DATA STRUCTURE LAB

1. **Program to Insert and Delete an element in an Array:**
   a. **Insert an element in an Array:**
   Python code to insert an element:

```
# given array (list)
arr = [1, 2, 3, 4, 5]
num=int(input("Enter a number to insert in array : "))
index=int(input("Enter a index to insert value : "))
if index >= len(arr):
    print("please enter index smaller than",len(arr))
else:
    # insering element ‘num’ at ‘index’ position
    arr.insert(index, num)
    print("Array after inserting",num,"=",arr)
```

   b. **Delete an element in an Array**
   Python code to delete an element:

```
#taking input to fix the size of the array
size=int(input("Enter the number of elements you want in array: "))
arr=[]
# adding elements to the array
for i in range(0,size):
        elem=int(input("Please give value for index "+str(i)+": "))
        arr.append(elem)
        num=int(input("Enter a number to remove from array : "))
        # removing element ‘num’ from the array.
        arr.remove(num)
        print("Array after removing",num,"=",arr)
```

2. Program to implement operations on a Singly Linked List

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


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

    def get_node(self, index):
        current = self.head
        for i in range(index):
            if current is None:
                return None
            current = current.next
        return current

    def get_prev_node(self, ref_node):
        current = self.head
        while (current and current.next != ref_node):
            current = current.next
        return current

    def insert_after(self, ref_node, new_node):
        new_node.next = ref_node.next
        ref_node.next = new_node

    def insert_before(self, ref_node, new_node):
        prev_node = self.get_prev_node(ref_node)
        self.insert_after(prev_node, new_node)

    def insert_at_beg(self, new_node):
        if self.head is None:
            self.head = new_node
        else:
            new_node.next = self.head
            self.head = new_node

    def insert_at_end(self, new_node):
        if self.head is None:
            self.head = new_node
```

```
        else:
            current = self.head
            while current.next is not None:
                current = current.next
            current.next = new_node

    def remove(self, node):
        prev_node = self.get_prev_node(node)
        if prev_node is None:
            self.head = self.head.next
        else:
            prev_node.next = node.next

    def display(self):
        current = self.head
        while current:
            print(current.data, end = ' ')
            current = current.next


a_llist = LinkedList()

print('Menu')
print('insert <data> after <index>')
print('insert <data> before <index>')
print('insert <data> at beg')
print('insert <data> at end')
print('remove <index>')
print('quit')

while True:
    print('The list: ', end = '')
    a_llist.display()
    print()
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()

    if operation == 'insert':
        data = int(do[1])
        position = do[3].strip().lower()
        new_node = Node(data)
        suboperation = do[2].strip().lower()
        if suboperation == 'at':
            if position == 'beg':
```

```
            a_llist.insert_at_beg(new_node)
        elif position == 'end':
            a_llist.insert_at_end(new_node)
    else:
        index = int(position)
        ref_node = a_llist.get_node(index)
        if ref_node is None:
            print('No such index.')
            continue
        if suboperation == 'after':
            a_llist.insert_after(ref_node, new_node)
        elif suboperation == 'before':
            a_llist.insert_before(ref_node, new_node)

elif operation == 'remove':
    index = int(do[1])
    node = a_llist.get_node(index)
    if node is None:
        print('No such index.')
        continue
    a_llist.remove(node)

elif operation == 'quit':
    break
```

3. Program to implement operations on a Doubly Linked List

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


class DoublyLinkedList:
    def __init__(self):
        self.first = None
        self.last = None

    def get_node(self, index):
        current = self.first
        for i in range(index):
            if current is None:
                return None
            current = current.next
        return current

    def insert_after(self, ref_node, new_node):
        new_node.prev = ref_node
        if ref_node.next is None:
            self.last = new_node
        else:
            new_node.next = ref_node.next
            new_node.next.prev = new_node
        ref_node.next = new_node

    def insert_before(self, ref_node, new_node):
        new_node.next = ref_node
        if ref_node.prev is None:
            self.first = new_node
        else:
            new_node.prev = ref_node.prev
            new_node.prev.next = new_node
        ref_node.prev = new_node

    def insert_at_beg(self, new_node):
        if self.first is None:
            self.first = new_node
```

```
            self.last = new_node
        else:
            self.insert_before(self.first, new_node)

    def insert_at_end(self, new_node):
        if self.last is None:
            self.last = new_node
            self.first = new_node
        else:
            self.insert_after(self.last, new_node)

    def remove(self, node):
        if node.prev is None:
            self.first = node.next
        else:
            node.prev.next = node.next

        if node.next is None:
            self.last = node.prev
        else:
            node.next.prev = node.prev

    def display(self):
        current = self.first
        while current:
            print(current.data, end = ' ')
            current = current.next


a_dllist = DoublyLinkedList()

print('Menu')
print('insert <data> after <index>')
print('insert <data> before <index>')
print('insert <data> at beg')
print('insert <data> at end')
print('remove <index>')
print('quit')

while True:
    print('The list: ', end = '')
    a_dllist.display()
    print()
    do = input('What would you like to do? ').split()
```

```
operation = do[0].strip().lower()

if operation == 'insert':
    data = int(do[1])
    position = do[3].strip().lower()
    new_node = Node(data)
    suboperation = do[2].strip().lower()
    if suboperation == 'at':
        if position == 'beg':
            a_dllist.insert_at_beg(new_node)
        elif position == 'end':
            a_dllist.insert_at_end(new_node)
    else:
        index = int(position)
        ref_node = a_dllist.get_node(index)
        if ref_node is None:
            print('No such index.')
            continue
        if suboperation == 'after':
            a_dllist.insert_after(ref_node, new_node)
        elif suboperation == 'before':
            a_dllist.insert_before(ref_node, new_node)

elif operation == 'remove':
    index = int(do[1])
    node = a_dllist.get_node(index)
    if node is None:
        print('No such index.')
        continue
    a_dllist.remove(node)

elif operation == 'quit':
    break
```

4. Program to Sort the Elements using Insertion Sort

```
def insertion_sort(alist):
    for i in range(1, len(alist)):
        temp = alist[i]
        j = i - 1
        while (j >= 0 and temp < alist[j]):
            alist[j + 1] = alist[j]
            j = j - 1
        alist[j + 1] = temp
alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
insertion_sort(alist)
print('Sorted list: ', end='')
print(alist)
```

5. Program to Sort the Elements using Quick Sort:

```
def quicksort(alist, start, end):
    '''Sorts the list from indexes start to end - 1 inclusive.'''
    if end - start > 1:
        p = partition(alist, start, end)
        quicksort(alist, start, p)
        quicksort(alist, p + 1, end)

def partition(alist, start, end):
    pivot = alist[start]
    i = start + 1
    j = end - 1

    while True:
        while (i <= j and alist[i] <= pivot):
            i = i + 1
        while (i <= j and alist[j] >= pivot):
            j = j - 1
        if i <= j:
            alist[i], alist[j] = alist[j], alist[i]
        else:
            alist[start], alist[j] = alist[j], alist[start]
            return j

alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
quicksort(alist, 0, len(alist))
print('Sorted list: ', end='')
print(alist)
```

6. Program to Sort the Elements using Merge Sort:

```
def merge_sort(alist, start, end):
    '''Sorts the list from indexes start to end - 1 inclusive.'''
    if end - start > 1:
        mid = (start + end)//2
        merge_sort(alist, start, mid)
        merge_sort(alist, mid, end)
        merge_list(alist, start, mid, end)

def merge_list(alist, start, mid, end):
    left = alist[start:mid]
    right = alist[mid:end]
    k = start
    i = 0
    j = 0
    while (start + i < mid and mid + j < end):
        if (left[i] <= right[j]):
            alist[k] = left[i]
            i = i + 1
        else:
            alist[k] = right[j]
            j = j + 1
        k = k + 1
    if start + i < mid:
        while k < end:
            alist[k] = left[i]
            i = i + 1
            k = k + 1
    else:
        while k < end:
            alist[k] = right[j]
            j = j + 1
            k = k + 1

alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
merge_sort(alist, 0, len(alist))
print('Sorted list: ', end='')
print(alist)
```

7.

a. Python Program to Implement Stack using an array

```
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, data):
        self.items.append(data)

    def pop(self):
        return self.items.pop()

s = Stack()
while True:
    print('push <value>')
    print('pop')
    print('quit')
    do = input('What would you like to do? ').split()
    operation = do[0].strip().lower()
    if operation == 'push':
        s.push(int(do[1]))
    elif operation == 'pop':
        if s.is_empty():
            print('Stack is empty.')
        else:
            print('Popped value: ', s.pop())
    elif operation == 'quit':
        break
```

b. Python Program to Implement Stack using Linked List

```
class Stack:
    def __init__(self):
        self.head = None

    def push(self, data):
        if self.head is None:
            self.head = Node(data)
        else:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node

    def pop(self):
        if self.head is None:
            return None
        else:
            popped = self.head.data
            self.head = self.head.next
            return popped

a_stack = Stack()
while True:
    print('push <value>')
    print('pop')
    print('quit')
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()
    if operation == 'push':
        a_stack.push(int(do[1]))
    elif operation == 'pop':
        popped = a_stack.pop()
        if popped is None:
            print('Stack is empty.')
        else:
            print('Popped value: ', int(popped))
    elif operation == 'quit':
        break
```

8.

a. Python Program to Implement Queue using array

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

    def is_empty(self):
        return self.items == []

    def enqueue(self, data):
        self.items.append(data)

    def dequeue(self):
        return self.items.pop(0)

q = Queue()
while True:
    print('enqueue <value>')
    print('dequeue')
    print('quit')
    do = input('What would you like to do? ').split()
    operation = do[0].strip().lower()
    if operation == 'enqueue':
        q.enqueue(int(do[1]))
    elif operation == 'dequeue':
        if q.is_empty():
            print('Queue is empty.')
        else:
            print('Dequeued value: ', q.dequeue())
    elif operation == 'quit':
        break
```

b. Python Program to Implement Queue using Linked List

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

class Queue:
    def __init__(self):
        self.head = None
        self.last = None
    def enqueue(self, data):
        if self.last is None:
            self.head = Node(data)
            self.last = self.head
        else:
            self.last.next = Node(data)
            self.last = self.last.next
    def dequeue(self):
        if self.head is None:
            return None
        else:
            to_return = self.head.data
            self.head = self.head.next
            return to_return
a_queue = Queue()
while True:
    print('enqueue <value>')
    print('dequeue')
    print('quit')
    do = input('What would you like to do? ').split()
    operation = do[0].strip().lower()
    if operation == 'enqueue':
        a_queue.enqueue(int(do[1]))
    elif operation == 'dequeue':
        dequeued = a_queue.dequeue()
        if dequeued is None:
            print('Queue is empty.')
        else:
            print('Dequeued element: ', int(dequeued))
    elif operation == 'quit':
        break
```

9. Program to implement Circular Queue:

```
class CircularQueue():

    # constructor
    def __init__(self, size): # initializing the class
        self.size = size

        # initializing queue with none
        self.queue = [None for i in range(size)]
        self.front = self.rear = -1

    def enqueue(self, data):

        # condition if queue is full
        if ((self.rear + 1) % self.size == self.front):
            print(" Queue is Full\n")

        # condition for empty queue
        elif (self.front == -1):
            self.front = 0
            self.rear = 0
            self.queue[self.rear] = data
        else:

            # next position of rear
            self.rear = (self.rear + 1) % self.size
            self.queue[self.rear] = data

    def dequeue(self):
        if (self.front == -1): # condition for empty queue
            print ("Queue is Empty\n")

        # condition for only one element
        elif (self.front == self.rear):
            temp=self.queue[self.front]
            self.front = -1
            self.rear = -1
            return temp
        else:
            temp = self.queue[self.front]
            self.front = (self.front + 1) % self.size
            return temp
```

```
    def display(self):

        # condition for empty queue
        if(self.front == -1):
            print ("Queue is Empty")

        elif (self.rear >= self.front):
            print("Elements in the circular queue are:", end = " ")
            for i in range(self.front, self.rear + 1):
                print(self.queue[i], end = " ")
            print ()
        else:
            print ("Elements in Circular Queue are:", end = " ")
            for i in range(self.front, self.size):
                print(self.queue[i], end = " ")
            for i in range(0, self.rear + 1):
                print(self.queue[i], end = " ")
            print ()

        if ((self.rear + 1) % self.size == self.front):
            print("Queue is Full")

# Driver Code
ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

10.program to convert infix to postfix expression

```
def infixToPostfix(expression):

    stack = [] # initialization of empty stack

    output = ''

    for character in expression:

        if character not in Operators:  # if an operand append in postfix expression

            output+= character

        elif character=='(':  # else Operators push onto stack

            stack.append('(')

        elif character==')':

            while stack and stack[-1]!= '(':

                output+=stack.pop()

            stack.pop()

        else:

            while stack and stack[-1]!='(' and Priority[character]<=Priority[stack[-1]]:

                output+=stack.pop()

            stack.append(character)

    while stack:

        output+=stack.pop()

    return output
expression = input('Enter infix expression ')
print('infix notation: ',expression)
print('postfix notation: ',infixToPostfix(expression))
```

11. <u>Program to implement display elements of a queue according to their priority</u>

```
# A simple implementation of Priority Queue
# using Queue.
class PriorityQueue(object):
	def __init__(self):
		self.queue = []

	def __str__(self):
		return ' '.join([str(i) for i in self.queue])

	# for checking if the queue is empty
	def isEmpty(self):
		return len(self.queue) == 0

	# for inserting an element in the queue
	def insert(self, data):
		self.queue.append(data)

	# for popping an element based on Priority
	def delete(self):
		try:
			max_val = 0
			for i in range(len(self.queue)):
				if self.queue[i] > self.queue[max_val]:
					max_val = i
			item = self.queue[max_val]
			del self.queue[max_val]
			return item
		except IndexError:
			print()
			exit()

if __name__ == '__main__':
	myQueue = PriorityQueue()
	myQueue.insert(12)
	myQueue.insert(1)
	myQueue.insert(14)
	myQueue.insert(7)
	print(myQueue)
	while not myQueue.isEmpty():
		print(myQueue.delete())
```