Practical-1

a) Write a program to store the elements in 1-D array and perform the operations like searching, sorting and reversing the elements. [Menu Driven]

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

def search_element(arr, element):
    for i in range(len(arr)):
        if arr[i] == element:
            return i
    return -1

def reverse_elements(arr):
    reversed_arr = []
    for item in arr:
        if isinstance(item, int):
            reversed_arr.append(int(str(item)[::-1]))
        elif isinstance(item, str):
            reversed_arr.append(item[::-1])
    return reversed_arr

# Input list from the user
input_list = []
n = int(input("Enter the number of elements in the list: "))
print("Enter elements for the list:")
for i in range(n):
    input_list.append(input())

# Sorting the list
bubble_sort(input_list)
print("Sorted List (Ascending Order):", input_list)

# Searching an element
search_elem = input("Enter an element to search: ")
index = search_element(input_list, search_elem)
if index != -1:
    print("Element '%s' found at index %d" % (search_elem, index))
```

```
else:
    print("Element '%s' not found in the list" % search_elem)

# Reversing elements
reversed_list = reverse_elements(input_list)
print("Reversed List:", reversed_list)
```

b) Read the two arrays from the user and merge them and display the elements in sorted order. [Menu Driven]

```
def merge_and_sort_arrays(arr1, arr2):
    merged_array = arr1 + arr2
    print("Merged Array:", merged_array)

    # Sorting the merged array using the Bubble Sort algorithm
    n = len(merged_array)
    for i in range(n):
        for j in range(0, n-i-1):
            if merged_array[j] > merged_array[j+1]:
                merged_array[j], merged_array[j+1] = merged_array[j+1],
merged_array[j]

    print("Sorted Array (Ascending Order):", merged_array)

# Input arrays from the user
arr1 = []
arr2 = []

# Input for the first array
n1 = int(input("Enter the number of elements for the first array: "))
print("Enter elements for the first array:")
for i in range(n1):
    arr1.append(int(input()))

# Input for the second array
n2 = int(input("Enter the number of elements for the second array: "))
print("Enter elements for the second array:")
for i in range(n2):
    arr2.append(int(input()))

# Call the merge_and_sort_arrays function
merge_and_sort_arrays(arr1, arr2)
```

c) Write a program to perform the Matrix addition, Multiplication and Transpose Operation. [Menu Driven]

```
def input_matrix(rows, cols):
    matrix = []
    for i in range(rows):
        row = []
        for j in range(cols):
            element = int(input("Enter element at row {} column {}: ".format(i +
1, j + 1)))
            row.append(element)
        matrix.append(row)
    return matrix

def print_matrix(matrix):
    for row in matrix:
        print(row)
def add_matrices(matrix1, matrix2):
    result = []
    for i in range(len(matrix1)):
        row = []
        for j in range(len(matrix1[0])):
            row.append(matrix1[i][j] + matrix2[i][j])
        result.append(row)
    return result
def multiply_matrices(matrix1, matrix2):
    result = []
    for i in range(len(matrix1)):
        row = []
        for j in range(len(matrix2[0])):
            sum = 0
            for k in range(len(matrix2)):
                sum += matrix1[i][k] * matrix2[k][j]
            row.append(sum)
        result.append(row)
    return result
def transpose_matrix(matrix):
    rows = len(matrix)
    cols = len(matrix[0])
    transposed = [[0 for _ in range(rows)] for _ in range(cols)]
    for i in range(rows):
        for j in range(cols):
            transposed[j][i] = matrix[i][j]
    return transposed
```

```
# Input dimensions of matrices
rows1 = int(input("Enter the number of rows for the first matrix: "))
cols1 = int(input("Enter the number of columns for the first matrix: "))
rows2 = int(input("Enter the number of rows for the second matrix: "))
cols2 = int(input("Enter the number of columns for the second matrix: "))

# Input elements for matrices
print("Enter elements for the first matrix:")
matrix1 = input_matrix(rows1, cols1)
print("Enter elements for the second matrix:")
matrix2 = input_matrix(rows2, cols2)

# Perform addition
if rows1 == rows2 and cols1 == cols2:
    print("Matrix Addition:")
    result_addition = add_matrices(matrix1, matrix2)
    print_matrix(result_addition)
else:
    print("Matrices cannot be added due to different dimensions.")

# Perform multiplication
if cols1 == rows2:
    print("Matrix Multiplication:")
    result_multiplication = multiply_matrices(matrix1, matrix2)
    print_matrix(result_multiplication)
else:
    print("Matrices cannot be multiplied due to incompatible dimensions.")

# Perform transpose
print("Transpose of the first matrix:")
result_transpose = transpose_matrix(matrix1)
print_matrix(result_transpose)

print("Transpose of the second matrix:")
result_transpose = transpose_matrix(matrix2)
print_matrix(result_transpose)
```

Practical –2

A) Write a program to create a single linked list and display the node elements in reverse order.

```
#Creating New Node
class node:
    def __init__(self,info):
        self.info=info
        self.next=None

#Perform Operation
class LinkedList:
    def __init__(self):
        self.begin=None #initializing a list
    #inserting elements in LinkList
    def insert(self,info):
        """
        Calling class "node" to create a new node and storing data
        inside variable "new_node"
        """
        new_node=node(info)
        """
        Checking whether LinkList is empty or not by "begin"
        if it is empty then storing address of first node inside the "begin"
        """
        if self.begin is None:
            self.begin=new_node
        else: #else storing address of node inside the "next part" of previous
node
            pointer=self.begin
            while pointer.next!=None:
                pointer=pointer.next
            pointer.next=new_node
    #Displaying data of LinkList
    def display(self):
        #Cheking whether LinkList is empty or not
        if self.begin is None:
            print("Linked List is empty")
        else:
            pointer = self.begin
            while pointer != None:
                print(pointer.info)
                pointer=pointer.next#pointing pointer to next node
    #Reversing the linked list
    def reversing(self):
        #Checking whether LinkList is empty or not
        if self.begin is None:
            print("Linked List is empty")
        else:
            #Creating pointers
            pointer = self.begin#For Adress of Current node
            pointer_prev = None  #For Adress of previous node
            pointer_next = self.begin#For Adress of Next node
            #Treversing thorugh list
            while pointer_next != None:
                pointer_next=pointer_next.next#Updating adress of next node
                pointer.next=pointer_prev     #Linking current node to previous
node
                pointer_prev=pointer          #Updating Adress of preveious node
                pointer=pointer_next          #Updating Adress of current node
            self.begin=pointer_prev          #Setting begine

l=LinkedList()
n=int(input("How many elements you want to enter: "))
for i in range(0,n):
    data=input("Enter element: ")
    l.insert(data)
l.reversing()
print("Elements in reversed order are: ")
l.display()
```

b) Write a program to search the elements in the linked list and display the same

```
class node:
    def __init__(self,info):
        self.info=info
        self.next=None

class LinkedList:
    def __init__(self):
        self.begin=None
    def insert(self,info):
        new_node=node(info)
        if self.begin is None:
            self.begin=new_node
        else:
            pointer=self.begin
            while pointer.next!=None:
                pointer=pointer.next
            pointer.next=new_node
    def display(self):
        if self.begin is None:
            print("Linked List is empty")
        else:
            pointer = self.begin
            while pointer != None:
                print(pointer.info)
                pointer=pointer.next
    def Search(self,n):
        if n == 0:
            print("There are no elements to search in this Linked List! ")
            return
        ele=input("Enter element you want to search in Linked List: ")
        pointer = self.begin
        while pointer != None:
            if pointer.info == ele:
                print(ele," is present at ",pointer)
                return
            pointer=pointer.next
        print("The entered element is not present in the list! ")

#Driver code
l=LinkedList()
n=int(input("How many elements you want to enter: "))
for i in range(0,n):
    data=input("Enter element: ")
    l.insert(data)
l.Search(n)
```

c) Write a program to create double linked list and sort the elements in the linked list.

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

class doublyLinkedList:
    def __init__(self):
        self.start_node = None
    def InsertToEmptyList(self, data):
        if self.start_node is None:
            new_node = Node(data)
            self.start_node = new_node
        else:
            print("The list is not empty")
    def InsertToEnd(self, data):
        if self.start_node is None:
            new_node = Node(data)
            self.start_node = new_node
            return
        n = self.start_node
        while n.next is not None:
            n = n.next
        new_node = Node(data)
        n.next = new_node
        new_node.prev = n
    def Display(self):
        if self.start_node is None:
            print("The list is empty")
            return
        else:
            n = self.start_node
            while n is not None:
                print("Element is:", n.item)
                n = n.next
        print("\n")
    def BubbleSort(self):
        if self.start_node is None:
            print("The list is empty")
            return
        else:
            end = None
            while end != self.start_node:
                n = self.start_node
                while n.next != end:
                    if n.item > n.next.item:
                        n.item, n.next.item = n.next.item, n.item
                    n = n.next
                end = n


# Create a new Doubly Linked List
```

```
NewDoublyLinkedList = doublyLinkedList()
# Insert elements
NewDoublyLinkedList.InsertToEmptyList(60)
NewDoublyLinkedList.InsertToEnd(30)
NewDoublyLinkedList.InsertToEnd(40)
NewDoublyLinkedList.InsertToEnd(20)
NewDoublyLinkedList.InsertToEnd(10)
NewDoublyLinkedList.InsertToEnd(50)
# Display initial data
print("Initial data for Linked List:")
NewDoublyLinkedList.Display()
# Sort using BubbleSort
NewDoublyLinkedList.BubbleSort()
# Display sorted data
print("Sorted data using bubble sort:")
NewDoublyLinkedList.Display()
```

Practical –3

a) Write a program to implement the concept of Stack with Push, Pop, Display and Exit operations

```
class stack():
   def __init__(self):
       self.stack=list()
       self.maxSize=8
       self.top=0

   def push(self,data):
     if self.top>=self.maxSize:
         return data,' can\'t be pushed in as stack is full !'
     self.stack.append(data)
     self.top +=1
     return data,' is pushed in the stack'

   def pop(self):
     if self.top<=0:
         return("stack is empty !")
     item=self.stack.pop()
     self.top-=1
     return item,' is popped out of the stack'
```

```
    def size(self):
        return self.top

s=stack()
print("\n before stacking the values, the size is: ",s.size(),"\n")
for i in range(1,4):
    print(s.push(i))
print("\n after pushing in the values, the size is: ",s.size(),"\n")
for i in range(1,2):
    print(s.pop())
print("\n after popping out the values, the size is: ",s.size(),"\n")
```

b) Write a program to convert an infix expression to postfix and prefix conversion

```
OPERATORS = set(['+', '-', '*', '/', '(', ')', '^'])
PRIORITY = {'+':1, '-':1, '*':2, '/':2, '^':3}

def infix_to_postfix(expression):
    stack = []
    output = ''
    for ch in expression:
        if ch not in OPERATORS:
            output+= ch
            print(output)
        elif ch=='(':
            stack.append('(')
            print(output)
        elif ch==')':
            while stack and stack[-1]!= '(':
                output+=stack.pop()
            stack.pop()
            print(output)
        else:
           while stack and stack[-1]!='(' and PRIORITY[ch]<=PRIORITY[stack[-1]]:
               output+=stack.pop()
           stack.append(ch)
           print(output)

    while stack:
        output+=stack.pop()
        print(output)
    return output

def infix_to_prefix(expression):
    stack = []
    output = ''
    expression = expression[::-1]

    for ch in expression:
        if ch not in OPERATORS:
            output += ch
            print(output)
        elif ch == ')':
            stack.append(')')
            print(output)
        elif ch == '(':
            while stack and stack[-1] != ')':
                output += stack.pop()
            stack.pop()
            print(output)
        else:
            while stack and stack[-1] != ')' and PRIORITY[ch] < PRIORITY[stack[-
1]]:
                output += stack.pop()
            stack.append(ch)
            print(output)

    while stack:
        output += stack.pop()
        print(output)

    return output[::-1]

expression = input('Enter infix expression: ')
print('infix expression: ',expression)
print('prefix expression: ',infix_to_prefix(expression))
print('postfix expression: ',infix_to_postfix(expression))
```

Practical-4

a) Write a program to implement the concept of Queue with Insert, Delete, Display and Exit operations.

```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        self.queue.append(item)

    def dequeue(self):
        return None if not self.queue else self.queue.pop(0)

    def display(self):
        print(self.queue)

    def is_empty(self):
        return "The queue is empty" if len(self.queue) == 0 else "The queue is 
not empty"

# Create a queue
queue = Queue()
# Insert elements into the queue
for i in range(1,6):
    queue.enqueue(i)
print("After inserting 5 elements in the queue: ")
# Display the queue
queue.display()
print("After deleting 5 elements in the queue: ")
# Delete an element from the queue
for i in range(6):
    queue.dequeue()
# Display the queue again
queue.display()
# Check if the queue is empty
queue.is_empty()
```

b) Write a program to implement the concept of Circular Queue

```
    self.queue = [None] * size
        self.front = 0
        self.rear = 0
        self.size = size

    def enqueue(self, item):
        if self.is_full():
            returnclass CircularQueue:
    def __init__(self, size):

        self.rear = (self.rear + 1) % self.size
        self.queue[self.rear] = item

    def dequeue(self):
        if self.is_empty():
            return None
        self.front = (self.front + 1) % self.size
        item = self.queue[self.front]
        self.queue[self.front] = None
        return item

    def is_empty(self):
        return self.front == self.rear

    def is_full(self):
        return (self.rear + 1) % self.size == self.front

# Create a circular queue
queue = CircularQueue(5)

# Insert elements into the queue
queue.enqueue('one')
queue.enqueue('two')
queue.enqueue('three')
print('after entering the values in the queue: ')
# Display the queue
print(queue.queue)

# Delete an element from the queue
for i in range(4):
    item = queue.dequeue()
print("after deleting the all the items in the queue: ")

# Display the queue again
print(queue.queue)

# Check if the queue is empty
if queue.is_empty():
    print("The queue is empty")
else:
    print("The queue is not empty")

```

c) Write a program to implement the concept of Deque

```
class Deque:
    def __init__(self):
        self.queue = []

    def enqueue_front(self, item):
        self.queue.insert(0, item)

    def enqueue_rear(self, item):
        self.queue.append(item)

    def dequeue_front(self):
        if not self.queue:
            return None
        return self.queue.pop(0)

    def dequeue_rear(self):
        if not self.queue:
            return None
        return self.queue.pop()

    def is_empty(self):
        return len(self.queue) == 0

    def size(self):
        return len(self.queue)

# Create a deque
deque = Deque()
print("by inserting values '1','2','3' from the front and '4','5','6' from back:
")
# Insert elements into the deque
for i in range(1,4):
    deque.enqueue_front(i)
for j in range(4,7):
    deque.enqueue_rear(j)
```

```
# Display the deque
print(deque.queue)

# Delete an element from the front of the deque
print("after deleting all the elements in the deque:")
for i in range(1,4):
    deque.dequeue_front()
for j in range(4,7):
    deque.dequeue_rear()

# Display the deque again
print(deque.queue)

# Check if the deque is empty
if deque.is_empty():
    print("The deque is empty")
else:
    print("The deque is not empty")
```

Pracical –5

a) Write a program to implement bubble sort.

```
# Creating a bubble sort function
def bubble_sort(list1):
    # Outer loop for traversing the entire list
    for i in range(0, len(list1) - 1):
        for j in range(0, len(list1) - 1 - i):  # Adjusted the inner loop range
            if list1[j] > list1[j + 1]:
                temp = list1[j]
                list1[j] = list1[j + 1]
                list1[j + 1] = temp
    return list1  # Moved return statement outside the inner loop

list1 = [43, 21, 76, 53, 98, 245, 20]
print("The unsorted list is:", list1)
# Calling the bubble sort function
print("The sorted list is:", bubble_sort(list1))
```

b) Write a program to implement selection sort.

```
# Selection sort in Python
def selectionSort(array, size):
    for step in range(size):
        min_idx = step
        for i in range(step + 1, size):
            # To sort in descending order, change > to <
            # Select the maximum element in each loop for descending order
            if array[i] < array[min_idx]:
                min_idx = i
        # Put the minimum at the correct position
        array[step], array[min_idx] = array[min_idx], array[step]
data = [-2, 45, 0, 11, -9]
size = len(data)
selectionSort(data, size)
print('Sorted Array in Ascending Order:')
print(data)
```

c) Write a program to implement insertion sort.

```
def insertionSort(array):
    for step in range(1, len(array)):
        key = array[step]
        j = step - 1
        # Compare key with each element on the left of it until an element
smaller than it is found
        # For descending order, change key < array[j] to key > array[j].
        while j >= 0 and key < array[j]:
            array[j + 1] = array[j]
            j = j - 1
        # Place key after the element just smaller than it.
        array[j + 1] = key

data = [9, 5, 1, 4, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

Practcial-6

a) Write a program to implement merge sort.

```
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_list = [12, 11, 13, 5, 6, 39, 7]
print("Original list:", input_list)
merge_sort(input_list)
print("\nSorted list:", input_list)
```

b) Write a program to search the element using sequential search.

```
def Sequential_Search(dlist, item):
    pos, found = 0, False
    while pos < len(dlist) and not found:
        if dlist[pos] == item:
            found = True
        else:
            pos = pos + 1
    return pos
list=[11,23,58,31,56,77,43,12,65,19]
answer= Sequential_Search(list, 77 )
if int(answer)+1 > int(len(list)):
    print("element is not found")
else:
    print("element is found at the position: ",answer)
```

c) Write a program to search the element using binary search.

```
def binary_search(list1, n):
    low, high, mid = 0, len(list1) - 1, 0
    while low <= high:
        mid = (high + low) // 2
        if list1[mid] < n:
            low = mid + 1
        elif list1[mid] > n:
            high = mid - 1
        else:
            return mid
    return -1
list1 = [12, 24, 32, 39, 45, 50, 54]
n = 50
result = binary_search(list1, n)
print("Element is present at index", str(result)) if result != -1 else
print("Element is not present in the list")
```

Practical-7

A) Write a program to create the tree and display the elements.

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

def create_tree():
    root = Node('A-Root')
    root.left = Node('B-Left node of the root A')
    root.right = Node('C-Right node of the root A')
    root.left.left = Node('D-Left node of parent B')
    root.left.right = Node('E-Right node of parent B')
    root.left.right.left = Node('F-Left node of parent E')
    root.left.right.right = Node('G-Right node of parent E')
    root.left.right.left.left = Node('H-Left node of parent F')
    return root

def print_tree(root):
    if root: print(root.data), print_tree(root.left), print_tree(root.right)

print_tree(create_tree())
```

b) Write a program to construct the binary tree.

```
class Node:
  def __init__(self, data):
      self.data = data
      self.left = None
      self.right = None
def create_tree(data_list):
    root = Node(data_list[0])
    queue = [root]
    i = 1
    while i < len(data_list):
        node = queue.pop(0)
        if data_list[i] is not None:
            node.left = Node(data_list[i])
            queue.append(node.left)
        i += 1
        if i < len(data_list):
            node.right = Node(data_list[i])
            queue.append(node.right)
            i += 1
    return root
def print_tree(root):
    if root: print(root.data), print_tree(root.left), print_tree(root.right)
print_tree(create_tree([1, 2, 3, None, None, 4, 5]))
```

c) Write a program for inorder, postorder and preorder traversal of tree

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
    def inorder(self):
        if self is not None:
            if self.left:
                self.left.inorder()
            print(self.data)
            if self.right:
                self.right.inorder()
    def postorder(self):
        if self is not None:
            if self.left:
                self.left.postorder()
            if self.right:
                self.right.postorder()
            print(self.data)
    def preorder(self):
        if self is not None:
            print(self.data)
            if self.left:
                self.left.preorder()
            if self.right:
                self.right.preorder()
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print("Inorder traversal:"),root.inorder()
print("Postorder traversal:"),root.postorder()
print("Preorder traversal:"),root.preorder()
```

Practical-8

a) Write a program to insert the element into maximum heap.

```
def insert_element(heap, element):
    heap.append(element)
    current_index = len(heap) - 1
    while current_index > 0:
        parent_index = (current_index - 1) // 2
        if heap[current_index] > heap[parent_index]:
            heap[current_index], heap[parent_index] = heap[parent_index],
heap[current_index]
            current_index = parent_index
        else:
            break
    return heap
heap = []
heap = insert_element(heap, 10)o
heap = insert_element(heap, 5)
heap = insert_element(heap, 15)
heap = insert_element(heap, 2)
print("elements in maximum heap (descending order): ",heap)
```

b) Write a program to insert the element into minimum heap.

```
def insert_element(heap, element):
    heap.append(element)
    current_index = len(heap) - 1
    while current_index > 0:
        parent_index = (current_index - 1) // 2
        if heap[current_index] < heap[parent_index]:
            heap[current_index], heap[parent_index] = heap[parent_index], 
heap[current_index]
            current_index = parent_index
        else:
            break
    return heap
heap = []
heap = insert_element(heap, 10)
heap = insert_element(heap, 5)
heap = insert_element(heap, 15)
heap = insert_element(heap, 2)
print("elements in minimum heap (ascending order): ",heap)
```

Practical-9

a) Write a program to implement the collision technique

```
def hash_function(key):
    return key % 10
def separate_chaining(keys, values):
    hash_table = [[] for _ in range(10)]
    for key, value in zip(keys, values):
        index = hash_function(key)
        hash_table[index].append((key, value))
```

```python
    return hash_table
def detect_collision(hash_table):
    for bucket in hash_table:
        if len(bucket) > 1:
            print("Collision detected at bucket:", bucket)
hash_table = separate_chaining([1, 2, 3, 4, 5, 1], ["apple", "banana", "cherry",
"apple", "orange", "strawberry"])
print("\n".join(map(str, hash_table)))
detect_collision(hash_table)
```