LAB DAY-3(06-06-24)

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.

PROGRAM:

```
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [1] * 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:
            if self.rank[rootX] > self.rank[rootY]:
                self.parent[rootY] = rootX
            elif self.rank[rootX] < self.rank[rootY]:
                self.parent[rootX] = rootY
            else:
                self.parent[rootY] = rootX
                self.rank[rootX] += 1

def smallestStringWithSwaps(s, pairs):
    n = len(s)
    uf = UnionFind(n)
    for a, b in pairs:
        uf.union(a, b)

    from collections import defaultdict
    groups = defaultdict(list)
    for i in range(n):
        root = uf.find(i)
        groups[root].append(s[i])

    for group in groups.values():
        group.sort(reverse=True)

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

    return ''.join(res)

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

OUTPUT:

```
 "abcd"
```

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.

PROGRAM:

```
def Break(s1, s2):
    s1, s2 = sorted(s1), sorted(s2)
    s1s2 = all(c1 >= c2 for c1, c2 in zip(s1, s2))
    s2s1 = all(c2 >= c1 for c1, c2 in zip(s1, s2))
    return s1s2 or s2s1


s1 = "abc"
s2 = "xya"
print(Break(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.

PROGRAM:

```
def minimizeCost(s):
    from collections import Counter
    count = Counter()
    result = []

    for ch in s:
        if ch == '?':
            for c in 'abcdefghijklmnopqrstuvwxyz':
                if count[c] == 0:
                    result.append(c)
                    count[c] += 1
                    break
        else:
            result.append(ch)
            count[ch] += 1

    return ''.join(result)


s = "a?b"
print(minimizeCost(s))
```

OUTPUT:

```
abb
```

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".

PROGRAM:

```
def valueBeforeLastOperation(s):
    import collections


    while True:
        seen = set()
        new_s = []
        for ch in s:
            if ch not in seen:
                seen.add(ch)
            else:
                new_s.append(ch)
        if len(seen) == 0:
            return ''.join(new_s)
        s = ''.join(new_s)
```

```
s = "aabcbbca"
print(valueBeforeLastOperation(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

Explanation: The subarray [4,-1,2,1] has the largest sum 6.

PROGRAM:

```
def max (n):
 i= n [0]
 j= n[0]
 for x in n[1:]:
       j= max(x, j+ x)
       i= max(i,j)
 return i
num = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(max (num))
```

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.

PROGRAM:

```
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.

PROGRAM:

```
def maxSubarraySumCircular(nums):
    def kadane(arr):
        max_so_far = arr[0]
        max_ending_here = arr[0]
        for x in arr[1:]:
            max_ending_here = max(x, max_ending_here + x)
            max_so_far = max(max_so_far, max_ending_here)
        return max_so_far

    max_kadane = kadane(nums)
    max_wrap = sum(nums) - kadane([-x for x in nums])

    return max(max_kadane, max_wrap) if max_wrap != 0 else max_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.

PROGRAM:

```
def maxSumAfterQueries(nums, queries):
    MOD = 10**9 + 7

    def max_sum_no_adj(nums):
        incl, excl = 0, 0
        for num in nums:
            new_excl = max(incl, excl)
            incl = excl + num
            excl = new_excl
        return max(incl, excl)

    total_sum = 0
    for pos, x in queries:
        nums[pos] = x
        total_sum += max_sum_no_adj(nums)
        total_sum %= MOD

    return total_sum

nums = [1, 2, 3, 4]
queries = [[0, 2], [1, 2]]
print(maxSumAfterQueries(nums, queries))
```

Output:

```
9
```

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).

PROGRAM:

```
import heapq

def kClosest(points, k):
    heap = []
    for (x, y) in points:
        dist = x*x + y*y
        heapq.heappush(heap, (dist, x, y))

    return [(x, y) for (dist, x, y) in heapq.nsmallest(k, heap)]

points = [[1,3],[-2,2]]
k = 1
print(kClosest(points, k))
```

OUTPUT:

```
[[-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)).

PROGRAM:

```
def median(sarr):

   n = len(sarr)

   if n % 2 == 1:

       return sarr[n // 2]

   else:

       middle1 = sarr[n // 2 - 1]

       middle2 = sarr[n // 2]

       return (middle1 + middle2) / 2


array = [5,1,6,7,9]

sarr=(sorted(array))

median =median(sarr)

print(f"The median is: {median}")
```

OUTPUT:

```
The median is: 6
```