Problem 1718: Construct the Lexicographically Largest Valid Sequence

Problem Information

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

Problem Description

Given an integer

n

, find a sequence with elements in the range

[1, n]

that satisfies all of the following:

The integer

1

occurs once in the sequence.

Each integer between

2

and

n

occurs twice in the sequence.

For every integer

i

between

2

and

n

, the

distance

between the two occurrences of

i

is exactly

i

.

The

distance

between two numbers on the sequence,

a[i]

and

a[j]

, is the absolute difference of their indices,

|j - i|

.

Return

the

lexicographically largest

sequence

. It is guaranteed that under the given constraints, there is always a solution.

A sequence

a

is lexicographically larger than a sequence

b

(of the same length) if in the first position where

a

and

b

differ, sequence

a

has a number greater than the corresponding number in

b

. For example,

[0,1,9,0]

is lexicographically larger than

[0,1,5,6]

because the first position they differ is at the third number, and

9

is greater than

5

.

Example 1:

Input:

n = 3

Output:

[3,1,2,3,2]

Explanation:

[2,3,2,1,3] is also a valid sequence, but [3,1,2,3,2] is the lexicographically largest valid sequence.

Example 2:

Input:

n = 5

Output:

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

Constraints:

1 <= n <= 20

Code Snippets

C++:

```
class Solution {
public:
vector<int> constructDistancedSequence(int n) {

}
};
```

Java:

```
class Solution {
public int[] constructDistancedSequence(int n) {

}
}
```

Python3:

```
class Solution:
def constructDistancedSequence(self, n: int) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function constructDistancedSequence(n: number): number[] {

};
```

C#:

```
public class Solution {
public int[] ConstructDistancedSequence(int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* constructDistancedSequence(int n, int* returnSize) {

}
```

Go:

```
func constructDistancedSequence(n int) []int {

}
```

Kotlin:

```
class Solution {
fun constructDistancedSequence(n: Int): IntArray {
}
}
```

Swift:

```
class Solution {
func constructDistancedSequence(_ n: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn construct_distanced_sequence(n: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[]}
def construct_distanced_sequence(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[]
*/
function constructDistancedSequence($n) {

}
}
```

Dart:

```
class Solution {
List<int> constructDistancedSequence(int n) {

}
}
```

Scala:

```
object Solution {
def constructDistancedSequence(n: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec construct_distanced_sequence(n :: integer) :: [integer]
def construct_distanced_sequence(n) do

end
end
```

Erlang:

```
-spec construct_distanced_sequence(N :: integer()) -> [integer()].
construct_distanced_sequence(N) ->
.
```

Racket:

```
(define/contract (construct-distanced-sequence n)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct the Lexicographically Largest Valid Sequence
* Difficulty: Medium
* Tags: array, graph
*
* 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:
vector<int> constructDistancedSequence(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Construct the Lexicographically Largest Valid Sequence
* Difficulty: Medium
* Tags: array, graph
*
* 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[] constructDistancedSequence(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct the Lexicographically Largest Valid Sequence
Difficulty: Medium
Tags: array, graph

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 constructDistancedSequence(self, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Construct the Lexicographically Largest Valid Sequence
* Difficulty: Medium
* Tags: array, graph
*
* 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 constructDistancedSequence = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct the Lexicographically Largest Valid Sequence
* Difficulty: Medium
* Tags: array, graph
*
* 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 constructDistancedSequence(n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Construct the Lexicographically Largest Valid Sequence
* Difficulty: Medium
* Tags: array, graph
*
* 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[] ConstructDistancedSequence(int n) {

}
}
```

C Solution:

```
/*
* Problem: Construct the Lexicographically Largest Valid Sequence
* Difficulty: Medium
* Tags: array, graph
*
* 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
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* constructDistancedSequence(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Construct the Lexicographically Largest Valid Sequence
// Difficulty: Medium
// Tags: array, graph
//
// 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 constructDistancedSequence(n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun constructDistancedSequence(n: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func constructDistancedSequence(_ n: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Construct the Lexicographically Largest Valid Sequence
// Difficulty: Medium
// Tags: array, graph
//
// 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 construct_distanced_sequence(n: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[]}
def construct_distanced_sequence(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[]
*/
function constructDistancedSequence($n) {

}
}
```

Dart Solution:

```
class Solution {
List<int> constructDistancedSequence(int n) {

}
}
```

Scala Solution:

```
object Solution {
def constructDistancedSequence(n: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec construct_distanced_sequence(n :: integer) :: [integer]
def construct_distanced_sequence(n) do

end
end
```

Erlang Solution:

```
-spec construct_distanced_sequence(N :: integer()) -> [integer()].
construct_distanced_sequence(N) ->
.
```

Racket Solution:

```
(define/contract (construct-distanced-sequence n)
(-> exact-integer? (listof exact-integer?))
)
```