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.

main.py Save Run

```python
import collections
def smallestStringWithSwaps(s, pairs):
    def find(x):
        if x != parent[x]:
            parent[x] = find(parent[x])
        return parent[x]
    def union(x, y):
        root_x, root_y = find(x), find(y)
        if rank[root_x] > rank[root_y]:
            parent[root_y] = root_x
        else:
            parent[root_x] = root_y
            if rank[root_x] == rank[root_y]:
                rank[root_y] += 1
    parent = list(range(len(s)))
    rank = [0] * len(s)
    for a, b in pairs:
        union(a, b)
    groups = collections.defaultdict(list)
    for i in range(len(s)):
        groups[find(i)].append(s[i])
    for group in groups:
        groups[group].sort(reverse=True)
    res = []
    for i in range(len(s)):
        res.append(groups[find(i)].pop())
```

Output

```
Output: bacd

=== Code Execution Successful ===
```

time complexities: O(N + M)

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.

main.py Save Run

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

    if all(s1_char >= s2_char for s1_char, s2_char in zip(s1_sorted, s2_sorted))
        or all(s2_char >= s1_char for s1_char, s2_char in zip(s1_sorted,
        s2_sorted)):
        return True
    else:
        return False


s1 = "abc"
s2 = "xya"

# Output
print(check_if_can_break(s1, s2))

```

Output Clear

```
True

=== Code Execution Successful ===
```

time complexities: O(nlogn).

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.

main.py

```python
from collections import Counter
def min_value_string(s):
    def cost(i, t):
        return sum(1 for j in range(i) if t[j] == t[i])
    def value(t):
        return sum(cost(i, t) for i in range(len(t)))
    letters = 'abcdefghijklmnopqrstuvwxyz'
    min_value = float('inf')
    result = ''
    for c in letters:
        t = s.replace('?', c)
        curr_value = value(t)
        if curr_value < min_value:
            min_value = curr_value
            result = t
    return result
s = "a?b?c?"
result = min_value_string(s)
print(result)
```

Output

```
adbdcd

=== Code Execution Successful ===
```

time complexities: O(N * 26)

3. 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".

main.py

```python
def last_remaining_string(s):
    for char in 'abcdefghijklmnopqrstuvwxyz':
        if char in s:
            s = s.replace(char, '', 1)
    return s

# Example
s = "aabcbbca"
result = last_remaining_string(s)
print(result)
```

Output

```
abbca

=== Code Execution Successful ===
```

time complexities: O(n)

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.

main.py

```python
def max_subarray_sum(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

# Example
nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(max_subarray_sum(nums))  # Output: 6
```

Output

```
6

=== Code Execution Successful ===
```

time complexities: O(n)

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.

main.py

```python
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)
def print_tree(node, level=0):
    if node:
        print_tree(node.right, level + 1)
        print('    ' * level + '->', node.val)
        print_tree(node.left, level + 1)
print_tree(root)
```

Output

```
    -> 5
        -> 0
-> 6
            -> 1
        -> 2
    -> 3

=== Code Execution Successful ===
```

time complexities: O(n^2)

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.

main.py

```
def maxSubarraySumCircular(nums):
    total_sum = max_sum = min_sum = max_temp = min_temp = nums[0]

    for num in nums[1:]:
        max_temp = max(num, max_temp + num)
        max_sum = max(max_sum, max_temp)

        min_temp = min(num, min_temp + num)
        min_sum = min(min_sum, min_temp)

        total_sum += num

    return max(max_sum, total_sum - min_sum) if max_sum > 0 else max_sum

# Example usage
nums = [1, -2, 3, -2]
output = maxSubarraySumCircular(nums)
print("Output:", output)
```

Output

```
Output: 3

=== Code Execution Successful ===
```

time complexities: O(n)

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.

main.py

```
def checkIfCanBreak(s1, s2):
    return all(ord(x) <= ord(y) for x, y in zip(sorted(s1), sorted(s2)))

s1 = "abc"
s2 = "xya"
print(checkIfCanBreak(s1, s2))
```

Output

```
True

=== Code Execution Successful ===
```

time complexities: O(Q * N).

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).

main.py

```python
import heapq

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

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

output = kClosest(points, k)
print("Output:", output)
```

Output

```
Output: [[-2, 2]]

=== Code Execution Successful ===
```

time complexities: O(N * log(k))

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)).

main.py

```python
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1 + nums2)
    n = len(nums)
    if n % 2 == 0:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2
    else:
        return nums[n // 2]

# Example usage
nums1 = [1, 3]
nums2 = [2]

output = findMedianSortedArrays(nums1, nums2)
print("Output:", output)
```

Output

```
Output: 2

=== Code Execution Successful ===
```

time complexities: O(log(min(m, n)))