***WEEK–2***

**Problem 1. 3Sum**

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]

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

Explanation:

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.

nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.

nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.

The distinct triplets are [-1,0,1] and [-1,-1,2].

Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input: nums = [0,1,1]

Output: []

Explanation: The only possible triplet does not sum up to 0.

Example 3:

Input: nums = [0,0,0]

Output: [[0,0,0]]

Explanation: The only possible triplet sums up to 0.

Constraints:

* 3 <= nums.length <= 3000
* -105 <= nums[i] <= 105

**Solution.**

class Solution:

def threeSum(self, nums: List[int]) -> List[List[int]]:

nums, res, n =sorted(nums), [], len(nums)

for i in range(n-2):

if nums[i] > 0:

break

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

continue

hi = n - 1

lo = bisect.bisect\_left(nums, - nums[i] - nums[hi], i + 1, hi) - 1

lo += (lo == i)

while lo < hi:

sum = nums[i] + nums[lo] + nums[hi]

if sum == 0:

res += (nums[i], nums[lo], nums[hi]),

while lo < hi and nums[lo] == nums[lo+1]:

lo += 1

while lo < hi and nums[hi] == nums[hi-1]:

hi -= 1

lo += sum <= 0

hi -= sum >= 0

return res

**Problem 2. 3Sum Closest**

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

Input: nums = [-1,2,1,-4], target = 1

Output: 2

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

Example 2:

Input: nums = [0,0,0], target = 1

Output: 0

Constraints:

* 3 <= nums.length <= 1000
* -1000 <= nums[i] <= 1000
* -104 <= target <= 104

**Solution.**

class Solution:

def threeSumClosest(self, nums: List[int], target: int) -> int:

minimum = 10000000

nums.sort()

for i in range(len(nums)-2):

j = i+1

k = len(nums)-1

while j < k:

res = nums[i] + nums[j] + nums[k] - target

if abs(res) <= abs(minimum):

minimum = res

#movement of two pointers based on above result

if res == 0:

return target

elif res < 0:

j += 1

else:

k -= 1

return minimum + target

**Problem 3. Sort Colors**

Given an array nums with n objects colored red, white, or blue, sort them [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) 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.

Example 1:

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

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

Example 2:

Input: nums = [2,0,1]

Output: [0,1,2]

Constraints:

* n == nums.length
* 1 <= n <= 300
* nums[i] is either 0, 1, or 2

**Solution.**

class Solution:

def sortColors(self, nums: List[int]) -> None:

"""

Do not return anything, modify nums in-place instead.

"""

shift = 0

for i in range(len(nums)):

i += shift

if nums[i] == 0:

nums.insert(0, nums.pop(i))

if nums[i] == 2:

nums.append(nums.pop(i))

shift -= 1

**Problem 4. 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.

The final sorted array should not be returned by the function, but instead be *stored inside the array*nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

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

Explanation: The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Example 2:

Input: nums1 = [1], m = 1, nums2 = [], n = 0

Output: [1]

Explanation: The arrays we are merging are [1] and [].

The result of the merge is [1].

Example 3:

Input: nums1 = [0], m = 0, nums2 = [1], n = 1

Output: [1]

Explanation: The arrays we are merging are [] and [1].

The result of the merge is [1].

Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

Constraints:

* nums1.length == m + n
* nums2.length == n
* 0 <= m, n <= 200
* 1 <= m + n <= 200
* -109 <= nums1[i], nums2[j] <= 109

**Solution.**

class Solution:

def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:

for i in range(0,n):

nums1[m]=nums2[i]

m+=1

for i in range(0,len(nums1)):

for j in range(i+1,len(nums1)):

if(nums1[i]>nums1[j]):

nums1[i],nums1[j]=nums1[j],nums1[i]

return nums1

**Problem 5. 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.

Calendar

Description automatically generated

Example 1:

Diagram

Description automatically generated

Input: head = [4,2,1,3]

Output: [1,2,3,4]

Example 2:

Diagram

Description automatically generated

Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Constraints:

* The number of nodes in the list is in the range [1, 5000].
* -5000 <= Node.val <= 5000

**Solution.**

class Solution:

def insertionSortList(self, head: Optional[ListNode]) -> Optional[ListNode]:

temp = head

nums = []

while temp:

nums.append(temp.val)

temp = temp.next

for i in range(len(nums)):

key = nums[i]

j = i- 1

while j >= 0 and nums[j] > key:

nums[j+1] = nums[j]

j -=1

nums[j+1] = key

i = 0

temp = head

while temp:

temp.val = nums[i]

i += 1

temp = temp.next

return head

**Problem 6. Sort List**

Given the head of a linked list, return *the list after sorting it in ascending order*.

Example 1:



Input: head = [4,2,1,3]

Output: [1,2,3,4]

Example 2:

Diagram

Description automatically generated

Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Example 3:

Input: head = []

Output: []

Constraints:

* The number of nodes in the list is in the range [0, 5 \* 104].
* -105 <= Node.val <= 105

**Solution.**

class Solution:

def sortList(self, head: Optional[ListNode]) -> Optional[ListNode]:

l1=[]

s=head

k=s

while(head):

l1.append(head.val)

head=head.next

l1.sort()

i=0

while(s):

s.val=l1[i]

i+=1

s=s.next

return(k)

**Problem 7. Largest Number**

Given a list of non-negative integers nums, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

Example 1:

Input: nums = [10,2]

Output: "210"

Example 2:

Input: nums = [3,30,34,5,9]

Output: "9534330"

Constraints:

* 1 <= nums.length <= 100
* 0 <= nums[i] <= 109

**Solution.**

class Solution:

def largestNumber(self, nums: List[int]) -> str:

nums = [str(i) for i in nums]

nums = self.mergesort(nums)

return str(int("".join(nums)))

def compare(self, n1, n2):

return n1 + n2 > n2 + n1

def mergesort(self, nums):

length = len(nums)

if length > 1:

middle = length //2

left\_list = self.mergesort(nums[:middle])

right\_list = self.mergesort(nums[middle:])

nums = self.merge(left\_list,right\_list)

return nums

def merge(self,list1,list2):

result = []

l1 = deque(list1)

l2 = deque(list2)

while l1 and l2:

if self.compare(l1[0],l2[0]):

result.append(l1[0])

l1.popleft()

else:

result.append(l2[0])

l2.popleft()

result.extend(l1 or l2)

return result

**Problem 8. Median of Two Sorted Arrays**

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Example 1:

Input: nums1 = [1,3], nums2 = [2]

Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2.

Example 2:

Input: nums1 = [1,2], nums2 = [3,4]

Output: 2.50000

Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

* nums1.length == m
* nums2.length == n
* 0 <= m <= 1000
* 0 <= n <= 1000
* 1 <= m + n <= 2000
* -106 <= nums1[i], nums2[i] <= 106

**Solution.**

class Solution:

def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:

merged = self.merge(nums1, nums2)

if len(merged) % 2 == 1:

return merged[((len(merged)+1)//2)-1]

else:

lind = ((len(merged))//2)-1

rind = lind+1

return (merged[lind] + merged[rind]) / 2

def merge(self, arr1, arr2):

result = []

ind1 = ind2 = 0

while ind1 < len(arr1) and ind2 < len(arr2):

currentArr1Val = arr1[ind1]

currentArr2Val = arr2[ind2]

if currentArr1Val < currentArr2Val:

result.append(currentArr1Val)

ind1 += 1

else:

result.append(currentArr2Val)

ind2 += 1

while ind1 < len(arr1):

result.append(arr1[ind1])

ind1 += 1

while ind2 < len(arr2):

result.append(arr2[ind2])

ind2 += 1

return result

**Problem 9. 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.

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]

Example 3:

Input: nums = [], target = 0

Output: [-1,-1]

Constraints:

* 0 <= nums.length <= 105
* -109 <= nums[i] <= 109
* nums is a non-decreasing array.
* -109 <= target <= 109

**Solution.**

class Solution:

def searchRange(self, nums: List[int], target: int) -> List[int]:

l = 0

r = len(nums)-1

res = [-1,-1]

while l<=r:

mid = (l+r)//2

if nums[mid] == target:

res[0]=mid

r = mid -1

elif nums[mid] > target:

r = mid -1

elif nums[mid] < target:

l = mid +1

l = 0

r = len(nums)-1

while l<=r:

mid = (l+r)//2

if nums[mid] == target:

res[1] = mid

l = mid +1

elif nums[mid] > target:

r = mid -1

elif nums[mid] < target:

l = mid +1

return res

**Problem 10. Search Insert Position**

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5

Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2

Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7

Output: 4

Constraints:

* 1 <= nums.length <= 104
* -104 <= nums[i] <= 104
* nums contains distinct values sorted in ascending order.
* -104 <= target <= 104

**Solution.**

class Solution:

def searchInsert(self, nums: List[int], target: int) -> int:

i = 0

while i < len(nums):

if nums[i] >= target:

return i

i += 1

return i

**Problem 11. Sqrt(x)**

Given a non-negative integer x, compute and return *the square root of* x.

Since the return type is an integer, the decimal digits are truncated, and only the integer part of the result is returned.

Note: You are not allowed to use any built-in exponent function or operator, such as pow(x, 0.5) or x \*\* 0.5.

Example 1:

Input: x = 4

Output: 2

Example 2:

Input: x = 8

Output: 2

Explanation: The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

Constraints:

* 0 <= x <= 231 – 1

**Solution.**

class Solution:

def mySqrt(self, x: int) -> int:

y=0

while y\*y<=x:

y+=1

if y \* y > x:

y-=1

return y

**Problem 12. Two Sum II - Input Array Is Sorted**

Given a 1-indexed array of integers numbers that is already *sorted in non-decreasing order*, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length.

Return*the indices of the two numbers,*index1*and*index2*, added by one as an integer array*[index1, index2]*of length 2.*

The tests are generated such that there is exactly one solution. You may not use the same element twice.

Your solution must use only constant extra space.

Example 1:

Input: numbers = [2,7,11,15], target = 9

Output: [1,2]

Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].

Example 2:

Input: numbers = [2,3,4], target = 6

Output: [1,3]

Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].

Example 3:

Input: numbers = [-1,0], target = -1

Output: [1,2]

Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

Constraints:

* 2 <= numbers.length <= 3 \* 104
* -1000 <= numbers[i] <= 1000
* numbers is sorted in non-decreasing order.
* -1000 <= target <= 1000
* The tests are generated such that there is exactly one solution.

**Solution.**

class Solution:

def twoSum(self, numbers: List[int], target: int) -> List[int]:

low, high = 0, len(numbers) - 1

while low < high:

sum = numbers[low] + numbers[high]

if sum == target:

return [low+1, high+1]

if sum < target:

low += 1

elif sum > target:

high -= 1

return -1, -1

**Problem 13.  Missing Number**

Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

Example 1:

Input: nums = [3,0,1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0,1]

Output: 2

Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]

Output: 8

Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

Constraints:

* n == nums.length
* 1 <= n <= 104
* 0 <= nums[i] <= n
* All the numbers of nums are unique.

**Solution.**

class Solution:

def missingNumber(self, nums: List[int]) -> int:

nums = sorted(nums)

if nums[0] != 0:

return 0

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

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

return nums[i] + 1

return nums[-1] + 1