Problem 1537: Get the Maximum Score

Problem Information

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

Problem Description

You are given two

sorted

arrays of distinct integers

nums1

and

nums2

.

A

valid

path

is defined as follows:

Choose array

nums1

or

nums2

to traverse (from index-0).

Traverse the current array from left to right.

If you are reading any value that is present in

nums1

and

nums2

you are allowed to change your path to the other array. (Only one repeated value is considered in the valid path).

The

score

is defined as the sum of unique values in a valid path.

Return

the maximum score you can obtain of all possible

valid paths

. Since the answer may be too large, return it modulo

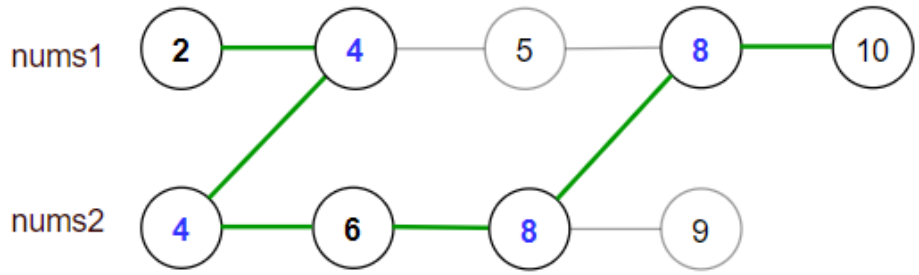
10

9

+ 7

.

Example 1:



Input:

nums1 = [2,4,5,8,10], nums2 = [4,6,8,9]

Output:

30

Explanation:

Valid paths: [2,4,5,8,10], [2,4,5,8,9], [2,4,6,8,9], [2,4,6,8,10], (starting from nums1) [4,6,8,9], [4,5,8,10], [4,5,8,9], [4,6,8,10] (starting from nums2) The maximum is obtained with the path in green

[2,4,6,8,10]

.

Example 2:

Input:

nums1 = [1,3,5,7,9], nums2 = [3,5,100]

Output:

109

Explanation:

Maximum sum is obtained with the path

[1,3,5,100]

.

Example 3:

Input:

nums1 = [1,2,3,4,5], nums2 = [6,7,8,9,10]

Output:

40

Explanation:

There are no common elements between nums1 and nums2. Maximum sum is obtained with the path [6,7,8,9,10].

Constraints:

1 <= nums1.length, nums2.length <= 10

5

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

7

nums1

and

nums2

are strictly increasing.

Code Snippets

C++:

```
class Solution {
public:
int maxSum(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int maxSum(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def maxSum(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxSum(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
```

```
 * @return {number}
 */
var maxSum = function(nums1, nums2) {

};
```

TypeScript:

```
function maxSum(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxSum(int[] nums1, int[] nums2) {

}
}
```

C:

```
int maxSum(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func maxSum(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun maxSum(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxSum(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def max_sum(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function maxSum($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int maxSum(List<int> nums1, List<int> nums2) {

}
```

```
}
```

Scala:

```
object Solution {
def maxSum(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sum(nums1 :: [integer], nums2 :: [integer]) :: integer
def max_sum(nums1, nums2) do

end
end
```

Erlang:

```
-spec max_sum(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
max_sum(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (max-sum nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Get the Maximum Score
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxSum(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Get the Maximum Score
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxSum(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Get the Maximum Score
Difficulty: Hard
Tags: array, dp, greedy, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""
```

```
class Solution:
def maxSum(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxSum(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Get the Maximum Score
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var maxSum = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Get the Maximum Score
```

```
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxSum(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Get the Maximum Score
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxSum(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Get the Maximum Score
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
int maxSum(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Get the Maximum Score
// Difficulty: Hard
// Tags: array, dp, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxSum(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSum(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSum(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Get the Maximum Score
// Difficulty: Hard
// Tags: array, dp, greedy, sort
//
```

```
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def max_sum(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function maxSum($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int maxSum(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def maxSum(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sum(nums1 :: [integer], nums2 :: [integer]) :: integer
def max_sum(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec max_sum(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
max_sum(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (max-sum nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```