Name:-M . Charan
Reg No:- 192324115
Sub Code:-CSA0676

OBSERVATION DAY - 3

1.YOU ARE GIVEN A STRING S, AND AN ARRAY OF PAIRS OF INDICES IN THE STRING PAIRS WHERE PAIRS[I] = [A, B] INDICATES 2 INDICES(0-INDEXED) OF THE STRING.YOU CAN SWAP THE CHARACTERS AT ANY PAIR OF INDICES IN THE GIVEN PAIRS ANY NUMBER OF TIMES. RETURN THE LEXICOGRAPHICALLY SMALLEST STRING THAT S CAN BE CHANGED TO AFTER USING THE SWAPS.

```
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        rootX = self.find(x)
        rootY = self.find(y)
        if rootX != rootY:
            self.parent[rootY] = rootX

def smallestStringWithSwaps(s, pairs):
    n = len(s)
    uf = UnionFind(n)

    for a, b in pairs:
        uf.union(a, b)

    from collections import defaultdict
    components = defaultdict(list)

    for i in range(n):
        root = uf.find(i)
        components[root].append(i)

    res = list(s)

    for comp in components.values():
        indices = sorted(comp)
        chars = sorted(res[i] for i in indices)

        for i, char in zip(indices, chars):
            res[i] = char

    return ''.join(res)

s = "dcab"
pairs = [[0, 3], [1, 2]]
print(smallestStringWithSwaps(s, pairs))
```

# OUTPUT: "bacd"

2.GIVEN TWO STRINGS: S1 AND S2 WITH THE SAME SIZE, CHECK IF SOME PERMUTATION OF STRING S1 CAN BREAK SOME PERMUTATION OF STRING S2 OR VICE-VERSA. IN OTHER WORDS S2 CAN BREAK S1 OR VICE-VERSA. A STRING X CAN BREAK STRING Y (BOTH OF SIZE N) IF X[I] >= Y[I] (IN ALPHABETICAL ORDER) FOR ALL I BETWEEN 0 AND N-1.

```
def checkIfCanBreak(s1, s2):
    s1_sorted = sorted(s1)
    s2_sorted = sorted(s2)

    def can_break(x, y):
        return all(x[i] >= y[i] for i in range(len(x)))

    return can_break(s1_sorted, s2_sorted) or can_break(s2_sorted, s1_sorted)
```

```
s1 = "abc"

s2 = "xya"

print(checkIfCanBreak(s1, s2))

# OUTPUT: True
```

3.YOU ARE GIVEN A STRING S. S[I] IS EITHER A LOWERCASE ENGLISH LETTER OR '?'. FOR A STRING T HAVING LENGTH M CONTAINING ONLY LOWERCASE ENGLISH LETTERS, WE DEFINE THE FUNCTION COST(I) FOR AN INDEX I AS THE NUMBER OF CHARACTERS EQUAL TO T[I] THAT APPEARED BEFORE IT, I.E. IN THE RANGE [0, I - 1]. THE VALUE OF T IS THE SUM OF COST(I) FOR ALL INDICES I. FOR EXAMPLE, FOR THE STRING T = "AAB":

COST(0) = 0

COST(1) = 1

COST(2) = 0

HENCE, THE VALUE OF "AAB" IS 0 + 1 + 0 = 1. YOUR TASK IS TO REPLACE ALL OCCURRENCES OF '?' IN S WITH ANY LOWERCASE ENGLISH LETTER SO AT THE VALUE OF S IS MINIMIZED.

```
def minimizeCost(s):

    res = []

    last_seen = {}

    alphabet = 'abcdefghijklmnopqrstuvwxyz'


    for char in s:

        if char == '?':

            min_cost_char = None

            min_cost = float('inf')


            for letter in alphabet:

                cost = last_seen.get(letter, 0)

                if cost < min_cost:

                    min_cost = cost

                    min_cost_char = letter


            res.append(min_cost_char)

            last_seen[min_cost_char] = last_seen.get(min_cost_char, 0) + 1
        else:

            res.append(char)

            last_seen[char] = last_seen.get(char, 0) + 1


    return ''.join(res)


s = "a?b?c?"

print(minimizeCost(s))
```

 # OUTPUT: "aabaca"

4.YOU ARE GIVEN A STRING S. CONSIDER PERFORMING THE FOLLOWING OPERATION UNTIL S BECOMES EMPTY: FOR EVERY ALPHABET CHARACTER FROM 'A' TO 'Z', REMOVE THE FIRST OCCURRENCE OF THAT CHARACTER IN S (IF IT EXISTS). FOR EXAMPLE, LET INITIALLY S = "AABCBBCA". WE DO THE FOLLOWING OPERATIONS: REMOVE THE UNDERLINED CHARACTERS S = "AABCBBCA". THE RESULTING STRING IS S = "ABBCA". REMOVE THE UNDERLINED CHARACTERS S = "ABBCA". THE RESULTING STRING IS S = "BA". REMOVE THE UNDERLINED CHARACTERS S = "BA". THE RESULTING STRING IS S = "". RETURN THE VALUE OF THE STRING S RIGHT BEFORE APPLYING THE LAST OPERATION. IN THE EXAMPLE ABOVE, ANSWER IS "BA".

```
def lastStringBeforeEmptying(s):

    while True:

        new_s = list(s)

        for c in 'abcdefghijklmnopqrstuvwxyz':

            if c in new_s:

                new_s.remove(c)

        new_s = ''.join(new_s)

        if new_s == s:

            return s

        s = new_s


s = "aabcbbca"

print(lastStringBeforeEmptying(s))
```

# OUTPUT: "ba"

5.GIVEN AN INTEGER ARRAY NUMS, FIND THE SUBARRAY WITH THE LARGEST SUM, AND RETURN ITS SUM.

EXAMPLE 1:

INPUT: NUMS = [-2,1,-3,4,-1,2,1,-5,4]

OUTPUT: 6

```
def maxSubArray(nums):
    max_current = max_global = nums[0]

    for num in nums[1:]:
        max_current = max(num, max_current + num)
        if max_current > max_global:
            max_global = max_current

    return max_global

# Example usage:
nums = [-2,1,-3,4,-1,2,1,-5,4]
print(maxSubArray(nums))
# OUTPUT: 6
```

6.YOU ARE GIVEN AN INTEGER ARRAY NUMS WITH NO DUPLICATES. A MAXIMUM BINARY TREE CAN BE BUILT RECURSIVELY FROM NUMS USING THE FOLLOWING ALGORITHM: CREATE A ROOT NODE WHOSE VALUE IS THE MAXIMUM VALUE IN NUMS. RECURSIVELY BUILD THE LEFT SUBTREE ON THE SUBARRAY PREFIX TO THE LEFT OF THE MAXIMUM VALUE. RECURSIVELY BUILD THE RIGHT SUBTREE ON THE SUBARRAY SUFFIX TO THE RIGHT OF THE MAXIMUM VALUE. RETURN THE MAXIMUM BINARY TREE BUILT FROM NUMS.

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def constructMaximumBinaryTree(nums):
    if not nums:
        return None
    max_val = max(nums)
    max_index = nums.index(max_val)

    root = TreeNode(max_val)
    root.left = constructMaximumBinaryTree(nums[:max_index])
    root.right = constructMaximumBinaryTree(nums[max_index + 1:])

    return root

nums = [3,2,1,6,0,5]
root = constructMaximumBinaryTree(nums)
```

7.GIVEN A CIRCULAR INTEGER ARRAY NUMS OF LENGTH N, RETURN THE MAXIMUM POSSIBLE SUM OF A NON-EMPTY SUBARRAY OF NUMS.A CIRCULAR ARRAY MEANS THE END OF THE ARRAY CONNECTS TO THE BEGINNING OF THE ARRAY. FORMALLY, THE NEXT ELEMENT OF NUMS[I] IS NUMS[(I + 1) % N] AND THE PREVIOUS ELEMENT OF NUMS[I] IS NUMS[(I - 1 + N) % N].A SUBARRAY MAY ONLY INCLUDE EACH ELEMENT OF THE FIXED BUFFER NUMS AT MOST ONCE. FORMALLY, FOR A SUBARRAY NUMS[I], NUMS[I + 1], ..., NUMS[J], THERE DOES NOT EXIST I <= K1, K2 <= J WITH K1 % N == K2 % N.

```
def maxSubArray(nums):
    max_current = max_global = nums[0]
    for num in nums[1:]:
        max_current = max(num, max_current + num)
        if max_current > max_global:
            max_global = max_current
    return max_global

def maxSubarraySumCircular(nums):
    total_sum = sum(nums)
    max_kadane = maxSubArray(nums)
    min_kadane = -maxSubArray([-num for num in nums])

    if min_kadane == total_sum:
        return max_kadane

    return max(max_kadane, total_sum + min_kadane)

nums = [1, -2, 3, -2]
print(maxSubarraySumCircular(nums))
```

# OUTPUT: 3

8.YOU ARE GIVEN AN ARRAY NUMS CONSISTING OF INTEGERS. YOU ARE ALSO GIVEN A 2D ARRAY QUERIES, WHERE QUERIES[I] = [POSI, XI].FOR QUERY I, WE FIRST SET NUMS[POSI] EQUAL TO XI, THEN WE CALCULATE THE ANSWER TO QUERY I WHICH IS THE MAXIMUM SUM OF A SUBSEQUENCE OF NUMS WHERE NO TWO ADJACENT ELEMENTS ARE SELECTED. RETURN THE SUM OF THE ANSWERS TO ALL QUERIES. SINCE THE FINAL ANSWER MAY BE VERY LARGE, RETURN IT MODULO 109 + 7. A SUBSEQUENCE IS AN ARRAY THAT CAN BE DERIVED FROM ANOTHER ARRAY BY DELETING SOME OR NO ELEMENTS WITHOUT CHANGING THE ORDER OF THE REMAINING ELEMENTS.

```
def maxNonAdjacentSum(nums):
    include, exclude = 0, 0
    for num in nums:
        new_exclude = max(include, exclude)
        include = exclude + num
        exclude = new_exclude
    return max(include, exclude)

def processQueries(nums, queries):
    MOD = 10**9 + 7
    total_sum = 0

    for pos, val in queries:
        nums[pos] = val
        total_sum = (total_sum + maxNonAdjacentSum(nums)) % MOD

    return total_sum

nums = [1, 2, 3, 4]
```

```
queries = [[1, 3], [2, 4]]

print(processQueries(nums, queries))
```

# OUTPUT: 7

9. GIVEN AN ARRAY OF POINTS WHERE POINTS[I] = [XI, YI] REPRESENTS A POINT ON THE X-Y PLANE AND AN INTEGER K, RETURN THE K CLOSEST POINTS TO THE ORIGIN (0, 0).THE DISTANCE BETWEEN TWO POINTS ON THE X-Y PLANE IS THE EUCLIDEAN DISTANCE (I.E., √(X1 - X2)2 + (Y1 - Y2)2). YOU MAY RETURN THE ANSWER IN ANY ORDER. THE ANSWER IS GUARANTEED TO BE UNIQUE (EXCEPT FOR THE ORDER THAT IT IS IN).

```
import heapq


def kClosest(points, k):

    max_heap = []

    for x, y in points:

        dist = -(x * x + y * y)

        if len(max_heap) < k:

            heapq.heappush(max_heap, (dist, x, y))

        else:

            heapq.heappushpop(max_heap, (dist, x, y))

    return [(x, y) for _, x, y in max_heap]


points = [[1, 3], [-2, 2], [2, -2]]

k = 2

print(kClosest(points, k))
```

# OUTPUT: [[-2, 2], [2, -2]]

10. 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)).

```
def findMedianSortedArrays(nums1, nums2):

    if len(nums1) > len(nums2):

        nums1, nums2 = nums2, nums1


    m, n = len(nums1), len(nums2)

    imin, imax, half_len = 0, m, (m + n + 1) // 2
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j-1]:
            imin = i + 1
        elif i > 0 and nums1[i-1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = nums2[j-1]
            elif j == 0: max_of_left = nums1[i-1]
            else: max_of_left = max(nums1[i-1], nums2[j-1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m: min_of_right = nums2[j]
            elif j == n: min_of_right = nums1[i]
            else: min_of_right = min(nums1[i], nums2[j])
            return (max_of_left + min_of_right) / 2.0


nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
# OUTPUT: 2.0
```