

Problem 3011: Find if Array Can Be Sorted

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

array of

positive

integers

nums

.

In one

operation

, you can swap any two

adjacent

elements if they have the

same

number of

set bits

. You are allowed to do this operation

any

number of times (

including zero

).

Return

true

if you can sort the array in ascending order, else return

false

.

Example 1:

Input:

nums = [8,4,2,30,15]

Output:

true

Explanation:

Let's look at the binary representation of every element. The numbers 2, 4, and 8 have one set bit each with binary representation "10", "100", and "1000" respectively. The numbers 15 and 30 have four set bits each with binary representation "1111" and "11110". We can sort the

array using 4 operations: - Swap `nums[0]` with `nums[1]`. This operation is valid because 8 and 4 have one set bit each. The array becomes `[4,8,2,30,15]`. - Swap `nums[1]` with `nums[2]`. This operation is valid because 8 and 2 have one set bit each. The array becomes `[4,2,8,30,15]`. - Swap `nums[0]` with `nums[1]`. This operation is valid because 4 and 2 have one set bit each. The array becomes `[2,4,8,30,15]`. - Swap `nums[3]` with `nums[4]`. This operation is valid because 30 and 15 have four set bits each. The array becomes `[2,4,8,15,30]`. The array has become sorted, hence we return true. Note that there may be other sequences of operations which also sort the array.

Example 2:

Input:

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

Output:

true

Explanation:

The array is already sorted, hence we return true.

Example 3:

Input:

`nums = [3,16,8,4,2]`

Output:

false

Explanation:

It can be shown that it is not possible to sort the input array using any number of operations.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 2

8

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

Python3:

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

Python:

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

JavaScript:

```

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

};

```

TypeScript:

```

function canSortArray(nums: number[]): boolean {

};

```

C#:

```

public class Solution {
    public bool CanSortArray(int[] nums) {

    }
}

```

C:

```

bool canSortArray(int* nums, int numsSize) {

}

```

Go:

```

func canSortArray(nums []int) bool {

}

```

Kotlin:

```

class Solution {
    fun canSortArray(nums: IntArray): Boolean {

    }
}

```

Swift:

```

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

  }
}

```

Rust:

```

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

  }
}

```

Ruby:

```

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

end

```

PHP:

```

class Solution {

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

  }
}

```

Dart:

```

class Solution {
  bool canSortArray(List<int> nums) {

  }
}

```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (can-sort-array nums)  
  (-> (listof exact-integer?) boolean?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Find if Array Can Be Sorted  
 * Difficulty: Medium  
 * 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 canSortArray(vector<int>& nums) {

    }
};

```

Java Solution:

```

/**
 * Problem: Find if Array Can Be Sorted
 * Difficulty: Medium
 * 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 canSortArray(int[] nums) {

    }
}

```

Python3 Solution:

```

"""
Problem: Find if Array Can Be Sorted
Difficulty: Medium
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 canSortArray(self, nums: List[int]) -> bool:
        # TODO: Implement optimized solution

```



```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**  
 * Problem: Find if Array Can Be Sorted  
 * Difficulty: Medium  
 * 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  
 * @return {boolean}  
 */  
var canSortArray = function(nums) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Find if Array Can Be Sorted  
 * Difficulty: Medium  
 * 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 canSortArray(nums: number[]): boolean {

};

```

C# Solution:

```

/*
 * Problem: Find if Array Can Be Sorted
 * Difficulty: Medium
 * 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 CanSortArray(int[] nums) {

    }
}

```

C Solution:

```

/*
 * Problem: Find if Array Can Be Sorted
 * Difficulty: Medium
 * 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 canSortArray(int* nums, int numsSize) {

}

```

Go Solution:

```

// Problem: Find if Array Can Be Sorted
// Difficulty: Medium
// 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 canSortArray(nums []int) bool {

}

```

Kotlin Solution:

```

class Solution {
    fun canSortArray(nums: IntArray): Boolean {

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```

// Problem: Find if Array Can Be Sorted
// Difficulty: Medium
// 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 can_sort_array(nums: Vec<i32>) -> bool {

    }
}

```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

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

  }
}
```

Scala Solution:

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

  }
}
```

Elixir Solution:

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

  end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-sort-array nums)
  (-> (listof exact-integer?) boolean?)
)
```