Problem 907: Sum of Subarray Minimums

Problem Information

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

Problem Description

Given an array of integers arr, find the sum of

min(b)

, where

b

ranges over every (contiguous) subarray of

arr

. Since the answer may be large, return the answer

modulo

10

9

+ 7

.

Example 1:

Input:

arr = [3,1,2,4]

Output:

17

Explanation:

Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2], [1,2,4], [3,1,2,4]. Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1. Sum is 17.

Example 2:

Input:

arr = [11,81,94,43,3]

Output:

444

Constraints:

1 <= arr.length <= 3 * 10

4

1 <= arr[i] <= 3 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int sumSubarrayMins(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int sumSubarrayMins(int[] arr) {

}
}
```

Python3:

```
class Solution:
def sumSubarrayMins(self, arr: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sumSubarrayMins(arr: number[]): number {
```

```
};
```

C#:

```
public class Solution {
public int SumSubarrayMins(int[] arr) {

}
}
```

C:

```
int sumSubarrayMins(int* arr, int arrSize) {

}
```

Go:

```
func sumSubarrayMins(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun sumSubarrayMins(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func sumSubarrayMins(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn sum_subarray_mins(arr: Vec<i32>) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def sum_subarray_mins(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function sumSubarrayMins($arr) {

}
}
```

Dart:

```
class Solution {
int sumSubarrayMins(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def sumSubarrayMins(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_subarray_mins(arr :: [integer]) :: integer
def sum_subarray_mins(arr) do

end
end
```

Erlang:

```
-spec sum_subarray_mins(Arr :: [integer()]) -> integer().
sum_subarray_mins(Arr) ->
.
```

Racket:

```
(define/contract (sum-subarray-mins arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Subarray Minimums
* Difficulty: Medium
* Tags: array, dp, stack
*
* 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 sumSubarrayMins(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Subarray Minimums
* Difficulty: Medium
* Tags: array, dp, stack
*
* 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 sumSubarrayMins(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Subarray Minimums
Difficulty: Medium
Tags: array, dp, stack

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 sumSubarrayMins(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Subarray Minimums
* Difficulty: Medium
* Tags: array, dp, stack
*
* 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[]} arr
* @return {number}
*/
var sumSubarrayMins = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Subarray Minimums
* Difficulty: Medium
* Tags: array, dp, stack
*
* 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 sumSubarrayMins(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Subarray Minimums
* Difficulty: Medium
* Tags: array, dp, stack
*
```

```
* 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 SumSubarrayMins(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Subarray Minimums
* Difficulty: Medium
* Tags: array, dp, stack
*
* 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 sumSubarrayMins(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Sum of Subarray Minimums
// Difficulty: Medium
// Tags: array, dp, stack
//
// 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 sumSubarrayMins(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sumSubarrayMins(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func sumSubarrayMins(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Subarray Minimums
// Difficulty: Medium
// Tags: array, dp, stack
//
// 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 sum_subarray_mins(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def sum_subarray_mins(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function sumSubarrayMins($arr) {

}
}
```

Dart Solution:

```
class Solution {
int sumSubarrayMins(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def sumSubarrayMins(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_subarray_mins(arr :: [integer]) :: integer
def sum_subarray_mins(arr) do

end
end
```

Erlang Solution:

```
-spec sum_subarray_mins(Arr :: [integer()]) -> integer().
sum_subarray_mins(Arr) ->
.
```

Racket Solution:

```
(define/contract (sum-subarray-mins arr)
(-> (listof exact-integer?) exact-integer?)
)
```