II B. TECH 2025-2026 ODD SEMESTER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB WORKBOOK

DESIGN AND ANALYSIS OF ALGORITHMS

(24CS2203)

Student Name	
University Id	
Section	
Instructor Name	

Koneru Lakshmaiah Education Foundation
(Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Off-Campus: Bachupally-Gandimaisamma Road, Bowrampet, Hyderabad, Telangana - 500 043.
Phone No: 7815926816, www.klh.edu.in

Experiment **#1**		Student ID	
Date		Student Name	

Experiment Title: Analysis of Algorithm based on Arrays

Aim/Objective: To understand the concept and implementation of Basic programs on arrays.

Description: The students will understand and able to implement programs on Arrays.

Pre-Requisites:

Knowledge: Arrays

Tools: Code Blocks/Eclipse IDE

Pre-Lab:

Calculate the time complexity of the following:

1. Algorithm **linear_search(arr, target):**

```
for i in range(len(arr)):
    if arr[i] == target:
        return i
return -1
```

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Algorithm function(arr):

```
n = len(arr)
for i in range(n):
    for j in range(0, n-i-1):
        if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
return arr;
```

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

3. The median of a list of numbers is essentially its middle element after sorting. The same number of elements occur after it as before. Given a list of numbers with an odd number of elements, find the median? Also find its time complexity.

   Example arr=[5,3,1,2,4]

   The sorted array arr=[1,2,3,4,5]. The middle element and the median is 3.

   Description: Write an algorithm findMedian with the following parameter(s):

   arr[n]: an unsorted array of integers

   Returns the median of the array

   Sample Input

   7

   0 1 2 4 6 5 3

   Sample Output

   3

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Experiment **#1**		Student ID	
Date		Student Name	

In-Lab:

1. Given an array of strings arr[], the task is to sort the array of strings according to frequency of each string, in ascending order. If two elements have same frequency, then they are to be sorted in alphabetical order.

   Input: arr[] = {“Ramesh”, “Mahesh”, “Mahesh”, “Ramesh”}

   Output: {“Mahesh”, ”Ramesh”}

   Explanation: As both the strings have the same frequency, the array is sorted in alphabetical order.

   Find the time and space complexity for the procedure/Program.

   - **Procedure/Program:**

   - **Data and Results:**

- **Analysis and Inferences:**

2. Given an array of strings words [] and the sequential order of alphabets, our task is to sort the array according to the order given. Assume that the dictionary and the words only contain lowercase alphabets.

   Input: words = {"word", "world", "row"},

   Order= "worldabcefghijkmnpqstuvxyz"

   Output: {"world", "word", "row"}

   Explanation: According to the given order 'l' occurs before 'd' hence the words "world" will be kept first.

   Find the time and space complexity for the procedure/Program.

- **Procedure/Program:**

- **Data and Results:**

Experiment **#1**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

Post-Lab:

Evaluate the time complexity of pre-lab using **master's theorem**.

- **Procedure/Program:**

- **Data and Results:**

Experiment **#1**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab):**
  1. What is the significance of analyzing both time complexity and space complexity when evaluating algorithms?
  2. How does the choice of data structures impact both time and space complexity in algorithm implementations? Give examples.
  3. When comparing two algorithms with different time and space complexities, how do you decide which one is more suitable for a particular problem or application?
  4. What is the difference between best-case, average-case, and worst-case time complexity? Provide examples.
  5. In the context of sorting algorithms, compare the time and space complexities of algorithms like Quick Sort and Merge Sort. Which one is more time-efficient, and which one is more space-efficient?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#2**		Student ID	
Date		Student Name	

Experiment Title: Performance analysis of Time and Space Complexity

Aim/Objective: Analysis of Time and Space Complexity of Algorithms

Description: The students will understand and find the Time and Space Complexity of Algorithms

Pre-Requisites

Knowledge: Basics of Data Structures and C Programming, Basic knowledge about algorithms in C and Data Structures.

Tools: Code Blocks / Eclipse IDE

Pre-Lab:

1. During lockdown Mothi gets bored by his daily routine while scrolling YouTube and he found an algorithm that looks different. Mothi is very crazy about algorithms, but as he cannot solve algorithms of multiple loops, he got struck and need your help to find the time complexity of that algorithm

```
Algoritm KLU(int n) {
   int count=0;
   for(int i=0;i<n;i=i*2) {
     for(int j=n;j>0;j=j/3) {
     for(int k=0;k<n;k++) {
         Count++;
      } } } }
```

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Suresh provided the following recursive algorithm to the students:

recursive algorithm:

```
int custom_recursive_function (int n)
{   if (n <= 1)
      return 1;
else
      return 3 * custom_recursive_function (n-1);
}
```

Determine the time complexity of the custom_recursive_function function.

- **Procedure/Program:**

Experiment **#2**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

1. During the final skill exam, the teacher gave the students a problem to calculate the factorial of a given number n. One student, Ravi, decided to write a recursive algorithm that is intentionally inefficient to ensure his approach is unique. Your task is to determine whether Ravi's algorithm for calculating the factorial is correct and to analyze its time complexity.

Here is Ravi's recursive algorithm:

```
int inefficient_factorial(int n) {    if (n == 0)
    return 1;    else if (n == 1)
    return 1;    else
```

```
    return n * inefficient_factorial(n-1) * inefficient_factorial(n-1);
}
```

Question: Determine if Ravi's algorithm for calculating the factorial is correct and analyze its time complexity.

- **Procedure/Program:**

- **Data and Results:**

Experiment **#2**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

Post-Lab:

1) In the city of KLU, there are numerous street lamps arranged in a straight line. These street lamps are equipped with sensors that can detect their respective positions. Professor Stefan has asked Mothi to find the street lamp that has a specific height using a recursive search algorithm. Mothi has written a recursive algorithm to find the height but needs help in determining the time complexity. Given an array of street lamp heights a, which is sorted in ascending order, write an algorithm to find the position of a lamp with a specific height using a recursive linear search.

```
int recursiveLinearSearch(int a[], int low, int high, int tar)
  {    if (low > high)
                    return -1;
       if (a[low] == tar)
          return low;
       return recursiveLinearSearch(a, low + 1, high, tar);
  }
```

Question: Determine the time complexity of the recursiveLinearSearch function in the best, worst, and average cases.

- **Procedure/Program:**

Experiment **#2**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab):**

1. Define time complexity in the context of algorithms. How does time complexity influence the efficiency of an algorithm?
2. Explain the concept of space complexity. Why is it important to consider space complexity along with time complexity when analyzing algorithms?
3. Differentiate between worst-case, best-case, and average-case time complexity. How does each scenario impact the performance of an algorithm?
4. What is Big-O notation? How is it used to express the time complexity of algorithms? Provide an example to illustrate.
5. Explain the process of Merge Sort with a detailed step-by-step example. How does Merge Sort ensure its time complexity of O(nlogn)?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#3**		Student ID	
Date		Student Name	

Experiment Title: Implementation of programs on Sorting and Searching problems.

Aim/Objective: To understand the concept and implementation of programs on Sorting and Searching problems.

Description: The students will understand and able to implement programs on Sorting and Searching problems.

Pre-Requisites:

Knowledge: Sorting, Searching, Arrays in C/C++/Python

Pre-Lab:

1. Sort the following elements using Quick sort technique:

   54, 26, 93, 17, 77, 31, 44, 55, 20

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Stefan is a guy who is suffering with OCD. He always like to align things in an order. He got a lot of strings for his birthday party as gifts. He like to sort the strings in a unique way. He wants his strings to be sorted based on the count of characters that are present in the string.

   Input

   aaabbc

   aabbcc

   Output

   cbbaaa

   aabbcc

   If in case when there are two characters is same, then the lexicographically smaller one will be printed first

   Input:

   aabbccdd

   aabcc

   Output:

   aabbccdd

   baacc

- **Procedure/Program:**

Experiment **#3**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

1. Given an array arr[] of N strings, the task is to sort these strings according to the number of upper-case letters in them try to use zip function to get the format.

Input arr[] = {poiNtEr, aRRAy, cOde, foR}

Output: [('cOde', 1), ('foR', 1), ('poiNtEr', 2), ('aRRAy', 3)]

The input array describes the following:

“aRRAy” R, R, A->3 Upper Case Letters

“poiNtEr” N, E->2 Upper Case Letters

“cOde” O->2 Upper Case Letters

“foR” R->3 Upper Case Letters

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Andrea is working in a photo studio where his boss has given him a task to arrange the photos of family members. He is French and he do not know English somehow, he managed to send the list of names to you (his friend). Help Andrea to sort the photos.
(Note: implement the odd even merge algorithm)

Input

5

Neil Katherine Harry Stefan Dennis

Output

Dennis Harry Katherine Neil Stefan

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

1. James is sharing his information with his friend secretly in a chat. But he thinks that message should not understandable to anyone only for him and his friend. So he sent the message in the following format.

Input

a1b2c3d4e

Output

abbdcfdhe

Explanation:

The digits are replaced as follows:

- s[1] -> shift('a',1) = 'b'

- s[3] -> shift('b',2) = 'd'

- s[5] -> shift('c',3) = 'f'

- s[7] -> shift('d',4) = 'h'

- **Procedure/Program:**

Experiment **#4**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab):**

1. Can linear search be used on unsorted arrays? Why or why not?
2. How would you modify a linear search to return the index of all occurrences of a target value?
3. What are the advantages and disadvantages of linear search compared to other search algorithms?
4. What are the time complexities of binary search in the best, worst, and average cases?
5. Compare binary search and linear search in terms of efficiency and use cases.

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#4**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Program on Naïve Based and KMP Algorithm.

Aim/Objective: To implement and understand the functioning of the Naive Bayes classifier and Knuth-Morris-Pratt (KMP) algorithm for pattern matching.

Description: Naive Bayes Classifier:

A probabilistic classifier based on Bayes' theorem with the assumption of independence between every pair of features.

Pre-Requisites:

- Basic understanding of C programming language.
- Knowledge of fundamental concepts in probability and statistics.
- Familiarity with algorithms and data structures.
- Understanding of Bayes' theorem and probability distributions.
- Knowledge of string manipulation and pattern matching algorithms.

Pre-Lab:

1. We must find if a string is present in another string, as an example, the string "algorithm" is present within the string "naive algorithm". If it is found, then its location (i.e. position it is present at) is displayed. We tend to create a function that receives 2 character arrays and returns the position if matching happens otherwise returns -1.

   Input:

   txt = "HAVE A NICE DAY"

   pattern = "NICE"

   Output: Pattern found at index 8

   Input:

   txt = "WELCOME EVERYONE"

   pattern = "ONE"

   Output: Pattern found at index 14

Experiment **#4**		Student ID	
Date		Student Name	

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Experiment **#4**		Student ID	
Date		Student Name	

In-Lab:

1. Write a program to implement naive string matching algorithm for the below example.

Consider txt[] = "AAAAABBAABAAAACC", pat[] = "AAAA"

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Naive method and KMP are two string comparison methods. Write a program for Naïve method and KMP to check whether a pattern is present in a string or not. Using clock function find execution time for both and compare the time complexities of both the programs (for larger inputs) and discuss which one is more efficient and why? Sample program with function which calculate execution time:

```
#include<stdio.h>
#include<time.h>
void fun() {
//some statements here
}
int main() {
//calculate time taken by fun()
clock_t t;
t=clock();
fun();
t=clock()-t;
double time_taken=((double)t)/CLOCK_PER_SEC; //in seconds
printf("fun() took %f seconds to execute \n",time_taken);
return 0;
}
```

- **Procedure/Program:**

Experiment **#4**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Experiment **#4**		Student ID	
Date		Student Name	

Post-Lab:

Given a pattern of length- 5 window, find the valid match in the given text by step-by-step process using Robin-Karp algorithm

Pattern: 2 1 9 3 6

Modulus: 21

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

Text: 9 2 7 2 1 8 3 0 5 7 1 2 1 2 1 9 3 6 2 3 9 7

- **Procedure/Program:**

Experiment **#4**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab):**
  1. What is the main advantage of using the Knuth-Morris-Pratt (KMP) algorithm over the naive string matching algorithm?
  2. How does the naive algorithm search for a pattern in a text?
  3. What are the time complexities of the naive pattern matching algorithm in the best, worst, and average cases?
  4. What is the prefix function (or partial match table) in the context of the KMP algorithm?
  5. What are the time complexities of the KMP algorithm for preprocessing and searching?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#5**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Divide and Conquer Problems.

Aim/Objective: To understand and implement Divide and Conquer algorithms, and to analyze their performance in solving computational problems.

Description: Divide and Conquer is a powerful algorithmic paradigm used to solve complex problems by breaking them down into simpler sub-problems, solving each sub-problem recursively, and then combining their solutions to solve the original problem. This approach is used in many classical algorithms, such as Strassen's Multiplication, and convex hull algorithms for finding the closest pair of points.

- **Pre-Requisites:**

Understanding of Recursion: Familiarity with the concept of recursion and how recursive functions work. Ability to trace and debug recursive functions.

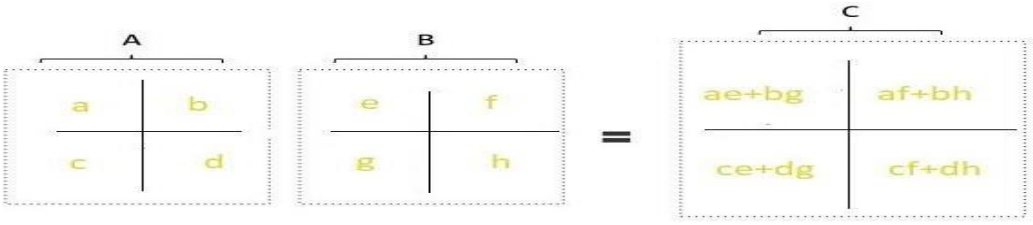
Basic Algorithm Analysis: Knowledge of Big-O notation and how to analyze the time complexity of algorithms. Understanding of recurrence relations and how to solve them.

Basic Data Structures: Proficiency in using arrays, lists, and other fundamental data structures. Understanding of data structures that can be used to implement Divide and Conquer algorithms.

Programming Skills: Competence in a programming language like Python, Java, C++, etc. Familiarity with writing and testing code in an integrated development environment (IDE).

Pre-Lab:

1. Trace the output of the following matrix multiplication using Strassen's Multiplication Method



A, B and C are the Matrices of Size NxN

a, b, c and d are the sub-Matrices of A of size N/2xN/2

e, f, g and h are the sub-Matrices of B of size N/2xN/2

- **Procedure/Program:**

Experiment **#5**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

2. Write a divide and conquer algorithm for finding the maximum and minimum in the sequence of numbers. Find the time complexity.

- **Procedure/Program:**

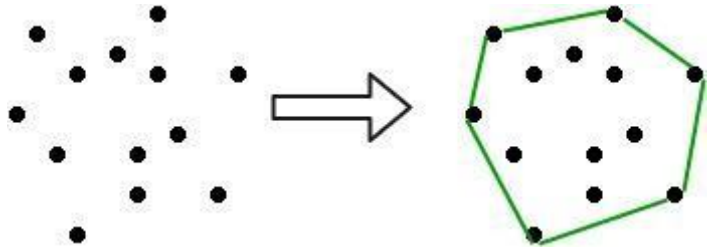
Experiment **#5**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

1. Given an input is an array of points specified by their x and y co-ordinates. The output is the convex hull of this set of points by using Divide and Conquer algorithm.



Input : points[] = {(0, 0), (0, 4), (-4, 0), (5, 0), (0, -6), (1, 0)};

Output : (-4, 0), (5, 0), (0, -6), (0, 4)

Experiment **#5**		Student ID	
Date		Student Name	

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Harry's Aunt and family treat him badly and make him work all the time. Dudley, his cousin got homework from school and he as usual handed it over to Harry but Harry has a lot of work and his own homework to do.

The homework is to solve the problems which are numbered in numerical he tries to solve random question after solving random questions he did not put those questions in order Dudley will return in a time of n*logn Harry has to arrange them as soon as possible. Help Harry to solve this problem so that he can go on and do his own homework.

Example

Input

9

15,5,24,8,1,3,16,10,20

Output

1, 3, 5, 8, 10, 15, 16, 20, 24

- **Procedure/Program:**

Experiment **#5**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

1. Matrix Chain Multiplication

Problem Statement: Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not to perform the multiplications, but to determine the order in which to multiply the matrices such that the total number of scalar multiplications is minimized.

```
def matrix_chain_order_recursive(p, i, j):
 if i == j:
 return 0
 min_cost = sys.maxsize
for k in range(i, j):
cost = (matrix_chain_order_recursive(p, i, k) + matrix_chain_order_recursive(p, k+1, j) + p[i-1] * p[k] * p[j])
if cost < min_cost:
```

min_cost = cost

return min_cost

- **Procedure/Program:**

Experiment **#5**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the divide and conquer approach?
2. Can you explain the three main steps involved in the divide and conquer strategy?
3. What are the seven submatrix multiplications used in Strassen's algorithm?
4. What is the convex hull of a set of points?
5. What are the key steps involved in the Graham scan algorithm?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#6**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Greedy method Problems - Job Sequence with Deadlines and Knapsack Problems.

Aim/Objective: Students can able to apply and analyze the Job Sequence with Deadlines and knapsack problems on greedy method. The aim of these algorithm to find the optimal solution.

Description: The Job Sequencing with Deadlines problem is a scheduling problem where the goal is to maximize the total profit by selecting a subset of jobs to complete within their respective deadlines. Each job has a deadline and a profit associated with it.

The Knapsack Problem is a classic optimization problem where the goal is to maximize the total profit of items that can be placed into a knapsack of fixed weights. Each item has a specific weight and profit, and the knapsack has a weight limit.

Pre-Requisites:

Pre-Lab:

1. Given the jobs, their deadlines and associated profits as shown:

   Jobs, J1, J2, J3, J4, J5, J6

   Deadlines, 5, 3, 3, 2, 4, 2

   Profits, 200, 180, 190, 300, 120, 100

   Answer the following questions:

   Write the optimal schedule that gives maximum profit.

   Are all the jobs completed in the optimal schedule?

   What is the maximum earned profit?

- **Procedure/Program:**

Experiment **#6**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

2. Explain why 0-1 Knapsack problems cannot be solved using greedy method unlike fractional knapsack. (Students may attempt this exercise after completion of 0/1 knapsack in dynamic approach)

- **Procedure/Program:**

Experiment **#6**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

1. Given an array of size n that has the following specifications:

   a. Each element in the array contains either a police officer or a thief.

   b. Each police officer can catch only one thief.

   c. A police officer cannot catch a thief who is more than K units away from the police officer.

   We need to find the maximum number of thieves that can be caught.

   Input

   arr [] = {'P', 'T', 'T', 'P', 'T'},

   k = 1.

   Output

   2

   Here maximum 2 thieves can be caught; first police officer catches first thief and second police officer can catch either second or third thief.

- **Procedure/Program:**

Experiment **#6**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

1. Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

   Input

   4

Job	**ID**	**Deadline Profit**
a	4	20
b	1	10
c	1	40
d	1	30

   Output

   60

   Profit sequence of jobs is c, a

Experiment **#6**		Student ID	
Date		Student Name	

- **Procedure/Program:**

- **Data and Results:**

Experiment **#6**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions:**

1. Describe the steps involved in the greedy algorithm for Job Sequencing with Deadlines.
2. Why do we sort the jobs in decreasing order of profit?
3. Under what circumstances might the greedy algorithm be less effective?
4. What is the time complexity of the greedy algorithm for the Fractional Knapsack problem?
5. What is the significance of the value-to-weight ratio in the selection process?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#7**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Greedy method Problems – MST and Single Source Shortest Path.

Aim/Objective:

To implement algorithms for solving Minimum Spanning Tree (MST) and Single Source Shortest Path (SSSP) problems using greedy methods.

Description:

The goal is to understand and apply greedy algorithms to solve MST and SSSP problems efficiently. For MST, we will use Kruskal's and Prim's algorithms. For SSSP, we will use Dijkstra's algorithm.

Pre-Requisites:

- Basic understanding of graphs and their representations (adjacency matrix/list).
- Familiarity with greedy algorithm concepts.
- Knowledge of data structures like heaps and disjoint-set data structures.

Pre-Lab:

Given a graph with vertices A, B, C, D, and E, and the following edges with weights: (A-B, 1), (A-C, 3), (B-C, 2), (B-D, 4), (C-D, 5), (C-E, 6), (D-E, 7). Perform Prim's algorithm step-by-step to find the MST. Illustrate your steps and show the final MST.

- **Procedure/Program:**

Experiment **#7**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

In a city consider the Apartments as nodes and the possible cable connections as edges with costs. Implement Kruskal's algorithm to determine the optimal way to connect all Apartments. Write a program the implications of using a greedy algorithm for this task.

Input:

Apartments = ["Aparna Amaravati ", "Jayabheri", "Vajra Residency", "Sunrise Towers", "Rams enclave"]

cable_connections = [("Aparna Amaravati ", "Jayabheri", 10), ("Aparna Amaravati ", " Vajra

Experiment **#7**		Student ID	
Date		Student Name	

Residency ", 20), ("Jayabheri ", “Vajra Residency", 30), ("Jayabheri ", “Sunrise Towers ", 40), ("Vajra Residency", “Sunrise Towers ", 50), ("Sunrise Towers ", “Rams enclave ", 60), ("Vajra Residency ", " Rams enclave ", 70)]

Output:

Total cost: 130

MST: [('Aparna Amaravati', 'Jayabheri', 10), ('Aparna Amaravati', 'Vajra Residency', 20), ('Jayabheri', 'Sunrise Towers', 40), ('Sunrise Towers', 'Rams enclave', 60)]

- **Procedure/Program:**

Experiment **#7**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

Write a program to model the delivery points as nodes and travel times as edges. Use Dijkstra's algorithm to find the shortest route from the warehouse to each delivery point. Explain how this optimization can reduce delivery times and improve customer satisfaction.

Input:

delivery_points = ["W", "C1", "C2", "C3", "C4"]

travel_times = { "W": {"C1": 3, "C2": 6}, "C1": {"W": 3, "C3": 2}, "C2": {"W": 6, "C3": 4, "C4": 2}, "C3": {"C1": 2, "C2": 4, "C4": 1}, "C4": {"C2": 2, "C3": 1} }

start_point = "W"

Output:

Shortest paths from W:

W-> W : 0

W-> C1 : 3

W -> C3 : 5

W -> C2 : 6

W -> C4 : 6

- **Procedure/Program:**

Experiment **#7**		Student ID	
Date		Student Name	

- **Data and Results:**

Experiment **#7**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab)**:

1. What is a greedy algorithm?
2. What are the characteristics of problems suitable for greedy algorithms?
3. What is the time complexity of Prim's algorithm, Kruskal's algorithm and Dijkstra's algorithm?
4. What are the limitations or constraints of Greedy Method approaches in solving MST and SSSP problems?
5. Explain how the time complexity is derived.

Evaluator Remark (if Any):	**Marks Secured ___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#8**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Dynamic Programing – OBST, 0/1 Knapsack Problems and TSP.

Aim/Objective: To implement and understand the Optimal Binary Search Tree (OBST) algorithm using dynamic programming techniques.

Description: In the context of a library catalog system, the Optimal Binary Search Tree (OBST) algorithm is employed to organize book titles and their corresponding access frequencies. By utilizing dynamic programming techniques, the algorithm constructs a binary search tree that minimizes the average search cost, ensuring efficient retrieval of books based on their frequencies of access.

Pre-Requisites:

- Basic understanding of binary search trees, 0/1 Knapsack Problems and TSP.
- Familiarity with dynamic programming principles.
- Proficiency in a programming language such as Python or C++.

Pre-Lab:

In KL University, every student has to participate in NCC activity regularly. Each day NCC activity has its own duration. You will be given number of days schedule and the number of hours you need to spend on each day. The rule you need to follow is that you cannot skip NCC activity 3 days in a row.

Your task is to find the minimum number of hours that you spend on activities by following given rule.

Input:

Line 1: An integer "n" representing number of days

Line 2: n non negative integers representing the hours to spend on each day

Output:

A single non negative integer representing number of hours that you want to spend by following the given rule.

Sample Input:

10

3 4 1 1 2 3 2 3 2 1

8

3 2 3 2 3 5 1 3

7

3 2 5 5 4 2 4

Sample Output:

5 (1+1+2+1)

5(2+2+1)

6(2+4+2)

- **Procedure/Program:**

Experiment **#8**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

1. You are given a set of keys and their frequencies, representing the number of times each key is accessed. The goal is to construct a binary search tree with these keys in such a way that the average search cost is minimized. This involves finding an optimal way to arrange the keys in the tree using dynamic programming techniques.

Input:

Keys: [10, 20, 30, 40, 50]

Frequencies: [4, 2, 6, 3, 5]

Output:

Minimum Average Search Cost: 34

- **Procedure/Program:**

Experiment **#8**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

2. The 0/1 Knapsack problem involves selecting items with given weights and values to maximize the total value without exceeding a specified weight limit. Each item can either be included in the knapsack (1) or excluded (0). Dynamic programming is used to solve this problem by breaking it down into simpler subproblems and solving each subproblem only once.

Input:

Weights of Items: [2, 3, 4, 5]

Values of Items: [3, 4, 5, 6]

Knapsack Capacity: 8

Output:

Maximum Value: 10

- **Procedure/Program:**

Experiment **#8**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

A delivery person needs to visit 4 locations (Home, Office, Grocery Store, Park) with known distances between each pair of locations. The goal is to determine the shortest route that visits each location exactly once and returns to the starting location.

Input:

Locations: [Home, Office, Grocery Store, Park]

Distance Matrix:

0	5	10	15
5	0	8	20
10	8	0	12
15	20	12	0

Output:

Shortest Route: [Home, Office, Grocery Store, Park, Home]

Total Distance: 37

- **Procedure/Program:**

Experiment **#8**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Experiment **#8**		Student ID	
Date		Student Name	

- **Sample VIVA-VOCE Questions (In-Lab):**

1. Explain the concept of memorization in dynamic programming.
2. How does the 0/1 Knapsack problem differ from the fractional knapsack problem?
3. Describe the Optimal Binary Search Tree (OBST) problem and its significance.
4. What is the main challenge in solving the Travelling Salesman Problem (TSP)?
5. What are the advantages of using dynamic programming over brute-force methods?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#9**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Dynamic Programming – Graph coloring Problems and Sum of Subset Problem.

Aim/Objective: Students able to apply and analyze Graph Coloring and Sum of subset problem on Dynamic Programing approach.

Description: The Graph Coloring Problem involves assigning colors to vertices of a graph such that no two adjacent vertices share the same color, using the minimum number of colors possible.

The Sum of Subset Problem involves finding whether there exists a subset of a given set S such that the sum of elements in the subset equals a specified target sum T.

Pre-Requisites:

- Basics of Data Structures and C Programming.
- Basic knowledge about arrays.
- Familiarity with the concept of backtracking.

Pre-Lab:

1. Define X(N,K) be number of subsets of K distinct elements of S where N is the size of S. Given that P(<=N), Val=X(N,0)+X(N,1)+….+X(N,P). Your task is to print Val using dynamic programming.

   Input:

   First line contains, T, the number of test cases. Each test case consists of N and P in one line.

   Output:

   Print required answer in one line for each test case.

   Sample Input:

   2

   2 2

   2 0

   Sample Output:

   4

   1

Experiment **#9**		Student ID	
Date		Student Name	

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Given N cities numbered from 1 to N. Your task is to visit the cities. Here K cities are already visited. You can visit i^{th} city if $(i-1)^{th}$ or $(i+1)^{th}$ city is already visited. Your task is to determine the number of ways you can visit all the remaining cities.

   Input format:

   First line: Two space-separated integers N and K

   Second line: K space-separated integers each denoting the city that is already visited

   Output format:

   Print an integer denoting the number of ways to visit the remaining cities.

   Sample Input:

   6 3

   1 2 6

   Sample Output:

   4 ({3, 4, 5}, {3, 5, 4}, {5, 3, 4}, {5, 4, 3})

   - **Procedure/Program:**

Experiment **#9**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

1. Given an undirected graph and N colors, the problem is to find if it is possible to color the graph with at most N colors, which means assigning colors to the vertices of the graph such that no two adjacent vertices of the graph are colored with the same color. Print "Possible" if it is possible to color the graph as mentioned above, else print "Not Possible".

Input

1
3 2
0 2
1 2
2

Output

Possible

Experiment **#9**		Student ID	
Date		Student Name	

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

2. Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum. A subarray is a contiguous part of an array.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

Example 2:

Input: nums = [1]

Output: 1

Example 3:

Input: nums = [5,4,-1,7,8]

Output: 23

- **Procedure/Program:**

Experiment **#9**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

Karthik was given a problem in an online interview, but he cannot solve the solution help him solve the question. There are n students whose ids vary between 0 to 9 digits; two students can have same id's. You will be given x numbers which also vary from 0 to 9. You need to find the total number of student id's subsets which contains all the x numbers.

Input:

Student ID's: "333", "1", "3"

X = {3, 1, 3}

Output

6

- **Procedure/Program:**

Experiment **#9**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences**

- **Sample VIVA-VOCE Questions (In-Lab):**

1. Can you explain the graph coloring problem and its significance?
2. How does backtracking help in solving the graph coloring problem?
3. What is dynamic programming, and how is it applied to the graph coloring problem?
4. Explain the subset sum problem and its relevance.
5. How does backtracking assist in solving the subset sum problem?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#10**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Backtracking Problems – Eight queens and 0/1 knapsack Problems.

Aim/Objective: To understand the backtracking algorithm and implement eight queens and 0/1 knapsack problems**.** The aim of this experiment is to implement the Eight Queens problem using backtracking to find all possible ways to place eight queens on an 8x8 chessboard so that no two queens threaten each other.

Description: The Eight Queens problem is a classic example of backtracking where the objective is to place eight queens on a chessboard such that no two queens can attack each other. This means no two queens should be in the same row, column, or diagonal.

Pre-Requisites:

→ Basic understanding of arrays and recursion.

→ Knowledge of chessboard geometry.

→ Familiarity with the concept of backtracking.

Pre-Lab:

1. There's an array A consisting of N non-zero integers A1..N. A subarray of A is called alternating if any two adjacent elements in it have different signs (i.e. one of them should be negative and the other should be positive). For each x from 1 to N, compute the length of the longest alternating subarray that starts at x - that is, a subarray Ax..y for the maximum possible $y \geq x$. The length of such a subarray is y-x+1.

Input

The first line of the input contains an integer T - the number of test cases.

The first line of each test case contains N.

The following line contains N space-separated integers A1..N.

Output

For each test case, output one line with N space-separated integers - the lengths of the longest alternating subarray starting at x, for each x from 1 to N.

Constraints

$1 \leq T \leq 10$

$1 \leq N \leq 10^5$

$-10^9 \leq Ai \leq 10^9$

Experiment **#10**		Student ID	
Date		Student Name	

Sample 1:

Input	Output
4 1 2 3 4 4 1 -5 1 -5 6 -5 -1 -1 2 -2 -3	1 1 1 1 4 3 2 1 1 1 3 2 1 1

Explanation:

Example case 1. No two elements have different signs, so any alternating sub array may only consist of a single number.

Example case 2. Every sub array is alternating.

Example case 3. The only alternating sub array of length 3 is A3..5.

- **Procedure/Program:**

Experiment **#10**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

Problem Statement: Real-World Scenario Utilizing Backtracking **(Sum of Subsets)**

Problem: Optimal Packing in Logistics

1) In logistics and transportation, optimizing the packing of goods in containers or vehicles to utilize available space efficiently is critical. The problem is akin to the "sum of subsets" where you aim to find subsets of items whose total weight or volume equals a target value, ensuring the most efficient use of available capacity.

- **Procedure/Program:**

Experiment **#10**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Experiment **#10**		Student ID	
Date		Student Name	

Post-Lab:

Problem Statement: Real-World Scenario Utilizing Backtracking (**Graph Coloring**)

Problem: Scheduling Examinations in Educational Institutes

Scenario:

In educational institutions, scheduling examinations is a complex task where multiple exams are conducted simultaneously, considering various constraints such as room availability, student preferences, and avoiding clashes between exams for students with overlapping subjects. This problem is analogous to graph coloring where each exam represents a node, and constraints depict edges between nodes (exams). Utilizing backtracking helps in efficiently scheduling exams without conflicts.

- **Procedure/Program:**

Experiment **#10**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the Eight Queens problem?
2. Why is the Eight Queens problem significant in computer science?
3. What is backtracking in the context of algorithm design?
4. How does backtracking help in solving the Eight Queens problem?
5. Can you explain the difference between backtracking and brute force?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#11**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Branch and Bound Problems – TSP and 0/1 Knapsack.

Aim/Objective: : To understand the concept and implementation of Basic programs on Branch and Bound Technique Problems.

Description: The students will understand and able to implement programs on Branch and Bound Technique Problems.

Pre-Requisites:

Knowledge: Branch and Bound Technique in C/C++/Python

Tools: CodeBlocks/EclipseIDE/Python IDLE

Pre-Lab:

1. XYZ Logistics is a medium-sized logistics company that provides delivery services across multiple cities. The company aims to minimize travel costs and delivery times to improve efficiency and customer satisfaction. They face the challenge of determining the optimal route for their delivery trucks that need to visit several cities and return to the starting point. The company decides to implement the Branch and Bound method to solve their Traveling Salesman Problem (TSP).

Problem Formulation:

- Objective: Minimize the total distance travelled by a delivery truck.
- Constraints: Each city must be visited exactly once, and the truck must return to the starting city.

Let's consider a scenario where there are 4 cities (A, B, C, and D) and the distances between them are as follows:

- Distance from A to B: 10
- Distance from A to C: 15
- Distance from A to D: 20
- Distance from B to C: 35
- Distance from B to D: 25
- Distance from C to D: 30

The goal is to find the shortest possible route that visits each city exactly once and returns to the starting city.

- **Procedure/Program:**

Experiment **#11**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

2. Consider the following 0/1 Knapsack Problem:

You are a thief planning to rob a store. You have a backpack with a maximum weight capacity of 15 kilograms. The store contains the following items:

Item	Weight (kg)	Value ($)
A	2	10
B	4	25
C	6	15
D	3	20
E	5	30

You can only take whole items (either you take an item completely or you don't take it at all). Determine the maximum value of items you can steal without exceeding the weight capacity of your backpack using the branch and bound algorithm.

Solution Approach:

- **Formulate the problem:** Define the objective function and constraints.
- **Apply the branch and bound algorithm:** Break down the problem into subproblems and use bounding techniques to narrow down the search space.
- **Find the optimal solution:** Determine the maximum value of items that can be stolen without exceeding the weight capacity.

- **Procedure/Program:**

Experiment **#11**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

Task1: Traveling Salesman Problem with Least Cost Branch and Bound

Problem: Imagine you're a delivery driver tasked with visiting all customers in a city but minimizing the total distance traveled. You need to find the most efficient route that ensures you visit each customer and return to the starting point.

Objective:

Implement the Traveling Salesman Problem (TSP) using the Least Cost Branch and Bound algorithm to find the shortest route a salesperson can take to visit all cities exactly once and return to the starting point.

Approach: You decide to use the Lower Cost Branch and Bound (LCBB) algorithm to efficiently search the solution space.

Input:

- **Distance Matrix:** A 2D array of size (n x n), where n is the number of cities. Each entry

distanceMatrix[i][j] represents the cost/distance between city i and city j. The matrix should be symmetric (distance from i to j is the same as j to i).

Output:

- **Minimum Tour Cost:** The total cost of the shortest route visiting all cities exactly once and returning to the starting point.
- **Tour Path (Optional):** An ordered list of city indices representing the optimal route for the salesman.

Additional Considerations:

- You can decide whether to accept the distance matrix directly as input or allow the user to enter the distances for each pair of cities.
- The program should handle invalid inputs like non-symmetric distance matrices or negative distances.
- Error messages or appropriate handling should be implemented for such cases.
- The output format for the minimum tour cost can be a simple numeric value.
- The optional tour path output can be an array or list of city indices in the optimal visiting order.

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Task2: 0/1 knapsack problem with Least Cost Branch and Bound

Problem: Imagine you're a thief planning a heist and need to choose the most valuable items from a safe that has a weight limit. Each item has a value and a weight. You can only take an item entirely (not a fraction) and aim to maximize the total value of stolen items without exceeding the safe's weight capacity.

Objective:

Implement the 0/1 Knapsack Problem using the Branch and Bound algorithm to find the optimal selection of items with maximum total value that fits within a limited knapsack capacity.

Approach: You decide to use the Lower Cost Branch and Bound (LCBB) algorithm to efficiently search the solution space.

Input:

- **Item Values:** An array of size n where values[i] represents the value of item i.
- **Item Weights:** An array of size n where weights[i] represents the weight of item i.
- **Knapsack Capacity:** The weight limit of the knapsack.

Experiment **#11**		Student ID	
Date		Student Name	

Output:

- **Maximum Total Value:** The total value of the optimal selection of items that fits within the knapsack capacity.
- **Optimal Selection (Optional):** An ordered list of item indices representing the items included in the optimal solution.
- **Additional Considerations:**
- You can decide whether to accept the item values, weights, and capacity directly as input or allow the user to enter them individually.
- The program should handle invalid inputs like negative values or a capacity that cannot accommodate any items.
- Error messages or appropriate handling should be implemented for such cases.
- The output format for the maximum total value can be a simple numeric value.
- The optional optimal selection output can be an array or list of item indices in the knapsack for the optimal solution.

- **Procedure/Program:**

Experiment **#11**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post Lab:

1. Problem Statement: There are N cities. The time it takes to travel from City i to City j is $T_{i,j}$. Among those paths that start at City 1, visit all other cities exactly once, and then go back to City 1, how many paths take the total time of exactly K to travel along?

Constraints:

$2 \leq N \leq 8$
if $i \neq j, 1 \leq T_{i,j} \leq 10^8$
$T_{i,i} = 0$
$T_{i,j} = T_{j,i}$
$1 \leq K \leq 10^9$
All the Input values are integers

Input

Input is given from Standard Input in the following format:

N K
$T_{1,1} \ldots\ldots T_{1,N}$
$\vdots\vdots$
$T_{1,N} \ldots\ldots T_{N,N}$

Output

Print the answer as an integer.

Sample Input 1

```
4 330
0 1 10 100
1 0 20 200
10 20 0 300
100 200 300 0
```

Sample Output 1

2

There are six paths that start at City 1, visit all other cities exactly once, and then go back to City 1:

1→2→3→4→1
1→2→4→3→1
1→3→2→4→1
1→3→4→2→1
1→4→2→3→1
1→4→3→2→1

The times it takes to travel along these paths are 421, 511, 330, 511, 330, and 421, respectively, among which two are exactly 330.

Sample Input 2

```
5 5
0 1 1 1 1
1 0 1 1 1
1 1 0 1 1
1 1 1 0 1
1 1 1 1 0
```

Sample Output 2

24

In whatever order we visit the cities, it will take the total time of 5 to travel.

- **Program:**

Experiment **#11**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inference:**

2. Problem Statement

There are *N* items remaining in the company. The weight of the *i*-th item is 1<=i<=N and Wi
Takahashi will sell these items as *D* lucky bags.
He wants to minimize the variance of the total weights of the items in the lucky bags.
Here, the variance is defined as

$V = \frac{1}{D}\sum_{i=1}^{D}((x_i - \bar{x})^2)$

where x1, x2,..xD are the total weights of the items in the lucky bags, and

$\bar{x} = \frac{1}{D}(x_1 + x_2 + \ldots x_D)$

is the average of x1, x2,..,xD . Find the variance of the total weights of the items in the lucky bags when the items are divided to minimize this value.

It is acceptable to have empty lucky bags (in which case the total weight of the items in that bag is defined as *0*), but **each item must be in exactly one of the *D* lucky bags**.

Constraints

- $2 \leq D \leq N \leq 15$
- $1 \leq D \leq 10^8$
- All input values are integers.

Input

The input is given from Standard Input in the following format:

```
N D
W1, W2 ..... WN
Output
```

Print the variance of the total weights of the items in the lucky bags when the items are divided to minimize this value.

Your output will be considered correct if the absolute or relative error from the true value is at most *10−6*.

Sample Input 1

```
5 3
3 5 3 6 3
```

Sample Output 1

```
0.888888888888889
```

If you put the first and third items in the first lucky bag, the second and fifth items in the second lucky bag, and the fourth item in the third lucky bag, the total weight of the items in the bags are *6*, *8*, and *6*, respectively.

Then, the average weight is $\frac{1}{3}\left(6+8+6=\frac{20}{3}\right)$

and the variance is $\frac{1}{3}\left\{\left(6-\frac{20}{3}\right)^2+\left(8-\frac{20}{3}\right)^2+\left(6-\frac{20}{3}\right)^2\right\}=0.88888\ldots$ which is the minimum.

Note that multiple items may have the same weight, and that each item must be in one of the lucky bags.

Program:

Experiment **#11**		Student ID	
Date		Student Name	

- **Data/ Results:**

- **Analysis and Inference:**

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the difference between backtracking and branch and bound?
2. List the applications of Brach and Bound approach
3. List the methods of Branch and bound
4. Define state space tree
5. Define E-Node and Live node
6. In what real-world scenarios might TSP be applicable?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#12**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on NP Hard and NP Complete Problems.

Aim/Objective: To understand the concept and implementation of Basic Programs on NP-Hard and NP-Complete.

Description: The can able to understand the concepts of NP-Hard and NP-Complete

Pre-Requisites:

Knowledge: Non-deterministic Algorithms, 3 SAT Problems, Clique Decision Problems, Node Cover Decision Problems implemented in C/C++/Java/Python.

Tools: Code Blocks/ Eclipse IDE/Python

Pre-Lab: Read the following conversation

Sagar: Is Conjunctive Normal Formula an NP-Hard Problem?

Deepa: Yes, CNF is NP-Hard Problem ?

Sagar: Can you prove...!

You are Deepa's friend. Help her to prove the conjunctive normal formula is a NP-Hard Problem.

- **Procedure/Program**:

Experiment **#12**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

In-Lab:

Given an undirected graph G returns, the minimum number of vertices needed so that every vertex is adjacent to the selected one. In short, return the size of the vertex cover of the graph.

Input:

N=5

M=6

Edges [ ] [ ] = {{1,2}, {4, 1}, {2, 4}, {3, 4}, {5, 2}, {1, 3}}

Output: 3

Explanation: Just pick 2, 3, 4.

- **Procedure/Program:**

Experiment **#12**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

Post-Lab:

Write the procedure to prove that the clique decision problem and Node cover decision problems are NP-Hard Problems.

- **Procedure/Program:**

Experiment **#12**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions :**

1. Differentiation between Deterministic and Non-Deterministic Algorithms?
2. What is NP-Complete Problems?
3. Write one word statement about P and NP Class Problems?
4. Mention the relation between P and NP-Hard Problems?
5. Write the Cook's Theorem?

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment **#12**		Student ID	
Date		Student Name	

Experiment **#13**		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs/ Algorithms on CDP, NCDP and AOG problems.

Aim/Objective: To understand the concept and implementation of Basic program on NCDP as NP Hard problem

Description: The students will understand and able to implement programs on NCDP as NP Hard problem.

Pre-Requisites:

Knowledge: NCDP as NP Hard problem in C/C++/PythonTools: Code Blocks/Eclipse IDE

Pre-Lab:

1) Vijval has an array A of length N.

In one operation, Vijval can choose any two **distinct** indices i,j ($1 \le i,j \le N, i=j$) and **either** change Ai to Aj **or** change Aj to Ai.

Find the **minimum** number of operations required to make all the elements of the array **equal**.

Input Format

- First line will contain T, number of test cases. Then the test cases follow.
- First line of each test case consists of an integer N - denoting the size of array A.
- Second line of each test case consists of N space-separated integers 1,2,…,A1,A2,…,AN - denoting thearray A.

Output Format

For each test case, output the minimum number of operations required to make all the elements equal.Constraints

$1 \le T \le 100$

$2 \le N \le 1000$

$1 \le Ai \le 1000$Sample 1:

Input4

4

4 5 6 7

3

6 6 6

4

3 3 2 2

3

2 2 3

Output3

0

2

1

Explanation:

Test Case 1**:** You can make all the elements equal in 22 operations. In the first operation, you can choose indices 4,5 and convert A1 to A2. So the array becomes [4,4,6,7]. Now you can choose indices 4,6 and convert A6 to A4, etc., so the final array becomes [4,4,4,4].

Test Case 2**:** Since all the elements are already equal there is no need to perform any operation.

- **Procedure/Program:**

- **Data and Results:**

Experiment **#13**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

In-Lab:

1. The Clique Decision Problem belongs to NP-Hard. Prove that the Boolean Satisfiability problem reduces to the Clique Decision Problem

- **Procedure/Program:**

- **Data and Results:**

Experiment **#13**		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

Post-Lab

1) You are given two integer A and B.

You need to compute and output the Greatest common divisor and Least common multiple of these 2numbers and store them in the variables GCD and LCM.

Input Format

The first line of input will contain a single integer T, denoting the number of test cases.Each test case consists of a single line of input - the integer A and B.

Output Format

For each test case, output the GCD and LCM on one line

Sample 1:

Input

2

4 9

24 32

Output

1 36

8 96

- **Procedure/Program:**

Experiment **#13**		Student ID	
Date		Student Name	

- **Data and Results:**

- **Analysis and Inferences:**

- **Sample VIVA-VOCE Questions:**

1) Differentiate between CDP and NCDP?

2) List the NP-Hard Graph problems?

3) What is Reducibility?

4) Identify one difference between Satisfiability and Reducibility?

5) What is AOG?

Evaluator Remark (if Any):	**Marks Secured ___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment **#13**		Student ID	
Date		Student Name	