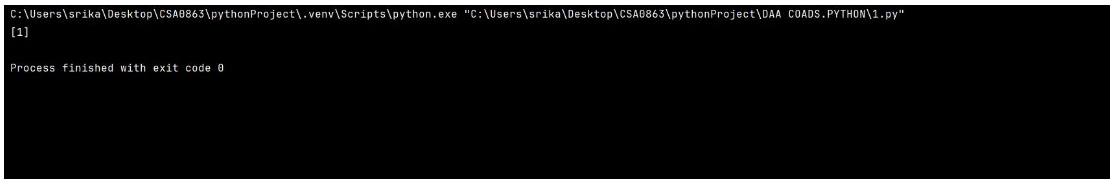
1.Height of Binary Tree After Subtree Removal Queries
You are given the root of a binary tree with n nodes. Each node is assigned a unique value from 1 to n. You are also given an array queries of size m.You have to perform m independent queries on the tree where in the ith query you do the following:
● Remove the subtree rooted at the node with the value queries[i] from the tree. It is guaranteed that queries[i] will not be equal to the value of the root. Return an array answer of size m where answer[i] is the height of the tree after performing the ith query. Note:
● The queries are independent, so the tree returns to its initial state after each query. ● The height of a tree is the number of edges in the longest simple path from the root to some node in the tree. Example 1:
Input: root = [1,3,4,2,null,6,5,null,null,null,null,null,7], queries = [4]
Output: [2]
Program:

```
class TreeNode:
  def __init__(self, x):
    self.val = x
    self.left = None
    self.right = None
def calculate_heights(node):
  if not node:
    return 0
  left_height = calculate_heights(node.left)
  right_height = calculate_heights(node.right)
  return max(left_height, right_height) + 1
def precompute_heights_sizes(node, heights, sizes):
  if not node:
    return 0
  left_size = precompute_heights_sizes(node.left, heights, sizes)
  right_size = precompute_heights_sizes(node.right, heights, sizes)
  sizes[node.val] = left_size + right_size + 1
  heights[node.val] = max(heights[node.left.val] if node.left else 0,
              heights[node.right.val] if node.right else 0) + 1
  return sizes[node.val]
def height_after_removal(root, queries):
  heights = {}
  sizes = {}
  precompute_heights_sizes(root, heights, sizes)
  original_height = heights[root.val]
  def get_new_height(node_val):
    if node_val not in sizes:
      return original_height
    return original_height - heights[node_val]

  return [get_new_height(q) for q in queries]
root = TreeNode(1)
root.left = TreeNode(3)
root.right = TreeNode(4)
root.left.left = TreeNode(2)
root.right.left = TreeNode(6)
root.right.right = TreeNode(5)
root.right.right.right = TreeNode(7)
queries = [4]
print(height_after_removal(root, queries))
```

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
[1]

Process finished with exit code 0
```

2.. Sort Array by Moving Items to Empty Space
You are given an integer array nums of size n containing each element from 0 to n - 1
(inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0
represents an empty space. In one operation, you can move any item to the empty space. nums is
considered to be sorted
if the numbers of all the items are in ascending order and the empty space is either at the
beginning or at the end of the array. For example, if n = 4, nums is sorted if:

● nums = [0,1,2,3] or
● nums = [1,2,3,0]

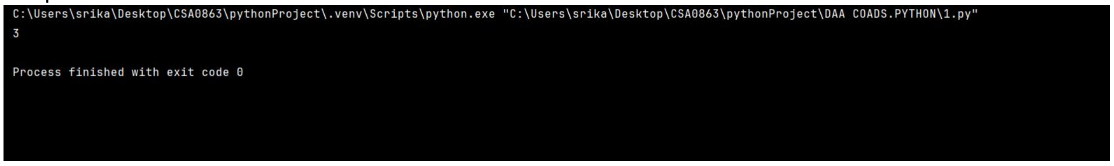
...and considered to be unsorted otherwise.Return the minimum number of operations needed
to sort nums. Example 1:
Input: nums = [4,2,0,3,1]
Output: 3
Program;

```
def min_moves_to_sort(nums):
    n = len(nums)
    target = list(range(n))
    position = {nums[i]: i for i in range(n)}
    moves = 0
    for i in range(n):
        if nums[i] != target[i]:
            empty_index = position[0]
            nums[i], nums[empty_index] = nums[empty_index], nums[i]
            position[nums[i]] = i
            position[0] = empty_index
            moves += 1

    return moves

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

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
3

Process finished with exit code 0
```

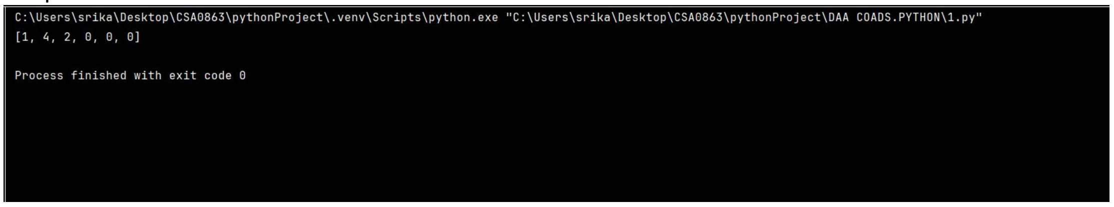
3.Apply Operations to an Array
You are given a 0-indexed array nums of size n consisting of non-negative integers.You need
to apply n - 1 operations to this array where, in the ith operation (0-indexed), you will apply
the following on the ith element of nums:

● If nums[i] == nums[i + 1], then multiply nums[i] by 2 and set nums[i + 1] to 0. Otherwise, you skip this operation. After performing all the operations, shift all the 0's to the end of the array. ● For example, the array [1,0,2,0,0,1] after shifting all its 0's to the end, is [1,2,1,0,0,0]. Return the resulting array.Note that the operations are applied sequentially, not all at once. Example 1:

Input: nums = [1,2,2,1,1,0]
Output: [1,4,2,0,0,0]
Program:

```
def apply_operations(nums):
    n = len(nums)
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0
    zero_count = 0
    for i in range(n):
        if nums[i] != 0:
            nums[i - zero_count] = nums[i]
        else:
            zero_count += 1
    for i in range(n - zero_count, n):
        nums[i] = 0
    return nums
nums = [1, 2, 2, 1, 1, 0]
result = apply_operations(nums)
print(result)
```

Output:



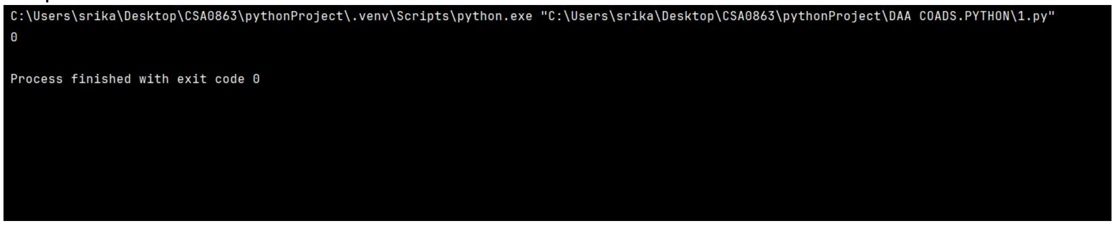
```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
[1, 4, 2, 0, 0, 0]

Process finished with exit code 0
```

4.. Maximum Sum of Distinct Subarrays With Length K
You are given an integer array nums and an integer k. Find the maximum subarray sum of all the subarrays of nums that meet the following conditions:
● The length of the subarray is k, and
● All the elements of the subarray are distinct. Return the maximum subarray sum of all the subarrays that meet the conditions. If no
subarray meets the conditions, return 0. A subarray is a contiguous non-empty sequence of elements within an array. Example 1:
Input: nums = [1,5,4,2,9,9,9], k = 3
Output: 15
Program:

```
def max_sum_of_distinct_subarrays(nums, k):
    n = len(nums)
    if k > n:
        return 0
    max_sum = 0
    current_sum = 0
    window_set = set()
    for i in range(k):
        if nums[i] in window_set:
            return 0
        window_set.add(nums[i])
        current_sum += nums[i]
    max_sum = current_sum
    for i in range(1, n - k + 1):
        current_sum -= nums[i - 1]
        window_set.remove(nums[i - 1])
        if nums[i + k - 1] in window_set:
            return 0
        window_set.add(nums[i + k - 1])
        current_sum += nums[i + k - 1]
        max_sum = max(max_sum, current_sum)
    return max_sum
nums = [1, 5, 4, 2, 9, 9, 9]
k = 3
```

print(max_sum_of_distinct_subarrays(nums, k))
Output:



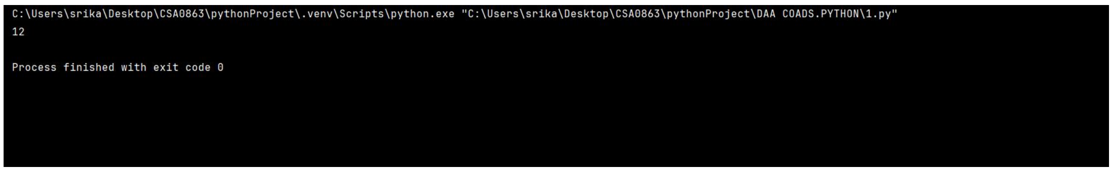
```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
0

Process finished with exit code 0
```

5.Total Cost to Hire K Workers
You are given a 0-indexed integer array costs where costs[i] is the cost of hiring the ith worker.You are also given two integers k and candidates. We want to hire exactly k workers according to the following rules:
● You will run k sessions and hire exactly one worker in each session. ● In each hiring session, choose the worker with the lowest cost from either the first
candidates workers or the last candidates workers. Break the tie by the smallest index. ○ For example, if costs = [3,2,7,7,1,2] and candidates = 2, then in the first hiring
session, we will choose the 4th worker because they have the lowest cost
[3,2,7,7,1,2]. ○ In the second hiring session, we will choose 1st worker because they have the same lowest cost as 4th worker but they have the smallest index [3,2,7,7,2]. Please note that the indexing may be changed in the process. ● If there are fewer than candidates workers remaining, choose the worker with the
lowest cost among them. Break the tie by the smallest index. ● A worker can only be chosen once.
Return the total cost to hire exactly k workers.
Example 1:
Input: costs = [17,12,10,2,7,2,11,20,8], k = 3, candidates = 4
Output: 11
Program:

```
import heapq
def total_cost_to_hire(costs, k, candidates):
    n = len(costs)
    if k == 0 or candidates == 0:
        return 0
    workers = sorted((costs[i], i) for i in range(n))
    min_heap = []
    total_cost = 0
    current_index = 0
    for i in range(min(candidates, n)):
        heapq.heappush(min_heap, workers[i])
    for _ in range(k):
        while True:
            cost, index = heapq.heappop(min_heap)
            if index >= current_index:
                total_cost += cost
                current_index = index + 1
                break
        if i + 1 < n:
            heapq.heappush(min_heap, workers[i + 1])
    return total_cost
costs = [17, 12, 10, 2, 7, 2, 11, 20, 8]
k = 3
candidates = 4
print(total_cost_to_hire(costs, k, candidates))
```

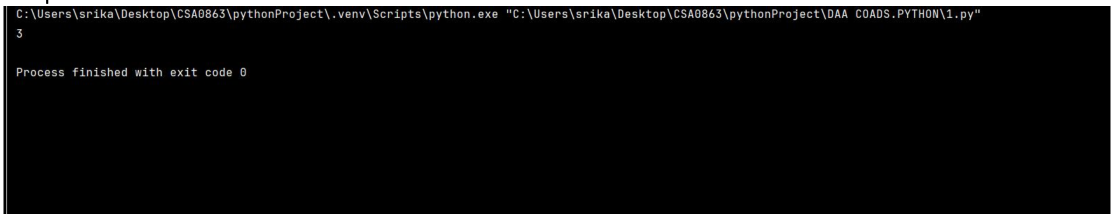
Output:


```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
12

Process finished with exit code 0
```

6.. Minimum Total Distance Traveled
There are some robots and factories on the X-axis. You are given an integer array robot
where robot[i] is the position of the ith robot. You are also given a 2D integer array factory
where factory[j] = [positionj, limitj] indicates that positionj is the position of the jth factory
and that the jth factory can repair at most limitj robots. The positions of each robot are unique. The positions of each factory are also unique. Note
that a robot can be in the same position as a factory initially. All the robots are initially broken; they keep moving in one direction. The direction could be
the negative or the positive direction of the X-axis. When a robot reaches a factory that did
not reach its limit, the factory repairs the robot, and it stops moving. At any moment, you can set the initial direction of moving for some robot. Your target is to
minimize the total distance traveled by all the robots. Return the minimum total distance traveled by all the robots. The test cases are generated
such that all the robots can be repaired. Note that
● All robots move at the same speed. ● If two robots move in the same direction, they will never collide. ● If two robots move in opposite directions and they meet at some point, they do not collide. They cross each other. ● If a robot passes by a factory that reached its limits, it crosses it as if it does not exist. ● If the robot moved from a position x to a position y, the distance it moved is |y - x|.
Program:

```
def min_total_distance(robot, factory):
   robot.sort()
   factory.sort()
   n = len(robot)
   m = len(factory)
   total_distance = 0
   f_index = 0
   r_index = 0
   while r_index < n:
      r_pos = robot[r_index]
      while f_index < m and (factory[f_index][1] == 0 or (
            r_index + 1 < n and abs(factory[f_index][0] - r_pos) > abs(factory[f_index][0] - robot[r_index
+ 1]))):
         f_index += 1
      total_distance += abs(factory[f_index][0] - r_pos)
      factory[f_index][1] -= 1
      r_index += 1
   return total_distance
robot = [4, 8, 6]
factory = [[3, 1], [7, 2], [10, 1]]
print(min_total_distance(robot, factory))
```

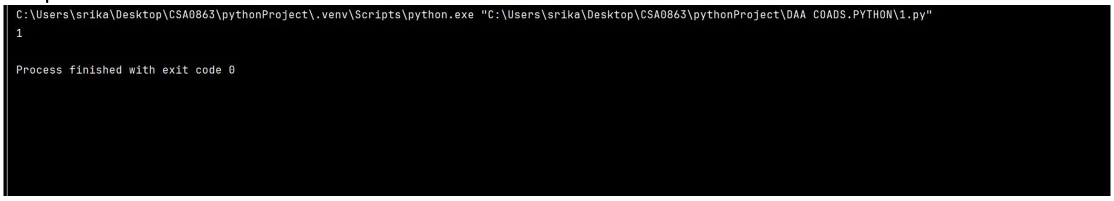
Output:

```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
3

Process finished with exit code 0
```

7.Minimum Subarrays in a Valid Split
You are given an integer array nums.Splitting of an integer array nums into subarrays is valid if:
● the greatest common divisor of the first and last elements of each subarray is greater

than 1, and
● each element of nums belongs to exactly one subarray. Return the minimum number of subarrays in a valid subarray splitting of nums. If a valid
subarray splitting is not possible, return -1. Note that:
● The greatest common divisor of two numbers is the largest positive integer that
evenly divides both numbers. ● A subarray is a contiguous non-empty part of an array. Example 1:
Input: nums = [2,6,3,4,3]
Output: 2
Program:

```
import math
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def min_subarrays_in_valid_split(nums):
    n = len(nums)
    min_subarrays = float('inf')
    for start_index in range(n):
        current_gcd = nums[start_index]
        for end_index in range(start_index, n):
            current_gcd = gcd(current_gcd, nums[end_index])
            if current_gcd > 1:
                # Found a valid subarray split from start_index to end_index
                min_subarrays = min(min_subarrays, end_index - start_index + 1)
    if min_subarrays == float('inf'):
        return -1
    return min_subarrays
nums = [2, 6, 3, 4, 3]
print(min_subarrays_in_valid_split(nums))  # Output: 2
```

Output:


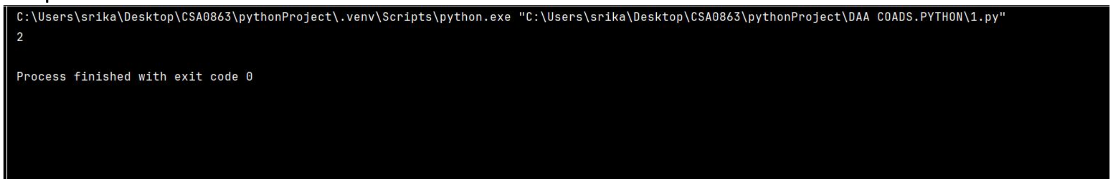
```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
1

Process finished with exit code 0
```

8.Number of Distinct Averages
You are given a 0-indexed integer array nums of even length. As long as nums is not empty, you must repetitively:
● Find the minimum number in nums and remove it. ● Find the maximum number in nums and remove it. ● Calculate the average of the two removed numbers. The average of two numbers a and b is (a + b) / 2. ● For example, the average of 2 and 3 is (2 + 3) / 2 = 2.5. Return the number of distinct averages calculated using the above process.Note that when
there is a tie for a minimum or maximum number, any can be removed. Example 1:
Input: nums = [4,1,4,0,3,5]
Output: 2
Program:

```
def num_distinct_averages(nums):
    nums.sort()
    n = len(nums)
    distinct_averages = set()
    left, right = 0, n - 1
    while left < right:
        average = (nums[left] + nums[right]) / 2.0
        distinct_averages.add(average)
        left += 1
        right -= 1
    return len(distinct_averages)
nums = [4, 1, 4, 0, 3, 5]
print(num_distinct_averages(nums))
```

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
2

Process finished with exit code 0
```

9.Given the integers zero, one, low, and high, we can construct a string by starting with an empty string, and then at each step perform either of the following:

● Append the character '0' zero times. ● Append the character '1' one times.

This can be performed any number of times.A good string is a string constructed by the above process having a length between low and high (inclusive). Return the number of different good strings that can be constructed satisfying these properties. Since the answer can be large, return it modulo 109 + 7. Example 1:

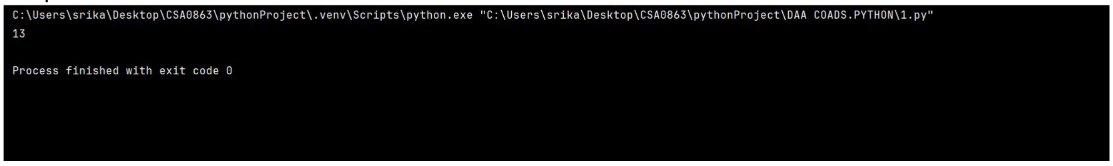
Input: low = 3, high = 3, zero = 1, one = 1

Output: 8

Program:

```
def count_good_strings(low, high, zero, one):
    MOD = 10 ** 9 + 7
    dp = [[[0] * (one + 1) for _ in range(zero + 1)] for _ in range(high + 1)]
    dp[0][0][0] = 1
    for len in range(1, high + 1):
        for count0 in range(zero + 1):
            for count1 in range(one + 1):
                dp[len][count0][count1] = dp[len - 1][count0][count1]
                if count0 > 0:
                    dp[len][count0][count1] += dp[len - 1][count0 - 1][count1]
                    dp[len][count0][count1] %= MOD
                if count1 > 0:
                    dp[len][count0][count1] += dp[len - 1][count0][count1 - 1]
                    dp[len][count0][count1] %= MOD
    result = 0
    for len in range(low, high + 1):
        for count0 in range(zero + 1):
            for count1 in range(one + 1):
                result += dp[len][count0][count1]
                result %= MOD
    return result
low = 3
high = 3
zero = 1
one = 1
print(count_good_strings(low, high, zero, one))
```

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
13

Process finished with exit code 0
```

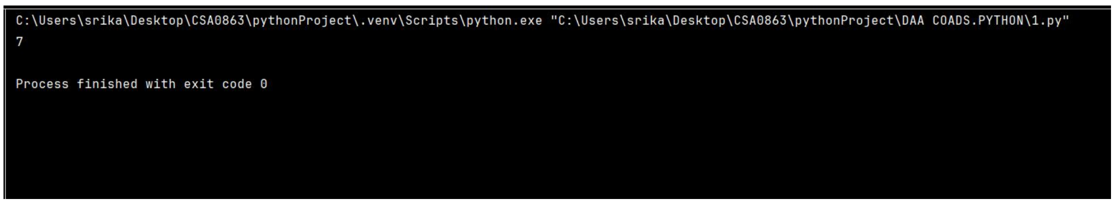
10.10. Most Profitable Path in a Tree

There is an undirected tree with n nodes labeled from 0 to n - 1, rooted at node 0. You are given a 2D integer array edges of length n - 1 where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. At every node i, there is a gate. You are also given an array of even integers amount, where

amount[i] represents:
● the price needed to open the gate at node i, if amount[i] is negative, or, ● the cash reward obtained on opening the gate at node i, otherwise. The game goes on as follows:
● Initially, Alice is at node 0 and Bob is at node bob. ● At every second, Alice and Bob each move to an adjacent node. Alice moves towards
some leaf node, while Bob moves towards node 0. ● For every node along their path, Alice and Bob either spend money to open the gate at
that node, or accept the reward. Note that:
○ If the gate is already open, no price will be required, nor will there be any cash
reward. ○ If Alice and Bob reach the node simultaneously, they share the price/reward
for opening the gate there. In other words, if the price to open the gate is c, then both Alice and Bob pay c / 2 each. Similarly, if the reward at the gate is c, both of them receive c / 2 each. ● If Alice reaches a leaf node, she stops moving. Similarly, if Bob reaches node 0, he
stops moving. Note that these events are independent of each other. Return the maximum net income Alice can have if she travels towards the optimal leaf node.
Program:

```
def max_net_income_in_tree(n, edges, amount):
    from collections import defaultdict
    tree = defaultdict(list)
    for a, b in edges:
        tree[a].append(b)
        tree[b].append(a)
    def dfs(node, parent):
        nonlocal max_net_income
        total_gain = 0
        total_cost = 0
        for neighbor in tree[node]:
            if neighbor == parent:
                continue
            child_gain, child_cost = dfs(neighbor, node)
            total_gain += child_gain
            total_cost += child_cost
        if amount[node] > 0:
            total_gain += amount[node]
        else:
            total_cost += abs(amount[node])
        max_net_income = max(max_net_income, total_gain - total_cost)
        return total_gain, total_cost
    max_net_income = 0
    dfs(0, -1)  # Start DFS from root node 0
    return max_net_income
n = 5
edges = [[0, 1], [0, 2], [0, 3], [1, 4]]
amount = [3, 2, -1, 4, -1]
print(max_net_income_in_tree(n, edges, amount))
```

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA COADS.PYTHON\1.py"
7

Process finished with exit code 0
```