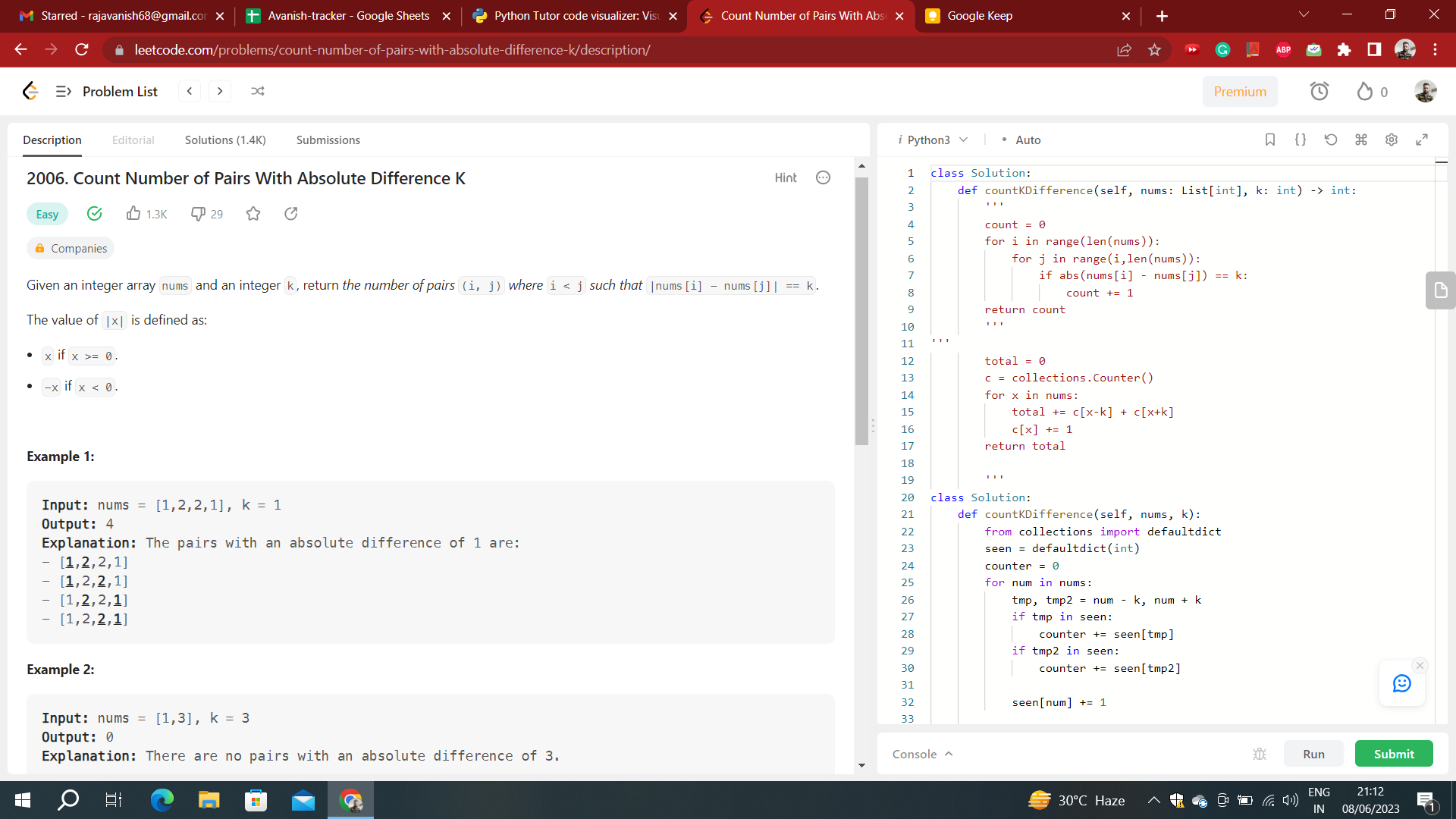
Count Number of Pairs With Absolute Difference K

[https://leetcode.com/problems/count-number-of-pairs-with-absolute-difference-k/](https://leetcode.com/problems/count-number-of-pairs-with-absolute-difference-k/description/)

**

**Time Complexity:** O(N2)  
**Auxiliary Space:** O(1)

class Solution:

    def countKDifference(self, nums: List[int], k: int) -> int:

        '''

        count = 0

        for i in range(len(nums)):

            for j in range(i,len(nums)):

                if abs(nums[i] - nums[j]) == k:

                    count += 1

        return count

**Time Complexity:** O(N)  
**Auxiliary Space:** O(N)

class Solution:

    def countKDifference(self, nums, k):

        from collections import defaultdict

        seen = defaultdict(int)

        counter = 0

        for num in nums:

            tmp, tmp2 = num - k, num + k

            if tmp in seen:

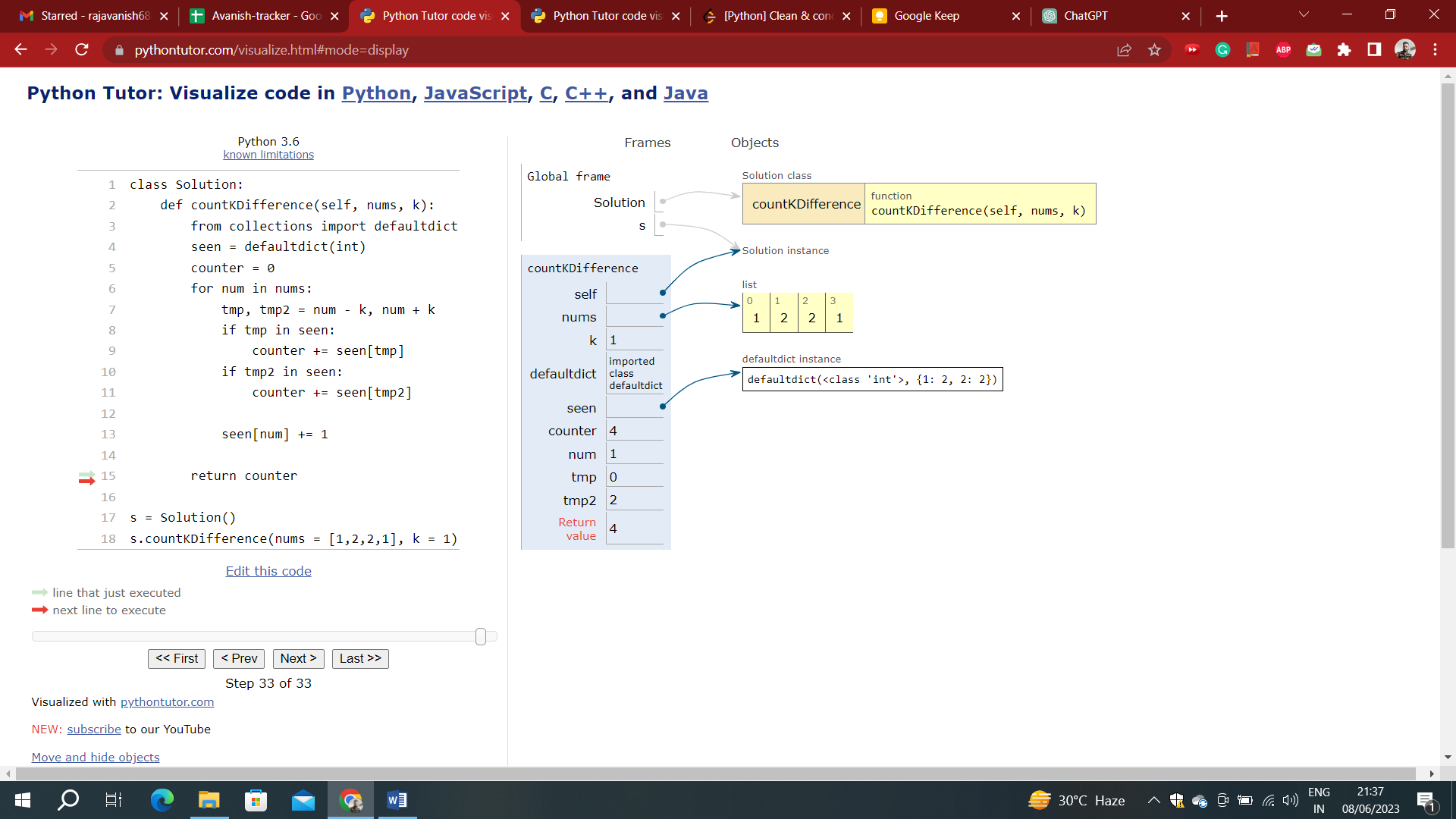
                counter += seen[tmp]

            if tmp2 in seen:

                counter += seen[tmp2]

            seen[num] += 1

        return counter



shorter version:

class Solution:

def countKDifference(self, nums: List[int], k: int) -> int:

seen = defaultdict(int)

counter = 0

for num in nums:

counter += seen[num-k] + seen[num+k]

seen[num] += 1

return counter

without using defaultdict or counter:

class Solution:

def countKDifference(self, nums, k):

seen = {}

counter = 0

for num in nums:

tmp, tmp2 = num - k, num + k

if tmp in seen:

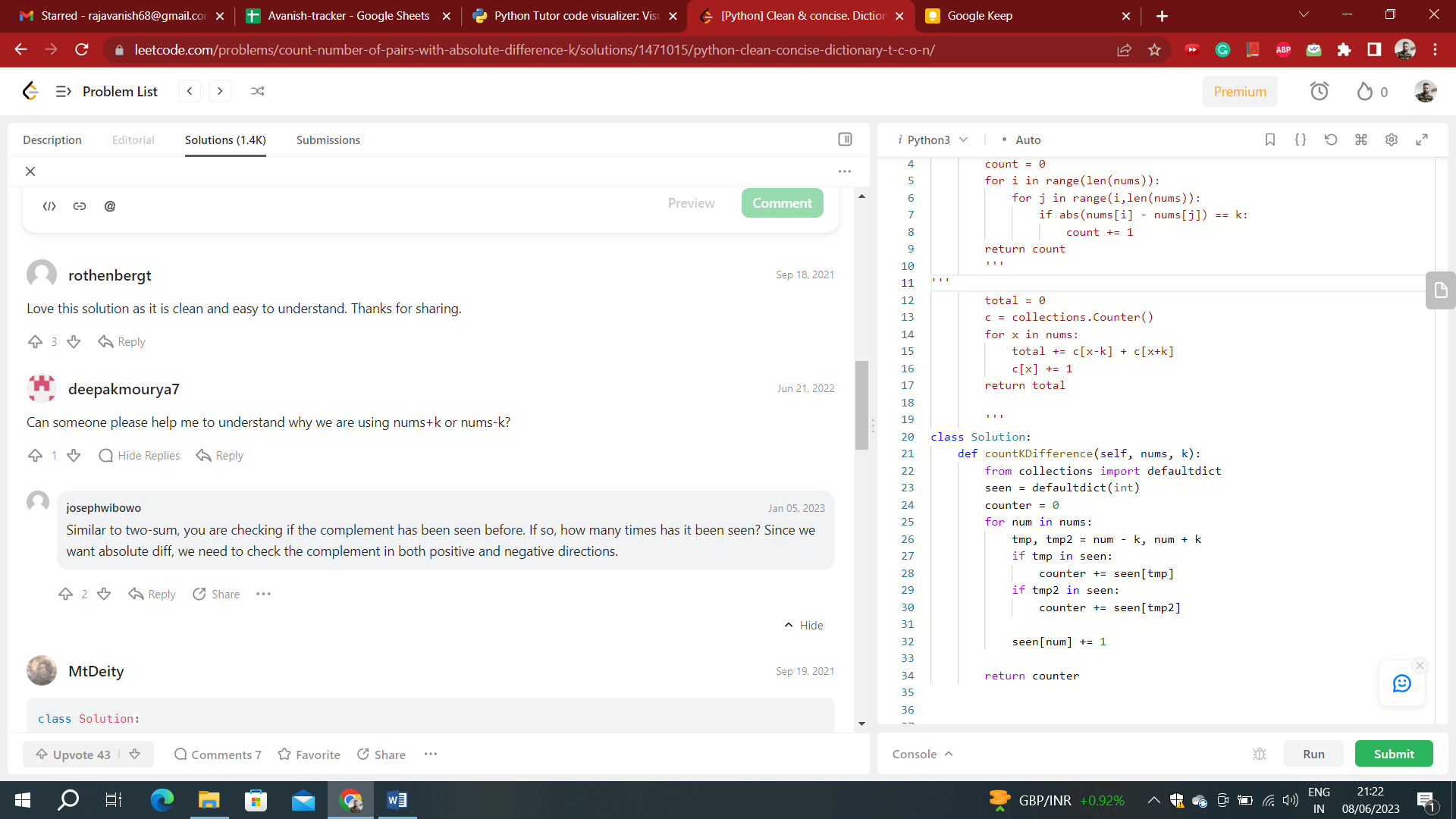
counter += seen[tmp]

if tmp2 in seen:

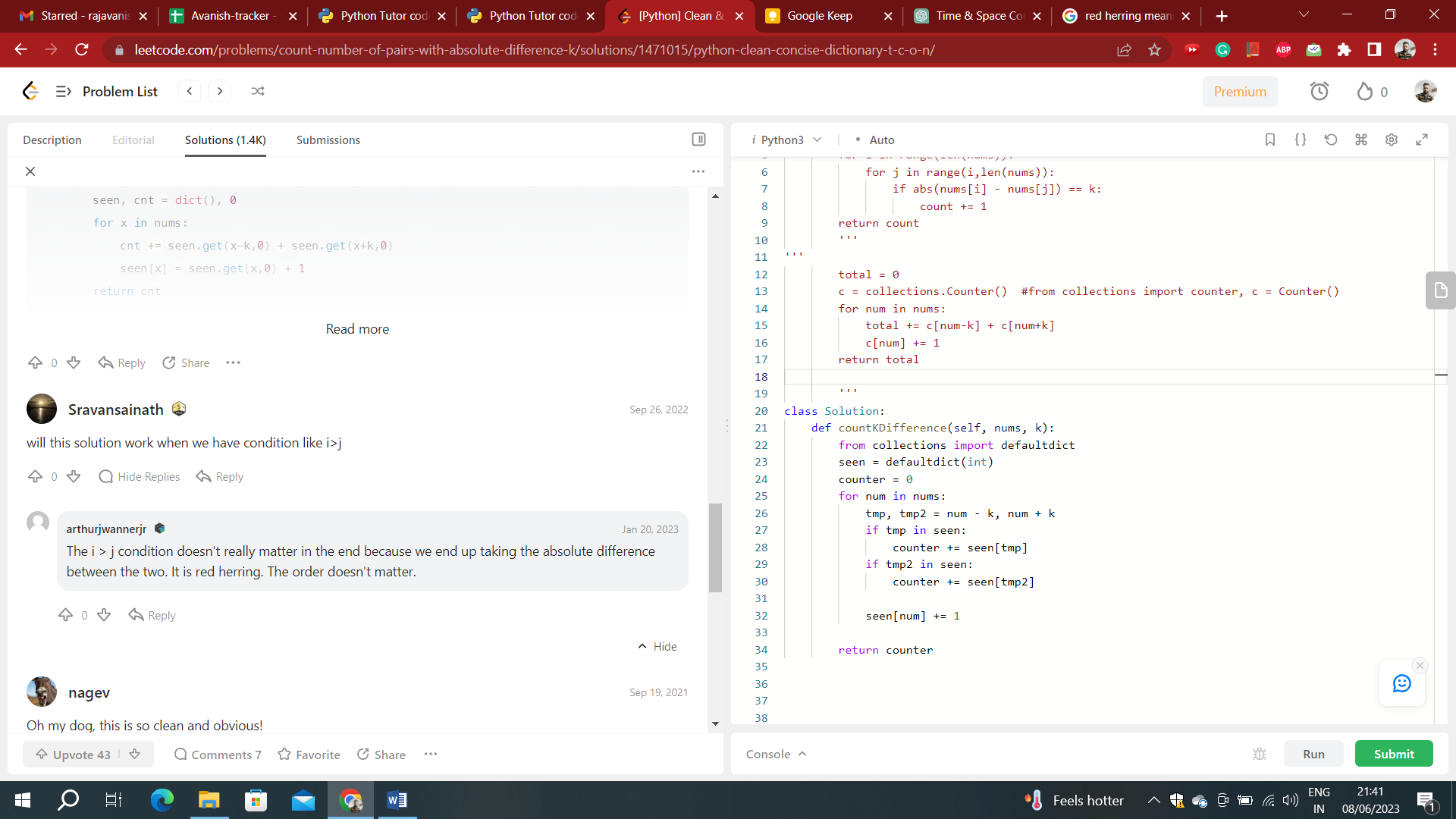
counter += seen[tmp2]

seen[num] = seen.get(num, 0) + 1

return counter



The space complexity of this code is O(n), where n is the length of the **nums** list. This is because the code uses a **defaultdict** named **seen** to store the frequency of each number in **nums**. In the worst case, all distinct elements in **nums** will be stored in **seen**, resulting in space complexity proportional to the size of **nums**.



        total = 0

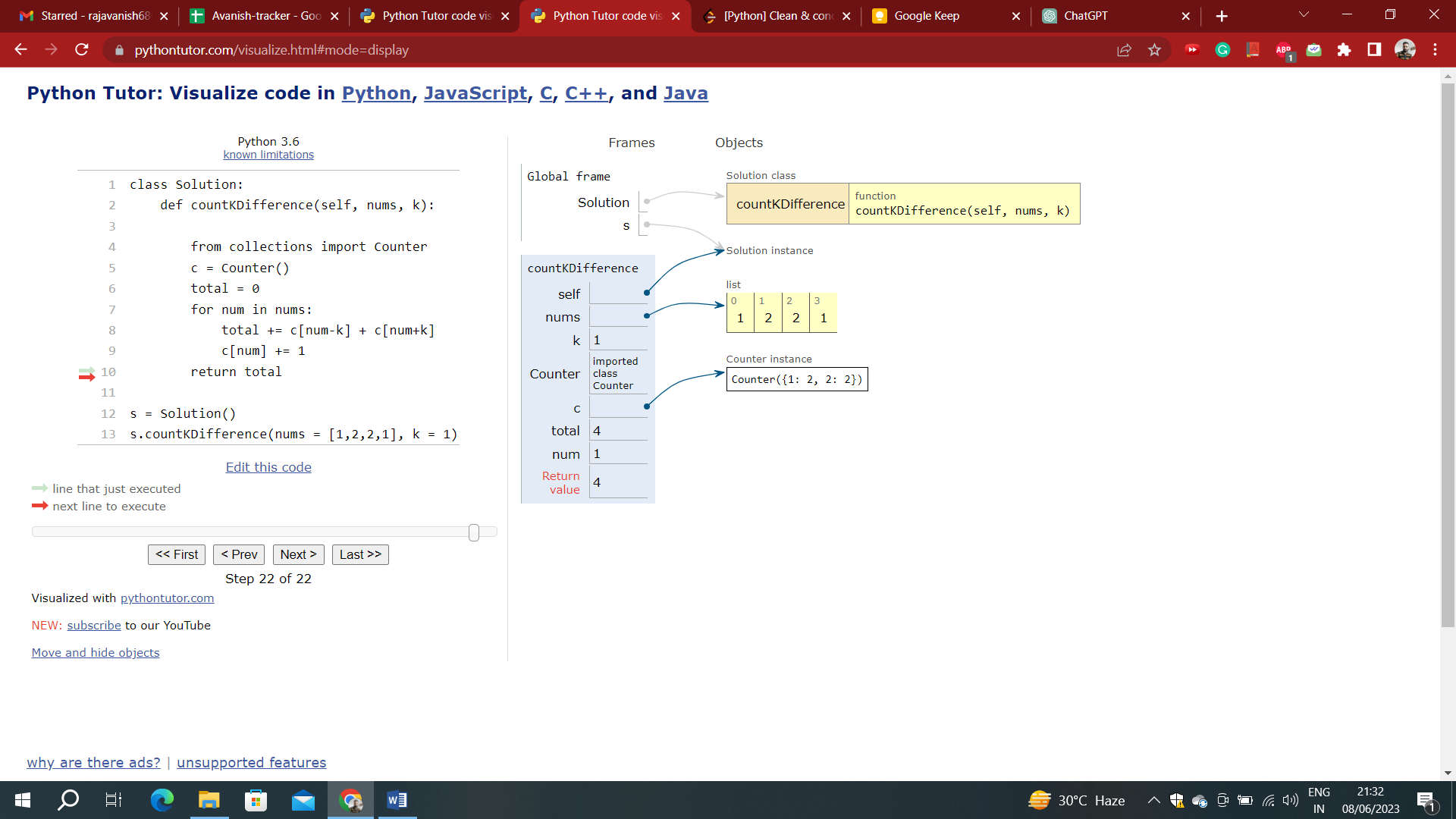
        c = collections.Counter()  #from collections import counter, c = Counter()

        for num in nums:

            total += c[num-k] + c[num+k]

            c[num] += 1

        return total



**NOTE:**

Similar to **defaultdict**, **Counter** provides a default value of 0 for missing keys. When accessing the count of a key that hasn't been encountered before, it returns 0 instead of raising a **KeyError**.

#We count the number of times each number shows up in the array, then for each number, we check if it + k is in the dict.

#If it is, then we generated # of pairs equal to the number of times val + k showed up.

#e.g., for [2, 2, 1, 1], when num = 1, we have two 2s, so we add 2 pairs. when num = 1 again, we add another 2 pairs.

#We don't need to check if it - k is in the array because that value has already incremented counter in another part of the array.

nums\_dict, count = {}, 0

for num in nums:

if num not in nums\_dict:

nums\_dict[num] = 1

else:

nums\_dict[num] += 1

for num in nums:

if num + k in nums\_dict:

count += nums\_dict[num + k]

return count