Strength definition: min(subarray) * Sum(subarray)

Intuition:

Find subarrays using the same method as shown in 8/26/2022-2104: with each element in total array, find subarrays in which it is the smallest element.

Here, we will use increasing stack to get PLE and NLE

- minimum element is already known.
- How to find sum?

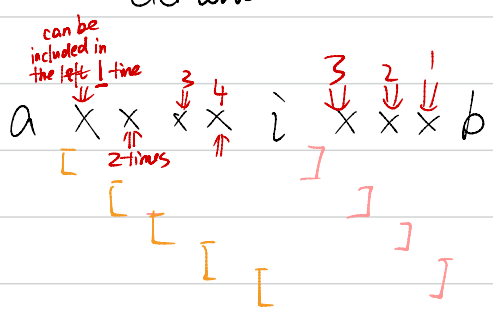
eg. a x x x x i x x x b

Say i is smallest element, a is PLE, b is its NLE

there are (i-a) positions for left boundary, let x = i-a

(b-i) positions for right boundary, let y = b-i

∴ in total we can find x*y subarrays with i as smallest element



∴ Total Sum

= ① (nums[a+1]×1 + nums[a+2]×2 + nums[a+3]×3 + nums[a+4]×4) * y

+ ② (nums[i+1]×3 + nums[i+2]×2 + nums[i+3]×1) * x

+ nums[i] * x * y

part ①:

$(nums[a+1]\times 1 + nums[a+2]\times 2 + nums[a+3]\times 3 + nums[a+4]\times 4) * y$

coeff 2 index

can define: $presum2 = \sum_{1}^{n} nums[i] \cdot i$

then $presum2[i-1] - presum[a]$

$= nums[a+1](a+1) + nums[a+2](a+2) + \dots + nums[i-1]\cdot(i-1)$

just need to subtract $\sum_{j=a+1}^{i} nums[j] \times a$ to get we want (what)

↓

can define $presum = \sum_{1}^{n} nums[i]$ and use $(presum[i-1] - presum[a])$

part ②

$(nums[i+1]\times 3 + nums[i+2]\times 2 + nums[i+3]\times 1) \times x$

do same trick but in reverse

$(presum[b-1] - presum[i]) \times b$

① $= nums[i+1]\times b + nums[i+2]\times b + nums[i+3]\times b$

$presum2[b-1] - presum2[i]$

② $= nums[i+1]\times(i+1) + nums[i+2](i+2) + nums[i+3]\times(i+3)$

① - ② ⇒ done

```cpp
using LL = long long;
LL M = 1e9+7;
class Solution {
public:
    int totalStrength(vector<int>& nums) {
        int n = nums.size();
        nums.insert(nums.begin(),0);

        vector<LL> presum(n+2,0);
        for(int i = 1; i <=n ; i++) presum[i] = (presum[i-1] + nums[i]) % M;


        vector<LL> presum2(n+2,0);
        for(int i = 1; i <=n ; i++) presum2[i] = (presum2[i-1] + (LL)nums[i]*i) % M;


        stack<int> stack;
        vector<int> nextSmaller(n+2, n+1);
        vector<int> prevSmaller(n+2, 0);
        for(int i = 1; i <=n ; i++)
        {
            while(!stack.empty() && nums[stack.top()] > nums[i])
            {
                nextSmaller[stack.top()] = i;
                stack.pop();
            }
            if(!stack.empty()) prevSmaller[i] = stack.top();
            stack.push(i);
        }

        LL result = 0;
        for(int i = 1; i <= n; i++)
        {
            LL a = prevSmaller[i], b = nextSmaller[i];
            LL x = i-a, y = b-i;
            LL left = ( (presum2[i-1] - presum2[a]) - (presum[i-1] -presum[a])*a % M + M ) % M;
            left = left * y % M;
            LL right =( (presum[b-1] - presum[i])*b % M - (presum2[b-1] -presum2[i]) + M ) % M;
            right = right * x % M;
            LL mid = (LL)nums[i] * x * y % M;
            result = ( result + (left + mid + right) * nums[i] ) % M;

        }

        return result;


    }
};
```