Problem 891: Sum of Subsequence Widths

Problem Information

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

Problem Description

The

width

of a sequence is the difference between the maximum and minimum elements in the sequence.

Given an array of integers

nums

, return

the sum of the

widths

of all the non-empty

subsequences

of

nums

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

A

subsequence

is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example,

[3,6,2,7]

is a subsequence of the array

[0,3,1,6,2,2,7]

.

Example 1:

Input:

nums = [2,1,3]

Output:

6 Explanation: The subsequences are [1], [2], [3], [2,1], [2,3], [1,3], [2,1,3]. The corresponding widths are 0, 0, 0, 1, 1, 2, 2. The sum of these widths is 6.

Example 2:

Input:

nums = [2]

Output:

0

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int sumSubseqWidths(int[] nums) {

}
}
```

Python3:

```
class Solution:
def sumSubseqWidths(self, nums: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sumSubseqWidths(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int SumSubseqWidths(int[] nums) {

}
}
```

C:

```
int sumSubseqWidths(int* nums, int numsSize) {

}
```

Go:

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

```
}
```

Kotlin:

```
class Solution {
fun sumSubseqWidths(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func sumSubseqWidths(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn sum_subseq_widths(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def sum_subseq_widths(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
```

```
function sumSubseqWidths($nums) {

}
}
```

Dart:

```
class Solution {
int sumSubseqWidths(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_subseq_widths(nums :: [integer]) :: integer
def sum_subseq_widths(nums) do

end
end
```

Erlang:

```
-spec sum_subseq_widths(Nums :: [integer()]) -> integer().
sum_subseq_widths(Nums) ->
.
```

Racket:

```
(define/contract (sum-subseq-widths nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Subsequence Widths
* Difficulty: Hard
* Tags: array, math, 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:
int sumSubseqWidths(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Subsequence Widths
* Difficulty: Hard
* Tags: array, math, 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 int sumSubseqWidths(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Subsequence Widths
Difficulty: Hard
Tags: array, math, 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 sumSubseqWidths(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Subsequence Widths
* Difficulty: Hard
* Tags: array, math, 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 {number}
*/
var sumSubseqWidths = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Sum of Subsequence Widths
* Difficulty: Hard
* Tags: array, math, 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
*/

function sumSubseqWidths(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Subsequence Widths
* Difficulty: Hard
* Tags: array, math, 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 int SumSubseqWidths(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Subsequence Widths
* Difficulty: Hard
```

```
* Tags: array, math, 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
*/

int sumSubseqWidths(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Subsequence Widths
// Difficulty: Hard
// Tags: array, math, 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 sumSubseqWidths(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sumSubseqWidths(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func sumSubseqWidths(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Subsequence Widths
// Difficulty: Hard
// Tags: array, math, 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 sum_subseq_widths(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def sum_subseq_widths(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function sumSubseqWidths($nums) {

}
}
```

Dart Solution:

```
class Solution {
int sumSubseqWidths(List<int> nums) {
```

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_subseq_widths(nums :: [integer]) :: integer
def sum_subseq_widths(nums) do

end
end
```

Erlang Solution:

```
-spec sum_subseq_widths(Nums :: [integer()]) -> integer().
sum_subseq_widths(Nums) ->
.
```

Racket Solution:

```
(define/contract (sum-subseq-widths nums)
(-> (listof exact-integer?) exact-integer?)
)
```