Number of Subarrays with Bounded Maximum

We are given an array `A` of positive integers, and two positive integers `L` and `R` (`L <= R`).

Return the number of (contiguous, non-empty) subarrays such that the value of the maximum array element in that subarray is at least `L` and at most `R`.

```
Example :
Input:
A = [2, 1, 4, 3]
L = 2
R = 3
Output: 3
Explanation: There are three subarrays that meet the requirements: [2], [2, 1], [3].
```

Note:

- L, R and `A[i]` will be an integer in the range `[0, 10^9]`.
- The length of `A` will be in the range of `[1, 50000]`.

Solution 1

```
class Solution {
    public int numSubarrayBoundedMax(int[] A, int L, int R) {
        int j=0,count=0,res=0;

        for(int i=0;i<A.length;i++){
            if(A[i]>=L && A[i]<=R){
                res+=i-j+1;count=i-j+1;
            }
            else if(A[i]<L){
                res+=count;
            }
            else{
                j=i+1;
                count=0;
            }
        }
        return res;
    }
}
```

written by kay_deep original link here

Solution 2

```
class Solution {
public:
    int numSubarrayBoundedMax(vector<int>& A, int L, int R) {
        int result=0, left=-1, right=-1;
        for (int i=0; i<A.size(); i++) {
            if (A[i]>R) left=i;
            if (A[i]>=L) right=i;
            result+=right-left;
        }
        return result;
    }
};
```

written by JohnsonTau original link here

Solution 3

The idea is to keep track of 3 things while iterating: the number of valid subarrays (`res`), the number of valid subarray starting points (`heads`), and the number of not-yet valid starting points (`tails`).

- Values between `L` and `R` are heads. Every contiguous value afterwards that is less than R afterwards can extend them, creating new subarrays.
- Values less than `L` are tails. If they connect to a head later in the array, they become a head for valid subarrays.
- Values greater than `R` are combo breakers. They stop all heads and tails from forming from subarrays.

Therefore, we keep a rolling count of valid subarrays as we iterate through `A` , the main array.

- If a `head` is encountered, it joins the existing heads to form subarrays at each iteration. All `tails` are promoted to heads. All existing heads create a new valid subarray.
    - The new head creates subarray of a single element ([head])
    - Each promoted head creates subarrays from its tail index to current index (e.g. [tail1, tail2, head, ...], encountering head promotes tail1 and tail2 to heads and creates [tail1, tail2, head] and [tail2, head])
- If a tail is encountered, all existing heads can create another subarray with it. The tail remains useless until it encounters a head (see above).
- If a combo breaker is met, all existing heads and tails become useless, and are reset to 0.

Counts of new subarrays (i.e. head count) are added to `res` at each iteration, if valid.

```
class Solution {
public:
    int numSubarrayBoundedMax(vector<int>& A, int L, int R) {
        int res = 0, heads = 0, tails = 0;
        for (int val : A) {
            if (L <= val && val <= R) {
                // val is a head. All tails promoted to heads
                heads+= tails + 1;
                tails = 0;
                res += heads;
            }
            else if (val < L) {
                // val is a tail, can extend existing subarrays
                tails++;
                res += heads;
            }
            else {
                // combo breaker
                heads = 0;
                tails = 0;
            }
        }
        return res;
    }
};
```

written by nickyh original link here