Link:

https://leetcode.com/problems/sum-of-subarray-minimums/discuss/178876/stack-solution-with-very-detailed-explanation-step-by-step

Before diving into the solution, we first introduce a very important stack type, which is called **monotone stack** .

What is monotonous increase stack?

Roughly speaking, the elements in the an monotonous increase stack keeps an increasing order.

The typical paradigm for monotonous increase stack:

```
for(int i = 0; i < A.size(); i++){

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

    in_stk.pop();

  }

  in_stk.push(A[i]);

}
```

What can monotonous increase stack do?

(1) find the **previous less** element of each element in a vector **with O(n) time**:

- What is the previous less element of an element?
  For example:
  [3, 7, 8, 4]
  The previous less element of 7 is 3.
  The previous less element of 8 is 7.
  The previous less element of 4 is 3.
  There is no previous less element for 3.

For simplicity of notation, we use abbreviation **PLE** to denote **P**revious **L**ess **E**lement.

- C++ code (by slitghly modifying the paradigm):
  Instead of directly pushing the element itself, here for simplicity, we push the **index**.
  We do some record when the index is pushed into the stack.

```
// previous_less[i] = j means A[j] is the previous less element of A[i].

// previous_less[i] = -1 means there is no previous less element of A[i].

vector<int> previous_less(A.size(), -1);

for(int i = 0; i < A.size(); i++){

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

    in_stk.pop();

  }

  previous_less[i] = in_stk.empty()? -1: in_stk.top();

  in_stk.push(i);

}
```

(2) find the **next less** element of each element in a vector with **O(n) time**:

- What is the next less element of an element?
  For example:
  [3, 7, 8, 4]

The next less element of 8 is 4.
The next less element of 7 is 4.
There is no next less element for 3 and 4.

For simplicity of notation, we use abbreviation **NLE** to denote **N**ext **L**ess **E**lement.

- C++ code (by slighly modifying the paradigm):
  We do some record when the index is poped out from the stack.

```
// next_less[i] = j means A[j] is the next less element of A[i].

// next_less[i] = -1 means there is no next less element of A[i].

vector<int> previous_less(A.size(), -1);

for(int i = 0; i < A.size(); i++){

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

    auto x = in_stk.top(); in_stk.pop();

    next_less[x] = i;

  }

  in_stk.push(i);

}
```

How can the monotonous increase stack be applied to this problem?

For example:
Consider the element `3` in the following vector:

```
                        [2, 9, 7, 8, 3, 4, 6, 1]

                            |                 |

                  the previous less      the next less

                     element of 3         element of 3
```

After finding both **NLE** and **PLE** of `3`, we can determine the
distance between `3` and `2`(previous less) , and the distance between `3` and `1`(next less).
In this example, the distance is `4` and `3` respectively.

How many subarrays with 3 being its minimum value?
The answer is `4*3`.

```
9 7 8 3

9 7 8 3 4

9 7 8 3 4 6

7 8 3

7 8 3 4

7 8 3 4 6

8 3

8 3 4

8 3 4 6

3

3 4
```

```
3 4 6
```

How much the element 3 contributes to the final answer?
It is `3*(4*3)`.
What is the final answer?
Denote by `left[i]` the distance between element `A[i]` and its **PLE**.
Denote by `right[i]` the distance between element `A[i]` and its **NLE**.

The final answer is,
`sum(A[i]*left[i]*right[i] )`

The solution (One pass)

```
class Solution {

public:

  int sumSubarrayMins(vector<int>& A) {

    stack<pair<int, int>> in_stk_p, in_stk_n;

    // left is for the distance to previous less element

    // right is for the distance to next less element

    vector<int> left(A.size()), right(A.size());


    //initialize

    for(int i = 0; i < A.size(); i++) left[i] =  i + 1;

    for(int i = 0; i < A.size(); i++) right[i] = A.size() - i;


    for(int i = 0; i < A.size(); i++){

      // for previous less

      while(!in_stk_p.empty() && in_stk_p.top().first > A[i]) in_stk_p.pop();

      left[i] = in_stk_p.empty()? i + 1: i - in_stk_p.top().second;

      in_stk_p.push({A[i],i});


      // for next less

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

        auto x = in_stk_n.top();in_stk_n.pop();

        right[x.second] = i - x.second;

      }

      in_stk_n.push({A[i], i});

    }


    int ans = 0, mod = 1e9 +7;

    for(int i = 0; i < A.size(); i++){

      ans = (ans + A[i]*left[i]*right[i])%mod;

    }

    return ans;
```

```
  }

};
```

The last thing that needs to be mentioned for handling duplicate elements:

Method: Set **strict less** and **non-strict less**(less than **or equal to**) for finding **NLE** and **PLE** respectively. The order doesn't matter.

For example, the above code for finding **NLE** is **strict less**, while **PLE** is actually **non-strict less**.
Remark: Although in both loop conditions the signs are set as `>`, for NLE, we make records **inside** the loop, while for PLE, records are done **outside** the loop.

More:

- What can monotonous **decrease** stack do?
- Some applications of monotone (increase/decrease) stack in leetcode:
  Next Greater Element II (a very basic one)
  Largest Rectangle in Histogram(almost the same as this problem)
  Maximal Rectangle(please do this problem after you solve the above one)
  Trapping Rain Water (challenge)
  Remove Duplicate Letters(challenge)
  Remove K Digits
  Create Maximum Number
  132 Pattern(challenge, instead of focusing on the elements in the stack, this problem focuses on the elements poped from the monotone stack)
  sliding window maximum(challenge, monotone **queue**)
  Max Chunks To Make Sorted II

Hope this helps.