

Problem 3644: Maximum K to Sort a Permutation

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

`nums`

of length

`n`

, where

`nums`

is a

permutation

of the numbers in the range

`[0..n - 1]`

.

You may swap elements at indices

`i`

and

j

only if

$\text{nums}[i] \text{ AND } \text{nums}[j] == k$

, where

AND

denotes the bitwise AND operation and

k

is a

non-negative

integer.

Return the

maximum

value of

k

such that the array can be sorted in

non-decreasing

order using any number of such swaps. If

nums

is already sorted, return 0.

Example 1:

Input:

nums = [0,3,2,1]

Output:

1

Explanation:

Choose

k = 1

. Swapping

nums[1] = 3

and

nums[3] = 1

is allowed since

nums[1] AND nums[3] == 1

, resulting in a sorted permutation:

[0, 1, 2, 3]

.

Example 2:

Input:

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

Output:

2

Explanation:

Choose

k = 2

. Swapping

```
nums[2] = 3
```

and

```
nums[3] = 2
```

is allowed since

```
nums[2] AND nums[3] == 2
```

, resulting in a sorted permutation:

```
[0, 1, 2, 3]
```

.

Example 3:

Input:

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

Output:

0

Explanation:

Only

$k = 0$

allows sorting since no greater

k

allows the required swaps where

$\text{nums}[i] \text{ AND } \text{nums}[j] == k$

.

Constraints:

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

5

$0 \leq \text{nums}[i] \leq n - 1$

nums

is a permutation of integers from

0

to

$n - 1$

.

Code Snippets

C++:

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

    }
};
```

Java:

```
class Solution {
    public int sortPermutation(int[] nums) {

    }
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

C#:

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

C:

```
int sortPermutation(int* nums, int numsSize) {  
  
}
```

Go:

```
func sortPermutation(nums []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun sortPermutation(nums: IntArray): Int {  
  
    }  
}
```

Swift:

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

Rust:

```

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

  }
}

```

Ruby:

```

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

end

```

PHP:

```

class Solution {

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

    }

}

```

Dart:

```

class Solution {
  int sortPermutation(List<int> nums) {

  }
}

```

Scala:

```

object Solution {
  def sortPermutation(nums: Array[Int]): Int = {

  }
}

```

Elixir:

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

  end

end
```

Erlang:

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

```

Racket:

```
(define/contract (sort-permutation nums)
  (-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum K to Sort a Permutation
 * 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:
    int sortPermutation(vector<int>& nums) {

    }

};
```

Java Solution:

```
/**
 * Problem: Maximum K to Sort a Permutation
 * 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 int sortPermutation(int[] nums) {

}

}
```

Python3 Solution:

```
"""
Problem: Maximum K to Sort a Permutation
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 sortPermutation(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
 * Problem: Maximum K to Sort a Permutation
 * 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 {number}
 */
var sortPermutation = function(nums) {

};
```

TypeScript Solution:

```
/**
 * Problem: Maximum K to Sort a Permutation
 * 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 sortPermutation(nums: number[]): number {

};
```

C# Solution:

```

/*
 * Problem: Maximum K to Sort a Permutation
 * 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 int SortPermutation(int[] nums) {

    }
}

```

C Solution:

```

/*
 * Problem: Maximum K to Sort a Permutation
 * 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
 */

int sortPermutation(int* nums, int numsSize) {

}

```

Go Solution:

```

// Problem: Maximum K to Sort a Permutation
// 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 sortPermutation(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
    fun sortPermutation(nums: IntArray): Int {

    }
}
```

Swift Solution:

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

    }
}
```

Rust Solution:

```
// Problem: Maximum K to Sort a Permutation
// 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 sort_permutation(nums: Vec<i32>) -> i32 {

    }
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

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

Dart Solution:

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

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (sort-permutation nums)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```