Problem 386: Lexicographical Numbers

Problem Information

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

Problem Description

Given an integer

n

, return all the numbers in the range

[1, n]

sorted in lexicographical order.

You must write an algorithm that runs in

O(n)

time and uses

O(1)

extra space.

Example 1:

Input:

n = 13

Output:

[1,10,11,12,13,2,3,4,5,6,7,8,9]

Example 2:

Input:

n = 2

Output:

[1,2]

Constraints:

1 <= n <= 5 * 10

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> lexicalOrder(int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> LexicalOrder(int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* lexicalOrder(int n, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun lexicalOrder(n: Int): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def lexicalOrder(n: Int): List[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (lexical-order n)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographical Numbers
* Difficulty: Medium
* Tags: graph, sort, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Lexicographical Numbers
* Difficulty: Medium
* Tags: graph, sort, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<Integer> lexicalOrder(int n) {
}
}
```

Python3 Solution:

```
"""
Problem: Lexicographical Numbers
Difficulty: Medium
Tags: graph, sort, search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def lexicalOrder(self, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographical Numbers
* Difficulty: Medium
* Tags: graph, sort, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographical Numbers
* Difficulty: Medium
* Tags: graph, sort, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function lexicalOrder(n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Lexicographical Numbers
* Difficulty: Medium
* Tags: graph, sort, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<int> LexicalOrder(int n) {

}

}
```

C Solution:

```
/*
* Problem: Lexicographical Numbers
* Difficulty: Medium
* Tags: graph, sort, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

Go Solution:

```
// Problem: Lexicographical Numbers
// Difficulty: Medium
// Tags: graph, sort, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func lexicalOrder(n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun lexicalOrder(n: Int): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographical Numbers
// Difficulty: Medium
// Tags: graph, sort, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def lexicalOrder(n: Int): List[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (lexical-order n)
(-> exact-integer? (listof exact-integer?))

)
```