Problem 1806: Minimum Number of Operations to Reinitialize a Permutation

Problem Information

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

Problem Description

You are given an

even

integer

n

. You initially have a permutation

perm

of size

n

where

perm[i] == i

(0-indexed)

.

In one operation, you will create a new array

arr

, and for each

i

:

If

i % 2 == 0

, then

arr[i] = perm[i / 2]

.

If

i % 2 == 1

, then

arr[i] = perm[n / 2 + (i - 1) / 2]

.

You will then assign

arr

to

perm

.

Return

the minimum

non-zero

number of operations you need to perform on

perm

to return the permutation to its initial value.

Example 1:

Input:

n = 2

Output:

1

Explanation:

perm = [0,1] initially. After the 1

st

operation, perm = [0,1] So it takes only 1 operation.

Example 2:

Input:

n = 4

Output:

2

Explanation:

perm = [0,1,2,3] initially. After the 1

st

operation, perm = [0,2,1,3] After the 2

nd

operation, perm = [0,1,2,3] So it takes only 2 operations.

Example 3:

Input:

n = 6

Output:

4

Constraints:

2 <= n <= 1000

n

is even.

Code Snippets

C++:

```
class Solution {
public:
```

```
int reinitializePermutation(int n) {

}
};
```

Java:

```
class Solution {
public int reinitializePermutation(int n) {

}
}
```

Python3:

```
class Solution:
def reinitializePermutation(self, n: int) -> int:
```

Python:

```
class Solution(object):
def reinitializePermutation(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var reinitializePermutation = function(n) {

};
```

TypeScript:

```
function reinitializePermutation(n: number): number {

};
```

C#:

```
public class Solution {
public int ReinitializePermutation(int n) {

}
}
```

C:

```
int reinitializePermutation(int n) {

}
```

Go:

```
func reinitializePermutation(n int) int {

}
```

Kotlin:

```
class Solution {
fun reinitializePermutation(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func reinitializePermutation(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn reinitialize_permutation(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def reinitialize_permutation(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function reinitializePermutation($n) {

}
}
```

Dart:

```
class Solution {
int reinitializePermutation(int n) {

}
}
```

Scala:

```
object Solution {
def reinitializePermutation(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reinitialize_permutation(n :: integer) :: integer
def reinitialize_permutation(n) do
```

```
end
end
```

Erlang:

```
-spec reinitialize_permutation(N :: integer()) -> integer().
reinitialize_permutation(N) ->
.
```

Racket:

```
(define/contract (reinitialize-permutation n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Reinitialize a Permutation
* Difficulty: Medium
* Tags: array, math
*
* 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 reinitializePermutation(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Reinitialize a Permutation
```

```
* Difficulty: Medium
* Tags: array, math
*
* 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 reinitializePermutation(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Reinitialize a Permutation
Difficulty: Medium
Tags: array, math

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 reinitializePermutation(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def reinitializePermutation(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Reinitialize a Permutation
* Difficulty: Medium
* Tags: array, math
*
* 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} n
* @return {number}
*/
var reinitializePermutation = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Reinitialize a Permutation
* Difficulty: Medium
* Tags: array, math
*
* 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 reinitializePermutation(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Reinitialize a Permutation
* Difficulty: Medium
* Tags: array, math
*
* 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 ReinitializePermutation(int n) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Reinitialize a Permutation
* Difficulty: Medium
* Tags: array, math
*
* 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 reinitializePermutation(int n) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Reinitialize a Permutation
// Difficulty: Medium
// Tags: array, math
//
// 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 reinitializePermutation(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun reinitializePermutation(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func reinitializePermutation(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Reinitialize a Permutation
// Difficulty: Medium
// Tags: array, math
//
// 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 reinitialize_permutation(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def reinitialize_permutation(n)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $n
* @return Integer
*/
function reinitializePermutation($n) {

}
}
```

Dart Solution:

```
class Solution {
int reinitializePermutation(int n) {

}
}
```

Scala Solution:

```
object Solution {
def reinitializePermutation(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reinitialize_permutation(n :: integer) :: integer
def reinitialize_permutation(n) do

end
end
```

Erlang Solution:

```
-spec reinitialize_permutation(N :: integer()) -> integer().
reinitialize_permutation(N) ->
.
```

Racket Solution:

```
(define/contract (reinitialize-permutation n)
(-> exact-integer? exact-integer?)
)
```