Problem 456: 132 Pattern

Problem Information

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

Problem Description

Given an array of

n

integers

nums

, a

132 pattern

is a subsequence of three integers

nums[i]

,

nums[j]

and

nums[k]

such that

i < j < k

and

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

.

Return

true

if there is a

132 pattern

in

nums

, otherwise, return

false

.

Example 1:

Input:

nums = [1,2,3,4]

Output:

false

Explanation:

There is no 132 pattern in the sequence.

Example 2:

Input:

nums = [3,1,4,2]

Output:

true

Explanation:

There is a 132 pattern in the sequence: [1, 4, 2].

Example 3:

Input:

nums = [-1,3,2,0]

Output:

true

Explanation:

There are three 132 patterns in the sequence: [-1, 3, 2], [-1, 3, 0] and [-1, 2, 0].

Constraints:

n == nums.length

1 <= n <= 2 * 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
bool find132pattern(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public boolean find132pattern(int[] nums) {

}
}
```

Python3:

```
class Solution:
def find132pattern(self, nums: List[int]) -> bool:
```

Python:

```
class Solution(object):
def find132pattern(self, nums):
"""
:type nums: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {boolean}
*/
var find132pattern = function(nums) {

};
```

TypeScript:

```
function find132pattern(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool Find132pattern(int[] nums) {

}
}
```

C:

```
bool find132pattern(int* nums, int numsSize) {

}
```

Go:

```
func find132pattern(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun find132pattern(nums: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func find132pattern(_ nums: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn find132pattern(nums: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Boolean}
def find132pattern(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function find132pattern($nums) {

}
}
```

Dart:

```
class Solution {
bool find132pattern(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def find132pattern(nums: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find132pattern(nums :: [integer]) :: boolean
def find132pattern(nums) do

end
end
```

Erlang:

```
-spec find132pattern(Nums :: [integer()]) -> boolean().
find132pattern(Nums) ->
.
```

Racket:

```
(define/contract (find132pattern nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 132 Pattern
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 find132pattern(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: 132 Pattern
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 find132pattern(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: 132 Pattern
Difficulty: Medium
Tags: array, search, stack

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 find132pattern(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def find132pattern(self, nums):
"""
:type nums: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: 132 Pattern
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 find132pattern = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: 132 Pattern
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 find132pattern(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: 132 Pattern
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 Find132pattern(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: 132 Pattern
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 find132pattern(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: 132 Pattern
// Difficulty: Medium
// Tags: array, search, stack
//
// 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 find132pattern(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun find132pattern(nums: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func find132pattern(_ nums: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: 132 Pattern
// Difficulty: Medium
// Tags: array, search, stack
//
// 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 find132pattern(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean}
def find132pattern(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function find132pattern($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool find132pattern(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def find132pattern(nums: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find132pattern(nums :: [integer]) :: boolean
def find132pattern(nums) do

end
end
```

Erlang Solution:

```
-spec find132pattern(Nums :: [integer()]) -> boolean().
find132pattern(Nums) ->
.
```

Racket Solution:

```
(define/contract (find132pattern nums)
(-> (listof exact-integer?) boolean?)
)
```