.weight;
            }
        });

        Subset[] subsets = new Subset[graph.V];
        for (int i = 0; i < graph.V; i++) {
            subsets[i] = new Subset();
            subsets[i].parent = i;
            subsets[i].rank = 0;
        }

        int i = 0;
        int j = 0;
        while (j < graph.V - 1) {
            Edge e = edges[i++];
            int x = find(subsets, e.src);
            int y = find(subsets, e.dest);
            if (x != y) {
                MST.addEdge(e);
                j++;
                union(subsets, x, y);
            }
        }
        return MST;
    }

    public static void printGraph(Graph graph) {
        for (int i = 0; i < graph.V; i++) {
            System.out.print(i + " -> ");
            for (int j = 0; j < graph.edges.size(); j++) {
```

```
                if (graph.edges.get(j).src == i) {
                    System.out.print(graph.edges.get(j).dest + "(" +
graph.edges.get(j).weight + ") ");
                }
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {

        int V = 4;
        int E = 5;

        Graph graph = new Graph(V, E);

        graph.addEdge(new Edge(0, 1, 10));
        graph.addEdge(new Edge(0, 2, 6));
        graph.addEdge(new Edge(0, 3, 5));
        graph.addEdge(new Edge(1, 3, 15));
        graph.addEdge(new Edge(2, 3, 4));

        System.out.println("Given Graph: \n");
        printGraph(graph);

        Graph MST = kruskalMinimumSpanningTree(graph);
        System.out.println("Minimum Spanning Tree of Given Graph:
\n");
        printGraph(MST);
    }
}
```

Output:

```
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 6> javac .\experiment_6_problem_statement_prims.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 6> java experiment_6_problem_statement_prims
Graph Adjacency Matrix:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Prims Minimum Spanning Tree:
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 6>
```

Code:

```
/*
Implement Prim’s and Kruskal’s algorithms for finding minimum spanning
tree of a given graph.
*/

import java.util.*;

/**
 * experiment_6_problem_statement_prims
 */
public class experiment_6_problem_statement_prims {

    public static void primsMST(int[][] graph) {
        int V = graph.length;
        int[] parent = new int[V];
        int[] key = new int[V];
        boolean[] mstSet = new boolean[V];

        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;

            for (int v = 0; v < V; v++) {
```

```
                if (graph[u][v] != 0 && mstSet[v] == false &&
graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }

        printMST(parent, V, graph);
    }

    public static int minKey(int[] key, boolean[] mstSet) {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < key.length; v++) {
            if (mstSet[v] == false && key[v] < min) {
                min = key[v];
                min_index = v;
            }
        }

        return min_index;
    }

    public static void printMST(int[] parent, int n, int[][] graph) {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < n; i++) {
            System.out.println(parent[i] + " - " + i + "\t" +
graph[i][parent[i]]);
        }
    }

    public static void main(String[] args) {
        int[][] graph = new int[][] {
                { 0, 2, 0, 6, 0 },
```

```
                { 2, 0, 3, 8, 5 },
                { 0, 3, 0, 0, 7 },
                { 6, 8, 0, 0, 9 },
                { 0, 5, 7, 9, 0 }
        };

        // Priting Graph Adjacency Matrix
        System.out.println("Graph Adjacency Matrix:");
        for (int i = 0; i < graph.length; i++) {
            for (int j = 0; j < graph[i].length; j++) {
                System.out.print(graph[i][j] + " ");
            }
            System.out.println();
        }

        // Printing Prims Minimum Spanning Tree
        System.out.println("Prims Minimum Spanning Tree:");
        primsMST(graph);
    }
}
```

Output:

```
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 6> javac .\experiment_6_problem_statement_kruskel.java
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 6> java experiment_6_problem_statement_kruskel
Given Graph:

0 -> 1(10) 2(6) 3(5)
1 -> 3(15)
2 -> 3(4)
3 ->
Minimum Spanning Tree of Given Graph:

0 -> 3(5) 1(10)
1 ->
2 -> 3(4)
3 ->
PS C:\Users\mainp\Google Drive - NCU\My Drive\College Material\4. 2nd Year (EVEN Se
mester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Ma
nual\Experiment 6>
```

EXPERIMENT NO. 7

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 14.04.2023	
Faculty Signature:	
Marks:	

Objective(s):

Perform problems based on Dynamic Programming.

Outcome:

Students will be able to understand the concept of dynamic programming.

Problem Statement:

Write a program for 0/1 Knapsack problem using Dynamic Programming.

Background Study:

This dynamic-0-1-kanpsack algorithm takes θ(*nw*) times, broken up as follows: θ(*nw*) times to fill the *c*-table, which has (*n*+*1*).(*w*+1) entries, each requiring θ(1) time to compute. *O*(*n*) time to trace the solution, because the tracing process starts in row *n* of the table and moves up 1 row at each step.

Question Bank:

1. Time complexity of knapsack 0/1 where n is the number of items and W is the capacity of knapsack.
a. O(W)
b. O(n)
c. O(nW)

2. In dynamic programming, the output to stage n become the input to
a. Objective function
b. Feasible solution
c. Decision stages
d. Optimum solution

3. Which of the following problems is equivalent to the 0-1 Knapsack problem?
a) You are given a bag that can carry a maximum weight of W. You are given N items which have a weight of {w1, w2, w3,…., wn} and a value of {v1, v2, v3,…., vn}. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

b) You are studying for an exam and you have to study N questions. The questions take {t1, t2, t3,…., tn} time(in hours) and carry {m1, m2, m3,…., mn} marks. You can study for a maximum of T hours. You can either study a question or leave it. Choose the questions in such a way that your score is maximized

c) You are given infinite coins of denominations {v1, v2, v3,….., vn} and a sum S. You have to find the minimum number of coins required to get the sum S

d) You are given a suitcase that can carry a maximum weight of 15kg. You are given 4 items which have a weight of {10, 20, 15,40} and a value of {1, 2, 3,4}. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

4 What is the time complexity of the brute force algorithm used to solve the Knapsack problem?
a) O(n)
b) O(n!)
c) $O(2^n)$
d) $O(n^3)$

5. Which of the following methods can be used to solve the Knapsack problem?
a) Brute force algorithm
b) Recursion
c) Dynamic programming
d) Brute force, Recursion and Dynamic Programming

6. You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?
a) 160
b) 200
c) 170
d) 90

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Write a program for 0/1 Knapsack problem using Dynamic Programming.

The 0-1 knapsack problem is a variation of the knapsack problem in
which each item can only be included in the knapsack once or not at
all (i.e., it must be either 0 or 1). The problem is stated as
follows:

Given a set of n items, each with a weight w_i and a value v_i, and
a knapsack with a maximum capacity W, the goal is to find the subset
of items that maximizes the total value of the knapsack while
keeping the total weight less than or equal to W.
*/

import java.util.Scanner;

/**
 * experiment_7_problem_statement
 */
public class experiment_7_problem_statement {

    // Function to find the maximum value that can be put in a
knapsack of capacity
    // capacity
    public static int knspsack(int[] weight, int[] value, int n, int
maxWeight) {
        int[][] K = new int[n + 1][maxWeight + 1];
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= maxWeight; j++) {
                if (i == 0 || j == 0) {
                    K[i][j] = 0;
                } else if (weight[i - 1] <= j) {
                    K[i][j] = Math.max(value[i - 1] + K[i - 1][j -
weight[i - 1]], K[i - 1][j]);
```

```
                } else {
                    K[i][j] = K[i - 1][j];
                }
            }
        }

        return K[n][maxWeight];
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of items");
        int n = sc.nextInt();
        int[] weight = new int[n];
        int[] value = new int[n];
        System.out.println("Enter the weight of each item");
        for (int i = 0; i < n; i++) {
            weight[i] = sc.nextInt();
        }
        System.out.println("Enter the value of each item");
        for (int i = 0; i < n; i++) {
            value[i] = sc.nextInt();
        }
        System.out.println("Enter the capacity of the knapsack");
        int capacity = sc.nextInt();

        System.out.println("The maximum value that can be put in a
knapsack of capacity " + capacity + " is "
                + knspsack(weight, value, n, capacity));
        sc.close();
    }
}
```

Output:

```
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algori
thms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 7> javac .\experiment_7_problem_statement.java
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algori
thms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 7> java experiment_7_problem_statement
Enter the number of items
5
Enter the weight of each item
6
2
2
12
23
Enter the value of each item
13
23
12
5
5
Enter the capacity of the knapsack
18
The maximum value that can be put in a knapsack of capacity 18 is 48
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algori
thms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 7>
```

EXPERIMENT NO. 8

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 14.04.2023	
Faculty Signature:	
Marks:	

Objective:

Implement travelling salesman problem.

Traveling Salesperson Problem:
Given *n* cities, a salesperson starts at a specified city (source), visit all n-1 cities only once and return to the city from where he has started. The objective of this problem is to find a route through the cities that minimizes the cost and thereby maximizing the profit.

Outcome:

Students will be able to understand the concept of dynamic programming.

Problem Statement:

Write a program for Travelling salesman problem using Dynamic Programming.

Background Study:

Complexity Analysis:
An algorithm that proceeds to find an optimal tour by making use of (1) and (2) will require $O(n^2 2^n)$ time since the computation of $g(i, S)$ with $|S| = k$ requires k -1 comparisons when solving (2). It's better than enumerating all $n!$ different tours to find the best one. The most serious drawback of this dynamic programming solution is the space needed. The space needed in $O(n2^n)$. This is too large even for modest values of n.

Question Bank:

1. The traveling salesman problem involves visiting each city how many times?
a. 0
b. 1
c. 2
d. 3

2. What is the traveling salesman problem equivalent to in graph theory?
a. Any circuit.
b. A Hamilton circuit in a non-weighted graph.
c. A round trip airfare.

d. A Hamilton circuit in a weighted graph.
e. A connect the dots game.

3. A weighted graph has what associated with each edge?
a. A cost
b. Nothing
c. Direction
d. Size

4 What is the time complexity of the dynamic programming used to solve travelling salesman problem?
a) O(n)
b) O(n!)
c) $O(2^n)$
d) $O(n^3)$

5. If an optimal solution can be created for a problem by constructing optimal solutions for its subproblems, the problem possesses ____________ property.
a) Overlapping subproblems
b) Optimal substructure
c) Memoization
d) Greedy

6. When a top-down approach of dynamic programming is applied to a problem, it usually _____________
a) Decreases both, the time complexity and the space complexity
b) Decreases the time complexity and increases the space complexity
c) Increases the time complexity and decreases the space complexity
d) Increases both, the time complexity and the space complexity

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Write a program for Travelling salesman problem using Dynamic
Programming.

Traveling Salesperson Problem:
Given n cities, a salesperson starts at a specified city
(source), visit all n-1 cities only once and return to the city
from where he has started. The objective of this problem is to
find a route through the cities that minimizes the cost and
thereby maximizing the profit.
*/

import java.util.*;

/**
 * experiment_8_problem_statement
 */
public class experiment_8_problem_statement {

    static int tsp(int[][] cost, int[] tour, int start, int n)
{
        int[] temp = new int[n];
        int[] mintour = new int[n];
        int mincost = Integer.MAX_VALUE;
        int ccost;
        if (start == n - 2) {
            return cost[tour[n - 2]][tour[n - 1]] + cost[tour[n
- 1]][0];
        }
        for (int i = start + 1; i < n; i++) {
            for (int j = 0; j < n; j++) {
```

```
                temp[j] = tour[j];
            }
            temp[start + 1] = tour[i];
            temp[i] = tour[start + 1];
            if (cost[tour[start]][tour[i]] + (ccost = tsp(cost,
temp, start + 1, n)) < mincost) {
                mincost = cost[tour[start]][tour[i]] + ccost;
                for (int j = 0; j < n; j++) {
                    mintour[j] = temp[j];
                }
            }
        }
        for (int i = 0; i < n; i++) {
            tour[i] = mintour[i];
        }
        return mincost;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of cities: ");
        int n = sc.nextInt();
        int[][] cost = new int[n][n];
        System.out.println("Enter the cost matrix: ");
        for (int i = 0; i < n; i++) {
            System.out.println("Enter the cost for city " + (i
+ 1) + ": ");
            for (int j = 0; j < n; j++) {
                cost[i][j] = sc.nextInt();
            }
        }
        int[] tour = new int[n];
        for (int i = 0; i < n; i++) {
            tour[i] = i;
        }
```

```
        int minCost = tsp(cost, tour, 0, n);
        System.out.println("Minimum cost: " + minCost);
        sc.close();
    }
}
```

Output:

```
PowerShell 7.3.4
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 8> javac .\experiment_8_problem_statement.java
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 8> java experiment_8_problem_statement
Enter the number of cities:
5
Enter the cost matrix:
Enter the cost for city 1:
23
12
23
56
23
Enter the cost for city 2:
12
23
54

21
15
Enter the cost for city 3:
563
23
15
2
23
Enter the cost for city 4:
23
56
45
89
54
```

```
23
12
23
56
23
Enter the cost for city 2:
12
23
54

21
15
Enter the cost for city 3:
563
23
15
2
23
Enter the cost for city 4:
23
56
45
89
54
Enter the cost for city 5:
66
56
22
24
56
Minimum cost: 74
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL230-ADA-Lab_Manual\Experiment 8>
```

EXPERIMENT NO. 9

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 21.04.2023	
Faculty Signature:	
Marks:	

Objective(s):

To Implement 8 Queen's Problem

Outcome:

Students will be able to understand the concept of Backtracking algorithm.

Problem Statement:

Write a program for N-Queen's problem using Backtracking.

Background Study:

Complexity Analysis:
The power of the set of all possible solutions of the n queen's problem is n! and the bounding function takes a linear amount of time to calculate, therefore the running time of the n queens problem is O(n!).

Question Bank:

1. Backtracking algorithm is implemented by constructing a tree of choices called as?
a) State-space tree
b) State-chart tree
c) Node tree
d) Backtracking tree

2. How many solutions are there for 8 queens on 8*8 board?
a) 12
b) 91
c) 92
d) 93

3. In how many directions do queens attack each other?
a) 1
b) 2
c) 3
d) 4

4. Of the following given options, which one of the following is a correct option that provides an optimal solution for 4-queens problem?

a) (3,1,4,2)
b) (2,3,1,4)
c) (4,3,2,1)
d) (4,2,3,1)

5. In what manner is a state-space tree for a backtracking algorithm constructed?
a) Depth-first search
b) Breadth-first search
c) Twice around the tree
d) Nearest neighbour first

6. Which one of the following is an application of the backtracking algorithm?
a) Finding the shortest path
b) Finding the efficient quantity to shop
c) Ludo
d) Crossword

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Write a program for N-Queen’s problem using Backtracking.
*/

/*
Complexity Analysis:
Time Complexity: O(N!).
Explanation: There are N possibilities to put the first queen, not
more than N-1 to put the second one, not more than N-2 for the third
one etc. In total that results in O(N!) time complexity.

Space Complexity: O(N).
Explanantion: We use O(N) extra space to store the solution.
*/

import java.util.Scanner;
import java.util.ArrayList;

/**
 * experiment_9_problem_statement
 */
public class experiment_9_problem_statement {

    // isSafe function to check if a queen can be placed on
board[row][column]
    private static boolean isSafe(int[][] board, int row, int
column) {
        // check if there is a queen in the same row
        for (int i = 0; i < column; i++) {
            if (board[row][i] == 1) {
                return false;
            }
        }
```

```
        // check if there is a queen in the same column
        for (int i = 0; i < row; i++) {
            if (board[i][column] == 1) {
                return false;
            }
        }

        // check if there is a queen in the upper left diagonal
        for (int i = row, j = column; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }
        // check if there is a queen in the lower left diagonal
        for (int i = row, j = column; i < board.length && j >= 0;
i++, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }

        // check if there is a queen in the lower right diagonal
        for (int i = row, j = column; i < board.length && j <
board.length; i++, j++) {
            if (board[i][j] == 1) {
                return false;
            }
        }

        // check if there is a queen in the upper right diagonal
        for (int i = row, j = column; i >= 0 && j < board.length; i-
-, j++) {
            if (board[i][j] == 1) {
                return false;
            }
        }

        return true;
```

```
    }

    // printSolution function to print the solution
    private static void printSolution(int[][] board) {
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board.length; j++) {
                if (board[i][j] == 1) {
                    System.out.print("Q ");
                } else {
                    System.out.print(".");
                }
            }
            System.out.println();
        }
        System.out.println("\n");
    }

    // saveSolution function to save the solution
    private static void saveSolution(int[][] board,
ArrayList<int[][]> solutions) {
        int[][] solution = new int[board.length][board.length];
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board.length; j++) {
                solution[i][j] = board[i][j];
            }
        }
        solutions.add(solution);
    }

    // solveNQueenHelper function to solve N Queen problem
    private static void solveNQueenHelper(int[][] board, int column,
ArrayList<int[][]> solutions) {
        // if all queens are placed then return true and print the
solution
        if (column >= board.length) {
            saveSolution(board, solutions);
            return;
        }
```

```
        // place the queen in the column and check if it is safe
        for (int i = 0; i < board.length; i++) {
            if (isSafe(board, i, column)) {
                board[i][column] = 1;
                solveNQueenHelper(board, column + 1, solutions);
                board[i][column] = 0;
            }
        }
    }

    // solveNQueen function to solve N Queen problem
    public static void solveNQueen(int n) {
        int[][] board = new int[n][n];
        ArrayList<int[][]> solutions = new ArrayList<int[][]>();
        solveNQueenHelper(board, 0, solutions);
        if (solutions.size() == 0) {
            System.out.println("Solution does not exist");
            return;
        }
        System.out.println("Total Solutions: " + solutions.size() +
"\n");

        for (int i = 0; i < solutions.size(); i++) {
            System.out.println("Solution " + (i + 1) + ":");
            printSolution(solutions.get(i));
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Number of Queens: ");
        int n = sc.nextInt();
        System.out.println();
        solveNQueen(n);
        sc.close();
    }
}
```

Output:

```
PowerShell 7.3.4
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 9>               javac .\experiment_9_problem_statement.java
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 9> java experiment_9_problem_statement
Enter the Number of Queens: 4

Total Solutions: 2

Solution 1:
..Q .
Q ...
...Q
.Q ..


Solution 2:
.Q ..
...Q
Q ...
..Q .


PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 9>
```

EXPERIMENT NO. 10

Student Name and Roll Number: Piyush Gambhir – 21CSU349	
Semester/Section: Semester-IV – CS-IV-AIML-B (AL-3)	
Link to Code: Piyush-Gambhir/CSL230-ADA-Lab_Manual (github.com)	
Date: 21.04.2023	
Faculty Signature:	
Marks:	

Objective:

Implement 0/1 Knapsack problem using Branch and Bound

Outcome:

Students will be able to apply Branch and Bound Algorithm to the similar problems and appreciate how it differs from greedy approach and dynamic programming approach already done.

Problem Statement:

Write a program for 0/1 Knapsack using Branch and Bound

Background Study:

Complexity Analysis:
Time Complexity: O(N*W).
As redundant calculations of states are avoided.

Question Bank:

1. Branch and bound is a ___________
a) problem solving technique
b) data structure
c) sorting algorithm
d) type of tree

2. Which data structure is used for implementing a LIFO branch and bound strategy?
a) stack
b) queue
c) array
d) linked list

3. Which data structure is used for implementing a FIFO branch and bound strategy?
a) stack
b) queue
c) array
d) linked list

4. Which data structure is most suitable for implementing best first branch and bound strategy?
a) stack
b) queue

c) priority queue
d) linked list

5. Which of the following branch and bound strategy leads to breadth first search?
a) LIFO branch and bound
b) FIFO branch and bound
c) Lowest cost branch and bound
d) Highest cost branch and bound

6. Which of the following branch and bound strategy leads to depth first search?
a) LIFO branch and bound
b) FIFO branch and bound
c) Lowest cost branch and bound
d) Highest cost branch and bound

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Code:

```
/*
Write a program for 0/1 Knapsack using Branch and Bound
*/

import java.util.*;

/**
 * experiment_10_problem_statement
 */
public class experiment_10_problem_statement {
    static class Item {
        int weight;
        int value;

        Item(int weight, int value) {
            this.weight = weight;
            this.value = value;
        }
    }

    static class Node {
        int level;
        int profit;
        int bound = 0;
        int weight = 0;
    }

    private static int bound(Node u, Item[] items, int n, int
maxWeight) {
        if (u.weight >= maxWeight) {
            return 0;
        }
        int profitBound = u.profit;
        int j = u.level + 1;
```

```
        int totalWeight = u.weight;
        while (j < n && totalWeight + items[j].weight <= maxWeight)
{
            totalWeight += items[j].weight;
            profitBound += items[j].value;
            j++;
        }
        if (j < n) {
            profitBound += (maxWeight - totalWeight) *
items[j].value / items[j].weight;
        }
        return profitBound;
    }

    public static void knapsackUsingBranchBound(Item[] items, int n,
int maxWeight) {
        Queue<Node> queue = new LinkedList<>();
        Node u = new Node();
        Node v = new Node();
        queue.add(u);
        int maxProfit = 0;
        while (!queue.isEmpty()) {
            u = queue.poll();
            if (u.level == -1) {
                v.level = 0;
            }
            if (u.level == n - 1) {
                continue;
            }
            v.level = u.level + 1;
            v.weight = u.weight + items[v.level].weight;
            v.profit = u.profit + items[v.level].value;
            if (v.weight <= maxWeight && v.profit > maxProfit) {
                maxProfit = v.profit;
            }
            v.bound = bound(v, items, n, maxWeight);
            if (v.bound > maxProfit) {
                queue.add(v);
```

```
            }
            v.weight = u.weight;
            v.profit = u.profit;
            v.bound = bound(v, items, n, maxWeight);
            if (v.bound > maxProfit) {
                queue.add(v);
            }
        }
        System.out.println("Maximum profit is " + maxProfit + ".");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of items: ");
        int n = sc.nextInt();
        System.out.print("Enter the capacity of knapsack: ");
        int capacity = sc.nextInt();

        Item[] items = new Item[n];
        System.out.print("Enter the weight and value of items: ");
        for (int i = 0; i < n; i++) {
            int weight = sc.nextInt();
            int value = sc.nextInt();
            items[i] = new Item(weight, value);
        }

        knapsackUsingBranchBound(items, n, capacity);
        sc.close();
    }
}
```

Output:

```
PowerShell
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 10> javac .\experiment_10_problem_statement.java
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 10> javac .\experiment_10_problem_statement.java
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 10> java experiment_10_problem_statement
Enter the number of items: 5
Enter the capacity of knapsack: 61
Enter the weight and value of items: 12
65
14
25
98
47
65
65
12
23
Maximum profit is 25.
PS C:\Users\mainp\OneDrive\College Material\4. 2nd Year (EVEN Semester)\CSL230 - Analysis and Design of Algorithms\Lab Manual\NCU-CSL2
30-ADA-Lab_Manual\Experiment 10>
```