Problem 31: Next Permutation

Problem Information

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

Problem Description

A

permutation

of an array of integers is an arrangement of its members into a sequence or linear order.

For example, for

arr = [1,2,3]

, the following are all the permutations of

arr

:

[1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]

.

The

next permutation

of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their

lexicographical order, then the

next permutation

of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

For example, the next permutation of

arr = [1,2,3]

is

[1,3,2]

.

Similarly, the next permutation of

arr = [2,3,1]

is

[3,1,2]

.

While the next permutation of

arr = [3,2,1]

is

[1,2,3]

because

[3,2,1]

does not have a lexicographical larger rearrangement.

Given an array of integers

nums

,

find the next permutation of

nums

.

The replacement must be

in place

and use only constant extra memory.

Example 1:

Input:

nums = [1,2,3]

Output:

[1,3,2]

Example 2:

Input:

nums = [3,2,1]

Output:

[1,2,3]

Example 3:

Input:

nums = [1,1,5]

Output:

[1,5,1]

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public void nextPermutation(int[] nums) {

}
}
```

Python3:

```
class Solution:
def nextPermutation(self, nums: List[int]) -> None:
"""
Do not return anything, modify nums in-place instead.
"""
```

Python:

```
class Solution(object):
def nextPermutation(self, nums):
"""
:type nums: List[int]
:rtype: None Do not return anything, modify nums in-place instead.
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {void} Do not return anything, modify nums in-place instead.
*/
var nextPermutation = function(nums) {

};
```

TypeScript:

```
/**
Do not return anything, modify nums in-place instead.
*/
function nextPermutation(nums: number[]): void {

};
```

C#:

```
public class Solution {
public void NextPermutation(int[] nums) {

}
}
```

C:

```
void nextPermutation(int* nums, int numsSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun nextPermutation(nums: IntArray): Unit {

}
}
```

Swift:

```
class Solution {
func nextPermutation(_ nums: inout [Int]) {

}
}
```

Rust:

```
impl Solution {
pub fn next_permutation(nums: &mut Vec<i32>) {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Void} Do not return anything, modify nums in-place instead.
def next_permutation(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return NULL
*/
function nextPermutation(&$nums) {

}
}
```

Dart:

```
class Solution {
void nextPermutation(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Next Permutation
* Difficulty: Medium
* Tags: array, graph, 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:
void nextPermutation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Next Permutation
* Difficulty: Medium
* Tags: array, graph, 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 void nextPermutation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Next Permutation
Difficulty: Medium
Tags: array, graph, 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 nextPermutation(self, nums: List[int]) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def nextPermutation(self, nums):
"""
:type nums: List[int]
:rtype: None Do not return anything, modify nums in-place instead.
"""
```

JavaScript Solution:

```
/**
* Problem: Next Permutation
* Difficulty: Medium
* Tags: array, graph, 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 {void} Do not return anything, modify nums in-place instead.
*/
var nextPermutation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Next Permutation
* Difficulty: Medium
* Tags: array, graph, 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
*/

/**
```

```
Do not return anything, modify nums in-place instead.
*/
function nextPermutation(nums: number[]): void {

};
```

C# Solution:

```
/*
* Problem: Next Permutation
* Difficulty: Medium
* Tags: array, graph, 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 void NextPermutation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Next Permutation
* Difficulty: Medium
* Tags: array, graph, 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
*/

void nextPermutation(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Next Permutation
// Difficulty: Medium
// Tags: array, graph, 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 nextPermutation(nums []int) {

}
```

Kotlin Solution:

```
class Solution {
fun nextPermutation(nums: IntArray): Unit {

}
}
```

Swift Solution:

```
class Solution {
func nextPermutation(_ nums: inout [Int]) {

}
}
```

Rust Solution:

```
// Problem: Next Permutation
// Difficulty: Medium
// Tags: array, graph, 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 next_permutation(nums: &mut Vec<i32>) {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Void} Do not return anything, modify nums in-place instead.
def next_permutation(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return NULL
*/
function nextPermutation(&$nums) {

}
}
```

Dart Solution:

```
class Solution {
void nextPermutation(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```