K-diff Pairs in an Array

Given an array of integers and an integer **k**, you need to find the number of **unique** k-diff pairs in the array. Here a **k-diff** pair is defined as an integer pair (i, j), where **i** and **j** are both numbers in the array and their absolute difference is **k**.

Example 1:

```
Input: [3, 1, 4, 1, 5], k = 2
Output: 2
Explanation: There are two 2-diff pairs in the array, (1, 3) and (3, 5).Although we
 have two 1s in the input, we should only return the number of unique pairs.
```

Example 2:

```
Input:[1, 2, 3, 4, 5], k = 1
Output: 4
Explanation: There are four 1-diff pairs in the array, (1, 2), (2, 3), (3, 4) and (
4, 5).
```

Example 3:

```
Input: [1, 3, 1, 5, 4], k = 0
Output: 1
Explanation: There is one 0-diff pair in the array, (1, 1).
```

Note:

1. The pairs (i, j) and (j, i) count as the same pair.
2. The length of the array won't exceed 10,000.
3. All the integers in the given input belong to the range: [-1e7, 1e7].

Solution 1

```java
public class Solution {
    public int findPairs(int[] nums, int k) {
        if (nums == null || nums.length == 0 || k < 0)   return 0;

        Map<Integer, Integer> map = new HashMap<>();
        int count = 0;
        for (int i : nums) {
            map.put(i, map.getOrDefault(i, 0) + 1);
        }

        for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
            if (k == 0) {
                //count how many elements in the array that appear more than twice.
                if (entry.getValue() >= 2) {
                    count++;
                }
            } else {
                if (map.containsKey(entry.getKey() + k)) {
                    count++;
                }
            }
        }

        return count;
    }
}
```

written by tankztc original link here

Solution 2

The problem is just a variant of 2-sum.
Update: Fixed a bug that can cause integer subtraction overflow.
Update: The code runs in `O(n log n)` time, using `O(1)` space.

```java
public int findPairs(int[] nums, int k) {
    int ans = 0;
    Arrays.sort(nums);
    for (int i = 0, j = 0; i < nums.length; i++) {
        for (j = Math.max(j, i + 1); j < nums.length && (long) nums[j] - nums[i]
< k; j++) ;
        if (j < nums.length && (long) nums[j] - nums[i] == k) ans++;
        while (i + 1 < nums.length && nums[i] == nums[i + 1]) i++;
    }
    return ans;
}
```

written by lixx2100 original link here

Solution 3

```
    def findPairs(self, nums, k):
       return len(set(nums)&{n+k for n in nums}) if k>0 else sum(v>1 for v in collections.Counter(nums).values()) if k==0 else 0
```

which is equivalent to:

```
    def findPairs(self, nums, k):
        if k>0:
            return len(set(nums)&set(n+k for n in nums))
        elif k==0:
            sum(v>1 for v in collections.Counter(nums).values())
        else:
            return 0
```

written by o_sharp original link here