# Problem 2717: Semi-Ordered Permutation

## Problem Information

**Difficulty:** Easy
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You are given a

0-indexed

permutation of

n

integers

nums

.

A permutation is called

semi-ordered

if the first number equals

1

and the last number equals

n

. You can perform the below operation as many times as you want until you make

nums

a

semi-ordered

permutation:

Pick two adjacent elements in

nums

, then swap them.

Return

the minimum number of operations to make

nums

a

semi-ordered permutation

.

A

permutation

is a sequence of integers from

1

to

n

of length

n

containing each number exactly once.

Example 1:

Input:

nums = [2,1,4,3]

Output:

2

Explanation:

We can make the permutation semi-ordered using these sequence of operations: 1 - swap i = 0 and j = 1. The permutation becomes [1,2,4,3]. 2 - swap i = 2 and j = 3. The permutation becomes [1,2,3,4]. It can be proved that there is no sequence of less than two operations that make nums a semi-ordered permutation.

Example 2:

Input:

nums = [2,4,1,3]

Output:

3

Explanation:

We can make the permutation semi-ordered using these sequence of operations: 1 - swap i = 1 and j = 2. The permutation becomes [2,1,4,3]. 2 - swap i = 0 and j = 1. The permutation becomes [1,2,4,3]. 3 - swap i = 2 and j = 3. The permutation becomes [1,2,3,4]. It can be proved that there is no sequence of less than three operations that make nums a

semi-ordered permutation.

Example 3:

Input:

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

Output:

0

Explanation:

The permutation is already a semi-ordered permutation.

Constraints:

2 <= nums.length == n <= 50

1 <= nums[i] <= 50

nums is a permutation.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

```
}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function semiOrderedPermutation(nums: number[]): number {

};
```

**C#:**

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

}
}
```

**C:**

```
int semiOrderedPermutation(int* nums, int numsSize) {

}
```

**Go:**

```
func semiOrderedPermutation(nums []int) int {

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

```
class Solution {
int semiOrderedPermutation(List<int> nums) {

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

### Racket:

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

## Solutions

### C++ Solution:

```
/*
* Problem: Semi-Ordered Permutation
* 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:
int semiOrderedPermutation(vector<int>& nums) {

}
};
```

### Java Solution:

```
/**
* Problem: Semi-Ordered Permutation
* 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 int semiOrderedPermutation(int[] nums) {
```

```
}
}
```

### Python3 Solution:

```
"""
Problem: Semi-Ordered Permutation
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 semiOrderedPermutation(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Semi-Ordered Permutation
* 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 {number}
*/
var semiOrderedPermutation = function(nums) {

};
```

### TypeScript Solution:

```
/**
* Problem: Semi-Ordered Permutation
* 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 semiOrderedPermutation(nums: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Semi-Ordered Permutation
* 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 int SemiOrderedPermutation(int[] nums) {

}
```

```
}
```

### C Solution:

```
/*
* Problem: Semi-Ordered Permutation
* 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
*/

int semiOrderedPermutation(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Semi-Ordered Permutation
// 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 semiOrderedPermutation(nums []int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Semi-Ordered Permutation
// 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 semi_ordered_permutation(nums: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

```
class Solution {
int semiOrderedPermutation(List<int> nums) {

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

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