SAVITRIBAI PHULE PUNE UNIVERSITY
MASTER OF COMPUTER APPLICATION
DR.D.Y. PATIL SCHOOL OF MCA
Charoli (bk) pune-412105

STUDENT NAME:____

CLASS :-………… DIVISION :-……….. ROLL NO :-…………… REMARK :-…………….

DATE OF SUBMISSION :-………………………………………………………………………….

TITLE :- 1 Write a program to perform using NumPy to perform Various functions in Array.

Program :

```
import numpy as np

# Creating an array

arr = np.array([10, 20, 30, 40, 50])

# Various operations

print("Original Array:", arr)

print("Sum of array:", np.sum(arr))

print("Mean of array:", np.mean(arr))

print("Max value:", np.max(arr))

print("Min value:", np.min(arr))

print("Sorted Array:", np.sort(arr))

print("Square of each element:", np.square(arr))
```

Output :

```
Original Array: [10 20 30 40 50]
Sum of array: 150
Mean of array: 30.0
Max value: 50
Min value: 10
Sorted Array: [10 20 30 40 50]
Square of each element: [ 100  400  900 1600 2500]
```

SAVITRIBAI PHULE PUNE UNIVERSITY

MASTER OF COMPUTER APPLICATION
DR.D.Y. PATIL SCHOOL OF MCA
Charoli (bk) pune-412105

STUDENT NAME:____

CLASS :-………… DIVISION :-……….. ROLL NO :-…………… REMARK :-…………….

DATE OF SUBMISSION :-…………………………………………………………………….

TITLE :- 2 Write a program to implement a Sparse Matrix.

Program :

```
import numpy as np
def sparse_matrix(matrix):
    sparse = []
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if matrix[i][j] != 0:
                sparse.append((i, j, matrix[i][j]))
    return sparse
# Input Matrix
matrix = [
    [5, 0, 0],
    [0, 8, 0],
    [0, 0, 3]
]
print("Original Matrix:")
for row in matrix:
    print(row)
print("Sparse Matrix Representation:")
```

```
print(sparse_matrix(matrix))
```

Output :

```
Original Matrix:
[5, 0, 0]
[0, 8, 0]
[0, 0, 3]
Sparse Matrix Representation:
[(0, 0, 5), (1, 1, 8), (2, 2, 3)]
```

TITLE :- 3 Write a program to perform String Manipulations using Array.

Program :

```
from array import array

def string_manipulations(s):
    arr = array('u', s)  # Unicode array
    print("Original String Array:", "".join(arr))
    # Reversing the string
    arr.reverse()
    print("Reversed String:", "".join(arr))
    # Adding a character
    arr.append('!')
    print("After Adding a Character:", "".join(arr))
    # Removing a character
    arr.pop()
    print("After Removing Last Character:", "".join(arr))

string_manipulations("hello")
```

Output :

```
Original String Array: hello
Reversed String: olleh
After Adding a Character: olleh!
After Removing Last Character: olleh
```

SAVITRIBAI PHULE PUNE UNIVERSITY
MASTER OF COMPUTER APPLICATION
DR.D.Y. PATIL SCHOOL OF MCA
Charoli (bk) pune-412105

STUDENT NAME:____

CLASS :-………… DIVISION :-……….. ROLL NO :-…………… REMARK :-…………….

DATE OF SUBMISSION :-………………………………………………………………………….

TITLE :- 4 Write a menu driven program to perform thefollowing operations on singly linked list

i)Creation ii) Insertion iii) Deletion iv) Searching v) Display

Program :

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class SinglyLinkedList:
    def __init__(self):
        self.head = None
    def create(self, data):
        self.head = Node(data)
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
```

```
    def delete(self, key):
        current = self.head
        if current and current.data == key:
            self.head = current.next
            return
        while current.next and current.next.data != key:
            current = current.next
        if current.next:
            current.next = current.next.next
    def search(self, key):
        current = self.head
        while current:
            if current.data == key:
                return True
            current = current.next
        return False
    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
# Menu Driven Program
sll = SinglyLinkedList()
while True:
    print("\n1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit")
    choice = int(input("Enter your choice: "))
```

```
if choice == 1:
    data = int(input("Enter data: "))
    sll.create(data)
elif choice == 2:
    data = int(input("Enter data: "))
    sll.insert(data)
elif choice == 3:
    key = int(input("Enter element to delete: "))
    sll.delete(key)
elif choice == 4:
    key = int(input("Enter element to search: "))
    print("Found" if sll.search(key) else "Not Found")
elif choice == 5:
    sll.display()
elif choice == 6:
    break
```

Output :

```
1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit
Enter your choice:  1
Enter data:  1

1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit
Enter your choice:  2
Enter data:  2

1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit
Enter your choice:  5
1 -> 2 -> None

1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit
Enter your choice:  3
Enter element to delete:  1

1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit
Enter your choice:  5
2 -> None

1. Create 2. Insert 3. Delete 4. Search 5. Display 6. Exit
Enter your choice:  6
```

SAVITRIBAI PHULE PUNE UNIVERSITY

MASTER OF COMPUTER APPLICATION
DR.D.Y. PATIL SCHOOL OF MCA
Charoli (bk) pune-412105

STUDENT NAME:____

CLASS :-………… DIVISION :-……….. ROLL NO :-…………… REMARK :-…………….

DATE OF SUBMISSION :-……………………………………………………………………….

TITLE :- 5 Write a menu driven program to perform the following operations on doubly linked list

i)Creation ii) Insertion iii) Deletion iv) Searching v) Display

Program :

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
class DoublyLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
```

```python
            new_node.prev = current


    def delete(self, key):
        current = self.head
        while current:
            if current.data == key:
                if current.prev:
                    current.prev.next = current.next
                if current.next:
                    current.next.prev = current.prev
                if current == self.head:
                    self.head = current.next
                return
            current = current.next
    def display(self):
        current = self.head
        while current:
            print(current.data, end=" <-> ")
            current = current.next
        print("None")
# Menu Driven Program
dll = DoublyLinkedList()
while True:
    print("\n1. Insert 2. Delete 3. Display 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter data: "))
```

```
        dll.insert(data)
    elif choice == 2:
        key = int(input("Enter element to delete: "))
        dll.delete(key)
    elif choice == 3:
        dll.display()
    elif choice == 4:
        break
```

Output :

```
1. Insert 2. Delete 3. Display 4. Exit
Enter your choice:  1
Enter data:  10

1. Insert 2. Delete 3. Display 4. Exit
Enter your choice:  1
Enter data:  20

1. Insert 2. Delete 3. Display 4. Exit
Enter your choice:  3
10 <-> 20 <-> None

1. Insert 2. Delete 3. Display 4. Exit
Enter your choice:  2
Enter element to delete:  10

1. Insert 2. Delete 3. Display 4. Exit
Enter your choice:  3
20 <-> None

1. Insert 2. Delete 3. Display 4. Exit
Enter your choice:  4
```

TITLE :- 6 Write a menu driven program to perform the following operations on circular linked List

i) Creation ii) Insertion iii) Deletion iv) Searching v) Display

Program :

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
```

```
            new_node.next = self.head

    def display(self):
        if not self.head:
            print("List is empty")
            return
        temp = self.head
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("HEAD")
# Menu Driven Program
cll = CircularLinkedList()
while True:
    print("\n1. Insert 2. Display 3. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter data: "))
        cll.insert(data)
    elif choice == 2:
        cll.display()
    elif choice == 3:
        break
```

Output :

```
1. Insert 2. Display 3. Exit
Enter your choice:  1
Enter data:  10

1. Insert 2. Display 3. Exit
Enter your choice:  1
Enter data:  20

1. Insert 2. Display 3. Exit
Enter your choice:  1
Enter data:  30

1. Insert 2. Display 3. Exit
Enter your choice:  2
10 -> 20 -> 30 -> HEAD

1. Insert 2. Display 3. Exit
Enter your choice:  3
```

TITLE :- 7 Write a program that implement stack using

i) Arrays

Program :

```
class StackArray:
    def __init__(self):
        self.stack = []
    def push(self, data):
        self.stack.append(data)
        print(f"{data} pushed into stack")
    def pop(self):
        if not self.stack:
            print("Stack Underflow")
        else:
            print(f"Popped Element: {self.stack.pop()}")
    def display(self):
        if not self.stack:
            print("Stack is empty")
        else:
            print("Stack Elements:", self.stack)
# Menu Driven Program
stack = StackArray()
```

```
while True:
    print("\n1. Push 2. Pop 3. Display 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter data: "))
        stack.push(data)
    elif choice == 2:
        stack.pop()
    elif choice == 3:
        stack.display()
    elif choice == 4:
        break
```

Output :

```
1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  1
Enter data:  10
10 pushed into stack

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  1
Enter data:  20
20 pushed into stack

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  1
Enter data:  30
30 pushed into stack

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  3
Stack Elements: [10, 20, 30]

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  2
Popped Element: 30

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  3
Stack Elements: [10, 20]

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  4
```

ii) Linked list

Program :

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class StackLinkedList:
    def __init__(self):
        self.top = None
    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node
        print(f"{data} pushed into stack")
    def pop(self):
        if not self.top:
            print("Stack Underflow")
        else:
            print(f"Popped Element: {self.top.data}")
            self.top = self.top.next
    def display(self):
        if not self.top:
            print("Stack is empty")
        else:
```

```
        temp = self.top
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
# Menu Driven Program
stack = StackLinkedList()
while True:
    print("\n1. Push 2. Pop 3. Display 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter data: "))
        stack.push(data)
    elif choice == 2:
        stack.pop()
    elif choice == 3:
        stack.display()
    elif choice == 4:
        break
```

Output :

```
1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  1
Enter data:  10
10 pushed into stack

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  1
Enter data:  20
20 pushed into stack

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  1
Enter data:  30
30 pushed into stack

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  3
30 -> 20 -> 10 -> None

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  2
Popped Element: 30

1. Push 2. Pop 3. Display 4. Exit
Enter your choice:  4
```

TITLE :- 8 Write a program that implement Queue using

i) Arrays

Program :

```
class QueueArray:
    def __init__(self):
        self.queue = []
    def enqueue(self, data):
        self.queue.append(data)
        print(f"{data} added to queue")
    def dequeue(self):
        if not self.queue:
            print("Queue Underflow")
        else:
            print(f"Dequeued Element: {self.queue.pop(0)}")
    def display(self):
        if not self.queue:
            print("Queue is empty")
        else:
            print("Queue Elements:", self.queue)
# Menu Driven Program
```

```
queue = QueueArray()

while True:

    print("\n1. Enqueue 2. Dequeue 3. Display 4. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:

        data = int(input("Enter data: "))

        queue.enqueue(data)

    elif choice == 2:

        queue.dequeue()

    elif choice == 3:

        queue.display()

    elif choice == 4:

        break
```

Output :

```
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  10
10 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  20
20 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  30
30 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  3
Queue Elements: [10, 20, 30]

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  2
Dequeued Element: 10

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  3
Queue Elements: [20, 30]

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  4
```

ii) Linked list

Program :

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class QueueLinkedList:
    def __init__(self):
        self.front = self.rear = None
    def enqueue(self, data):
        new_node = Node(data)
        if not self.rear:
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node
        print(f"{data} added to queue")
    def dequeue(self):
        if not self.front:
            print("Queue Underflow")
        else:
            print(f"Dequeued Element: {self.front.data}")
            self.front = self.front.next
            if not self.front:
                self.rear = None
    def display(self):
        if not self.front:
```

```
            print("Queue is empty")
        else:
            temp = self.front
            while temp:
                print(temp.data, end=" -> ")
                temp = temp.next
            print("None")
# Menu Driven Program
queue = QueueLinkedList()
while True:
    print("\n1. Enqueue 2. Dequeue 3. Display 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter data: "))
        queue.enqueue(data)
    elif choice == 2:
        queue.dequeue()
    elif choice == 3:
        queue.display()
    elif choice == 4:
        break
```

Output :

```
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  10
10 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  20
20 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  30
30 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  3
10 -> 20 -> 30 -> None

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  2
Dequeued Element: 10

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  3
20 -> 30 -> None

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  4
```

TITLE :- 9 Write a program that implement Circular Queue using Arrays & Linked List

Program :

```
class CircularQueue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = self.rear = -1
    def enqueue(self, data):
        if (self.rear + 1) % self.size == self.front:
            print("Queue Overflow")
        elif self.front == -1:  # First element
            self.front = self.rear = 0
            self.queue[self.rear] = data
        else:
            self.rear = (self.rear + 1) % self.size
            self.queue[self.rear] = data
        print(f"{data} added to queue")
    def dequeue(self):
        if self.front == -1:
            print("Queue Underflow")
```

```
        elif self.front == self.rear:  # Only one element
            print(f"Dequeued Element: {self.queue[self.front]}")
            self.front = self.rear = -1
        else:
            print(f"Dequeued Element: {self.queue[self.front]}")
            self.front = (self.front + 1) % self.size
    def display(self):
        if self.front == -1:
            print("Queue is empty")
        else:
            print("Queue Elements:", end=" ")
            i = self.front
            while True:
                print(self.queue[i], end=" ")
                if i == self.rear:
                    break
                i = (i + 1) % self.size
            print()
# Menu Driven Program
size = int(input("Enter size of Circular Queue: "))
cq = CircularQueue(size)
while True:
    print("\n1. Enqueue 2. Dequeue 3. Display 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter data: "))
        cq.enqueue(data)
```

```
    elif choice == 2:
        cq.dequeue()
    elif choice == 3:
        cq.display()
    elif choice == 4:
        break
```

Output :

```
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  10
10 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  20
20 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  1
Enter data:  30
30 added to queue

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  3
Queue Elements: 10 20 30

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  2
Dequeued Element: 10

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  3
Queue Elements: 20 30

1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice:  4
```

STUDENT NAME:____

CLASS :-………… DIVISION :-……….. ROLL NO :-…………… REMARK :-…………….

DATE OF SUBMISSION :-……………………………………………………………………….

TITLE :- 10 Write a program to perform the following operations:

a) Insert an element into a binary search tree.

b) Delete an element from a binary search tree.

c) Search for a key element in a binary search tree

Program :

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.key = key
# Insert function
def insert(root, key):
    if root is None:
        return Node(key)
    elif key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root
# Delete function
def delete(root, key):
```

```python
    if root is None:
        return root
    if key < root.key:
        root.left = delete(root.left, key)
    elif key > root.key:
        root.right = delete(root.right, key)
    else:
        # Node with one or no child
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        # Node with two children: get the inorder successor
        temp = find_min(root.right)
        root.key = temp.key
        root.right = delete(root.right, temp.key)
    return root
# Find the minimum value (used in deletion)
def find_min(node):
    current = node
    while current.left is not None:
        current = current.left
    return current
# Search function
def search(root, key):
    if root is None or root.key == key:
        return root
```

```
    if key < root.key:
        return search(root.left, key)
    return search(root.right, key)
# Inorder Traversal (to display BST elements)
def inorder(root):
    if root:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)
# Menu-Driven Program
root = None
while True:
    print("\n1. Insert 2. Delete 3. Search 4. Display (Inorder) 5. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        key = int(input("Enter key to insert: "))
        root = insert(root, key)
    elif choice == 2:
        key = int(input("Enter key to delete: "))
        root = delete(root, key)
    elif choice == 3:
        key = int(input("Enter key to search: "))
        result = search(root, key)
        if result:
            print(f"Key {key} found in the BST")
        else:
            print(f"Key {key} not found in the BST")
```

```
    elif choice == 4:
        print("BST Elements (Inorder Traversal): ", end="")
        inorder(root)
        print()
    elif choice == 5:
        break
```

Output :

```
1. Insert 2. Delete 3. Search 4. Display (Inorder) 5. Exit
Enter your choice:  1
Enter key to insert:  20

1. Insert 2. Delete 3. Search 4. Display (Inorder) 5. Exit
Enter your choice:  1
Enter key to insert:  10

1. Insert 2. Delete 3. Search 4. Display (Inorder) 5. Exit
Enter your choice:  4
BST Elements (Inorder Traversal): 10 20

1. Insert 2. Delete 3. Search 4. Display (Inorder) 5. Exit
Enter your choice:  3
Enter key to search:  20
Key 20 found in the BST

1. Insert 2. Delete 3. Search 4. Display (Inorder) 5. Exit
Enter your choice:  5
```

TITLE :- 11 Write a program to search an element by using Linear search method

Program :

```
def linear_search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i  # Return index if found
    return -1  # Return -1 if not found
# Input and Testing
arr = list(map(int, input("Enter array elements: ").split()))
x = int(input("Enter element to search: "))
result = linear_search(arr, x)
if result != -1:
    print(f"Element found at index {result}")
else:
    print("Element not found")
```

Output :

```
Enter array elements:  2 3 4
Enter element to search:  3
Element found at index 1
```

TITLE :- 12 Write a program to search an element by using Binary search method

Program :

```
def binary_search(arr, x):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
# Input and Testing
arr = sorted(list(map(int, input("Enter sorted array elements: ").split())))
x = int(input("Enter element to search: "))
result = binary_search(arr, x)
if result != -1:
    print(f"Element found at index {result}")
else:
```

```
print("Element not found")
```

Output :

```
Enter sorted array elements:  1 2 4 6 10 23
Enter element to search:  6
Element found at index 3
```

TITLE :- 13 Write a program to implement the tree traversal methods.

Program :

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.key = key
# Traversal Functions
def preorder(root):
    if root:
        print(root.key, end=" ")
        preorder(root.left)
        preorder(root.right)
def inorder(root):
    if root:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)
def postorder(root):
    if root:
        postorder(root.left)
```

```
        postorder(root.right)
        print(root.key, end=" ")
# Insert into Binary Tree
def insert(root, key):
    if root is None:
        return Node(key)
    elif key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root
# Menu-Driven Program
root = None
while True:
    print("\n1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        key = int(input("Enter key to insert: "))
        root = insert(root, key)
    elif choice == 2:
        print("Preorder Traversal: ", end="")
        preorder(root)
        print()
    elif choice == 3:
        print("Inorder Traversal: ", end="")
        inorder(root)
        print()
```

```
    elif choice == 4:
        print("Postorder Traversal: ", end="")
        postorder(root)
        print()
    elif choice == 5:
        break
```

Output :

```
1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit
Enter your choice:  1
Enter key to insert:  30

1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit
Enter your choice:  1
Enter key to insert:  10

1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit
Enter your choice:  1
Enter key to insert:  20

1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit
Enter your choice:  3
Inorder Traversal: 10 20 30

1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit
Enter your choice:  4
Postorder Traversal: 20 10 30

1. Insert 2. Preorder 3. Inorder 4. Postorder 5. Exit
Enter your choice:  5
```

TITLE :- 14 Write a program to perform the following operations:

Insert an element into a AVL tree.

Delete an element from a AVL tree.

Search for a key element in a AVL tree.

Program :

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1
def get_height(node):
    if not node:
        return 0
    return node.height
def get_balance(node):
    if not node:
        return 0
    return get_height(node.left) - get_height(node.right)
def rotate_right(y):
```

```
    x = y.left
    T2 = x.right
    x.right = y
    y.left = T2
    y.height = 1 + max(get_height(y.left), get_height(y.right))
    x.height = 1 + max(get_height(x.left), get_height(x.right))
    return x
def rotate_left(x):
    y = x.right
    T2 = y.left
    y.left = x
    x.right = T2
    x.height = 1 + max(get_height(x.left), get_height(x.right))
    y.height = 1 + max(get_height(y.left), get_height(y.right))
    return y
# Insert operation
def insert(node, key):
    if not node:
        return Node(key)
    if key < node.key:
        node.left = insert(node.left, key)
    else:
        node.right = insert(node.right, key)

    node.height = 1 + max(get_height(node.left), get_height(node.right))
    balance = get_balance(node)
    # Left heavy
```

```
    if balance > 1 and key < node.left.key:
        return rotate_right(node)
    # Right heavy
    if balance < -1 and key > node.right.key:
        return rotate_left(node)
    # Left-Right heavy
    if balance > 1 and key > node.left.key:
        node.left = rotate_left(node.left)
        return rotate_right(node)
    # Right-Left heavy
    if balance < -1 and key < node.right.key:
        node.right = rotate_right(node.right)
        return rotate_left(node)

    return node
# Inorder Traversal
def inorder(node):
    if node:
        inorder(node.left)
        print(node.key, end=" ")
        inorder(node.right)
# Search operation
def search(node, key):
    if not node or node.key == key:
        return node
    if key < node.key:
        return search(node.left, key)
```

```
    return search(node.right, key)
# Menu-driven Program
root = None
while True:
    print("\n1. Insert 2. Search 3. Display Inorder 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        key = int(input("Enter key to insert: "))
        root = insert(root, key)
    elif choice == 2:
        key = int(input("Enter key to search: "))
        result = search(root, key)
        print(f"Key {key} found" if result else f"Key {key} not found")
    elif choice == 3:
        print("Inorder Traversal of AVL Tree: ", end="")
        inorder(root)
        print()
    elif choice == 4:
        break
```

Output :

```
1. Insert 2. Search 3. Display Inorder 4. Exit
Enter your choice:  1
Enter key to insert:  30

1. Insert 2. Search 3. Display Inorder 4. Exit
Enter your choice:  1
Enter key to insert:  10

1. Insert 2. Search 3. Display Inorder 4. Exit
Enter your choice:  1
Enter key to insert:  20

1. Insert 2. Search 3. Display Inorder 4. Exit
Enter your choice:  3
Inorder Traversal of AVL Tree: 10 20 30

1. Insert 2. Search 3. Display Inorder 4. Exit
Enter your choice:  2
Enter key to search:  20
Key 20 found

1. Insert 2. Search 3. Display Inorder 4. Exit
Enter your choice:  4
```

TITLE :- 15 Write a program to implement the Graph methods.

- Adjacency Matrix
- Adjacency List

Program :

```
# Graph using Adjacency Matrix
def adjacency_matrix(vertices, edges):
    matrix = [[0] * vertices for _ in range(vertices)]
    for edge in edges:
        u, v = edge
        matrix[u][v] = 1
        matrix[v][u] = 1  # For undirected graph
    return matrix
# Graph using Adjacency List
def adjacency_list(vertices, edges):
    adj_list = {i: [] for i in range(vertices)}
    for edge in edges:
        u, v = edge
        adj_list[u].append(v)
        adj_list[v].append(u)  # For undirected graph
    return adj_list
```

```python
# Input
vertices = int(input("Enter number of vertices: "))
edges_count = int(input("Enter number of edges: "))
edges = []
for _ in range(edges_count):
    u, v = map(int, input("Enter edge (u, v): ").split())
    edges.append((u, v))
# Output
print("\nAdjacency Matrix:")
matrix = adjacency_matrix(vertices, edges)
for row in matrix:
    print(row)
print("\nAdjacency List:")
adj_list = adjacency_list(vertices, edges)
for key, value in adj_list.items():
    print(f"{key}: {value}")
```

Output :

```
Enter number of vertices:  5
Enter number of edges:  4
Enter edge (u, v):  1 1
Enter edge (u, v):  2 2
Enter edge (u, v):  3 3
Enter edge (u, v):  4 4

Adjacency Matrix:
[0, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 0, 1]

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

TITLE :- 16 Write a program to implement the Graph traversal methods

- BFS
- DFS

Program :

```
from collections import deque

def bfs(graph, start):

    visited = set()

    queue = deque([start])

    print("BFS Traversal: ", end="")

    while queue:

        node = queue.popleft()

        if node not in visited:

            print(node, end=" ")

            visited.add(node)

            for neighbor in graph[node]:

                if neighbor not in visited:

                    queue.append(neighbor)

def dfs(graph, start, visited=None):

    if visited is None:

        visited = set()

    visited.add(start)
```

```
    print(start, end=" ")
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
# Input
vertices = int(input("Enter number of vertices: "))
edges_count = int(input("Enter number of edges: "))
graph = {i: [] for i in range(vertices)}
for _ in range(edges_count):
    u, v = map(int, input("Enter edge (u, v): ").split())
    graph[u].append(v)
    graph[v].append(u)
# Traversal
start = int(input("Enter starting vertex: "))
bfs(graph, start)
print("\nDFS Traversal: ", end="")
dfs(graph, start)
```

Output :

```
Enter number of vertices:  5
Enter number of edges:  6
Enter edge (u, v):  0 1
Enter edge (u, v):  0 2
Enter edge (u, v):  1 2
Enter edge (u, v):  1 3
Enter edge (u, v):  2 4
Enter edge (u, v):  3 4
Enter starting vertex:  0
BFS Traversal: 0 1 2 3 4
DFS Traversal: 0 1 2 4 3
```

TITLE :- 17 Write a program to sort an elements by using Bubble sort method .

Program :

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
# Input and Testing
arr = list(map(int, input("Enter elements to sort: ").split()))
bubble_sort(arr)
print("Sorted Array:", arr)
```

Output :

```
Enter elements to sort:  1 6 9 3 4 8 12 34
Sorted Array: [1, 3, 4, 6, 8, 9, 12, 34]
```

TITLE :- 18 Write a program to sort an elements by using Merge sort method .

Program :

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
```

```
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
# Input and Testing
arr = list(map(int, input("Enter elements to sort: ").split()))
merge_sort(arr)
print("Sorted Array:", arr)
```

Output :

```
Enter elements to sort:  67 34 23 89 3 6 4 90
Sorted Array: [3, 4, 6, 23, 34, 67, 89, 90]
```

STUDENT NAME:____

CLASS :-………… DIVISION :-……….. ROLL NO :-…………… REMARK :-…………….

DATE OF SUBMISSION :-……………………………………………………………………….

TITLE :- 19 Write a program to sort an elements by using Quick sort method.

Program :

```
def partition(arr, low, high):
    pivot = arr[high]  # Choose the last element as the pivot
    i = low - 1  # Index of smaller element
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]  # Swap
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)  # Partitioning index
        quick_sort(arr, low, pi - 1)  # Sort elements before partition
        quick_sort(arr, pi + 1, high)  # Sort elements after partition
# Input and Testing
arr = list(map(int, input("Enter elements to sort: ").split()))
quick_sort(arr, 0, len(arr) - 1)
print("Sorted Array:", arr)
```

Output :

```
Enter elements to sort:  3 67 2 45 56 8 9 6
Sorted Array: [2, 3, 6, 8, 9, 45, 56, 67]
```

TITLE :- 20 Write a program that implements the following methods

Heap sort.

Program :

```
def heapify(arr, n, i):
    largest = i  # Initialize the largest as root
    left = 2 * i + 1
    right = 2 * i + 2
    if left < n and arr[left] > arr[largest]:
        largest = left

    if right < n and arr[right] > arr[largest]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)
def heap_sort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):  # Build a max heap
        heapify(arr, n, i)
```

```
    for i in range(n - 1, 0, -1):  # Extract elements
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
# Input and Testing
arr = list(map(int, input("Enter elements to sort: ").split()))
heap_sort(arr)
print("Sorted Array:", arr)
```

Output :

```
Enter elements to sort:  34 56 23 87 59 37 4 8 9
Sorted Array: [4, 8, 9, 23, 34, 37, 56, 59, 87]
```

TITLE :- 21 Write a program that implements the Hash Methods

Program :

```
class HashTable:

    def __init__(self, size):

        self.size = size

        self.table = [None] * size

    def hash_function(self, key):

        return key % self.size

    def insert(self, key):

        index = self.hash_function(key)

        if self.table[index] is None:

            self.table[index] = key

        else:

            print(f"Collision occurred for key {key} at index {index}")

    def search(self, key):

        index = self.hash_function(key)

        if self.table[index] == key:

            print(f"Key {key} found at index {index}")

        else:

            print(f"Key {key} not found")
```

```
    def display(self):
        print("Hash Table:")
        for i, value in enumerate(self.table):
            print(f"Index {i}: {value}")
# Menu-driven program
size = int(input("Enter size of hash table: "))
hash_table = HashTable(size)
while True:
    print("\n1. Insert 2. Search 3. Display 4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        key = int(input("Enter key to insert: "))
        hash_table.insert(key)
    elif choice == 2:
        key = int(input("Enter key to search: "))
        hash_table.search(key)
    elif choice == 3:
        hash_table.display()
    elif choice == 4:
        break
```

Output :

```
Enter size of hash table:  3

1. Insert 2. Search 3. Display 4. Exit
Enter your choice:  1
Enter key to insert:  34

1. Insert 2. Search 3. Display 4. Exit
Enter your choice:  1
Enter key to insert:  23

1. Insert 2. Search 3. Display 4. Exit
Enter your choice:  1
Enter key to insert:  12

1. Insert 2. Search 3. Display 4. Exit
Enter your choice:  3
Hash Table:
Index 0: 12
Index 1: 34
Index 2: 23

1. Insert 2. Search 3. Display 4. Exit
Enter your choice:  2
Enter key to search:  23
Key 23 found at index 2

1. Insert 2. Search 3. Display 4. Exit
Enter your choice:  4
```