Rashtreeya Sikshana Samithi Trust

RV UNIVERSITY

School of Computer Science and Engineering Bengaluru – 560059

FUNDAMENTALS OF DATA STRUCTURES AND ALGORITHMS

COURSE CODE: CS1082

II SEMESTER B.C.A (Hons.)

LABORATORY RECORD

2024-2025

RV UNIVERSITY

School of Computer Science and Engineering Bengaluru – 560 059

LABORATORY CERTIFICATE

This is to certify that Mr./Ms.

has satisfactorily completed the course of experiments in Practical

FUNDAMENTALS OF DATA STRUCTURES AND ALGORITHMS

(CS1082) prescribed by the **School of Computer Science and Engineering**

during the year **2024-25.**

Name of the Candidate:

USN: Semester:

Marks	
Maximum	**Obtained**
25	

Signature of Faculty in-charge Program Director

Date:

Vision and Mission of the School of Computer Science and

Engineering

Vision

To be a pioneering school of Computer Science and Engineering committed to fostering liberal education and empowering the next generation of technologists to make a positive global socio-economic impact.

Mission

- To be a pioneer in computer science education and benchmark ourselves with the world's top computer science and engineering institutions.

- To provide state-of-the-art facilities that enable exemplary pedagogy, advanced research, innovation and entrepreneurship in emerging technologies of computer science.

- To promote a culture of cooperation and inclusiveness among students and faculty from diverse communities enabling them to take part in interdisciplinary and multidisciplinary research, contributing to institution building.

- To foster excellence through national and international academic, industry collaborations, bringing in diverse perspectives to drive innovation.

- To nurture a talented pool of ethical, self-driven and empathetic problem solvers to achieve sustainable development goals.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will demonstrate proficiency in solving industrial and societal problems, conducting research, innovation and entrepreneurship.

PEO2: Graduates will demonstrate strong leadership by excelling in interdisciplinary and multidisciplinary teamwork f98789 or sustainable global development.

PEO3: Graduates will embrace lifelong learning and digital proficiency with ethical standards

PROGRAM OUTCOMES (POs)

BCA(Hons) Graduates will be able to:

PO1: Complex Problem-Solving and Critical Thinking: Graduates should be proficient in using software tools to solve diverse problems in both familiar and unfamiliar contexts. They should apply their learning to real-life situations through software applications, demonstrating the ability to utilize analytic thought in processing a body of knowledge. This includes the use of software for the analysis and evaluation of policies, practices, evidence, and arguments, as well as assessing the reliability and relevance of evidence. Graduates should be adept at using software to identify assumptions and implications, formulate coherent arguments, detect logical flaws in others' arguments, and synthesize data from various sources. Ultimately, they should be able to draw valid conclusions supported by evidence and examples through the effective use of software tools.

PO2: Creative and Effective Communication: Description: Graduates should be able to demonstrate creativity by thinking in diverse ways, dealing with complex problems, innovating, and viewing situations from multiple perspectives. They should also effectively communicate by listening carefully, presenting complex information clearly, expressing thoughts and ideas both in writing and orally, sharing views confidently, constructing logical arguments using correct technical language, and conveying ideas respectfully and sensitively to different audiences.

PO3: Analytical and Research Competence: Graduates should be able to evaluate the reliability and relevance of evidence, identify logical flaws in arguments, and analyze and synthesize data from various sources to draw valid conclusions and support them with evidence. They should also demonstrate research-related skills, including a keen sense of observation and inquiry, the ability to define problems and formulate relevant research questions, design research proposals, use appropriate methodologies and tools, and apply statistical and analytical techniques.

PO4: Teamwork and Leadership: Graduates should be able to work effectively and respectfully with diverse teams, facilitating cooperative efforts and acting together towards common goals. They should demonstrate leadership by mapping out tasks, setting directions, formulating inspiring visions, building

and motivating teams to achieve these visions, and using management skills to guide team members to the right destination.

PO5: Lifelong Learning and Digital Proficiency: Graduates should demonstrate the ability to acquire new knowledge and skills for lifelong learning, adapt to changing workplace demands, work independently, and manage time and resources effectively. They should also exhibit digital and technological proficiency by using ICT in various learning and work situations, accessing and evaluating information sources, and using appropriate software for data analysis.

PO6: Multicultural Competence and Ethical Values: Graduates should demonstrate knowledge of multiple cultures, interact respectfully with diverse groups, lead diverse teams, and adopt a gender neutral and empathetic approach. They should also embrace and practice constitutional, humanistic, ethical, and moral values, engage in responsible global citizenship, recognize and address ethical issues, follow ethical practices, and promote sustainability and integrity in all aspects of their work.

PO7: Responsible Autonomy and Environmental Stewardship: Graduates should demonstrate the ability to apply knowledge and skills independently, manage projects to completion, exercise responsibility, and ensure workplace safety and security. They should also be able to take appropriate actions to mitigate environmental degradation, climate change, and pollution, practice effective waste management, conserve biodiversity, and promote sustainable development and living.

PO8: Community Engagement and Empathy: Graduates should demonstrate the capability to participate in community-engaged services and activities to promote societal well-being, and the ability to empathize by understanding the perspectives, experiences, and emotions of others.

Course Outcomes: After completing the course, the students will be able to:	
CO1	Understand the basic concepts of Programming required for data structures.
CO2	Apply user defined data types to implement single linked list, stacks and queues
CO3	Execute the doubly linked list and circular linked list with various operations on it
CO4	Implement the Binary Tree Data structure for different traversal methods

CO5	Create graph data structure and incorporate cycle detection

CO-PO Mapping								
CO/PO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**
CO1	3	-	-	-	2	-	-	-
CO2	3	2	-	-	1	-	-	-
CO3	3	2	-	-	1	-	-	-
CO4	3	2	-	-	1	-	-	-
CO5	3	2	-		1	-	-	-

LIST OF PROGRAMS

PART – A

Sl. No.	**Program**	**Week**
1.	Write a program to find the largest and smallest element in an array.	Week 2
2.	Write a program to find the factorial of a number using recursion. Write a program to find the nth Fibonacci number using recursion.	Week 2
3.	Write a program to implement the Sudoku game structure and implement searching operations along rows, columns and grids.	Week 3

4.	Write a program to create a class/structure Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Implement the following operations: 1. insert_at_beginning(data): Insert a new node with the given data at the beginning of the list. 2. .insert_at_end(data): Insert a new node with the given data at the end of the list. 3. delete_node(data): Delete the first node in the list that contains the given data 4. traverse (): Traverse the list and print the data of each node	Week 4
5.	Write a program to create a class/structure Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Then, create a class/structure Stack to represent the stack itself. Implement the following operations: 1. push(data): Push a new node with the given data onto the stack. 2. pop(): Remove and return the top node from the stack. If the stack is empty, return None. 3. peek(): Return the data of the top node without removing it. If the stack is empty, return None 4. empty(): Return True if the stack is empty, otherwise return False	Week 4

6.	Write a program to create a class/structure Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Then, create a class/structure Queue to represent the queue itself. Implement the following operations: 1. enqueue(data): Add a new node with the given data to the end of the queue. 2. front(): Return the data of the front node without removing it. If the queue is empty, return None. 3. is_empty(): Return True if the queue is empty, otherwise return False	Week 5

7.	Write a program to create a class/structure DoublyNode to represent a node in a doubly linked list. Each node should have three attributes: data, next, and prev. Then, create a class/structure DoublyLinkedList to represent the linked list itself. Implement the following operations: 1. insert_at_beginning(data): Insert a new node with the given data at the beginning of the list. 2. insert_at_end(data): Insert a new node with the given data at the end of the list. 3. traverse_forward (): Traverse the list forward and print the data of each node.	Week 5
8.	Write a program to a class/structure CircularNode to represent a node in a circular linked list. Each node should have two attributes: data and next. Then, create a class/structure CircularLinkedList to represent the linked list itself. Implement the following operations: 1.insert_at_beginning(data): Insert a new node with the given data at the beginning of the list. 2. traverse(): Traverse the list and print the data of each node.	Week 6

PART – B

Sl. No.	Program	Week
1.	Write a program to create a class/structure TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself. Implement the following operations: 1. insert(data): Insert a new node with the given data into the binary tree (assume a simple insertion without balancing). 2. pre_order_traversal(): Perform a pre-order traversal of the tree and print the data of each node.	Week 7

2.	Write a program to create a class TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself. include a method/function find_lca(node1, node2) that takes two node values as input and returns the value of their lowest common ancestor. Assume all values in the tree are unique.	Week 8
3.	Write a program to create a class TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself, to include a method/function find_grandchildren(node) that takes a node value as input and returns a list of values of all the grandchildren of the given node.	Week 9
4.	Create a class/structure Graph to represent a graph using an adjacency list. Implement the following operations: 1. add_edge(v, w): Add an edge between vertices v and w.	Week 10
5.	Create a class/structure Graph to represent a graph using an adjacency list, to include a method/function dfs(start_vertex) that performs a Depth-First Search starting from start_vertex and prints the order of traversal.	Week 11
6.	Create a class/structure Graph to represent an undirected graph using an adjacency list. Include a method/function detect_cycle() that detects if there is a cycle in the graph. The method should return True if a cycle is detected, otherwise False.	Week 12
7.	A social network is represented as an unweighted graph where users are vertices and friendships are edges. Implement a class/structure SocialNetwork that supports the following operations: 1. add_friendship(user1, user2): Add a friendship (edge) between user1 and user2. 2. degrees_of_separation(user1, user2): Find the shortest path (in terms of the number of edges) between user1 and user2 using BFS and return the number of edges in the path.	Week 13

Lab Write-up and Execution rubrics (Max: 6 marks)

Sl. No	Criteria	Measuring Methods	Excellent	Good	Poor	Marks	Remarks
1	**Understanding of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Test Rubrics (Max: 20 marks)					
1	**Writing** **(10 Marks)**	Observations	The code works perfectly, meets all the problem requirements, passes all test cases, and correctly utilizes appropriate data structure and python knowledge. **(10 M)**	The code works partially, meets some of the problem requirements, passes some test cases, and shows some utilization of data structure and python knowledge. **(< 10 M and >= 1 M)**	The code does not meet the problem requirements, fails most or all test cases, and incorrectly utilizes or does not utilize any data structure and python knowledge. **(< 1 M and >= 0 M)**
2	**Viva** **(5 Marks)**	Viva voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(5 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**
3	**Execution and Format of Results** **(15 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. Appropriate validation with at least 2 cases are handled, and execution results are shown in the proper format. **(2 M)**	Student demonstrates the execution of the program without error correction and appropriate data structure with only a minimum 1 test cases handled. Execution results in without format. **(1 M)**	Execution Results without comments or output. **(0 M)**

INDEX

PART – A

Sl. no.	Program Name.	Date	Record Marks (max 6)	Viva Voice (max 4)	Total Marks	Sign
1.						
2.						

3.						
4.						
5.						
6.						
7						
8						
	Total (80)					
	Total (20)					

PART – B

Sl. no.	**Program Name**	**Date**	**Record Marks (max 6)**	**Viva Voice (max 4)**	**Total Marks**	**Sign**
1.						
2.						
3.						
4.						
5.						
6.						
7						
	Total (70)					

	Total (15)					

Part-A **Record and Viva**	**Max – 10**	
Part-B **Record and Viva**	**Max - 10**	
Test **(Part A + Part B)**	**Max - 30**	
TOTAL	**Max - 50**	
	Signature of the faculty	

PART – A

PROGRAM - 1

Write a program to find the largest and smallest element in an array.

```
def find_largest_and_smallest(arr):
 if not arr:
 return None, None # Return None if the array is empty
 largest = arr[0]
 smallest = arr[0]

 for num in arr:
 if num > largest:
 largest = num
 if num < smallest:
 smallest = num

 return largest, smallest

# Example usage:
array = [3, 5, 1, 8, -3, 7, 2]
largest, smallest = find_largest_and_smallest(array)
print("Largest element:", largest)
print("Smallest element:", smallest)
```

Input:

```
 [3, 5, 1, 8, -3, 7, 2]
```

Output:

```
Largest element : 8
 Smallest element : -3
```

Activity 1:
Write a program to find the second largest and second smallest element in an array.

Stick Data sheets of Program-1 here :

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl.	**Criteria**	**Measuring**	**Excellent**	**Good**	**Poor**	**Marks**	**Remarks**

No		Methods					
1	**Understandi ng of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation n with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases are covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: _______

PROGRAM - 2

Write a program to find the factorial of a number using recursion

```
def factorial(n):
 if n == 0 or n == 1:
 return 1
 else:
 return n * factorial(n - 1)
```

# Example usage:

```
number = 5
result = factorial(number)
print(f"Factorial of {number} is {result}")
```

Input: 5

Output: Factorial of 5 is 120

Write a program to find the nth fibonacci number using recursion.

```
def fibonacci(n):
 if n == 0:
 return 0
 elif n == 1:
 return 1
 else:
 return fibonacci(n - 1) + fibonacci(n - 2)
```

# Example usage:

```
number = 6

result = fibonacci(number)

print(f"The {number}th Fibonacci number is {result}")
```

Input: 6

Output: The 6th Fibonacci number is 8

Activity 2:

2.a Write a program to find the sum of the digits of a number using recursion.

2.b Write a program to calculate the power of a number using recursion.

Stick Data sheets of Program-2 here:

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl. No	**Criteria**	**Measuring Methods**	**Excellent**	**Good**	**Poor**	**Marks**	**Remarks**
1	**Understandi ng of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: ______ **PROGRAM -3**

Write a program to implement the Sudoku game structure and implement searching operations along rows, columns and grids.

```
class Sudoku:
 def __init__(self, board):
 self.board = board

 def is_in_row(self, row, num):
 return num in self.board[row]

 def is_in_col(self, col, num):
 return any(self.board[row][col] == num for row in range(9))

 def is_in_grid(self, row, col, num):
 start_row, start_col = 3 * (row // 3), 3 * (col // 3)
 return any(self.board[r][c] == num for r in range(start_row, start_row + 3) for c in range(start_col,
start_col + 3))

 def print_board(self):
 for row in self.board:
 print(" ".join(str(num) if num != 0 else "." for num in row))

# Example usage:
board = [
 [5, 3, 0, 0, 7, 0, 0, 0, 0],
 [6, 0, 0, 1, 9, 5, 0, 0, 0],
 [0, 9, 8, 0, 0, 0, 0, 6, 0],
 [8, 0, 0, 0, 6, 0, 0, 0, 3],
 [4, 0, 0, 8, 0, 3, 0, 0, 1],
 [7, 0, 0, 0, 2, 0, 0, 0, 6],
 [0, 6, 0, 0, 0, 0, 2, 8, 0],
 [0, 0, 0, 4, 1, 9, 0, 0, 5],
 [0, 0, 0, 0, 8, 0, 0, 7, 9]
]
```

```
sudoku = Sudoku(board)
sudoku.print_board()

# Check for number in row, column, and grid
row, col, num = 0, 2, 3
print(f"\nIs {num} in row {row}? {'Yes' if sudoku.is_in_row(row, num) else 'No'}") print(f"Is {num} in
column {col}? {'Yes' if sudoku.is_in_col(col, num) else 'No'}") print(f"Is {num} in grid containing cell
({row}, {col})? {'Yes' if sudoku.is_in_grid(row, col, num) else  'No'}")
```

Input : 0,2,3

Output: No

Stick Data sheets of Program-3 here:

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl. No	**Criteria**	**Measuring Methods**	**Excellent**	**Good**	**Poor**	**Marks**	**Remarks**
1	**Understa ndi ng of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documenta tion (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptu al Understa nding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in correspondi ng lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: ____ PROGRAM – 4

Write a program to create a class/structure Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Implement the following operations:

1. insert_at_beginning(data): Insert a new node with the given data at the beginning of the list.
2. insert_at_end(data): Insert a new node with the given data at the end of the list.
3. delete_node(data): Delete the first node in the list that contains the given data.
4. traverse(): Traverse the list and print the data of each node

Solution:

```
class Node:
 def __init__(self, data):
 self.data = data
 self.next = None

class LinkedList:
 def __init__(self):
 self.head = None

 def insert_at_beginning(self, data):
 new_node = Node(data)
 new_node.next = self.head
 self.head = new_node

 def insert_at_end(self, data):
 new_node = Node(data)
 if not self.head:
 self.head = new_node
 return
 last = self.head
 while last.next:
 last = last.next
 last.next = new_node

 def delete_node(self, data):
 current = self.head
 if current and current.data == data:
 self.head = current.next
 current = None
 return

 prev = None
 while current and current.data != data:
 prev = current
 current = current.next

 if current is None:
 return

 prev.next = current.next
 current = None

 def traverse(self):
 current = self.head
 while current:
 print(current.data, end=" -> " if current.next else "")
current = current.next
 print()

# Example usage:
linked_list = LinkedList()

# Insert elements at the beginning
linked_list.insert_at_beginning(10)
linked_list.insert_at_beginning(20)
print("Linked list after inserting 20, 10 at the beginning:")
linked_list.traverse()

# Insert elements at the end
linked_list.insert_at_end(30)
linked_list.insert_at_end(40)
print("\nLinked list after inserting 30, 40 at the end:")
```

```
linked_list.traverse()

# Delete a node
linked_list.delete_node(20)
print("\nLinked list after deleting node with data 20:")
linked_list.traverse()

# Traverse the list
print("\nTraversing the linked list:")
linked_list.traverse()
```

1.

Input:

1. Insert 10 at the beginning.
2. Insert 20 at the beginning.
3. Insert 30 at the beginning.

Output:

- Linked list should be: 30 -> 20 -> 10

2.

Input:

1. Insert 5 at the end.
2. Insert 10 at the end.
3. Insert 15 at the end.
4. Insert 20 at the end.

Output:

- Linked list should be: 5 -> 10 -> 15 -> 20

3.

Input:

1. Insert 10 at the end.
2. Insert 20 at the end.
3. Insert 30 at the end.
4. Insert 40 at the end.
5. Delete the node with data 20.

Output:

- Linked list should be: 10 -> 30 -> 40

Activity 4:

Write a program to create a class/structure Node to represent a node in a singly linked list. Implement the following operation:

1. append_node(data): appends node at the end
2. search_node(data): search for a node with a particular value
3. display_list(): prints the list

Stick Data sheets of Program-4 here:

OUTPUT:

<table>
<tr><th colspan="8">Lab Write-up and Execution rubrics (Max: 6 marks)</th></tr>
<tr><th>Sl. No</th><th>Criteria</th><th>Measuring Methods</th><th>Excellent</th><th>Good</th><th>Poor</th><th>Marks</th><th>Remarks</th></tr>
<tr><td>1</td><td><b>Understanding of problem statements (2 Marks)</b></td><td>Observations</td><td>Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. <b>(2 M)</b></td><td>Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. <b>(&lt;2 M and &gt;=1 M)</b></td><td>Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><td>2</td><td><b>Execution (2 Marks)</b></td><td>Observations</td><td>Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. <b>(2 M)</b></td><td>Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. <b>(1 M)</b></td><td>Student has not executed the program or the code fails to demonstrate correct use of data structure. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><td>3</td><td><b>Results and Documentation (2 Marks)</b></td><td>Observations</td><td>Documentation with appropriate comments and output is covered in manual. <b>(2 M)</b></td><td>Documentation with only few comments and only few output cases is covered in manual. <b>(1 M)</b></td><td>Documentation with no comments and no output cases is covered in manual. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><th colspan="8">Viva Voce Rubrics (Max: 4 marks)</th></tr>
<tr><td>1</td><td><b>Conceptual Understanding (4 Marks)</b></td><td>Viva Voce</td><td>Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. <b>(4 M)</b></td><td>Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding <b>(3 M)</b></td><td>Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. <b>(0 M)</b></td><td></td><td></td></tr>
</table>

Lab In-charge Signature: Total Marks: ______ **PROGRAM - 5**

Write a program to create a class/structure Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Then, create a class/structure Stack to represent the stack itself. Implement the following operations:

1. push(data): Push a new node with the given data onto the stack.

2. pop(): Remove and return the top node from the stack. If the stack is empty, return None. 3.

peek(): Return the data of the top node without removing it. If the stack is empty, return None. 4.

is_empty(): Return True if the stack is empty, otherwise return False

```
class Node:

 def __init__(self, data):
 self.data = data
 self.next = None


class Stack:

 def __init__(self):
 self.top = None

 def push(self, data):
 new_node = Node(data)
 new_node.next = self.top
 self.top = new_node

 def pop(self):
 if self.is_empty():
 return None
 popped_node = self.top
 self.top = self.top.next
 return popped_node.data
```

```
def peek(self):
if self.is_empty():
return None
return self.top.data


def is_empty(self):
return self.top is None

def __str__(self):
elements = []
current = self.top
while current:
elements.append(str(current.data))
current = current.next
return " -> ".join(elements) if elements else "Stack is empty"

# Example usage:

stack = Stack()


# Push elements onto the stack

stack.push(10)

stack.push(20)

stack.push(30)

print("Stack after pushing 10, 20, 30:")

print(stack)


# Peek at the top element

print("\nTop element (peek):", stack.peek())
```

```
# Pop elements from the stack

print("\nPopped element:", stack.pop())

print("Stack after popping an element:")

print(stack)


# Check if the stack is empty

print("\nIs the stack empty?", stack.is_empty())


# Pop remaining elements

print("\nPopped element:", stack.pop())

print("Popped element:", stack.pop())

print("Stack after popping all elements:")

print(stack)


# Check if the stack is empty again

print("\nIs the stack empty?", stack.is_empty())


# Try to pop from an empty stack

print("\nAttempt to pop from empty stack:", stack.pop())
```

Input:

1. Push 5 onto the stack.
2. Push 10 onto the stack.
3. Push 15 onto the stack.
4. Push 20 onto the stack.

Output:

- Stack should be: 20 -> 15 -> 10 -> 5 (top to bottom)

Input:

1. Push elements `10`, `20`, and `30` onto the stack.
2. Pop the top element from the stack and verify the returned value.

Output:

- Stack should be: 20 -> 10 -> (top to bottom)

ACTIVITY 5:

Write a program to create a class Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Then, create a class Stack to represent the stack itself. Implement the following operations:

1. push(data): Add a new node with the given data to the top of the stack.
2. peek(): Return the data of the top node without removing it. If the stack is empty, return None.
3. is_empty(): Return True if the stack is empty, otherwise return False.
4. pop(): Remove and return the top node from the stack. If the stack is empty, return None.

Stick Data sheets of Program-5 here:

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl. No	**Criteria**	**Measuring Methods**	**Excellent**	**Good**	**Poor**	**Marks**	**Remarks**
1	**Understanding of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: _______ PROGRAM-6

Write a program to create a class/structure Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Then, create a class/structure Queue to represent the queue itself. Implement the following operations:

1. enqueue(data): Add a new node with the given data to the end of the queue.

2. front(): Return the data of the front node without removing it. If the queue is empty, return None. 3.

is_empty(): Return True if the queue is empty, otherwise return False

```
class Node:

    def __init__(self, data):
    self.data = data
    self.next = None
class Queue:

    def __init__(self):
    self.front_node = None
    self.rear_node = None


    def enqueue(self, data):
    new_node = Node(data)
    if self.is_empty():
    self.front_node = self.rear_node = new_node
    else:
    self.rear_node.next = new_node
    self.rear_node = new_node


    def front(self):
    if self.is_empty():
    return None
    return self.front_node.data
```

```
     def is_empty(self):
     return self.front_node is None
    # Example usage:

    q = Queue()

    # Enqueue elements

    q.enqueue(10)
    q.enqueue(20)
    q.enqueue(30)

    # Front element

    print("Front element:", q.front())

# Check if queue is empty

    print("Is the queue empty?", q.is_empty())

    # Check if queue is empty again

    print("Is the queue empty?", q.is_empty())
```

1. Input:

Enqueue elements `10`, `20`, and `30` to the queue.

Output:

```
Queue is : [10]

[10, 20]

[10, 20, 30]
```

Activity 6:

Write a program to create a class Node to represent a node in a singly linked list. Each node should have two attributes: data and next. Then, create a class Queue to represent the queue itself. Implement the following operations:

1. enqueue(data): Add a new node with the given data to the end of the queue. 2. Front(): Return the data of the front node without removing it. If the queue is empty, return None. 3. is_empty(): Return True if the queue is empty, otherwise return False. 4. dequeue(): Remove and return the front node from the queue. If the queue is empty, return None.

Additionally, implement a function serve_customers() to simulate a customer service queue:

- Enqueue 5 customers with IDs: 101, 102, 103, 104, and 105.
- Serve customers by dequeuing one at a time and printing the ID of the customer being served. - After serving all customers, check if the queue is empty and display an appropriate message.

Stick Data sheets of Activity-6 here:

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl. No	Criteria	Measuring Methods	Excellent	Good	Poor	Marks	Remarks
1	**Understanding of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: _______

PROGRAM-7

Write a program to create a class/structure DoublyNode to represent a node in a doubly linked list. Each node should have three attributes: data, next, and prev. Then, create a class/structure DoublyLinkedList to represent the linked list itself. Implement the following operations:

1. insert_at_beginning(data): Insert a new node with the given data at the beginning of the list.

2. insert_at_end(data): Insert a new node with the given data at the end of the list. 3.

traverse_forward(): Traverse the list forward and print the data of each node.

```
class DoublyNode:

 def __init__(self, data):
 self.data = data
 self.next = None
 self.prev = None

class DoublyLinkedList:

 def __init__(self):
 self.head = None
 self.tail = None


 def insert_at_beginning(self, data):
 new_node = DoublyNode(data)
 if self.head is None:
 self.head = self.tail = new_node
 else:
 new_node.next = self.head
 self.head.prev = new_node
 self.head = new_node


 def insert_at_end(self, data):
 new_node = DoublyNode(data)
 if self.tail is None:
 self.head = self.tail = new_node
 else:
 new_node.prev = self.tail
 self.tail.next = new_node
 self.tail = new_node


 def traverse_forward(self):
 current = self.head
 while current:
 print(current.data, end=" -> " if current.next else "")
current = current.next
 print()
```

# Example usage:

```
dll = DoublyLinkedList()
```

# Insert elements at the beginning

```
dll.insert_at_beginning(10)

dll.insert_at_beginning(20)

print("Doubly linked list after inserting 20, 10 at the

beginning:") dll.traverse_forward()
```

# Insert elements at the end

```
dll.insert_at_end(30)

dll.insert_at_end(40)

print("\nDoubly linked list after inserting 30, 40 at the

end:") dll.traverse_forward()
```

# Traverse the list forward

```
print("\nTraversing the doubly linked list forward:")
```

```
dll.traverse_forward()
```

1. **Input:**

    1. Insert 10 at the beginning.
    2. Insert 20 at the beginning.
    3. Insert 30 at the beginning.

Output:

- Doubly linked list should be: 30 <-> 20 <-> 10

2. Input:

1. Insert 10 at the end.
2. Insert 20 at the end.
3. Insert 30 at the end.

Output:

- Doubly linked list should be: 10 <-> 20 <-> 30

3. Input:

1. Insert 10 at the end.
2. Insert 20 at the end.
3. Insert 30 at the end.
4. Traverse the list forward.

Output:

- Traversal output: 10 20 30

Activity 7:

Write a program to create a class/structure DoublyNode to represent a node in a doubly linked list. Each node should have three attributes: data, next, and prev. Then, create a class/structure DoublyLinkedList to represent the linked list itself. Implement the following operations:

1. delete_node(data): Delete the first node in the list that contains the given data. 2.

traverse_backward(): Traverse the list backward and print the data of each node.

Stick Data sheets of Activity-7 here:

OUTPUT:

<table>
<tr><th colspan="8">Lab Write-up and Execution rubrics (Max: 6 marks)</th></tr>
<tr><th>Sl. No</th><th>Criteria</th><th>Measuring Methods</th><th>Excellent</th><th>Good</th><th>Poor</th><th>Marks</th><th>Remarks</th></tr>
<tr><td>1</td><td><b>Understandi ng of problem statements (2 Marks)</b></td><td>Observations</td><td>Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. <b>(2 M)</b></td><td>Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. <b>(&lt;2 M and &gt;=1 M)</b></td><td>Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><td>2</td><td><b>Execution (2 Marks)</b></td><td>Observations</td><td>Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. <b>(2 M)</b></td><td>Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. <b>(1 M)</b></td><td>Student has not executed the program or the code fails to demonstrate correct use of data structure. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><td>3</td><td><b>Results and Documentation (2 Marks)</b></td><td>Observations</td><td>Documentation with appropriate comments and output is covered in manual. <b>(2 M)</b></td><td>Documentation with only few comments and only few output cases is covered in manual. <b>(1 M)</b></td><td>Documentation with no comments and no output cases is covered in manual. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><th colspan="8">Viva Voce Rubrics (Max: 4 marks)</th></tr>
<tr><td>1</td><td><b>Conceptual Understanding (4 Marks)</b></td><td>Viva Voce</td><td>Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. <b>(4 M)</b></td><td>Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding <b>(3 M)</b></td><td>Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. <b>(0 M)</b></td><td></td><td></td></tr>
</table>

Lab In-charge Signature: Total Marks: ______

PROGRAM-8

Write a program to a class/structure CircularNode to represent a node in a circular linked list. Each node should have two attributes: data and next. Then, create a class/structure CircularLinkedList to represent the linked list itself. Implement the following operations:

1. insert_at_beginning(data): Insert a new node with the given data at the beginning of the list.
2. traverse(): Traverse the list and print the data of each node.

```
class CircularNode:

 def __init__(self, data):
 self.data = data
 self.next = None

class CircularLinkedList:

 def __init__(self):
 self.head = None

 def insert_at_beginning(self, data):
 new_node = CircularNode(data)
 if self.head is None:
 self.head = new_node
 new_node.next = self.head
 else:
 current = self.head
 while current.next != self.head:
 current = current.next
 new_node.next = self.head
 current.next = new_node
 self.head = new_node

 def traverse(self):
 if self.head is None:
 print("Circular linked list is empty.")
 return
 current = self.head
 while True:
 print(current.data, end=" -> " if current.next != self.head else "")
current = current.next
 if current == self.head:
 break
 print()
```

# Example usage:

```
cll = CircularLinkedList()
```

# Insert elements at the beginning

```
cll.insert_at_beginning(10)
cll.insert_at_beginning(20)
print("Circular linked list after inserting 20, 10 at the beginning:")
cll.traverse()
```

# Insert elements at the end

```
cll.insert_at_end(30)
cll.insert_at_end(40)
print("\nCircular linked list after inserting 30, 40 at the
end:") cll.traverse()
```

Input:

1. Insert 10 at the beginning.
2. Insert 20 at the beginning.
3. Insert 30 at the beginning.

Output:

- Circular linked list should be: 30 -> 20 -> 10 -> (back to 30)

Input:

1. Create nodes with the following data: 10, 20, 30, 40.

Output: Traversal output: 10 20 30 40

Activity 8:

Write a program to a class/structure CircularNode to represent a node in a circular linked list. Each node should have two attributes: data and next. Then, create a class/structure CircularLinkedList to represent the linked list itself. Implement the following operations:

1. insert_at_end(data): Insert a new node with the given data at the end of the list.
2. delete_node(data): Delete the first node in the list that contains the given data.

Stick Data sheets of Activity-8 here:

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl.	**Criteria**	**Measuring**	**Excellent**	**Good**	**Poor**	**Marks**	**Remarks**

No		Methods					
1	**Understandi ng of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: _______

PART – B

PRACTICE PROGRAM- 1

Write a program to create a class/structure TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself. Implement the following operations:

1. .insert(data): Insert a new node with the given data into the binary tree (assume a simple insertion without balancing).

2. .pre_order_traversal(): Perform a pre-order traversal of the tree and print the data of each node.

```
class TreeNode:

 def __init__(self, data):
 self.data = data
 self.left = None
 self.right = None

class BinaryTree:

 def __init__(self):
 self.root = None

 def insert(self, data):
 new_node = TreeNode(data)
 if self.root is None:
 self.root = new_node
 else:
 self._insert_recursive(self.root, new_node)

 def _insert_recursive(self, current, new_node):

 if new_node.data < current.data:
 if current.left is None:
 current.left = new_node
 else:
 self._insert_recursive(current.left, new_node)
 else:
 if current.right is None:
```

```
 current.right = new_node
 else:
 self._insert_recursive(current.right, new_node)


 def pre_order_traversal(self):
 self._pre_order_recursive(self.root)
 print()

 def _pre_order_recursive(self, node):
 if node:
 print(node.data, end=" ")
 self._pre_order_recursive(node.left)
 self._pre_order_recursive(node.right)

# Example usage:

bt = BinaryTree()



# Insert elements into the binary tree

bt.insert(50)
bt.insert(30)
bt.insert(70)
bt.insert(20)
bt.insert(40)
bt.insert(60)
bt.insert(80)


# Pre-order traversal

print("Pre-order traversal:")
bt.pre_order_traversal()
```

Input:

1. Insert nodes with the following data: 10, 5, 15, 3, 7, 12, 20.

Output:

- Expected binary tree structure (in-order traversal): 3, 5, 7, 10, 12, 15, 20

Input:

1. Insert nodes with the following data: 50, 30, 20, 40, 70, 60, 80.

Output:

Pre-order traversal:

50 30 20 40 70 60 80

Activity 1:

Write a program to create a class/structure TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself. Implement the following operations:

1. in_order_traversal(): Perform an in-order traversal of the tree and print the data of each node. 2.

post_order_traversal(): Perform a post-order traversal of the tree and print the data of each node.

OUTPUT:

<table>
<tr><th colspan="8">Lab Write-up and Execution rubrics (Max: 6 marks)</th></tr>
<tr><th>Sl. No</th><th>Criteria</th><th>Measuring Methods</th><th>Excellent</th><th>Good</th><th>Poor</th><th>Marks</th><th>Remarks</th></tr>
<tr><td>1</td><td><b>Understanding of problem statements (2 Marks)</b></td><td>Observations</td><td>Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. <b>(2 M)</b></td><td>Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. <b>(<2 M and >=1 M)</b></td><td>Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><td>2</td><td><b>Execution (2 Marks)</b></td><td>Observations</td><td>Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. <b>(2 M)</b></td><td>Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. <b>(1 M)</b></td><td>Student has not executed the program or the code fails to demonstrate correct use of data structure. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><td>3</td><td><b>Results and Documentation (2 Marks)</b></td><td>Observations</td><td>Documentation with appropriate comments and output is covered in manual. <b>(2 M)</b></td><td>Documentation with only few comments and only few output cases is covered in manual. <b>(1 M)</b></td><td>Documentation with no comments and no output cases is covered in manual. <b>(0 M)</b></td><td></td><td></td></tr>
<tr><th colspan="8">Viva Voce Rubrics (Max: 4 marks)</th></tr>
<tr><td>1</td><td><b>Conceptual Understanding (4 Marks)</b></td><td>Viva Voce</td><td>Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. <b>(4 M)</b></td><td>Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding <b>(3 M)</b></td><td>Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. <b>(0 M)</b></td><td></td><td></td></tr>
</table>

Lab In-charge Signature: Total Marks: _______

PRACTICE PROGRAM - 2

Write a program to create a class TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself. include a method/function find_lca(node1, node2) that takes two node values as input and returns the value of their lowest common ancestor. Assume all values in the tree are unique.

```
class TreeNode:

 def __init__(self, data):
 self.data = data
 self.left = None
 self.right = None

class BinaryTree:

 def __init__(self):
 self.root = None


 def find_lca(self, node1, node2):
 # Check if both nodes are in the tree
 if not self._find_node(self.root, node1) or not self._find_node(self.root, node2):
return None
 return self._find_lca_recursive(self.root, node1, node2)


 def _find_lca_recursive(self, current, node1, node2):
 if current is None:
 return None

 # If current node is one of the nodes we are looking for, return it

 if current.data == node1 or current.data == node2:
 return current.data

 # Recursively search in the left and right subtrees

 left_lca = self._find_lca_recursive(current.left, node1, node2)
 right_lca = self._find_lca_recursive(current.right, node1, node2)
 # If both nodes are found in left and right subtrees, then current node is LCA
```

```
 if left_lca and right_lca:
 return current.data

 # Otherwise, return the non-None value (either left_lca or right_lca)

return left_lca if left_lca is not None else right_lca

 def _find_node(self, current, target):
 if current is None:
 return False
 if current.data == target:
 return True
 return self._find_node(current.left, target) or self._find_node(current.right, target)
```

# Example usage:

```
bt = BinaryTree()
```

# Constructing the binary tree

```
bt.root = TreeNode(1)
bt.root.left = TreeNode(2)
bt.root.right = TreeNode(3)
bt.root.left.left = TreeNode(4)
bt.root.left.right = TreeNode(5)
bt.root.right.left = TreeNode(6)
bt.root.right.right = TreeNode(7)


# Finding the Lowest Common Ancestor (LCA) of nodes 4 and 5

lca_value = bt.find_lca(4, 5)
print(f"LCA of nodes 4 and 5: {lca_value}")
```

```
# Finding the Lowest Common Ancestor (LCA) of nodes 4 and

6 lca_value = bt.find_lca(4, 6)
```

```
print(f"LCA of nodes 4 and 6: {lca_value}")

# Finding the Lowest Common Ancestor (LCA) of nodes 3 and

7 lca_value = bt.find_lca(3, 7)

print(f"LCA of nodes 3 and 7: {lca_value}")
```

Input:

1. Insert nodes with the following data: 50, 30, 20, 40, 70, 60, 80. 2. Find the lowest common ancestor (LCA) of nodes with values 20 and 40.

Output:

Lowest Common Ancestor of 20 and 40: 30

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)

Sl. No	Criteria	Measuring Methods	Excellent	Good	Poor	Marks	Remarks
1	**Understandi ng of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: _______

PRACTICE PROGRAM - 3

Write a program to create a class TreeNode to represent a node in a binary tree. Each node should have three attributes: data, left, and right. Then, create a class/structure BinaryTree to represent the binary tree itself, to include a method/function find_grandchildren(node) that takes a node value as input and returns a list of values of all the grandchildren of the given node.

```
class TreeNode:

 def __init__(self, data):
 self.data = data
 self.left = None
 self.right = None

class BinaryTree:

 def __init__(self):
 self.root = None



 def find_grandchildren(self, node):
 if not node:
 return []

 grandchildren = []
 if node.left:
 grandchildren.extend(self._get_children_values(node.left))
 if node.right:
 grandchildren.extend(self._get_children_values(node.right))
 return grandchildren



 def _get_children_values(self, parent):
 children_values = []
 if parent.left:
 children_values.append(parent.left.data)
 if parent.right:
 children_values.append(parent.right.data)
 return children_values
# Example usage:

bt = BinaryTree()

# Constructing the binary tree

bt.root = TreeNode(1)
bt.root.left = TreeNode(2)
bt.root.right = TreeNode(3)
bt.root.left.left = TreeNode(4)
bt.root.left.right = TreeNode(5)
bt.root.right.left = TreeNode(6)
bt.root.right.right = TreeNode(7)

# Function to find grandchildren of a given node

def find_grandchildren_values(bt, node_value):
 node = find_node(bt.root, node_value)
 if node:
 return bt.find_grandchildren(node)
 else:
 return []

def find_node(current, node_value):
 if current is None:
 return None
 if current.data == node_value:
 return current
 left_node = find_node(current.left, node_value)
 if left_node:
 return left_node
 return find_node(current.right, node_value)



# Finding grandchildren of node with value 1

print("Grandchildren of node with value 1:", find_grandchildren_values(bt, 1))
```

```
# Finding grandchildren of node with value 2

print("Grandchildren of node with value 2:", find_grandchildren_values(bt,

2)) # Finding grandchildren of node with value 3

print("Grandchildren of node with value 3:", find_grandchildren_values(bt,

3)) # Finding grandchildren of node with value 4

print("Grandchildren of node with value 4:", find_grandchildren_values(bt,

4)) # Finding grandchildren of node with value 5

print("Grandchildren of node with value 5:", find_grandchildren_values(bt,

5)) # Finding grandchildren of node with value 6

print("Grandchildren of node with value 6:", find_grandchildren_values(bt,

6)) # Finding grandchildren of node with value 7

print("Grandchildren of node with value 7:", find_grandchildren_values(bt, 7))
```

Input:

1. Insert nodes with the following data: 50, 30, 70, 20, 40, 60, 80, 15, 25, 35, 45, 65, 75, 85.
2. Find the grandchildren of the node with data 30.

Output:

```
Grandchildren of node with data 30: [15, 25, 35, 45]
```

OUTPUT:

Lab Write-up and Execution rubrics (Max: 6 marks)							
Sl.	Criteria	Measuring	Excellent	Good	Poor	Marks	Remarks

No		Methods					
1	**Understandi ng of problem statements (2 Marks)**	Observations	Student exhibits a thorough understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(2 M)**	Student has sufficient understanding of the problem statements and applies appropriate data structure and python knowledge to devise a solution. **(<2 M and >=1 M)**	Student does not have a clear understanding of the problem statements and is unable to apply appropriate data structure and python knowledge to devise a solution. **(0 M)**		
2	**Execution (2 Marks)**	Observations	Student demonstrates the execution of the program with appropriate data structure. All test cases are handled with appropriate validations. **(2 M)**	Student demonstrates the execution of the program with appropriate data structure with only a few test cases handled. **(1 M)**	Student has not executed the program or the code fails to demonstrate correct use of data structure. **(0 M)**		
3	**Results and Documentation (2 Marks)**	Observations	Documentation with appropriate comments and output is covered in manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in manual. **(1 M)**	Documentation with no comments and no output cases is covered in manual. **(0 M)**		
Viva Voce Rubrics (Max: 4 marks)							
1	**Conceptual Understanding (4 Marks)**	Viva Voce	Explains thoroughly the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(4 M)**	Adequately explains the core concepts and principles of data structure used in corresponding lab as well as python constructs with some understanding **(3 M)**	Unable to explain the core concepts and principles of data structure used in corresponding lab as well as python constructs. **(0 M)**		

Lab In-charge Signature: Total Marks: _______

PRACTICE PROGRAM - 4

Create a class/structure Graph to represent a graph using an adjacency list. Implement the following operations:

1. add_edge(v, w): Add an edge between vertices v and w.

Solution:

```
class Graph:
 def __init__(self):
 self.adj_list = {}


 def add_edge(self, u, v):
 if u not in self.adj_list:
 self.adj_list[u] = []
 if v not in self.adj_list:
 self.adj_list[v] = []
 self.adj_list[u].append(v)
 self.adj_list[v].append(u)


 # Example usage:
graph = Graph()
graph.add_edge(1, 2)
graph.add_edge(1, 3)
graph.add_edge(2, 4)
graph.add_edge(3, 4)
graph.add_edge(4, 5)
```

Input:

1. Add edges between vertices: (0, 1), (0, 2), (1, 2), (2, 3), (3, 4), (4, 0).

Expected Output:

After adding edges, the adjacency list representation of the graph should be:

```
0: [1, 2]
 1: [0, 2]
2: [0, 1, 3]
3: [2, 4]
4: [3, 0]
```