Documentation K-diff Pairs in an Array

1. Problem Statement

Given an array of integers nums and an integer k, return the number of unique k-diff pairs in the array. A k-diff pair is defined as a pair (nums[i], nums[j]) such that:

- 0 <= i, j < nums.length
- i != j
- |nums[i] - nums[j]| == k

2. Intuition

To efficiently count unique pairs with absolute difference k, we use a frequency map (hash table). Depending on whether k is zero or positive, we either:

- Count elements that appear more than once (for k = 0), or
- Check if num + k exists for every unique number in the array (for k > 0).

3. Key Observations

- If k < 0, return 0 immediately — absolute difference can't be negative.
- For k = 0, pairs must consist of the same number appearing at least twice.
- For k > 0, we only need to check if num + k exists to ensure unique pairs.

4. Approach

- Use collections.Counter to build a frequency map of all elements.
- If k == 0:
    - Iterate through the map and count how many numbers appear more than once.
- If k > 0:
    - For each number, check if num + k exists in the map.
- Return the total count of such unique pairs.

5. Edge Cases

- Empty array → output: 0
- k < 0 → output: 0
- All unique numbers, k = 0 → output: 0
- Multiple duplicates → ensure unique pairs only
- Large values of k → should still be handled in linear time

6. Complexity Analysis

Time Complexity

- O(n): where n is the length of the array.
    - Counter(nums) is O(n)
    - Loop through unique keys is O(n) in worst case

Space Complexity

- O(n): For storing frequency map in Counter

7. Alternative Approaches

- Brute-force (nested loops):
    - Time: $O(n^2)$
    - Space: O(1)
    - Not suitable for large inputs.

- Sorting + Two pointers:
    - Time: O(n log n)
    - Space: O(1) or O(n) depending on sort implementation
    - Needs extra care to avoid counting duplicates.

8. Test Cases

Input	k	Output	Explanation
[3,1,4,1,5]	2	2	Pairs: (1,3), (3,5)
[1,2,3,4,5]	1	4	Pairs: (1,2), (2,3), (3,4), (4,5)
[1,3,1,5,4]	0	1	Only one duplicate: (1,1)
[1,2,3,4,5]	0	0	No duplicates
[1,1,1,1,1]	0	1	Only one unique duplicate (1,1)
[]	1	0	Empty input
[1,2,3,4,5]	-1	0	Negative k not valid

9. Final Thoughts

This problem is a great example of using hash maps for fast lookups and uniqueness tracking. The key is handling the k = 0 case correctly and avoiding double-counting. The solution is efficient and scalable for large inputs.