**DAY-5 LAB PROGRAMS:-**

1. **You are given the heads of two sorted linked lists list1 and list2. Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.**

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mergeTwoLists(list1, list2):

    dummy = ListNode()
    current = dummy

    while list1 and list2:
        if list1.val <= list2.val:
            current.next = list1
            list1 = list1.next
        else:
            current.next = list2
            list2 = list2.next
        current = current.next

       if list1:
        current.next = list1
    elif list2:
        current.next = list2


    return dummy.next
def printList(node):
    while node:
        print(node.val, end=" -> ")
        node = node.next
    print("None")


list1 = ListNode(1, ListNode(2, ListNode(4)))
list2 = ListNode(1, ListNode(3, ListNode(4)))

merged_list = mergeTwoLists(list1, list2)
printList(merged_list)
```

**Output:**

```
1 -> 1 -> 2 -> 3 -> 4 -> 4 -> None
```

**2) Merge k Sorted Lists You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.**

```
    from heapq import heappop, heappush
class ListNode:
  def __init__(self, val=0, next=None):
    self.val = val
    self.next = next


  def __lt__(self, other):


    return self.val < other.val


def mergeKLists(lists):
    heap = []
  for i in range(len(lists)):
    if lists[i]:
      heappush(heap, lists[i])




  dummy = ListNode()
  current = dummy


  while heap:


    node = heappop(heap)
    current.next = node
    current = current.next


    if node.next:
      heappush(heap, node.next)
    return dummy.next


def printList(node):
    while node:
        print(node.val, end=" -> ")
        node = node.next
    print("None")


list1 = ListNode(1, ListNode(4, ListNode(5)))
list2 = ListNode(1, ListNode(3, ListNode (4)))
list3 = ListNode(2, ListNode(6))


lists = [list1, list2, list3]


merged_list = mergeKLists(lists)
printList(merged_list)
```

**<u>Output:</u>**

```
1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6 -> None
```

3. **Remove Duplicates from Sorted Array Given an integer array nums sorted in non-decreasing order, remove the duplicates in place such that each unique element appears only once. The relative order of the elements should be kept the same. Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements. Return k after placing the final result in the first k slots of nums.**

```
            def removeDuplicates(nums):
    if not nums:
        return 0
    write_index = 1


    for read_index in range(1, len(nums)):
        if nums[read_index] != nums[read_index - 1]:


            nums[write_index] = nums[read_index]

            write_index += 1


    return write_index

nums = [0,0,1,1,1,2,2,3,3,4]

k = removeDuplicates(nums)

print(f"After removing duplicates, k = {k}")

print ("Modified array:", nums[:k])
```

**Output:**

```
After removing duplicates, k = 5
Modified array: [0, 1, 2, 3, 4]
```

4. **Search in Rotated Sorted Array There is an integer array nums sorted in ascending order (with distinct values). Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n 1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2]. Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums. You must write an algorithm with O(log n) runtime complexity.**

```
        def search(nums, target):

    left, right = 0, len(nums) - 1


    while left <= right:

        mid = (left + right) // 2


        if nums[mid] == target:

            return mid

        if nums[left] <= nums[mid]:

            if nums[left] <= target < nums[mid]:

                right = mid - 1

            else:
```

```
            left = mid + 1
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1

    return
nums = [4,5,6,7,0,1,2]
target = 0
result = search (nums, target)
print(f"Index of target {target} is: {result}")

target = 3
result = search(nums, target)
print(f"Index of target {target} is: {result}")
```

**Output:**

```
Index of target 0 is: 4
Index of target 3 is: -1
```

**5. . Find First and Last Position of Element in Sorted Array Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value. If target is not found in the array, return [-1, -1]. You must write an algorithm with O(log n) runtime complexity.**

```
  def findFirstPosition(nums, target):
    left, right = 0, len(nums) - 1
    first_position = -1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            first_position = mid
            right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return first_position

def findLastPosition(nums, target):
    left, right = 0, len(nums) - 1
    last position = -1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            last_position = mid
            left = mid + 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return last_position

def searchRange(nums, target):
    first_position = findFirstPosition(nums, target)
    last_position = findLastPosition(nums, target)

    return [first_position, last_position]
nums = [5, 7, 7, 8, 8, 10]
target = 8
```

```
result = searchRange(nums, target)

print(f"First and last positions of target {target} are: {result}")


target = 6

result = searchRange(nums, target)

print(f"First and last positions of target {target} are: {result}")
```

**Output:**

```
First and last positions of target 8 are: [3, 4]
First and last positions of target 6 are: [-1, -1]
```

**6. Sort Colors Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.**

```
     def sortColors(nums):

  low, mid, high = 0, 0, len(nums) - 1


  while mid <= high:

    if nums[mid] == 0:

      nums[low], nums[mid] = nums[mid], nums[low]

      low += 1

      mid += 1

    elif nums[mid] == 1:

      mid += 1

    else:

      nums[high], nums[mid] = nums[mid], nums[high]

      high -= 1

nums = [2, 0, 2, 1, 1, 0]

sortColors(nums)

print(f"Sorted colors: {nums}")


nums = [2, 0, 1]

sortColors(nums)
```

```
print(f"Sorted colors: {nums}")
```

**Output:**

```
Sorted colors: [0, 0, 1, 1, 2, 2]
Sorted colors: [0, 1, 2]
```

**7. Remove Duplicates from Sorted List Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.**

```
        class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def deleteDuplicates(head):
    current = head


    while current and current.next:
        if current.val == current.next.val:


            current.next = current.next.next
        else:


            current = current.next


    return head
def printList(node):
    while node:
        print(node.val, end=" -> ")
        node = node.next
    print("None")
head = ListNode(1, ListNode(1, ListNode(2, ListNode(3, ListNode(3)))))


print("Original list:")
```

```
printList(head)


head = deleteDuplicates(head)


print("List after removing duplicates:")

printList(head)
```

**<u>Output:</u>**

```
Original list:
1 -> 1 -> 2 -> 3 -> 3 -> None
List after removing duplicates:
1 -> 2 -> 3 -> None
```

**8. Merge Sorted Array You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively. Merge nums1 and nums2 into a single array sorted in non-decreasing order.**

```
     def merge(nums1, m, nums2, n):


  p1, p2 = m - 1, n - 1

    p = m + n - 1

  while p1 >= 0 and p2 >= 0:

    if nums1[p1] > nums2[p2]:

      nums1[p] = nums1[p1]

      p1 -= 1

    else:

      nums1[p] = nums2[p2]

      p2 -= 1

    p -= 1

  while p2 >= 0:

    nums1[p] = nums2[p2]

    p2 -= 1

    p -= 1

nums1 = [1, 2, 3, 0, 0, 0]

m = 3
```

```
nums2 = [2, 5, 6]
n = 3

merge(nums1, m, nums2, n)
print(f"Merged array: {nums1}")
```

**Output:**

```
Merged array: [1, 2, 2, 3, 5, 6]
```

**9. Convert Sorted Array to Binary Search Tree Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.**

```
    class TreeNode:
  def __init__(self, val=0, left=None, right=None):
    self.val = val
    self.left = left
    self.right = right

def sortedArrayToBST(nums):
  if not nums:
    return None
  mid = len(nums) // 2

  root = TreeNode(nums[mid])
  root.left = sortedArrayToBST(nums[:mid])

  root.right = sortedArrayToBST(nums[mid+1:])

  return root
def preOrder(node):
  if not node:
    return
    print(node.val, end=" ")

    preOrder(node.left)

    preOrder(node.right)

nums = [-10, -3, 0, 5, 9]

root = sortedArrayToBST(nums)


print("Pre-order traversal of the constructed BST:")

preOrder(root)
```

**Output:**

```
Pre-order traversal of the constructed BST:
0 -3 -10 9 5
```

**10. Insertion Sort List Given the head of a singly linked list, sort the list using insertion sort, and return the sorted list's head. The steps of the insertion sort algorithm: 1. Insertion sort iterates, consuming one input element each repetition and growing a sorted output list. 2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there. 3. It repeats until no input elements remain. The following is a graphical example of the insertion sort algorithm. The partially sorted list (black) initially contains only the first element in the list. One element (red) is removed from the input data and inserted in-place into the sorted list with each iteration.**

```
            class ListNode:

    def __init__(self, val=0, next=None):

        self.val = val

        self.next = next


def insertionSortList(head):

    if not head or not head.next:

        return head

    dummy = ListNode(0)

    dummy.next = head

    curr = head

    prev = dummy


    while curr:
        if curr.next and curr.next.val < curr.val:
            while prev.next and prev.next.val < curr.next.val:
                prev = prev.next

            temp = prev.next
            prev.next = curr.next
            curr.next = curr.next.next
            prev.next.next = temp

                prev = dummy
        else:
            curr = curr.next

    return dummy.next
def printList(node):
    while node:
        print(node.val, end=" -> ")
        node = node.next
    print("None")
head = ListNode(4, ListNode(2, ListNode(1, ListNode(3))))

print("Original list:")
printList(head)

sorted_head = insertionSortList(head)

print("Sorted list:")
printList(sorted_head)
```

**Output:**

```
Original list:
```

```
4 -> 2 -> 1 -> 3 -> None
Sorted list:
1 -> 2 -> 3 -> 4 -> None
```