**Assignment Questions 19**

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.

**CODE:**

**Import heapq**

**class** ListNode:

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

self**.**val **=** val

self**.**next **=** next

**def** mergeKLists(lists):

*# Create a min heap*

minHeap **=** []

*# Push the head nodes of all the linked lists into the min heap*

**for** head **in** lists:

**if** head:

heapq**.**heappush(minHeap, (head**.**val, head))

*# Create a dummy node to serve as the head of the merged list*

dummy **=** ListNode(0)

current **=** dummy

*# Merge the linked lists using the min heap*

**while** minHeap:

\_, node **=** heapq**.**heappop(minHeap)

current**.**next **=** node

current **=** current**.**next

**if** node**.**next:

heapq**.**heappush(minHeap, (node**.**next**.**val, node**.**next))

**return** dummy**.**next

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

**CODE:**

**class** Solution:

**def** countSmaller(self, nums):

**def** mergeSort(arr):

**if** len(arr) **<=** 1:

**return** arr

mid **=** len(arr) **//** 2

left **=** mergeSort(arr[:mid])

right **=** mergeSort(arr[mid:])

**return** merge(left, right)

**def** merge(left, right):

merged **=** []

count **=** 0

i, j **=** 0, 0

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

**if** left[i][0] **>** right[j][0]:

merged**.**append(right[j])

count **+=** len(left) **-** i

j **+=** 1

**else**:

merged**.**append(left[i])

i **+=** 1

merged**.**extend(left[i:])

merged**.**extend(right[j:])

**return** merged

nums **=** [(num, i) **for** i, num **in** enumerate(nums)]

result **=** [0] **\*** len(nums)

mergeSort(nums)

**for** num, index **in** nums:

result[index] **=** num

**return** result

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

**CODE:**

**class** Solution:

**def** sortArray(self, nums):

**if** len(nums) **<=** 1:

**return** nums

mid **=** len(nums) **//** 2

left **=** self**.**sortArray(nums[:mid])

right **=** self**.**sortArray(nums[mid:])

**return** self**.**merge(left, right)

**def** merge(self, left, right):

merged **=** []

i, j **=** 0, 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

merged**.**extend(left[i:])

merged**.**extend(right[j:])

**return** merged

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};

**CODE:**

**def** moveZeroesToEnd(arr):

n **=** len(arr)

zero\_index **=** 0 *# Pointer to track the position to place the next non-zero element*

*# Traverse the array*

**for** i **in** range(n):

**if** arr[i] **!=** 0:

*# If the current element is non-zero, swap it with the position indicated by zero\_index*

arr[i], arr[zero\_index] **=** arr[zero\_index], arr[i]

zero\_index **+=** 1

**return** arr

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}

CODE:

**def** rearrangeAlternate(arr):

n **=** len(arr)

positive **=** 0 *# Pointer for positive numbers*

negative **=** 1 *# Pointer for negative numbers*

*# Find the first positive number at an odd index*

**while** positive **<** n **and** arr[positive] **>** 0:

positive **+=** 2

*# Find the first negative number at an even index*

**while** negative **<** n **and** arr[negative] **<** 0:

negative **+=** 2

*# Rearrange the array by swapping positive and negative numbers*

**while** positive **<** n **and** negative **<** n:

*# Swap positive number at odd index with negative number at even index*

arr[positive], arr[negative] **=** arr[negative], arr[positive]

*# Find the next positive number at an odd index*

**while** positive **<** n **and** arr[positive] **>** 0:

positive **+=** 2

*# Find the next negative number at an even index*

**while** negative **<** n **and** arr[negative] **<** 0:

negative **+=** 2

**return** arr

**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}

**CODE:**

**def** mergeSortedArrays(arr1, arr2):

n1 **=** len(arr1)

n2 **=** len(arr2)

merged **=** []

i **=** 0 *# Pointer for arr1*

j **=** 0 *# Pointer for arr2*

*# Merge the arrays while comparing elements*

**while** i **<** n1 **and** j **<** n2:

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

merged**.**append(arr1[i])

i **+=** 1

**else**:

merged**.**append(arr2[j])

j **+=** 1

*# Append the remaining elements of arr1, if any*

**while** i **<** n1:

merged**.**append(arr1[i])

i **+=** 1

*# Append the remaining elements of arr2, if any*

**while** j **<** n2:

merged**.**append(arr2[j])

j **+=** 1

**return** merged

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

**CODE:**

**def** intersection(nums1, nums2):

*# Create sets to store unique elements*

set1 **=** set(nums1)

set2 **=** set(nums2)

*# Find the intersection of the two sets*

intersect **=** set1**.**intersection(set2)

*# Convert the intersection set to a list*

result **=** list(intersect)

**return** result

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

**CODE:**

**def** intersect(nums1, nums2):

*# Create a hashmap to store the count of each element in nums1*

hashmap **=** {}

**for** num **in** nums1:

hashmap[num] **=** hashmap**.**get(num, 0) **+** 1

*# Iterate over nums2 and check for common elements in the hashmap*

result **=** []

**for** num **in** nums2:

**if** num **in** hashmap **and** hashmap[num] **>** 0:

result**.**append(num)

hashmap[num] **-=** 1

**return** result