Problem 532: K-diff Pairs in an Array

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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 an integer pair

(nums[i], nums[j])

, where the following are true:

0 <= i, j < nums.length

i != j

|nums[i] - nums[j]| == k

Notice

that

|val|

denotes the absolute value of

val

.

Example 1:

Input:

nums = [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:

nums = [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:

nums = [1,3,1,5,4], k = 0

Output:

1

Explanation:

There is one 0-diff pair in the array, (1, 1).

Constraints:

1 <= nums.length <= 10

4

-10

7

<= nums[i] <= 10

7

0 <= k <= 10

7

Code Snippets

C++:

```
class Solution {
public:
int findPairs(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int findPairs(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def findPairs(self, nums: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def findPairs(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var findPairs = function(nums, k) {

};
```

TypeScript:

```
function findPairs(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int FindPairs(int[] nums, int k) {

}
}
```

C:

```
int findPairs(int* nums, int numsSize, int k) {

}
```

Go:

```
func findPairs(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun findPairs(nums: IntArray, k: Int): Int {

}
```

```
}
```

Swift:

```
class Solution {
func findPairs(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_pairs(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def find_pairs(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function findPairs($nums, $k) {

}
}
```

Dart:

```
class Solution {
int findPairs(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def findPairs(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_pairs(nums :: [integer], k :: integer) :: integer
def find_pairs(nums, k) do

end
end
```

Erlang:

```
-spec find_pairs(Nums :: [integer()], K :: integer()) -> integer().
find_pairs(Nums, K) ->
.
```

Racket:

```
(define/contract (find-pairs nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K-diff Pairs in an Array
```

```
* Difficulty: Medium
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int findPairs(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K-diff Pairs in an Array
* Difficulty: Medium
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int findPairs(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K-diff Pairs in an Array
Difficulty: Medium
Tags: array, hash, sort, search

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def findPairs(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findPairs(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: K-diff Pairs in an Array
* Difficulty: Medium
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var findPairs = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K-diff Pairs in an Array
* Difficulty: Medium
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function findPairs(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: K-diff Pairs in an Array
* Difficulty: Medium
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int FindPairs(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: K-diff Pairs in an Array
* Difficulty: Medium
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int findPairs(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: K-diff Pairs in an Array
// Difficulty: Medium
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findPairs(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findPairs(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findPairs(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: K-diff Pairs in an Array
// Difficulty: Medium
// Tags: array, hash, sort, search
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn find_pairs(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def find_pairs(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function findPairs($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findPairs(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findPairs(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_pairs(nums :: [integer], k :: integer) :: integer
def find_pairs(nums, k) do

end
end
```

Erlang Solution:

```
-spec find_pairs(Nums :: [integer()], K :: integer()) -> integer().
find_pairs(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (find-pairs nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```