**1. Given a 2D integer array matrix, return the transpose of matrix. The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices. Example 1:**

**Input: matrix = [[1,2,3],[4,5,6],[7,8,9]] Output: [[1,4,7],[2,5,8],[3,6,9]] Example 2: Input: matrix = [[1,2,3],[4,5,6]] Output: [[1,4],[2,5],[3,6]]**

**Program:**

def transpose(matrix):

rows, cols = len(matrix), len(matrix[0])

transposed = [[] for \_ in range(cols)]

for c in range(cols):

new\_row = [] for r in range(rows):

new\_row.append(matrix[r][c]) transposed[c] = new\_row

return transposed

matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(transpose(matrix1))

matrix2 = [[1, 2, 3], [4, 5, 6]] print(transpose(matrix2))

**2. You are given two 0-indexed integer arrays nums1 and nums2, each of size n, and an integer diff. Find the number of pairs (i, j) such that: 0 <= i < j <= n - 1 and nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff. Return the number of pairs that satisfy the conditions. Example 1: Input: nums1 = [3,2,5], nums2 = [2,2,1], diff = 1 Output: 3 Explanation: There are 3 pairs that satisfy the conditions: 1. i = 0, j = 1: 3 - 2 <= 2 - 2 + 1. Since i < j and 1 <= 1, this pair satisfies the conditions. 2. i = 0, j = 2: 3 - 5 <= 2 - 1 + 1. Since i < j and -2 <= 2, this pair satisfies the conditions. 3. i = 1, j = 2: 2 - 5 <= 2 - 1 + 1. Since i < j and -3 <= 2, this pair satisfies the conditions. Therefore, we return 3.**

**Program:**

class FenwickTree:

def \_\_init\_\_(self, size):

self.size = size self.tree = [0] \* (size + 1) def update(self, index, value):

index += 1 while index <= self.size: self.tree[index] += value index += index & -index

def query(self, index):

index += 1 sum = 0 while index > 0:

sum += self.tree[index] index -= index & -index return sum

def range\_query(self, left, right):

if left > right: return 0 return self.query(right) - self.query(left - 1)

def count\_pairs(nums1, nums2, diff):

n = len(nums1) new\_nums = [nums1[i] - nums2[i] for i in range(n)]

sorted\_new\_nums = sorted(new\_nums) rank\_map = {value: idx for idx, value in enumerate(sorted\_new\_nums)}

fenwick\_tree = FenwickTree(n) count = 0

for num in new\_nums:

rank = rank\_map[num] count += fenwick\_tree.range\_query(0, rank\_map[num + diff]) fenwick\_tree.update(rank, 1)

return count

nums1 = [3, 2, 5] nums2 = [2, 2, 1]

diff = 1 print(count\_pairs(nums1, nums2, diff))

1. **Given an integer n, return the nth digit of the infinite integer sequence [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...]. Example 1: Input: n = 3 Output: 3 Example 2: Input: n = 11 Output: 0 Explanation: The 11th digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.**

**Program:**

def find\_nth\_digit(n): # Step 1: Identify the range digit\_length = 1 count = 9 start = 1

while n > digit\_length \* count:

* 1. -= digit\_length \* count digit\_length += 1 count \*= 10 start \*= 10

# Step 2: Identify the exact number start += (n - 1) // digit\_length s = str(start)

# Step 3: Identify the exact digit within the number return int(s[(n - 1) % digit\_length])

# Example usage print(find\_nth\_digit(3)) # Output: 3 print(find\_nth\_digit(11)) # Output: 0

1. **A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not. Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string. Example 1: Input: s = "YazaAay" Output: "aAa"**

**Explanation: "aAa" is a nice string because 'A/a' is the only letter of the alphabet in s, and both 'A' and 'a' appear. "aAa" is the longest nice substring**.

**Program:**

def is\_nice(s):

unique\_chars = set(s) for char in unique\_chars: if char.swapcase() not in unique\_chars:

return False

return True

def longest\_nice\_substring(s):

* 1. = len(s) max\_len = 0

result = ""

for i in range(n): for j in range(i + 1, n + 1):

substring = s[i:j] if is\_nice(substring):

if len(substring) > max\_len: max\_len = len(substring) result = substring

return result

# Example usage print(longest\_nice\_substring("YazaAay")) # Output: "aAa" print(longest\_nice\_substring("abABB")) # Output: "abABB"

**5. Given a sentence that consists of some words separated by a single space, and a searchWord, check if searchWord is a prefix of any word in sentence. Return the index of the word in sentence (1-indexed) where searchWord is a prefix of this word. If searchWord is a prefix of more than one word, return the index of the first word (minimum index). If there is no such word return - 1. A prefix of a string s is any leading contiguous substring of s. Example 1: Input: sentence = "i love eating burger", searchWord = "burg" Output: 4 Explanation: "burg" is prefix of "burger" which is the 4th word in the sentence.**

**Program:**

def is\_prefix(word, prefix):

return word.startswith(prefix)

def index\_of\_prefix(sentence, searchWord):

words = sentence.split()

for index, word in enumerate(words): if is\_prefix(word, searchWord):

return index + 1

return -1

# Example usage sentence = "i love eating burger" searchWord = "burg" print(index\_of\_prefix(sentence, searchWord)) # Output: 4

sentence2 = "this is a simple test"

searchWord2 = "simp" print(index\_of\_prefix(sentence2, searchWord2)) # Output: 4

sentence3 = "hello world"

searchWord3 = "no" print(index\_of\_prefix(sentence3, searchWord3)) # Output: -1

**6. You are given an integer array nums and two integers indexDiff and valueDiff.Find a pair of indices (i, j) such that: i != j, abs(i - j) <= indexDiff. abs(nums[i] - nums[j]) <= valueDiff, and Return true if such pair exists or false otherwise. Example 1: Input: nums = [1,2,3,1], indexDiff = 3, valueDiff = 0 Output: true Explanation: We can choose (i, j) = (0, 3). We satisfy the three conditions: i != j --> 0 != 3 abs(i - j) <= indexDiff --> abs(0 - 3) <= 3 abs(nums[i] - nums[j]) <= valueDiff --> abs(1 - 1) <= 0 Program:**

from sortedcontainers import SortedList

def contains\_nearby\_almost\_duplicate(nums, indexDiff, valueDiff): if indexDiff <= 0 or valueDiff < 0:

return False

sorted\_list = SortedList()

for i in range(len(nums)):

# Remove the element that's out of the window

if i > indexDiff:

sorted\_list.remove(nums[i - indexDiff - 1])

pos1 = SortedList.bisect\_left(sorted\_list, nums[i] - valueDiff) if pos1 < len(sorted\_list) and abs(sorted\_list[pos1] - nums[i]) <= valueDiff:

return True

# Add the current number to the sorted list sorted\_list.add(nums[i])

return False

# Example usage

nums = [1, 2, 3, 1]

indexDiff = 3 valueDiff = 0 print(contains\_nearby\_almost\_duplicate(nums, indexDiff, valueDiff)) # Output: true

**7. 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**

**Program:** def min\_length\_after\_removals(nums):

i = 0 j = len(nums) - 1 pairs = 0

while i < j: if nums[i] < nums[j]:

pairs += 1 i += 1 j -= 1 else: j -= 1

return len(nums) - 2 \* pairs

# Example usage nums1 = [1, 2, 3, 4] print(min\_length\_after\_removals(nums1)) # Output: 0

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

print(min\_length\_after\_removals(nums2)) # Output: 0

nums3 = [1, 2, 2, 2, 3, 3, 4]

print(min\_length\_after\_removals(nums3)) # Output: 1\

**8. 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:**

**Program :**

from typing import List, Optional

# Definition for a binary tree node. class TreeNode: def \_init\_(self, val=0, left=None, right=None):

self.val = val self.left = left self.right = right

def sortedArrayToBST(nums: List[int]) -> Optional[TreeNode]: if not nums: return None

# Find the middle index mid = len(nums) // 2

# Create the root node with the middle element root = TreeNode(nums[mid])

# Recursively build the left subtree with the left half of the array root.left = sortedArrayToBST(nums[:mid])

# Recursively build the right subtree with the right half of the array root.right = sortedArrayToBST(nums[mid+1:])

return root

# Function to print the tree in level-order to validate the structure def printLevelOrder(root: Optional[TreeNode]):

if not root: return "[]"

result = [] queue = [root]

while queue:

current = queue.pop(0)

if current:

result.append(current.val) queue.append(current.left) queue.append(current.right)

else:

result.append(None)

# Remove trailing None values while result and result[-1] is None:

result.pop()

return result

# Example usage

nums = [-10, -3, 0, 5, 9] bst\_root = sortedArrayToBST(nums) print(printLevelOrder(bst\_root)) # Output: [0, -3, 9, -10, None, 5]

**9. 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.**

**Program:** def find\_substrings(words):

result = []

for i, word in enumerate(words): for j, other in enumerate(words): if i != j and word in other: result.append(word)

break

return result

# Example usage words = ["mass", "as", "hero", "superhero"] print(find\_substrings(words)) # Output: ["as", "hero"]

**10. 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]**

**Program:**

def wiggleSort(nums):

nums.sort() n = len(nums)

# Find the middle index mid = (n + 1) // 2

# Split the array into two halves left = nums[:mid] right = nums[mid:]

# Reverse the halves to place larger elements in the second half left.reverse() right.reverse()

# Interleave the elements nums[::2] = left nums[1::2] = right

# Example usage nums1 = [1, 5, 1, 1, 6, 4]

wiggleSort(nums1)

print(nums1) # Output: [1, 6, 1, 5, 1, 4]

nums2 = [1, 3, 2, 2, 3, 1] wiggleSort(nums2)

print(nums2) # Output: [2, 3, 1, 3, 1, 2]