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 re indexed. 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_operations(nums):
    n = len(nums)
    max_pairs = n // 2
    min_length = n - 2 * max_pairs
    return min_length


nums = [1, 2, 3, 4]
print(min_length_after_operations(nums))
```

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:

program:-

```
def sub_str(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])
 return result
words=['has','as','deepika','deep']
print(sub_str(words))
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
 def helper(left, right):
 if left > right:
 return None

 mid = (left + right) // 2
 root = TreeNode(nums[mid])
 root.left = helper(left, mid - 1)
 root.right = helper(mid + 1, right)
 return root
 return helper(0, len(nums) - 1)
```

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.

Program:-

```
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"]
```

```
output = find_substrings(words)
print(output)
```

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].

Program:-

```
def wiggleSort(nums):
    nums.sort()
    half = len(nums[::2])
    nums[::2], nums[1::2] = nums[:half][::-1], nums[half:][::-1]

nums1 = [1, 5, 1, 1, 6, 4]
wiggleSort(nums1)
print(nums1)
nums2 = [1, 3, 2, 2, 3, 1]
wiggleSort(nums2)
print(nums2)
```

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]]

Program :-

```
def updateMatrix(mat):
    m, n = len(mat), len(mat[0])
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    queue = []
    dist = [[float('inf')] * n for _ in range(m)]

    for i in range(m):
        for j in range(n):
            if mat[i][j] == 0:
                queue.append((i, j))
                dist[i][j] = 0

    index = 0
    while index < len(queue):
        x, y = queue[index]
        index += 1
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < m and 0 <= ny < n:
                if dist[nx][ny] > dist[x][y] + 1:
                    dist[nx][ny] = dist[x][y] + 1
                    queue.append((nx, ny))

    return dist

mat1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
print(updateMatrix(mat1))

mat2 = [[0, 0, 0], [0, 1, 0], [1, 1, 1]]
print(updateMatrix(mat2))
```

6. 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. 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

Program :-

```
import heapq

class ListNode:
    def _init_(self, val=0, next=None):
        self.val = val
        self.next = next

    def _repr_(self):
        return f"{self.val}->{self.next}"

def merge_k_lists(lists):
    heap = []
    for i in range(len(lists)):
        if lists[i]:
            heapq.heappush(heap, (lists[i].val, i, lists[i]))

    dummy = ListNode()
    current = dummy

    while heap:
        val, i, node = heapq.heappop(heap)
        current.next = ListNode(val)
        current = current.next
        if node.next:
            heapq.heappush(heap, (node.next.val, i, node.next))

    return dummy.next

def array_to_linked_list(arr):
    if not arr:
        return None
    head = ListNode(arr[0])
    current = head
    for value in arr[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


def linked_list_to_array(node):
    arr = []
    while node:
        arr.append(node.val)
        node = node.next
    return arr


lists = [[1,4,5],[1,3,4],[2,6]]
linked_lists = [array_to_linked_list(lst) for lst in lists]
merged_list = merge_k_lists(linked_lists)
output = linked_list_to_array(merged_list)
print(output)
```

7. Given two integer arrays arr1 and arr2, return the minimum number of operations (possibly zero) needed to make arr1 strictly increasing. In one operation, you can choose two indices 0 <= i < arr1.length and 0 <= j < arr2.length and do the assignment arr1[i] = arr2[j]. If there is no way to make arr1 strictly increasing, return -1. Example 1: Input: arr1 = [1,5,3,6,7], arr2 = [1,3,2,4] Output: 1 Explanation: Replace 5 with 2, then arr1 = [1, 2, 3, 6, 7].

Program :-

```
def makeArrayIncreasing(arr1, arr2):
    arr2 = sorted(set(arr2))


    dp = {-1: 0}


    for num in arr1:
        temp = {}
        for key in dp:
            if num > key:
                temp[num] = min(temp.get(num, float('inf')), dp[key])
            idx = binary_search(arr2, key)
            if idx < len(arr2):
                temp[arr2[idx]] = min(temp.get(arr2[idx], float('inf')), dp[key] + 1)

        if not temp:
            return -1
        dp = temp

    return min(dp.values())

def binary_search(arr, x):
    low, high = 0, len(arr)
    while low < high:
        mid = (low + high) // 2
        if arr[mid] <= x:
            low = mid + 1
        else:
            high = mid
    return low

arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
print(makeArrayIncreasing(arr1, arr2))
```