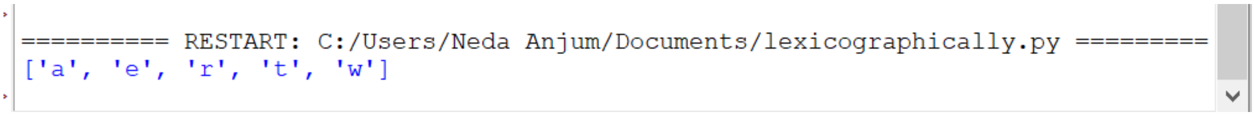
5-6-24

Lab3

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
   Code:

```
x="water"
sorted_x=sorted(x)
print(sorted_x)
```

output:



```
========== RESTART: C:/Users/Neda Anjum/Documents/lexicographically.py =========
['a', 'e', 'r', 't', 'w']
```

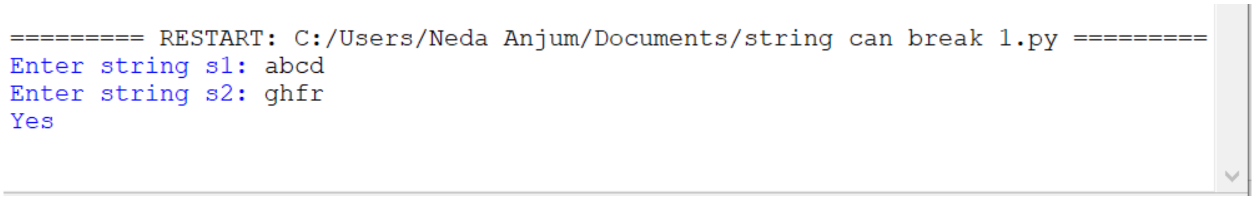
2. Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and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

Code:

```
def can_string_break(s1, s2):
    if len(s1) != len(s2):
        return False

    for i in range(len(s1)):
        if s1[i] < s2[i]:
            return False

    return True
s1 = input("Enter string s1: ")
s2 = input("Enter string s2: ")

if can_string_break(s1, s2) or can_string_break(s2, s1):
    print("Yes")
```

```
else:
    print("No")
```

output:



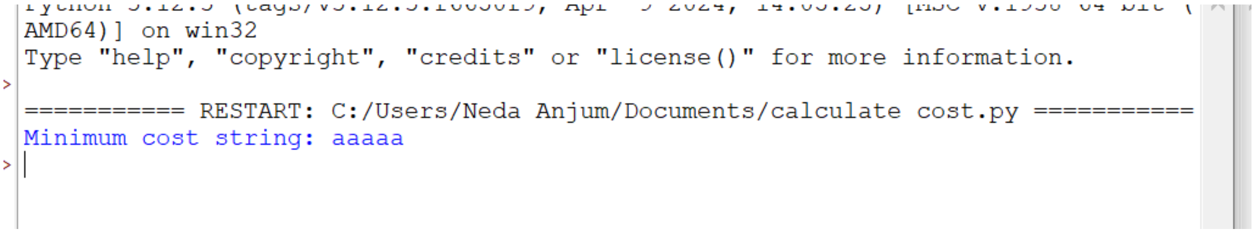
```
========= RESTART: C:/Users/Neda Anjum/Documents/string can break 1.py =========
Enter string s1: abcd
Enter string s2: ghfr
Yes
```

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.

```
def max_subarray_sum(a):
    max_sum = 0
    current_sum = 0

    for num in a:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)
        return max_sum  # Return the maximum sum found
a= [ 1, 4, 2, 1, 4]
print("Largest sum in a subarray:", max_subarray_sum(a))
def calculate_cost(s, t):
    total_cost = 0
    for i in range(len(t)):
        count = 0
        for j in range(i):
            if t[i] == t[j]:
                count += 1
        total_cost += count
    return total_cost
```

```
def find_minimum_cost_string(s):
    min_cost = float('inf')
    result = ""

    for i in range(26):
        char = chr(ord('a') + i)
        cost = calculate_cost(s, char * len(s))
        if cost < min_cost:
            min_cost = cost
            result = char * len(s)
 return result
s = "a?c?b"
result = find_minimum_cost_string(s)
print("Minimum cost string:", result)
```

output:



```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>
=========== RESTART: C:/Users/Neda Anjum/Documents/calculate cost.py ===========
Minimum cost string: aaaaa
>
```

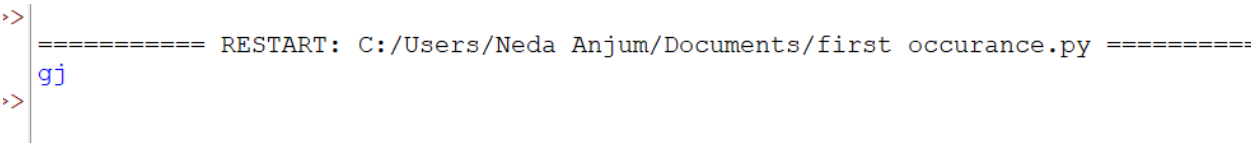
4. You are given a string s. Consider performing the following operation until s becomes empty: For every alphabet character from 'a' to 'z', remove the first occurrence of that character in s (if it exists). For example, let initially s = "aabcbbca". We do the following operations: Remove the underlined 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".

Code:

```
def rmv_fst_occurence(s):
    while s!=0:
```

```
    prev=s
    for char in 'abcdefghijklmnopqrstuvwxyz':
      if char in s:
        s=s.replace(char,'',1)
    if not s:
        return prev
 return ""
s="aajgggjjhshs"
print(rmv_fst_occurence(s))
```

output:

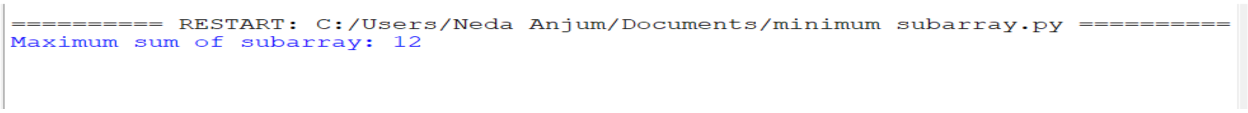


```
>
=========== RESTART: C:/Users/Neda Anjum/Documents/first occurance.py ==========
gj
>
```

5. Given an integer array nums, find the subarray with the largest sum, and return its sum.

Code:

```
def max_subarray_sum(nums):
  max_sum = 0
  current_sum =0
for num in nums:
    current_sum = max(num, current_sum + num)
    max_sum = max(max_sum, current_sum)


  return max_sum
nums = [ 1, 4,2, 1, 4]
result = max_subarray_sum(nums)
print("Maximum sum of subarray:", result)
```



```
========== RESTART: C:/Users/Neda Anjum/Documents/minimum subarray.py ==========
Maximum sum of subarray: 12
```

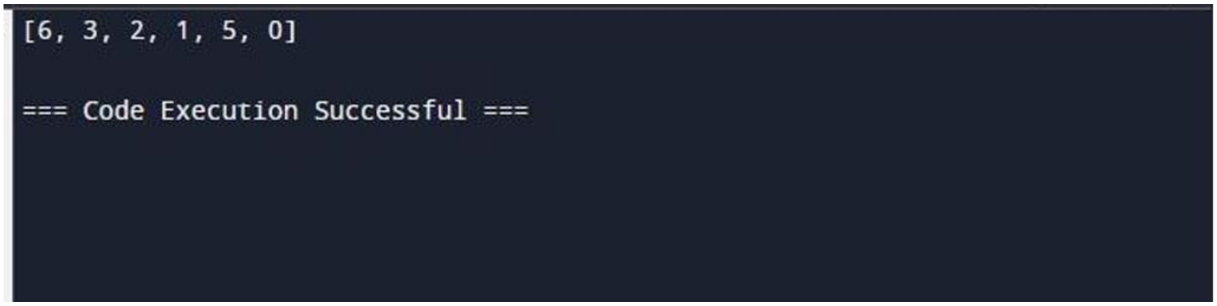
6. You are given an integer array nums with no duplicates. A maximum binary tree can be built recursively from nums using the following algorithm: Create a root node whose value is the maximum value in nums. Recursively build the left subtree on the subarray prefix to the left of the maximum value. Recursively build the right subtree on the subarray suffix to the right of the maximum value. Return the maximum binary tree built from nums.

Code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def constructMaxBinTree(nums):
    if not nums:
        return None
    max_index = nums.index(max(nums))
    root = TreeNode(nums[max_index])
    root.left = constructMaxBinTree(nums[:max_index])
    root.right = constructMaxBinTree(nums[max_index + 1:])
    return root
def preorderTraversal(root):
    if not root:
        return []
    return [root.val] + preorderTraversal(root.left) + preorderTraversal(root.right)
```

```
nums = [3, 2, 1, 6, 0, 5]

tree = constructMaxBinTree(nums)

print(preorderTraversal(tree))
```

Output:



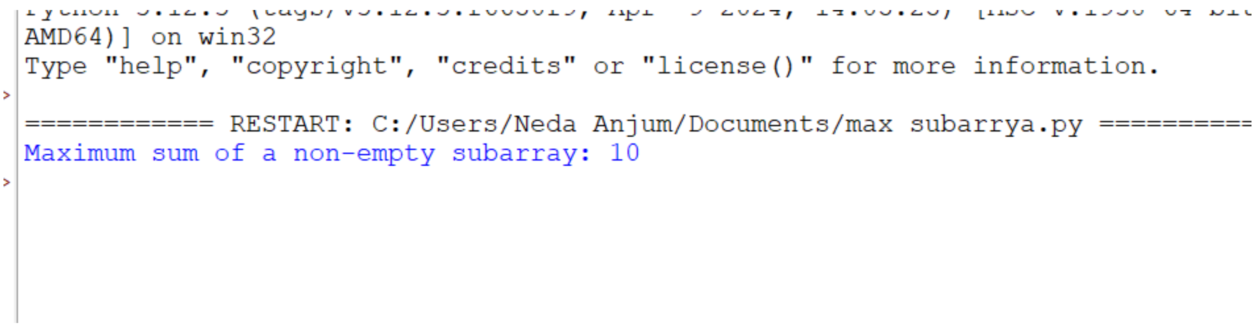
```
[6, 3, 2, 1, 5, 0]

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

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.

Code:

```
def max_subarray_sum(nums):

    max_sum = float('-inf')

    min_sum = float('inf')

    total_sum = 0

    curr_max = 0

    curr_min = 0

      for num in nums:

        curr_max = max(curr_max + num, num)

        max_sum = max(max_sum, curr_max)

         curr_min = min(curr_min + num, num)

        min_sum = min(min_sum, curr_min)

  total_sum += num

    if max_sum < 0:
        return max_sum

    return max(max_sum, total_sum - min_sum)

nums = [5, -3, 5]

result = max_subarray_sum(nums)

print("sum  no ofn-empty subarray:", result)
```

output:



```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>
============ RESTART: C:/Users/Neda Anjum/Documents/max subarrya.py ==========
Maximum sum of a non-empty subarray: 10
>
```

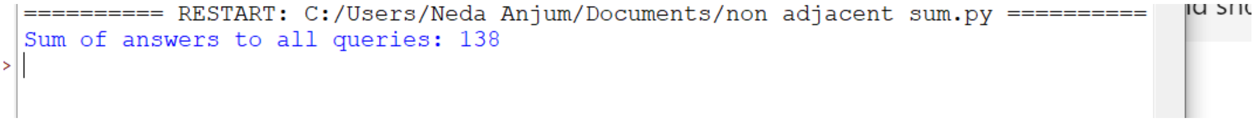
8. You are given an array nums consisting of integers. You are also given a 2D array queries, where queries[i] = [posi, xi].For query i, we first set nums[posi] equal to xi, then we calculate the answer to query i which is the maximum sum of a subsequence of nums where no two adjacent elements are selected.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

Code:

```
MOD = 10**9 + 7


def max_non_adjacent_sum(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 process_queries(nums, queries):

    total_sum = 0

       for pos, x in queries:

        nums[pos] = x

        total_sum = (total_sum + max_non_adjacent_sum(nums)) % MOD

      return total_sum

nums = [1, 2, 3, 4]

queries = [[0, 10], [1, 20], [2, 30], [3, 40]]

result = process_queries(nums, queries)

print("Sum of answers to all queries:", result)
```

output:

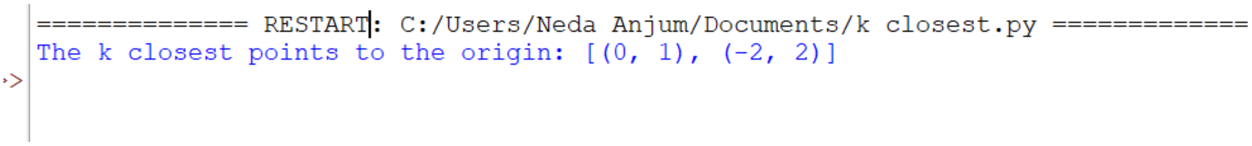


```
========== RESTART: C:/Users/Neda Anjum/Documents/non adjacent sum.py ==========
Sum of answers to all queries: 138
>
```

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 k_closest(points, k):

    heap = [(x*x + y*y, (x, y)) for x, y in points]

    heapq.heapify(heap)

        result = [heapq.heappop(heap)[1] for _ in range(k)]

    return result

points = [[1, 3], [-2, 2], [5, 8], [0, 1]]

k = 2

result = k_closest(points, k)

print("The k closest points to the origin:", result)
```

output:

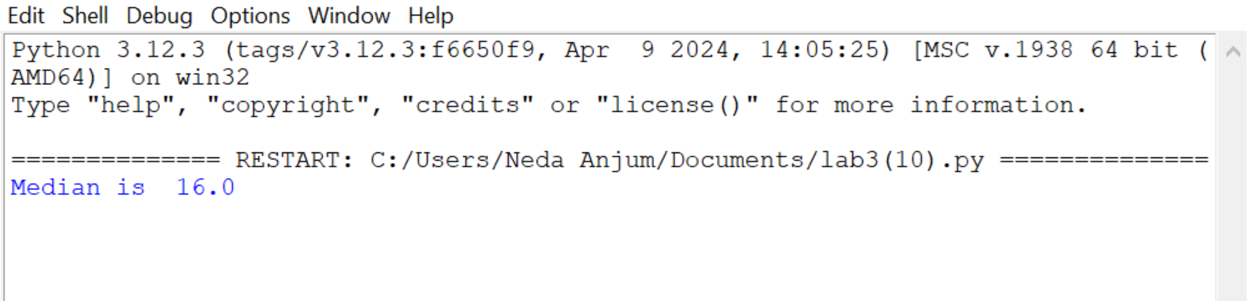


```
============== RESTART: C:/Users/Neda Anjum/Documents/k closest.py =============
The k closest points to the origin: [(0, 1), (-2, 2)]
>
```

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 getMedian( ar1, ar2 , n):
    i = 0
    j = 0
    m1 = -1
    m2 = -1
    count = 0
    while count < n + 1:
        count += 1
        if i == n:
            m1 = m2
            m2 = ar2[0]
            break
        elif j == n:
            m1 = m2
            m2 = ar1[0]
            break
        if ar1[i] <= ar2[j]:
            m1 = m2
            m2 = ar1[i]
            i += 1
        else:
            m1 = m2
            m2 = ar2[j]
            j += 1
    return (m1 + m2)/2
ar1 = [1, 12, 15, 26, 38]
```

```
ar2 = [2, 13, 17, 30, 45]

n1 = len(ar1)

n2 = len(ar2)

if n1 == n2:

        print("Median is ", getMedian(ar1, ar2, n1))

else:

        print("Doesn't work  unequal size")
```



```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

============== RESTART: C:/Users/Neda Anjum/Documents/lab3(10).py ==============
Median is  16.0
```