Problem 88: Merge Sorted Array

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

, sorted in

non-decreasing order

, and two integers

m

and

n

, representing the number of elements in

nums1

and

nums2

respectively.

Merge

nums1

and

nums2

into a single array sorted in

non-decreasing order

.

The final sorted array should not be returned by the function, but instead be

stored inside the array

nums1

. To accommodate this,

nums1

has a length of

m + n

, where the first

m

elements denote the elements that should be merged, and the last

n

elements are set to

0

and should be ignored.

nums2

has a length of

n

.

Example 1:

Input:

nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

Output:

[1,2,2,3,5,6]

Explanation:

The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [

1

,

2

,2,

3

,5,6] with the underlined elements coming from nums1.

Example 2:

Input:

nums1 = [1], m = 1, nums2 = [], n = 0

Output:

[1]

Explanation:

The arrays we are merging are [1] and []. The result of the merge is [1].

Example 3:

Input:

nums1 = [0], m = 0, nums2 = [1], n = 1

Output:

[1]

Explanation:

The arrays we are merging are [] and [1]. The result of the merge is [1]. Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

Constraints:

nums1.length == m + n

nums2.length == n

0 <= m, n <= 200

1 <= m + n <= 200

-10

9

<= nums1[i], nums2[j] <= 10

9

Follow up:

Can you come up with an algorithm that runs in

O(m + n)

time?

Code Snippets

C++:

```
class Solution {
public:
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

}
};
```

Java:

```
class Solution {
public void merge(int[] nums1, int m, int[] nums2, int n) {

}
}
```

Python3:

```
class Solution:
def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
"""
Do not return anything, modify nums1 in-place instead.
"""
```

Python:

```
class Solution(object):
def merge(self, nums1, m, nums2, n):
"""
:type nums1: List[int]
:type m: int
:type nums2: List[int]
:type n: int
:rtype: None Do not return anything, modify nums1 in-place instead.
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number} m
* @param {number[]} nums2
* @param {number} n
* @return {void} Do not return anything, modify nums1 in-place instead.
*/
var merge = function(nums1, m, nums2, n) {

};
```

TypeScript:

```
/**
Do not return anything, modify nums1 in-place instead.
*/
function merge(nums1: number[], m: number, nums2: number[], n: number): void
{

};
```

C#:

```
public class Solution {
public void Merge(int[] nums1, int m, int[] nums2, int n) {

}
}
```

C:

```
void merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size, int
n) {

}
```

Go:

```
func merge(nums1 []int, m int, nums2 []int, n int) {

}
```

Kotlin:

```
class Solution {
fun merge(nums1: IntArray, m: Int, nums2: IntArray, n: Int): Unit {

}
}
```

Swift:

```
class Solution {
func merge(_ nums1: inout [Int], _ m: Int, _ nums2: [Int], _ n: Int) {

}
}
```

Rust:

```
impl Solution {
pub fn merge(nums1: &mut Vec<i32>, m: i32, nums2: &mut Vec<i32>, n: i32) {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer} m
# @param {Integer[]} nums2
# @param {Integer} n
# @return {Void} Do not return anything, modify nums1 in-place instead.
def merge(nums1, m, nums2, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer $m
* @param Integer[] $nums2
* @param Integer $n
* @return NULL
*/
function merge(&$nums1, $m, $nums2, $n) {

}
}
```

Dart:

```
class Solution {
void merge(List<int> nums1, int m, List<int> nums2, int n) {

}
}
```

Scala:

```
object Solution {
def merge(nums1: Array[Int], m: Int, nums2: Array[Int], n: Int): Unit = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Sorted Array
* Difficulty: Easy
* Tags: array, 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 merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Sorted Array
* Difficulty: Easy
* Tags: array, 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 merge(int[] nums1, int m, int[] nums2, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge Sorted Array
Difficulty: Easy
Tags: array, 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 merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def merge(self, nums1, m, nums2, n):
"""
:type nums1: List[int]
:type m: int
:type nums2: List[int]
:type n: int
:rtype: None Do not return anything, modify nums1 in-place instead.
"""
```

JavaScript Solution:

```
/**
* Problem: Merge Sorted Array
* Difficulty: Easy
* Tags: array, 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[]} nums1
* @param {number} m
* @param {number[]} nums2
* @param {number} n
* @return {void} Do not return anything, modify nums1 in-place instead.
*/
var merge = function(nums1, m, nums2, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Sorted Array
* Difficulty: Easy
* Tags: array, 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 nums1 in-place instead.
*/
function merge(nums1: number[], m: number, nums2: number[], n: number): void
{

};
```

C# Solution:

```
/*
* Problem: Merge Sorted Array
* Difficulty: Easy
* Tags: array, 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 Merge(int[] nums1, int m, int[] nums2, int n) {

}
}
```

C Solution:

```
/*
* Problem: Merge Sorted Array
* Difficulty: Easy
* Tags: array, 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 merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size, int
n) {

}
```

Go Solution:

```
// Problem: Merge Sorted Array
// Difficulty: Easy
// Tags: array, 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 merge(nums1 []int, m int, nums2 []int, n int) {

}
```

Kotlin Solution:

```
class Solution {
fun merge(nums1: IntArray, m: Int, nums2: IntArray, n: Int): Unit {

}
}
```

Swift Solution:

```
class Solution {
func merge(_ nums1: inout [Int], _ m: Int, _ nums2: [Int], _ n: Int) {

}
}
```

Rust Solution:

```
// Problem: Merge Sorted Array
// Difficulty: Easy
// Tags: array, 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 merge(nums1: &mut Vec<i32>, m: i32, nums2: &mut Vec<i32>, n: i32) {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer} m
# @param {Integer[]} nums2
# @param {Integer} n
# @return {Void} Do not return anything, modify nums1 in-place instead.
def merge(nums1, m, nums2, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer $m
* @param Integer[] $nums2
* @param Integer $n
* @return NULL
*/
function merge(&$nums1, $m, $nums2, $n) {

}
}
```

Dart Solution:

```
class Solution {
void merge(List<int> nums1, int m, List<int> nums2, int n) {

}
}
```

Scala Solution:

```
object Solution {
def merge(nums1: Array[Int], m: Int, nums2: Array[Int], n: Int): Unit = {

}
}
```