**Assignment 19 - Searching and Sorting**

**Question:1. 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.\*

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]]

Output: [1,1,2,3,4,4,5,6]

Explanation: The linked-lists are:

[

1->4->5,

1->3->4,

2->6

]

merging them into one sorted list:

1->1->2->3->4->4->5->6

Example 2:

Input: lists = []

Output: []

Example 3:

Input: lists = [[]]

Output: []

Constraints:

- `k == lists.length`

- `0 <= k <= 10000`

- `0 <= lists[i].length <= 500`

- `-10000 <= lists[i][j] <= 10000`

- `lists[i]` is sorted in \*\*ascending order\*\*.

- The sum of `lists[i].length` will not exceed `10000`.

**Ans:**

import heapq

class ListNode:

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

self.val = val

self.next = next

def mergeKLists(lists):

# Create a min-heap

min\_heap = []

# Insert the head nodes of all linked lists into the min-heap

for linked\_list in lists:

if linked\_list:

heapq.heappush(min\_heap, (linked\_list.val, linked\_list))

# Create a dummy node as the head of the merged linked list

dummy = ListNode(0)

curr = dummy

# Merge the linked lists using the min-heap

while min\_heap:

# Remove the minimum element from the min-heap

val, node = heapq.heappop(min\_heap)

# Append the minimum element to the merged linked list

curr.next = node

curr = curr.next

# If the removed node has a next node, insert it into the min-heap

if node.next:

heapq.heappush(min\_heap, (node.next.val, node.next))

# Return the head of the merged linked list

return dummy.next  
  
# Example 1

lists = [[1,4,5],[1,3,4],[2,6]]

merged\_list = mergeKLists(lists)

# The expected output is [1,1,2,3,4,4,5,6]

# Example 2

lists = []

merged\_list = mergeKLists(lists)

# The expected output is []

# Example 3

lists = [[]]

merged\_list = mergeKLists(lists)

# The expected output is []

**Question:2. Count of Smaller Numbers After Self**

Given an integer array nums, return an integer array counts where counts[i] is the number of smaller elements to the right of nums[i].

Example 1:

Input: nums = [5,2,6,1]

Output: [2,1,1,0]

Explanation:

To the right of 5 there are2 smaller elements (2 and 1).

To the right of 2 there is only1 smaller element (1).

To the right of 6 there is1 smaller element (1).

To the right of 1 there is0 smaller element.

Example 2:

Input: nums = [-1]

Output: [0]

Example 3:

Input: nums = [-1,-1]

Output: [0,0]

Constraints:

- `1 <= nums.length <= 100000`

- `-10000 <= nums[i] <= 10000`

**Ans:**

def mergeAndCount(nums, start, mid, end, counts):

left = nums[start:mid+1]

right = nums[mid+1:end+1]

i = j = 0

k = start

count = 0

while i < len(left) and j < len(right):

if left[i] <= right[j]:

nums[k] = left[i]

i += 1

else:

nums[k] = right[j]

j += 1

count += len(left) - i

k += 1

while i < len(left):

nums[k] = left[i]

i += 1

k += 1

while j < len(right):

nums[k] = right[j]

j += 1

k += 1

for idx in range(start, end+1):

counts[idx] += count

return count

def mergeSortAndCount(nums, start, end, counts):

if start == end:

return 0

mid = (start + end) // 2

left\_count = mergeSortAndCount(nums, start, mid, counts)

right\_count = mergeSortAndCount(nums, mid + 1, end, counts)

merge\_count = mergeAndCount(nums, start, mid, end, counts)

return left\_count + right\_count + merge\_count

def countSmaller(nums):

counts = [0] \* len(nums)

mergeSortAndCount(nums, 0, len(nums) - 1, counts)

return counts  
  
nums = [5, 2, 6, 1]

print(countSmaller(nums)) # Output: [2, 1, 1, 0]

nums = [-1]

print(countSmaller(nums)) # Output: [0]

nums = [-1, -1]

print(countSmaller(nums)) # Output: [0, 0]

**Question:3. Sort an Array**

Given an array of integers `nums`, sort the array in ascending order and return it.

You must solve the problem \*\*without using any built-in\*\* functions in `O(nlog(n))` time complexity and with the smallest space complexity possible.

Example 1:

Input: nums = [5,2,3,1]

Output: [1,2,3,5]

Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

Example 2:

Input: nums = [5,1,1,2,0,0]

Output: [0,0,1,1,2,5]

Explanation: Note that the values of nums are not necessairly unique.

Constraints:

- `1 <= nums.length <= 5 \* 10000`

- `-5 \* 104 <= nums[i] <= 5 \* 10000`

**Ans:**

def merge\_sort(nums):

if len(nums) <= 1:

return nums

mid = len(nums) // 2

left = nums[:mid]

right = nums[mid:]

left = merge\_sort(left)

right = merge\_sort(right)

return merge(left, right)

def merge(left, right):

merged = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] <= right[j]:

merged.append(left[i])

i += 1

else:

merged.append(right[j])

j += 1

while i < len(left):

merged.append(left[i])

i += 1

while j < len(right):

merged.append(right[j])

j += 1

return merged

# Test the implementation with given examples

nums1 = [5, 2, 3, 1]

sorted\_nums1 = merge\_sort(nums1)

print(sorted\_nums1) # Output: [1, 2, 3, 5]

nums2 = [5, 1, 1, 2, 0, 0]

sorted\_nums2 = merge\_sort(nums2)

print(sorted\_nums2) # Output: [0, 0, 1, 1, 2, 5]

**Question:4. Move all zeroes to end of array**

Given an array of random numbers, Push all the zero’s of a given array to the end of the array. For example, if the given arrays is {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0}, it should be changed to {1, 9, 8, 4, 2, 7, 6, 0, 0, 0, 0}. The order of all other elements should be same. Expected time complexity is O(n) and extra space is O(1).

Example:

Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0};

Output : arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

Input : arr[] = {1, 2, 0, 0, 0, 3, 6};

Output : arr[] = {1, 2, 3, 6, 0, 0, 0};

**Ans:**

def move\_zeroes\_to\_end(arr):

left = 0

right = len(arr) - 1

while left < right:

if arr[left] == 0:

arr[left], arr[right] = arr[right], arr[left]

right -= 1

else:

left += 1

return arr

arr = [1, 2, 0, 4, 3, 0, 5, 0]

print(move\_zeroes\_to\_end(arr))  
  
[1, 2, 4, 3, 5, 0, 0, 0]  
  
arr = [1, 2, 0, 0, 0, 3, 6]

print(move\_zeroes\_to\_end(arr))  
  
arr = [1, 2, 0, 0, 0, 3, 6]

print(move\_zeroes\_to\_end(arr))

**Question:5. Rearrange array in alternating positive & negative items with O(1) extra space**

Given an array of positive and negative numbers, arrange them in an alternate fashion such that every positive number is followed by a negative and vice-versa maintaining the order of appearance. The number of positive and negative numbers need not be equal. If there are more positive numbers they appear at the end of the array. If there are more negative numbers, they too appear at the end of the array.

Examples:

Input:  arr[] = {1, 2, 3, -4, -1, 4}

Output: arr[] = {-4, 1, -1, 2, 3, 4}

Input:  arr[] = {-5, -2, 5, 2, 4, 7, 1, 8, 0, -8}

Output: arr[] = {-5, 5, -2, 2, -8, 4, 7, 1, 8, 0}

**Ans:**

def rearrange\_array(arr):

n = len(arr)

posPtr = 0

negPtr = 0

while posPtr < n and negPtr < n:

while posPtr < n and arr[posPtr] >= 0:

posPtr += 1

while negPtr < n and arr[negPtr] < 0:

negPtr += 1

if posPtr < n and negPtr < n:

arr[posPtr], arr[negPtr] = arr[negPtr], arr[posPtr]

posPtr += 1

negPtr += 1

return arr

arr1 = [1, 2, 3, -4, -1, 4]

print(rearrange\_array(arr1)) # Output: [-4, 1, -1, 2, 3, 4]

arr2 = [-5, -2, 5, 2, 4, 7, 1, 8, 0, -8]

print(rearrange\_array(arr2)) # Output: [-5, 5, -2, 2, -8, 4, 7, 1, 8, 0]

**Question:6. Merge two sorted arrays**

Given two sorted arrays, the task is to merge them in a sorted manner.

**Examples:**

Input: arr1[] = { 1, 3, 4, 5}, arr2[] = {2, 4, 6, 8}

Output: arr3[] = {1, 2, 3, 4, 4, 5, 6, 8}

Input: arr1[] = { 5, 8, 9}, arr2[] = {4, 7, 8}

Output: arr3[] = {4, 5, 7, 8, 8, 9}

**Ans:**

def merge\_sorted\_arrays(arr1, arr2):

n1 = len(arr1)

n2 = len(arr2)

arr3 = [0] \* (n1 + n2)

i = j = k = 0

while i < n1 and j < n2:

if arr1[i] <= arr2[j]:

arr3[k] = arr1[i]

i += 1

else:

arr3[k] = arr2[j]

j += 1

k += 1

while i < n1:

arr3[k] = arr1[i]

i += 1

k += 1

while j < n2:

arr3[k] = arr2[j]

j += 1

k += 1

return arr3

# Example usage:

arr1 = [1, 3, 4, 5]

arr2 = [2, 4, 6, 8]

merged\_array = merge\_sorted\_arrays(arr1, arr2)

print(merged\_array)

[1, 2, 3, 4, 4, 5, 6, 8]

arr1 = [5, 8, 9]

arr2 = [4, 7, 8]

merged\_array = merge\_sorted\_arrays(arr1, arr2)

print(merged\_array)

[4, 5, 7, 8, 8, 9]

**Question:7. Intersection of Two Arrays**

Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must be unique and you may return the result in any order.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Explanation: [4,9] is also accepted.

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`

- `0 <= nums1[i], nums2[i] <= 1000`

**Ans:**

def intersection(nums1, nums2):

# Create a hash set to store unique elements

set1 = set(nums1)

# Create a hash set to store the intersection

intersection\_set = set()

# Iterate over the second array

for num in nums2:

# Check if the element exists in the first array's hash set

if num in set1:

# Add the element to the intersection hash set

intersection\_set.add(num)

# Convert the intersection set to a list and return

return list(intersection\_set)

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersection(nums1, nums2))

# Output: [2]

nums1 = [4, 9, 5]

nums2 = [9, 4, 9, 8, 4]

print(intersection(nums1, nums2))

# Output: [9, 4]

**Question:8. Intersection of Two Arrays II**

Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2,2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [4,9]

Explanation: [9,4] is also accepted.

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`

- `0 <= nums1[i], nums2[i] <= 1000`

**Ans:**

from collections import defaultdict

def intersect(nums1, nums2):

freq\_map = defaultdict(int)

for num in nums1:

freq\_map[num] += 1

result = []

for num in nums2:

if freq\_map[num] > 0:

result.append(num)

freq\_map[num] -= 1

return result

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersect(nums1, nums2)) # Output: [2, 2]

nums1 = [4, 9, 5]

nums2 = [9, 4, 9, 8, 4]

print(intersect(nums1, nums2)) # Output: [4, 9]