Day 5 lab programs

2. 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.

Sol:- from queue import PriorityQueue

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def mergeKLists(lists):

dummy = ListNode(0)

curr = dummy

q = PriorityQueue()

for l in lists:

if l:

q.put((l.val, l))

while not q.empty():

val, node = q.get()

curr.next = ListNode(val)

curr = curr.next

node = node.next

if node:

q.put((node.val, node))

return dummy.next

3. Given an integer array nums sorted in non-decreasing order, remove the duplicates inplace such that each unique element appears only once. The relative order of the elements should be kept the same.

Sol:- def removeDuplicates(nums):

if not nums:

return 0

k = 1

for i in range(1, len(nums)):

if nums[i] != nums[i - 1]:

nums[k] = nums[i]

k += 1

return k

4. Search in Rotated Sorted Array

Sol:- def search(nums, target):

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

while left <= right:

mid = left + (right - left) // 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 -1

nums = [4, 5, 6, 7, 0, 1, 2]

target = 0

print(search(nums, target))

5. Find First and Last Position of Element in Sorted Array

Sol:-

class Solution:

def searchRange(self, nums, target):

def binarySearchLeft(nums, target):

left, right = 0, len(nums)

while left < right:

mid = left + (right - left) // 2

if nums[mid] < target:

left = mid + 1

else:

right = mid

return left

def binarySearchRight(nums, target):

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.

Sol:-

def sortColors(nums):

red, white, blue = 0, 0, len(nums) - 1

while white <= blue:

if nums[white] == 0:

nums[red], nums[white] = nums[white], nums[red]

red += 1

white += 1

elif nums[white] == 1:

white += 1

else:

nums[white], nums[blue] = nums[blue], nums[white]

blue -= 1

left, right = 0, len(nums)

while left < right:

mid = left + (right - left) // 2

if nums[mid] <= target:

left = mid + 1

else:

right = mid

return left

left\_idx = binarySearchLeft(nums, target)

right\_idx = binarySearchRight(nums, target

if left\_idx <= right\_idx:

return [left\_idx, right\_idx]

else:

return [-1, -1]

nums = [5, 7, 7, 8, 8, 10]

target = 8

solution = Solution()

print(solution.searchRange(nums, target))

7. Remove Duplicates from Sorted List

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

8. Merge Sorted Array

Sol:- def merge\_sorted\_arrays(nums1, m, nums2, n):

nums1[m:] = nums2

nums1.sort()

return nums1

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

m = 3

nums2 = [2, 5, 6]

n = 3

result = merge\_sorted\_arrays(nums1, m, nums2, n)

print(result)

9. Convert Sorted Array to 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

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.

Sol:- def improvedInsertionSortList(head):

if not head or not head.next:

return head

dummy = ListNode(0)

dummy.next = head

last\_sorted = head

current = head.next

while current:

if last\_sorted.val <= current.val:

last\_sorted = last\_sorted.next

else:

prev = dummy

while prev.next.val <= current.val:

prev = prev.next

last\_sorted.next = current.next

current.next = prev.next

prev.next = current

current = last\_sorted.next

return dummy.next