<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](https://leetcode.com/problems/Next-Greater-Element-II/description/) (a very basic one)  
  [Largest Rectangle in Histogram](https://leetcode.com/problems/Largest-Rectangle-in-Histogram/description/)(almost the same as this problem)  
  [Maximal Rectangle](https://leetcode.com/problems/Maximal-Rectangle/description/)(please do this problem after you solve the above one)  
  [Trapping Rain Water](https://leetcode.com/problems/Trapping-Rain-Water/description/) (challenge)  
  [Remove Duplicate Letters](https://leetcode.com/problems/remove-duplicate-letters/description/)(challenge)  
  [Remove K Digits](https://leetcode.com/problems/remove-k-digits/description/)  
  [Create Maximum Number](https://leetcode.com/problems/create-maximum-number/description/)  
  [132 Pattern](https://leetcode.com/problems/132-pattern/description/)(challenge, instead of focusing on the elements in the stack, this problem focuses on the elements poped from the monotone stack)  
  [sliding window maximum](https://leetcode.com/problems/sliding-window-maximum/description/)(challenge, monotone **queue**)  
  [Max Chunks To Make Sorted II](https://leetcode.com/problems/Max-Chunks-To-Make-Sorted-II/description/)

Java Version:

Best test: [2, 9, 7, 8, 3, 4, 6, 1]

class Solution {

public int sumSubarrayMins(int[] arr) {

int n = arr.length;

int[] left = new int[n];

int[] right = new int[n];

Stack<int[]> ple = new Stack<int[]>();

Stack<int[]> nle = new Stack<int[]>();

for(int i = 0; i < n; i++) {

// In PLE use ">=" to deal with duplicate elements, non-strict less

while(!ple.isEmpty() && ple.peek()[0] >= arr[i]) {

ple.pop();

}

left[i] = ple.isEmpty() ? i + 1 : i - ple.peek()[1];

ple.push(new int[] {arr[i], i});

}

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

// Since PLE handle duplicate elements, no need for NLE to handle again, strict less

while(!nle.isEmpty() && nle.peek()[0] > arr[i]) {

nle.pop();

}

right[i] = nle.isEmpty() ? n - i : nle.peek()[1] - i;

nle.push(new int[] {arr[i], i});

}

int mod = (int)1e9 + 7;

long sum = 0;

for(int i = 0; i < n; i++) {

sum = (sum + (long)arr[i] \* left[i] \* right[i]) % mod;

}

return (int)sum;

}

}