

Problem 896: Monotonic Array

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

An array is

monotonic

if it is either monotone increasing or monotone decreasing.

An array

nums

is monotone increasing if for all

$i \leq j$

,

$\text{nums}[i] \leq \text{nums}[j]$

. An array

nums

is monotone decreasing if for all

$i \leq j$

,

nums[i] >= nums[j]

Given an integer array

nums

, return

true

if the given array is monotonic, or

false

otherwise

.

Example 1:

Input:

nums = [1,2,2,3]

Output:

true

Example 2:

Input:

nums = [6,5,4,4]

Output:

true

Example 3:

Input:

nums = [1,3,2]

Output:

false

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

-10

5

$\leq \text{nums}[i] \leq 10$

5

Code Snippets

C++:

```
class Solution {
public:
    bool isMonotonic(vector<int>& nums) {
        }
};
```

Java:

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

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function isMonotonic(nums: number[]): boolean {  
  
};
```

C#:

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

C:

```
bool isMonotonic(int* nums, int numsSize) {  
    }  
}
```

Go:

```
func isMonotonic(nums []int) bool {  
    }  
}
```

Kotlin:

```
class Solution {  
    fun isMonotonic(nums: IntArray): Boolean {  
        }  
        }  
    }
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Boolean  
     */  
    function isMonotonic($nums) {  
  
    }  
}
```

Dart:

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

Scala:

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

Elixir:

```
defmodule Solution do  
  @spec is_monotonic(list :: [integer]) :: boolean  
  def is_monotonic(list) do  
  
  end  
end
```

Erlang:

```
-spec is_monotonic(list :: [integer()]) -> boolean().  
is_monotonic(List) ->  
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Monotonic Array
 * Difficulty: Easy
 * Tags: array
 *
 * 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 isMonotonic(vector<int>& nums) {
}
```

Java Solution:

```
/**
 * Problem: Monotonic Array
 * Difficulty: Easy
 * Tags: array
 *
 * 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 isMonotonic(int[] nums) {
```

```
}
```

```
}
```

Python3 Solution:

```
"""
Problem: Monotonic Array
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Monotonic Array
 * Difficulty: Easy
 * Tags: array
 *
 * 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 isMonotonic = function(nums) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Monotonic Array  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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 isMonotonic(nums: number[]): boolean {  
  
};
```

C# Solution:

```
/*  
 * Problem: Monotonic Array  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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 IsMonotonic(int[] nums) {  
  
    }
```

```
}
```

C Solution:

```
/*
 * Problem: Monotonic Array
 * Difficulty: Easy
 * Tags: array
 *
 * 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 isMonotonic(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Monotonic Array
// Difficulty: Easy
// Tags: array
//
// 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 isMonotonic(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
    fun isMonotonic(nums: IntArray): Boolean {
        }
    }
```

Swift Solution:

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

Rust Solution:

```
// Problem: Monotonic Array  
// Difficulty: Easy  
// Tags: array  
//  
// 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 is_monotonic(nums: Vec<i32>) -> bool {  
        }  
    }  
}
```

Ruby Solution:

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

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Boolean  
     */  
    function isMonotonic($nums) {  
        }  
    }
```

Dart Solution:

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

Scala Solution:

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

Elixir Solution:

```
defmodule Solution do  
  @spec is_monotonic(list :: [integer]) :: boolean  
  def is_monotonic(list) do  
  
  end  
end
```

Erlang Solution:

```
-spec is_monotonic(list :: [integer()]) -> boolean().  
is_monotonic(list) ->  
.
```

Racket Solution:

```
(define/contract (is-monotonic? list)  
  (-> (listof exact-integer?) boolean?)  
)
```