**Sum of Subarray Minimums**

**Medium**

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** 109 + 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 \* 104
* 1 <= arr[i] <= 3 \* 104

class Solution {

public:

int sumSubarrayMins(vector<int>& A) {

        int n=A.size(), mod=1e9+7;

        int left[n], right[n];

        stack<pair<int, int> > s1, s2;

        // getting number of element strictly larger

        // than A[i] on Left.

        for (int i = 0; i < n; ++i) {

            int cnt = 1;

            // get elements from stack until element

            // greater than A[i] found

            while (!s1.empty() && (s1.top().first) > A[i]) {

                cnt += s1.top().second;

                s1.pop();

            }

            s1.push({ A[i], cnt });

            left[i] = cnt;

        }

        // getting number of element larger than A[i] on Right.

        for (int i = n - 1; i >= 0; --i) {

            int cnt = 1;

            // get elements from stack until element greater

            // or equal to A[i] found

            while (!s2.empty() && (s2.top().first) >= A[i]) {

                cnt += s2.top().second;

                s2.pop();

            }

            s2.push({ A[i], cnt });

            right[i] = cnt;

        }

        int64\_t result = 0;

        // calculating required resultult

        for (int i = 0; i < n; ++i) {

            result+=((int64\_t)A[i] \* (int64\_t)left[i] \* (int64\_t)right[i]);

            result=(result%mod+mod)%mod;

        }

        return result;

    }

};