Problem 220: Contains Duplicate III

Problem Information

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

Problem Description

You are given an integer array

nums

and two integers

indexDiff

and

valueDiff

.

Find a pair of indices

(i, j)

such that:

i != j

,

abs(i - j) <= indexDiff

.

abs(nums[i] - nums[j]) <= valueDiff

, and

Return

true

if such pair exists or

false

otherwise

.

Example 1:

Input:

nums = [1,2,3,1], indexDiff = 3, valueDiff = 0

Output:

true

Explanation:

We can choose (i, j) = (0, 3). We satisfy the three conditions: i != j --> 0 != 3 abs(i - j) <= indexDiff --> abs(0 - 3) <= 3 abs(nums[i] - nums[j]) <= valueDiff --> abs(1 - 1) <= 0

Example 2:

Input:

nums = [1,5,9,1,5,9], indexDiff = 2, valueDiff = 3

Output:

false

Explanation:

After trying all the possible pairs (i, j), we cannot satisfy the three conditions, so we return false.

Constraints:

2 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

1 <= indexDiff <= nums.length

0 <= valueDiff <= 10

9

Code Snippets

C++:

```
class Solution {
public:
bool containsNearbyAlmostDuplicate(vector<int>& nums, int indexDiff, int
valueDiff) {

}
```

```
};
```

Java:

```
class Solution {
public boolean containsNearbyAlmostDuplicate(int[] nums, int indexDiff, int
valueDiff) {

}
}
```

Python3:

```
class Solution:
def containsNearbyAlmostDuplicate(self, nums: List[int], indexDiff: int,
valueDiff: int) -> bool:
```

Python:

```
class Solution(object):
def containsNearbyAlmostDuplicate(self, nums, indexDiff, valueDiff):
"""
:type nums: List[int]
:type indexDiff: int
:type valueDiff: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} indexDiff
* @param {number} valueDiff
* @return {boolean}
*/
var containsNearbyAlmostDuplicate = function(nums, indexDiff, valueDiff) {

};
```

TypeScript:

```
function containsNearbyAlmostDuplicate(nums: number[], indexDiff: number,
valueDiff: number): boolean {

};
```

C#:

```
public class Solution {
public bool ContainsNearbyAlmostDuplicate(int[] nums, int indexDiff, int
valueDiff) {

}
}
```

C:

```
bool containsNearbyAlmostDuplicate(int* nums, int numsSize, int indexDiff,
int valueDiff) {

}
```

Go:

```
func containsNearbyAlmostDuplicate(nums []int, indexDiff int, valueDiff int)
bool {

}
```

Kotlin:

```
class Solution {
fun containsNearbyAlmostDuplicate(nums: IntArray, indexDiff: Int, valueDiff:
Int): Boolean {

}
}
```

Swift:

```
class Solution {
func containsNearbyAlmostDuplicate(_ nums: [Int], _ indexDiff: Int, _
valueDiff: Int) -> Bool {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn contains_nearby_almost_duplicate(nums: Vec<i32>, index_diff: i32,
value_diff: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} index_diff
# @param {Integer} value_diff
# @return {Boolean}
def contains_nearby_almost_duplicate(nums, index_diff, value_diff)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $indexDiff
* @param Integer $valueDiff
* @return Boolean
*/
function containsNearbyAlmostDuplicate($nums, $indexDiff, $valueDiff) {

}
}
```

Dart:

```
class Solution {
bool containsNearbyAlmostDuplicate(List<int> nums, int indexDiff, int
valueDiff) {
```

```
}
}
```

Scala:

```
object Solution {
def containsNearbyAlmostDuplicate(nums: Array[Int], indexDiff: Int,
valueDiff: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec contains_nearby_almost_duplicate(nums :: [integer], index_diff ::
integer, value_diff :: integer) :: boolean
def contains_nearby_almost_duplicate(nums, index_diff, value_diff) do

end
end
```

Erlang:

```
-spec contains_nearby_almost_duplicate(Nums :: [integer()], IndexDiff ::
integer(), ValueDiff :: integer()) -> boolean().
contains_nearby_almost_duplicate(Nums, IndexDiff, ValueDiff) ->
.
```

Racket:

```
(define/contract (contains-nearby-almost-duplicate nums indexDiff valueDiff)
(-> (listof exact-integer?) exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Contains Duplicate III
* Difficulty: Hard
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool containsNearbyAlmostDuplicate(vector<int>& nums, int indexDiff, int
valueDiff) {

}
};
```

Java Solution:

```
/**
* Problem: Contains Duplicate III
* Difficulty: Hard
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean containsNearbyAlmostDuplicate(int[] nums, int indexDiff, int
valueDiff) {

}
}
```

Python3 Solution:

```
"""
Problem: Contains Duplicate III
Difficulty: Hard
```

```
Tags: array, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def containsNearbyAlmostDuplicate(self, nums: List[int], indexDiff: int,
valueDiff: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def containsNearbyAlmostDuplicate(self, nums, indexDiff, valueDiff):
"""
:type nums: List[int]
:type indexDiff: int
:type valueDiff: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Contains Duplicate III
* Difficulty: Hard
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} indexDiff
* @param {number} valueDiff
* @return {boolean}
```

```
*/
var containsNearbyAlmostDuplicate = function(nums, indexDiff, valueDiff) {

};
```

TypeScript Solution:

```
/**
* Problem: Contains Duplicate III
* Difficulty: Hard
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function containsNearbyAlmostDuplicate(nums: number[], indexDiff: number,
valueDiff: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Contains Duplicate III
* Difficulty: Hard
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool ContainsNearbyAlmostDuplicate(int[] nums, int indexDiff, int
valueDiff) {

}
}
```

C Solution:

```
/*
* Problem: Contains Duplicate III
* Difficulty: Hard
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool containsNearbyAlmostDuplicate(int* nums, int numsSize, int indexDiff,
int valueDiff) {

}
```

Go Solution:

```
// Problem: Contains Duplicate III
// Difficulty: Hard
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func containsNearbyAlmostDuplicate(nums []int, indexDiff int, valueDiff int)
bool {

}
```

Kotlin Solution:

```
class Solution {
fun containsNearbyAlmostDuplicate(nums: IntArray, indexDiff: Int, valueDiff:
Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func containsNearbyAlmostDuplicate(_ nums: [Int], _ indexDiff: Int, _
valueDiff: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Contains Duplicate III
// Difficulty: Hard
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn contains_nearby_almost_duplicate(nums: Vec<i32>, index_diff: i32,
value_diff: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} index_diff
# @param {Integer} value_diff
# @return {Boolean}
def contains_nearby_almost_duplicate(nums, index_diff, value_diff)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $indexDiff
* @param Integer $valueDiff
```

```
* @return Boolean
*/
function containsNearbyAlmostDuplicate($nums, $indexDiff, $valueDiff) {

}
}
```

Dart Solution:

```
class Solution {
bool containsNearbyAlmostDuplicate(List<int> nums, int indexDiff, int
valueDiff) {

}
}
```

Scala Solution:

```
object Solution {
def containsNearbyAlmostDuplicate(nums: Array[Int], indexDiff: Int,
valueDiff: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec contains_nearby_almost_duplicate(nums :: [integer], index_diff ::
integer, value_diff :: integer) :: boolean
def contains_nearby_almost_duplicate(nums, index_diff, value_diff) do

end
end
```

Erlang Solution:

```
-spec contains_nearby_almost_duplicate(Nums :: [integer()], IndexDiff ::
integer(), ValueDiff :: integer()) -> boolean().
contains_nearby_almost_duplicate(Nums, IndexDiff, ValueDiff) ->
.
```

Racket Solution:

```
(define/contract (contains-nearby-almost-duplicate nums indexDiff valueDiff)
(-> (listof exact-integer?) exact-integer? exact-integer? boolean?)
)
```