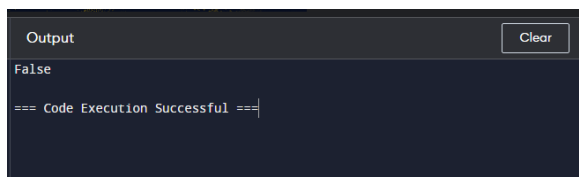


QUESTION 1:

IN AN GIVEN BINARY ARRAY CONSISTING OF N ELEMENTS LAST POSITION IS K ,IF K IS EQUAL TO 1 THEN THE RESULT SHOULD BE TRUE OTHERWISE FALSE.

CODE:

```
def k_length_apart(nums, k):  
    prev = -k - 1  
  
    for i, num in enumerate(nums):  
        if num == 1:  
            if i - prev <= k:  
                return False  
  
            prev = i  
  
    return True  
  
nums = [1, 0, 0, 1, 0, 1]  
  
k = 2  
  
print(k_length_apart(nums, k))
```



The screenshot shows a dark-themed output window. At the top, the word 'Output' is on the left and a 'Clear' button is on the right. Below the header, the text 'False' is displayed. At the bottom, a status message reads '=== Code Execution Successful ==='.

QUESTION 2

Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

Given an array of integers nums and an integer limit, return the size of the longest non-empty subarray such that the absolute difference between any two elements of this subarray is less than or equal to limit.

CODE:

```
def longest_subarray(nums, limit):  
  
    def helper(left, right, max_val, min_val):
```

```

if right == len(nums):
    return right - left

max_val = max(max_val, nums[right])
min_val = min(min_val, nums[right])

if max_val - min_val <= limit:
    return max(right - left + 1, helper(left, right + 1, max_val, min_val))

else:
    return helper(left + 1, right, nums[left + 1], nums[left + 1])

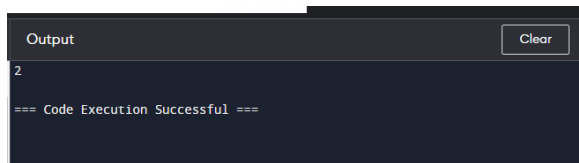
return helper(0, 0, nums[0], nums[0])

nums = [8, 2, 4, 7]

limit = 4

print(longest_subarray(nums, limit))

```



```

Output
2
=== Code Execution Successful ===

```

QUESTION 3:

Find the Kth Smallest Sum of a Matrix With Sorted Rows

You are given an $m \times n$ matrix `mat` that has its rows sorted in non-decreasing order and an integer `k`.

You are allowed to choose exactly one element from each row to form an array.

Return the `k`th smallest array sum among all possible arrays.

CODE:

```

from heapq import heappush, heappop

def kthSmallest(mat, k):
    m, n = len(mat), len(mat[0])

    def merge_rows(row1, row2):

```

```

min_heap = []

for num1 in row1:
    for num2 in row2:
        heappush(min_heap, num1 + num2)
        if len(min_heap) > k:
            heappop(min_heap)

result = []

while min_heap:
    result.append(heappop(min_heap))

result.sort()

return result

def recursive_combine(index):
    if index == 0:
        return mat[0]

    return merge_rows(recursive_combine(index - 1), mat[index])

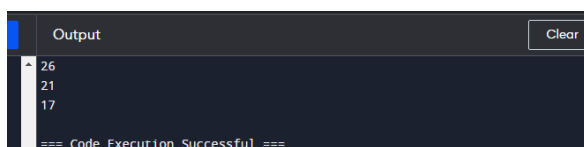
return recursive_combine(m - 1)[k - 1]

mat = [
    [1, 3, 11],
    [2, 4, 6],
    [5, 8, 9]
]

k = 5

print(kthSmallest(mat, k))

```



```

Output
26
21
17
=== Code Execution Successful ===

```

QUESTION 4:

Count Triplets That Can Form Two Arrays of Equal XOR

Given an array of integers arr.

We want to select three indices i, j and k where $(0 \leq i < j \leq k < \text{arr.length})$.

Let's define a and b as follows:

- $a = \text{arr}[i] \oplus \text{arr}[i + 1] \oplus \dots \oplus \text{arr}[j - 1]$
- $b = \text{arr}[j] \oplus \text{arr}[j + 1] \oplus \dots \oplus \text{arr}[k]$

Note that \oplus denotes the bitwise-xor operation.

Return the number of triplets (i, j and k) Where $a == b$.

CODE:

```
from heapq import heappush, heappop

def kthSmallest(mat, k):
    m, n = len(mat), len(mat[0])

    def merge_rows(row1, row2):
        min_heap = []

        for num1 in row1:
            for num2 in row2:
                heappush(min_heap, num1 + num2)

            if len(min_heap) > k:
                heappop(min_heap)

    result = []

    while min_heap:
        result.append(heappop(min_heap))

    result.sort()

    return result

def recursive_combine(index):
    if index == 0:
```

```

        return mat[0]

    return merge_rows(recursive_combine(index - 1), mat[index])

return recursive_combine(m - 1)[k - 1]

mat = [

    [1, 3, 11],

    [2, 4, 6],

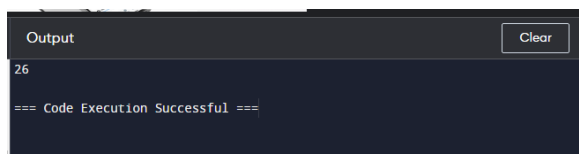
    [5, 8, 9]

]

k = 5

print(kthSmallest(mat, k))

```



The screenshot shows a dark-themed window titled "Output" with a "Clear" button in the top right corner. The output text is white and displays the number "26" on the first line, followed by "=== Code Execution Successful ===" on the second line.

QUESTION 5

Minimum Time to Collect All Apples in a Tree Given an undirected tree consisting of n vertices numbered from 0 to $n-1$, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at vertex 0 and coming back to this vertex. The edges of the undirected tree are given in the array `edges`, where `edges[i] = [ai, bi]` means that exists an edge connecting the vertices `ai` and `bi`. Additionally, there is a boolean array `hasApple`, where `hasApple[i] = true` means that vertex `i` has an apple; otherwise, it does not have any apple.

CODE:

```

def minTime(n, edges, hasApple):

    from collections import defaultdict

    tree = defaultdict(list)

    for u, v in edges:

        tree[u].append(v)

        tree[v].append(u)

```

```
def dfs(node, parent):  
    total_time = 0  
    for neighbor in tree[node]:  
        if neighbor == parent:  
            continue  
        time_spent = dfs(neighbor, node)  
        if time_spent > 0 or hasApple[neighbor]:  
            total_time += time_spent + 2  
    return total_time  
return dfs(0, -1)
```

n = 7

edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]]

hasApple = [False, False, True, False, True, True, False]

print(minTime(n, edges, hasApple))

```
Output Clear  
8  
=== Code Execution Successful ===
```