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.

Solution :

class ListNode:

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

self.val = val

self.next = next

def merge\_two\_lists(l1, l2):

dummy = ListNode()

current = dummy

while l1 and l2:

if l1.val <= l2.val:

current.next = l1

l1 = l1.next

else:

current.next = l2

l2 = l2.next

current = current.next

current.next = l1 if l1 else l2

return dummy.next

def merge\_k\_lists(lists):

if not lists:

return None

n = len(lists)

if n == 1:

return lists[0]

mid = n // 2

left\_lists = lists[:mid]

right\_lists = lists[mid:]

merged\_left = merge\_k\_lists(left\_lists)

merged\_right = merge\_k\_lists(right\_lists)

return merge\_two\_lists(merged\_left, merged\_right)

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].

Solution :

def countSmaller(nums):

def mergeSortWithCount(nums, start, end):

if start == end:

return [nums[start]], [0]

mid = (start + end) // 2

leftArr, leftCounts = mergeSortWithCount(nums, start, mid)

rightArr, rightCounts = mergeSortWithCount(nums, mid + 1, end)

mergedArr = []

smallerCounts = []

leftIndex, rightIndex = 0, 0

count = 0

while leftIndex < len(leftArr) and rightIndex < len(rightArr):

if leftArr[leftIndex] <= rightArr[rightIndex]:

mergedArr.append(leftArr[leftIndex])

smallerCounts.append(count)

leftIndex += 1

else:

mergedArr.append(rightArr[rightIndex])

smallerCounts.append(count + 1)

rightIndex += 1

count += 1

mergedArr.extend(leftArr[leftIndex:])

smallerCounts.extend([count] \* (len(leftArr) - leftIndex))

mergedArr.extend(rightArr[rightIndex:])

smallerCounts.extend([count

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.

Solution :

def sortArray(nums):

def mergeSort(nums, start, end):

if start == end:

return [nums[start]]

mid = (start + end) // 2

leftArr = mergeSort(nums, start, mid)

rightArr = mergeSort(nums, mid + 1, end)

mergedArr = []

leftIndex, rightIndex = 0, 0

while leftIndex < len(leftArr) and rightIndex < len(rightArr):

if leftArr[leftIndex] <= rightArr[rightIndex]:

mergedArr.append(leftArr[leftIndex])

leftIndex += 1

else:

mergedArr.append(rightArr[rightIndex])

rightIndex += 1

mergedArr.extend(leftArr[leftIndex:])

mergedArr.extend(rightArr[rightIndex:])

return mergedArr

return mergeSort(nums, 0, len(nums) - 1)

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).

Solution :

def pushZerosToEnd(nums):

left = 0

right = 0

while right < len(nums):

if nums[right] != 0:

nums[left], nums[right] = nums[right], nums[left]

left += 1

right += 1

return nums

**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.

Solution :

def rearrangeArray(nums):

left = 0

right = len(nums) - 1

# Move positive numbers to one side and negative numbers to the other side

while left <= right:

while left <= right and nums[left] > 0:

left += 1

while left <= right and nums[right] < 0:

right -= 1

if left <= right:

nums[left], nums[right] = nums[right], nums[left]

left += 1

right -= 1

# Determine the counts of positive and negative numbers

count\_positive = len(nums) - left

count\_negative = left

# Rearrange the remaining positive or negative numbers at the end of the array

if count\_positive > count\_negative:

left = 0

right = len(nums) - count\_positive - 1

else:

left = count\_negative

right = len(nums) - 1

while left <= right:

while left <= right and nums[left] < 0:

left += 1

while left <= right and nums[right] > 0:

right -= 1

if left <= right:

nums[left], nums[right] = nums[right], nums[left]

left += 1

right -= 1

return nums

**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**.

Solution :

def intersection(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

intersectionSet = set()

for num in set1:

if num in set2:

intersectionSet.add(num)

intersectionList = list(intersectionSet)

return intersectionList

**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**.

Solution :

def intersect(nums1, nums2):

countMap = {}

for num in nums1:

if num in countMap:

countMap[num] += 1

else:

countMap[num] = 1

intersection = []

for num in nums2:

if num in countMap and countMap[num] > 0:

intersection.append(num)

countMap[num] -= 1

return intersection