Experiment 20-30

-Prisha D(192311018)

1. Lexicographically smallest string after swaps from collections import defaultdict

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

def union(parent, rank, x, y):
    rootX = find(parent, x)
    rootY = find(parent, y)
    if rootX != rootY:
        if rank[rootX] > rank[rootY]:
            parent[rootY] = rootX
        elif rank[rootX] < rank[rootY]:
            parent[rootX] = rootY
        else:
            parent[rootY] = rootX
            rank[rootX] += 1

def smallestStringWithSwaps(s, pairs):
    parent = list(range(len(s)))
    rank = [0] * len(s)

    for a, b in pairs:
        union(parent, rank, a, b)

    groups = defaultdict(list)
```

```python
    for i in range(len(s)):
        root = find(parent, i)
        groups[root].append(i)

    res = list(s)
    for group in groups.values():
        indices = sorted(group)
        chars = sorted(s[i] for i in indices)
        for i, char in zip(indices, chars):
            res[i] = char

    return ''.join(res)
```

2. Check if one string can break another

```python
def checkIfCanBreak(s1, s2):
    s1 = sorted(s1)
    s2 = sorted(s2)

    def canBreak(a, b):
        return all(x >= y for x, y in zip(a, b))

    return canBreak(s1, s2) or canBreak(s2, s1)
```

3. Minimize the value of a string with '?'

```
def minimizeStringValue(s):
    def calculate_value(t):
        value = 0
        counts = [0] * 26
        for char in t:
            index = ord(char) - ord('a')
            value += counts[index]
            counts[index] += 1
        return value

    s = list(s)
    for i in range(len(s)):
        if s[i] == '?':
            min_value = float('inf')
            best_char = ''
            for c in 'abcdefghijklmnopqrstuvwxyz':
                s[i] = c
                current_value = calculate_value(s)
                if current_value < min_value:
                    min_value = current_value
                    best_char = c
            s[i] = best_char

    return ''.join(s)
```

4. Last value of the string before emptying

```
def lastValueBeforeEmptying(s):
```

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

5. Subarray with the largest sum

```
def maxSubArray(nums):
    max_sum = current_sum = nums[0]
    for num in nums[1:]:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)
    return max_sum
```

6. Maximum binary tree

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

```python
        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
```

7. Maximum sum of circular subarray

```python
def maxSubarraySumCircular(nums):
    def kadane(gen):
        current_sum = max_sum = next(gen)
        for x in gen:
            current_sum = x + max(current_sum, 0)
            max_sum = max(max_sum, current_sum)
        return max_sum

    S = sum(nums)
    max_kadane = kadane(iter(nums))
    min_kadane = kadane(-x for x in nums)
    return max(max_kadane, S + min_kadane if S + min_kadane != 0 else float('-inf'))
```

8. Maximum sum of non-adjacent subsequence after queries

```
def maxSubsequenceSum(nums, queries):
    mod = 10**9 + 7

    def max_sum_no_adjacent(nums):
        include, exclude = 0, 0
        for num in nums:
            new_include = exclude + num
            exclude = max(exclude, include)
            include = new_include
        return max(include, exclude)

    answer = 0
    for pos, x in queries:
        nums[pos] = x
        answer = (answer + max_sum_no_adjacent(nums)) % mod

    return answer
```

9. k closest points to the origin

```
import heapq

def kClosest(points, k):
```

```
    return heapq.nsmallest(k, points, key=lambda point: point[0] ** 2 + point[1] ** 2)
```

10. Median of two sorted arrays

```
def findMedianSortedArrays(nums1, nums2):
    A, B = nums1, nums2
    m, n = len(A), len(B)
    if m > n:
        A, B, m, n = B, A, n, m
    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 A[i] < B[j - 1]:
            imin = i + 1
        elif i > 0 and A[i - 1] > B[j]:
            imax = i - 1
        else:
            if i == 0:
                max_of_left = B[j - 1]
            elif j == 0:
                max_of_left = A[i - 1]
            else:
                max_of_left = max(A[i - 1], B[j - 1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m:
                min_of_right = B[j]
```

```
elif j == n:
    min_of_right = A[i]
else:
    min_of_right = min(A[i], B[j])
return (max_of_left + min_of_right) / 2.0
```