Question :

Count the number of mountain subarrays(length.>=3) in the array

1<=N<=100000

1<=A[i]<=100000000

[1 2 4 2 1]

O/p-> [1 2 4 2]

[2 4 2]

[2 4 2 1]

[1 2 4 2 1]



Prerequesties :

Q1) Find the longest increasing subarray in the array of size “N”

class Solution {

    public int increasingSubarray(int [] nums) {

        int max = 1;

        int n = nums.length;

        int [] prefix = new int[n];

        prefix[0] = nums[0];

        for(int i = 1 ; i < n ; i++){

            if(nums[i] > nums[i - 1]){

                prefix[i] = prefix[i - 1] + 1;

            }else{

                prefix[i] = 1;

            }

        }

        for(int i : prefix){

            max = Math.max(i , max);

        }

Return max;

    }

}

Q2) Find the longest decreasing subarray in the array of size “N”

class Solution {

    public int decreasingSubarray(int [] nums) {

        int max = 1;

        int n = nums.length;

        int [] suffix = new int[n];

        suffix[n - 1] = nums[n - 1];

        for(int i = n- 2 ; i >= 0 ; i--){

            if(nums[i] > nums[i + 1]){

                suffix[i] = suffix[i - 1] + 1;

            }else{

                suffix[i] = 1;

            }

        }

        for(int i : suffix){

            max = Math.max(i , max);

        }

    }

}

Q3 ) Find the largest mountain array for array of size “N”

class Solution {

    public int largestMountain ARray(int [] nums) {

        int max = 1;

        int n = nums.length;

        int [] prefix = new int[n];

        int [] suffix = new int[n];

        prefix[0] = 1

        suffix[n - 1] = 1;

        for(int i = 1 ; i < n ; i++){

            if(nums[i] > nums[i - 1]){

                prefix[i] = prefix[i - 1] + 1;

            }else{

                prefix[i] = 1;

            }

        }

        for(int i = n- 2 ; i >= 0 ; i--){

            if(nums[i] > nums[i + 1]){

                suffix[i] = suffix[i - 1] + 1;

            }else{

                suffix[i] = 1;

            }

        }

        // minus 1 because we calculate the current index twice.

        for(int i = 0 ; i < n ;i++){

            max = Math.max(max  , prefix[i]+ suffix[i] -1);

        }

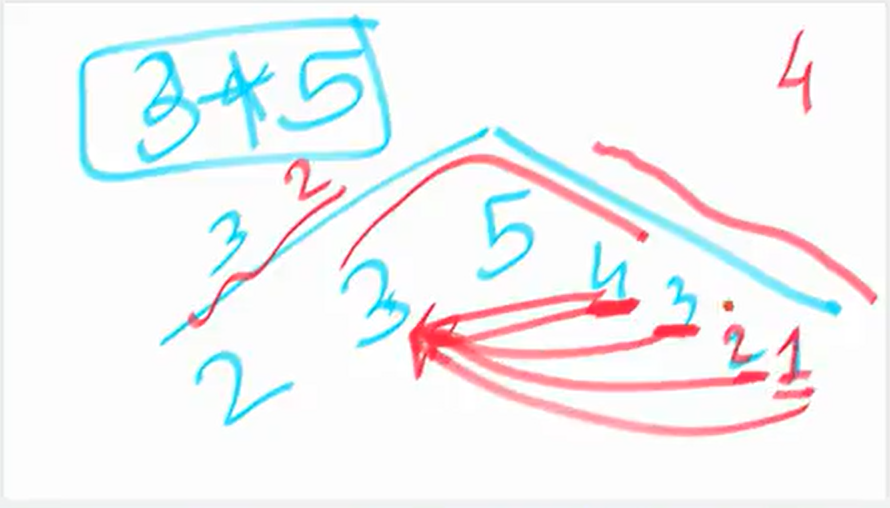
    }

}

Actual Question : Count the mountain subarray of array size “N”

Observation :

* For index , the number mountain is equal to prefix[i] – 1 \* suffix[I] – 1



In the example image , for the value 5 we have prefix value 3 & suffix value 5

The valid mountain subarrays are ,

3 , 5 , 4

3 , 5 , 4 , 3

3 , 5 , 4 , 3 , 2

3 , 5 , 4 , 3 , 2 , 1

2 , 3 , 5 , 4

2 , 3 , 5 , 4 , 3

2 , 3 , 5 , 4 , 3 , 2

2 , 3 , 5 , 4 , 3 , 2 , 1

So totally 8 subbarays are present , in other words **prefix[i] – 1 \* suffix[I] – 1**

class Solution {

    public int largestMountain ARray(int [] nums) {

        int count = 0;

        int n = nums.length;

        int [] prefix = new int[n];

        int [] suffix = new int[n];

        prefix[0] = 1

        suffix[n - 1] = 1;

        for(int i = 1 ; i < n ; i++){

            if(nums[i] > nums[i - 1]){

                prefix[i] = prefix[i - 1] + 1;

            }else{

                prefix[i] = 1;

            }

        }

        for(int i = n- 2 ; i >= 0 ; i--){

            if(nums[i] > nums[i + 1]){

                suffix[i] = suffix[i - 1] + 1;

            }else{

                suffix[i] = 1;

            }

        }

        for(int i = 0 ; i < n ;i++){

            count += ((perfix[i] - 1 ) \* (suffix[i] - 1));

        }

        return count;

    }

}