Problem 334: Increasing Triplet Subsequence

Problem Information

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

Problem Description

Given an integer array

nums

, return

true

if there exists a triple of indices

(i, j, k)

such that

i < j < k

and

nums[i] < nums[j] < nums[k]

. If no such indices exists, return

false

.

Example 1:

Input:

nums = [1,2,3,4,5]

Output:

true

Explanation:

Any triplet where i < j < k is valid.

Example 2:

Input:

nums = [5,4,3,2,1]

Output:

false

Explanation:

No triplet exists.

Example 3:

Input:

nums = [2,1,5,0,4,6]

Output:

true

Explanation:

One of the valid triplet is (1, 4, 5), because nums[1] == 1 < nums[4] == 4 < nums[5] == 6.

Constraints:

1 <= nums.length <= 5 * 10

5

-2

31

<= nums[i] <= 2

31

- 1

Follow up:

Could you implement a solution that runs in

O(n)

time complexity and

O(1)

space complexity?

Code Snippets

C++:

```
class Solution {
public:
bool increasingTriplet(vector<int>& nums) {


}
};
```

Java:

```
class Solution {
public boolean increasingTriplet(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {boolean}
*/
var increasingTriplet = function(nums) {

};
```

TypeScript:

```
function increasingTriplet(nums: number[]): boolean {

};
```

C#:

}

```
public class Solution {
public bool IncreasingTriplet(int[] nums) {

}
}
```

C:

```
bool increasingTriplet(int* nums, int numsSize) {

}
```

Go:

```
func increasingTriplet(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun increasingTriplet(nums: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func increasingTriplet(_ nums: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn increasing_triplet(nums: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Boolean}
def increasing_triplet(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function increasingTriplet($nums) {

}
}
```

Dart:

```
class Solution {
bool increasingTriplet(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def increasingTriplet(nums: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec increasing_triplet(nums :: [integer]) :: boolean
def increasing_triplet(nums) do

end
end
```

Erlang:

```
-spec increasing_triplet(Nums :: [integer()]) -> boolean().
increasing_triplet(Nums) ->
.
```

Racket:

```
(define/contract (increasing-triplet nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Increasing Triplet Subsequence
* Difficulty: Medium
* Tags: array, greedy
*
* 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 increasingTriplet(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Increasing Triplet Subsequence
* Difficulty: Medium
* Tags: array, greedy
*
* 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 increasingTriplet(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Increasing Triplet Subsequence
Difficulty: Medium
Tags: array, greedy

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 increasingTriplet(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Increasing Triplet Subsequence
* Difficulty: Medium
* Tags: array, greedy
*
* 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
* @return {boolean}
*/
var increasingTriplet = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Increasing Triplet Subsequence
* Difficulty: Medium
* Tags: array, greedy
*
* 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 increasingTriplet(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Increasing Triplet Subsequence
* Difficulty: Medium
* Tags: array, greedy
*
* 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 IncreasingTriplet(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Increasing Triplet Subsequence
* Difficulty: Medium
* Tags: array, greedy
*
* 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 increasingTriplet(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Increasing Triplet Subsequence
// Difficulty: Medium
// Tags: array, greedy
//
// 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 increasingTriplet(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun increasingTriplet(nums: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func increasingTriplet(_ nums: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Increasing Triplet Subsequence
// Difficulty: Medium
// Tags: array, greedy
//
// 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 increasing_triplet(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean}
def increasing_triplet(nums)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Boolean
*/
function increasingTriplet($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool increasingTriplet(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def increasingTriplet(nums: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec increasing_triplet(nums :: [integer]) :: boolean
def increasing_triplet(nums) do

end
end
```

Erlang Solution:

```
-spec increasing_triplet(Nums :: [integer()]) -> boolean().
increasing_triplet(Nums) ->
.
```

Racket Solution:

```
(define/contract (increasing-triplet nums)
(-> (listof exact-integer?) boolean?)
)
```