1).Given an integer array num sorted in non-decreasing order. You can perform the following operation any number of times: Choose two indices, i and j, where nums[i] < nums[j]. Then, remove the elements at indices i and j from nums. The remaining elements retain their original order, and the array is reindexed. Return the minimum length of nums after applying the operation zero or more times. Example 1: Input: nums = [1,2,3,4] Output: 0 Constraints: 1 <= nums.length <= 105 1 <= nums[i] <= 109 nums is sorted in non-decreasing order.

SOL:)

def min\_length\_after\_operations(nums):

n = len(nums)

if n % 2 == 0:

return 0

else:

return 1

nums = [1, 2, 3, 4]

print(min\_length\_after\_operations(nums)) # Output: 0

2.) Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree. Example 1: Input: nums = [-10,-3,0,5,9] Output: [0,-3,9,-10,null,5] Explanation: [0,-10,5,null,-3,null,9] is also accepted.

SOL:

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

nums = [-10, -3, 0, 5, 9]

result = sortedArrayToBST(nums)

3.) Given an array of string words, return all strings in words that is a substring of another word. You can return the answer in any order. A substring is a contiguous sequence of characters within a string Example 1: Input: words = ["mass","as","hero","superhero"] Output: ["as","hero"] Explanation: "as" is substring of "mass" and "hero" is substring of "superhero". ["hero","as"] is also a valid answer.

Sol:

def find\_substrings(words):

result = []

for i in range(len(words)):

for j in range(len(words)):

if i != j and words[i] in words[j]:

result.append(words[i])

break

return result

words = ["mass", "as", "hero", "superhero"]

print(find\_substrings(words)) # Output: ["as", "hero"]

4.) Given an integer array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3].... You may assume the input array always has a valid answer. Example 1: Input: nums = [1,5,1,1,6,4] Output: [1,6,1,5,1,4] Explanation: [1,4,1,5,1,6] is also accepted. Example 2: Input: nums = [1,3,2,2,3,1] Output: [2,3,1,3,1,2]

Sol:

def wiggleSort(nums):

nums.sort()

half = len(nums[::2])

nums[::2], nums[1::2] = nums[:half][::-1], nums[half:][::-1]

return nums

nums1 = [1, 5, 1, 1, 6, 4]

output1 = wiggleSort(nums1)

print(output1) # Output: [1, 6, 1, 5, 1, 4]

nums2 = [1, 3, 2, 2, 3, 1]

output2 = wiggleSort(nums2)

print(output2) # Output: [2, 3, 1, 3, 1, 2]

5.) Given an m x n binary matrix mat, return the distance of the nearest 0 for each cell. The distance between two adjacent cells is 1. Input: mat = [[0,0,0],[0,1,0],[0,0,0]] Output: [[0,0,0],[0,1,0],[0,0,0]] Input: mat = [[0,0,0],[0,1,0],[1,1,1]] Output: [[0,0,0],[0,1,0],[1,2,1]]

Sol:

from collections import deque

from typing import List

def updateMatrix(mat: List[List[int]]) -> List[List[int]]:

if not mat or not mat[0]:

return mat

m, n = len(mat), len(mat[0])

dist = [[float('inf')] \* n for \_ in range(m)]

queue = deque()

for i in range(m):

for j in range(n):

if mat[i][j] == 0:

dist[i][j] = 0

queue.append((i, j))

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

while queue:

x, y = queue.popleft()

for dx, dy in directions:

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < m and 0 <= new\_y < n:

if dist[new\_x][new\_y] > dist[x][y] + 1:

dist[new\_x][new\_y] = dist[x][y] + 1

queue.append((new\_x, new\_y))

return dist

# Example usage:

mat1 = [[0,0,0],[0,1,0],[0,0,0]]

mat2 = [[0,0,0],[0,1,0],[1,1,1]]

print(updateMatrix(mat1)) # Output: [[0,0,0],[0,1,0],[0,0,0]]

print(updateMatrix(mat2)) # Output: [[0,0,0],[0,1,0],[1,2,1]]