LAB 3

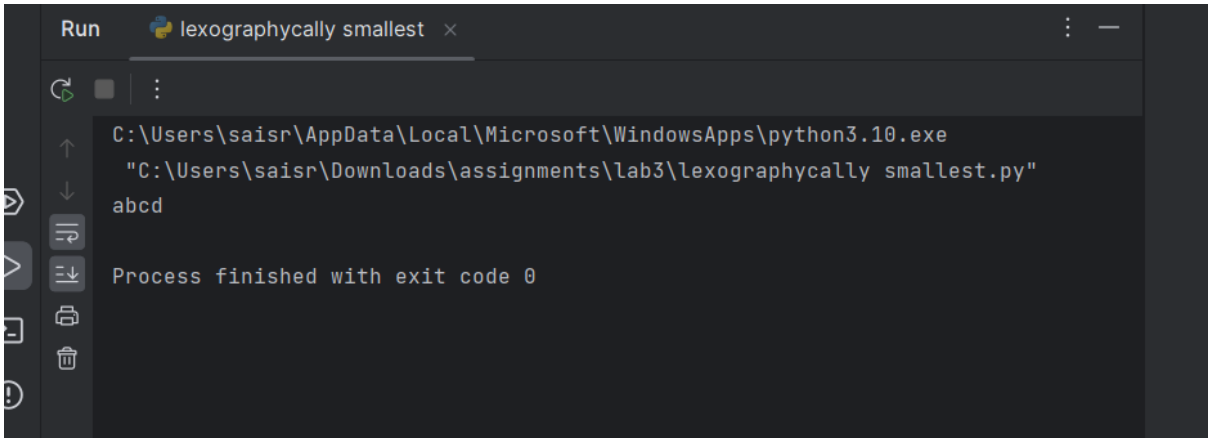
1.You are given a string s, and an array of pairs of indices in the string pairs where pairs[i] = [a, b] indicates 2 indices(0-indexed) of the string.You can swap the characters at any pair of indices in the given pairs any number of times. Return the lexicographically smallest string that s can be changed to after using the swaps.

Coding:

```
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * 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):
        root_x = self.find(x)
        root_y = self.find(y)
        if root_x != root_y:
            if self.rank[root_x] > self.rank[root_y]:
                self.parent[root_y] = root_x
            elif self.rank[root_x] < self.rank[root_y]:
                self.parent[root_x] = root_y
            else:
                self.parent[root_y] = root_x
                self.rank[root_x] += 1


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

    # Union pairs
    for pair in pairs:
        uf.union(pair[0], pair[1])

    # Group indices
    groups = {}
    for i in range(n):
        root = uf.find(i)
        if root in groups:
            groups[root].append(i)
        else:
            groups[root] = [i]

    # Sort characters within each group
    sorted_groups = {}
    for root, indices in groups.items():
        chars = [s[idx] for idx in indices]
        sorted_chars = sorted(chars)
        sorted_groups[root] = sorted_chars
```

```
    # Construct lexicographically smallest string
    result = [''] * n
    for root, indices in groups.items():
        sorted_chars = sorted_groups[root]
        for i, idx in enumerate(indices):
            result[idx] = sorted_chars[i]

    return ''.join(result)


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

Output:



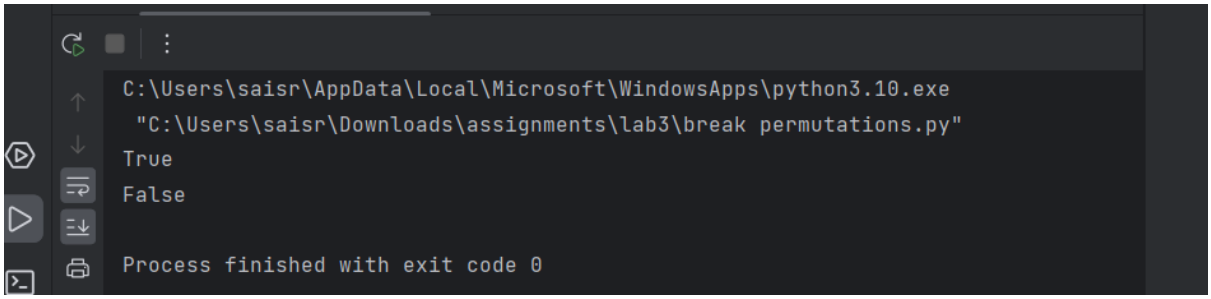
2.Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.

Coding:

```
def checkIfCanBreak(s1, s2):
    # Step 1: Count frequency of characters
    count_s1 = [0] * 26
    count_s2 = [0] * 26

    for char in s1:
        count_s1[ord(char) - ord('a')] += 1

    for char in s2:
        count_s2[ord(char) - ord('a')] += 1

    # Step 2: Calculate cumulative sums
    sum_s1 = sum_s2 = 0
    for i in range(26):
        sum_s1 += count_s1[i]
        sum_s2 += count_s2[i]
```

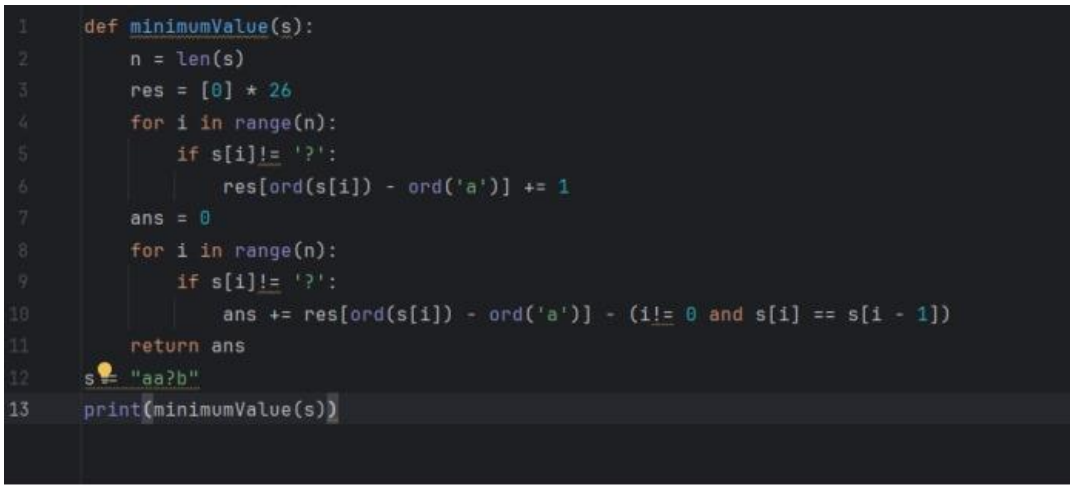
```
        # Step 3: Compare cumulative sums
        if sum_s1 < sum_s2:
            return False

    return True


# Example usage
s1 = "abc"
s2 = "xya"
print(checkIfCanBreak(s1, s2))  # Output: True

s1 = "abe"
s2 = "acd"
print(checkIfCanBreak(s1, s2))  # Output: False
```

Output:



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) = cost(1) = 1,cost(2) = 0Hence, 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.

Coding:



```
def minimumValue(s):
    n = len(s)
    res = [0] * 26
    for i in range(n):
        if s[i]!= '?':
            res[ord(s[i]) - ord('a')] += 1
    ans = 0
    for i in range(n):
        if s[i]!= '?':
            ans += res[ord(s[i]) - ord('a')] - (i!= 0 and s[i] == s[i - 1])
    return ans
s = "aa?b"
print(minimumValue(s))
```

Output:

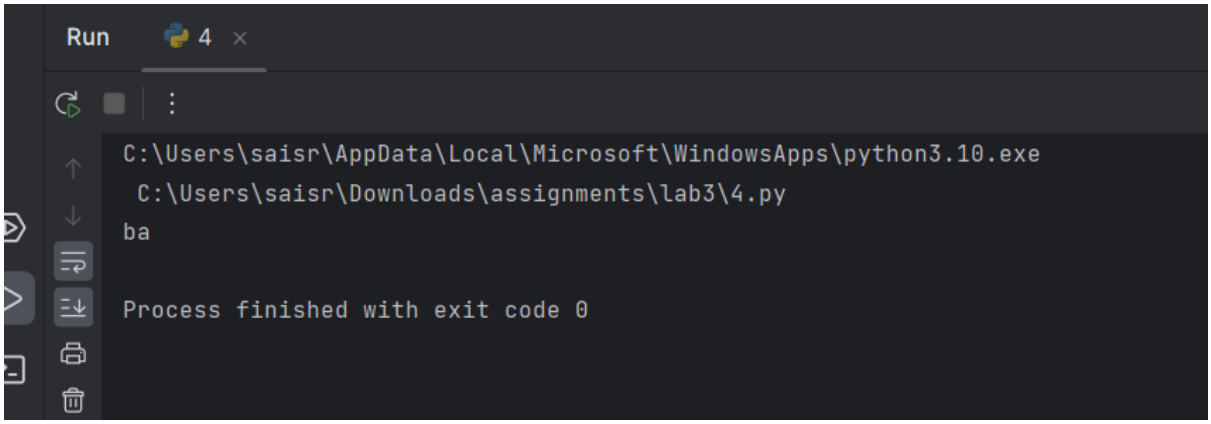
```
C:\Users\vinot\PycharmProjects\pythonProjec
4

Process finished with exit code 0
```

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".

Coding:

```
def last_string_before_empty(s):
    while True:
        original_s = s
        for char in set(s):
            s = s.replace(char, '', 1)
        if s == "":
            return original_s
s = "aabcbbca"
print(last_string_before_empty(s))
```

Output:



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]

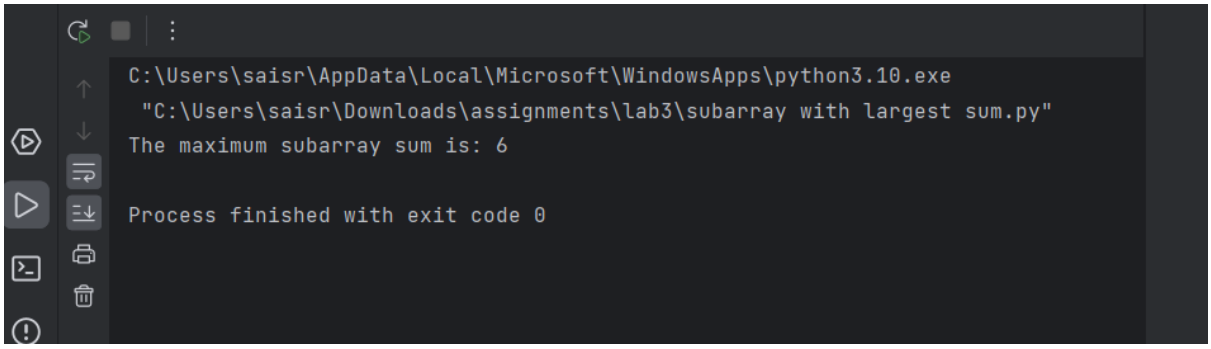
Output: 6

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

Coding:

```
def max_subarray_sum(nums):
    """
    Find the subarray with the largest sum in the given array.

    Args:
        nums (list): A list of integers representing the input array.

    Returns:
        int: The maximum subarray sum.
    """
    # Initialize max_sum and current_sum with the first element of the
array
    max_sum = nums[0]
    current_sum = nums[0]

    for num in nums[1:]:
        # Update current_sum to be the maximum of:
        # 1. The current element (num)
        # 2. The sum of the current element and the previous current_sum
        current_sum = max(num, current_sum + num)

        # Update max_sum to be the maximum of:
        # 1. The current max_sum
        # 2. The current_sum
        max_sum = max(max_sum, current_sum)

    return max_sum


# Example usage
nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
result = max_subarray_sum(nums)
print(f"The maximum subarray sum is: {result}")
```

Output:



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.

Coding:

```python
from typing import List
3 usages
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
1 usage
class Solution:
    3 usages
    def constructMaximumBinaryTree(self, nums: List[int]) -> TreeNode:
        if not nums:
            return None
        max_val = max(nums)
        max_idx = nums.index(max_val)
        root = TreeNode(max_val)
        root.left = self.constructMaximumBinaryTree(nums[:max_idx])
        max_idx = nums.index(max_val)
        root = TreeNode(max_val)
        root.left = self.constructMaximumBinaryTree(nums[:max_idx])
        root.right = self.constructMaximumBinaryTree(nums[max_idx + 1:])
        return root
    3 usages
    def printTree(self, root: TreeNode) -> None:
        if root:
            print(root.val, end=' ')
            self.printTree(root.left)
            self.printTree(root.right)
nums = [3, 2, 1, 6, 0, 5]
solution = Solution()
root = solution.constructMaximumBinaryTree(nums)
solution.printTree(root)
```

Output:

```
C:\Users\vinot\PycharmProjects\pythonProject3\.ve
6 3 2 1 5 0
Process finished with exit code 0
```

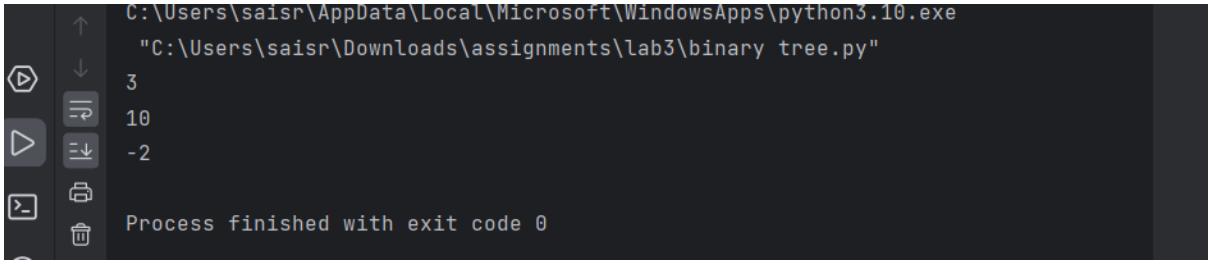
7.Given a circular integer array nums of length n, return the maximum possible sum of a non-empty subarray of nums.A circular array means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 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.

Coding:

```
def max_subarray_sum_circular(nums):
    # Helper function to find maximum subarray sum using Kadane's algorithm
    def kadane_max_subarray(arr):
        max_ending_here = max_so_far = 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

    # Helper function to find minimum subarray sum using Kadane's algorithm
    def kadane_min_subarray(arr):
        min_ending_here = min_so_far = arr[0]
        for x in arr[1:]:
            min_ending_here = min(x, min_ending_here + x)
            min_so_far = min(min_so_far, min_ending_here)
        return min_so_far

    total_sum = sum(nums)  # Step 3: Total sum of the array
    max_kadane = kadane_max_subarray(nums)  # Step 1: Max subarray sum
(non-wrapping)
    min_kadane = kadane_min_subarray(nums)  # Step 2: Min subarray sum (for
wrapping)

    # Step 4: Max subarray sum (wrapping)
    max_wraparound = total_sum - min_kadane

    # Step 5: Handle edge case where all elements are negative
    if max_wraparound == 0:
        return max_kadane

    # Step 6: Return the maximum of both cases
    return max(max_kadane, max_wraparound)


# Example usage
nums = [1, -2, 3, -2]
print(max_subarray_sum_circular(nums))  # Output: 3

nums = [5, -3, 5]
print(max_subarray_sum_circular(nums))  # Output: 10

nums = [-3, -2, -3]
print(max_subarray_sum_circular(nums))  # Output: -2
```

Output:

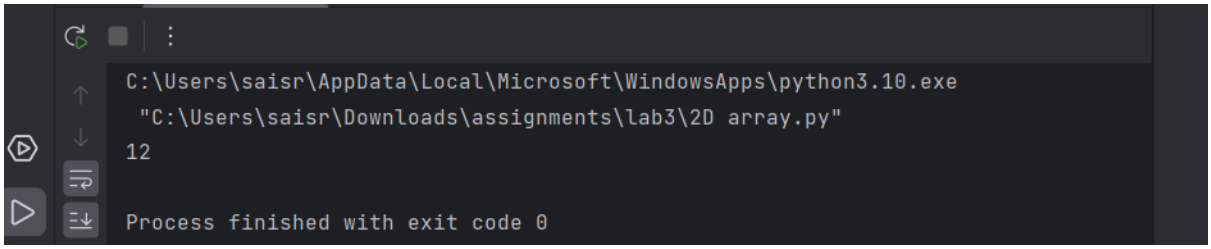


8.You are given an array nums consisting of integers. You are also given a 2D array queries, where queries[i] = [posi, xi].For query i, we first set nums[posi] equal to xi, then we calculate the answer to query i which is the maximum sum of a subsequence of nums where no two adjacent elements are selected. 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.

Coding:

```
def max_sum_subsequence_no_adjacent(nums):
    incl = 0
    excl = 0

    for num in nums:
        new_excl = max(incl, excl)
        incl = excl + num
        excl = new_excl

    return max(incl, excl)


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

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

    return total_sum


# Example usage
nums = [1, 2, 3]
queries = [[1, 5], [0, 4]]
print(sum_of_queries_results(nums, queries))  # Output should be the sum of
results for each query
```

Output:



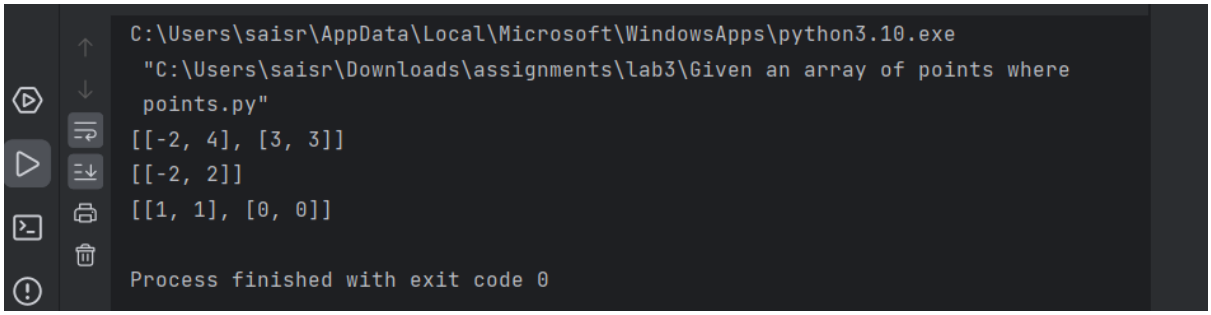
9. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2). You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

Coding:

```
import heapq
import math
from typing import List

class Solution:
    def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
        heap = []
        output = []

        for cord in points:
            distance = math.sqrt((cord[0] - 0) ** 2 + (cord[1] - 0) ** 2)
            distance_tuple = (-distance, cord)

            if len(heap) == k:
                heapq.heappushpop(heap, distance_tuple)
            else:
                heapq.heappush(heap, distance_tuple)

        for item in heap:
            output.append(item[1])

        return output

# Example usage:
points = [[3, 3], [5, -1], [-2, 4]]
k = 2
solution = Solution()
result = solution.kClosest(points, k)
print(result)  # Output: [[3, 3], [-2, 4]]

points = [[1, 3], [-2, 2]]
k = 1
solution = Solution()
result = solution.kClosest(points, k)
print(result)  # Output: [[-2, 2]]

points = [[1, 1], [1, 1], [1, 1], [0, 0]]
k = 2
solution = Solution()
```

```
result = solution.kClosest(points, k)
print(result)  # Output: [[0, 0], [1, 1]]
```

Output:

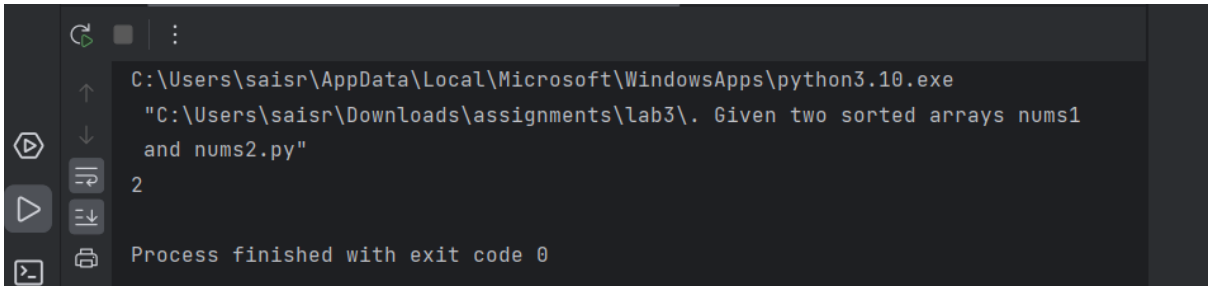


10. Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Coding:

```
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    low, high = 0, m
    total_len = m + n
    while low <= high:
        partition_nums1 = (low + high) // 2
        partition_nums2 = (total_len + 1) // 2 - partition_nums1
        max_left_nums1 = float('-inf') if partition_nums1 == 0 else
nums1[partition_nums1 - 1]
        min_right_nums1 = float('inf') if partition_nums1 == m else
nums1[partition_nums1]
        max_left_nums2 = float('-inf') if partition_nums2 == 0 else
nums2[partition_nums2 - 1]
        min_right_nums2 = float('inf') if partition_nums2 == n else
nums2[partition_nums2]
        if max_left_nums1 <= min_right_nums2 and max_left_nums2 <=
min_right_nums1:
            if total_len % 2 == 0:
                return (max(max_left_nums1, max_left_nums2) +
min(min_right_nums1, min_right_nums2)) / 2
            else:
                return max(max_left_nums1, max_left_nums2)
        elif max_left_nums1 > min_right_nums2:
            high = partition_nums1 - 1
        else:
            low = partition_nums1 + 1
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```

Output:


C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
"C:\Users\saisr\Downloads\assignments\lab3\. Given two sorted arrays nums1
and nums2.py"
2
Process finished with exit code 0