<https://leetcode.com/problems/longest-mountain-in-array/>

You may recall that an array arr is a **mountain array** if and only if:

arr.length >= 3

There exists some index i (**0-indexed**) with 0 < i < arr.length - 1 such that:

arr[0] < arr[1] < ... < arr[i - 1] < arr[i]

arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

Given an integer array arr, return *the length of the longest subarray, which is a mountain*. Return 0 if there is no mountain subarray.

**Example 1:**

Input: arr = [2,1,4,7,3,2,5]

Output: 5

Explanation: The largest mountain is [1,4,7,3,2] which has length 5.

**Example 2:**

Input: arr = [2,2,2]

Output: 0

Explanation: There is no mountain.

**Constraints:**

* 1 <= arr.length <= 104
* 0 <= arr[i] <= 104

**Follow up:**

* Can you solve it using only one pass?
* Can you solve it in O(1) space?

**Attempt 1: 2023-02-22**

**Solution 1: Two passes from left to right/right to left with O(N) space (30 min)**

class Solution {

public int longestMountain(int[] arr) {

int count = 0;

int len = arr.length;

int[] up = new int[len];

int[] down = new int[len];

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

if(i > 0 && arr[i] > arr[i - 1]) {

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

}

}

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

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

down[i] = down[i + 1] + 1;

}

if(up[i] > 0 && down[i] > 0) {

count = Math.max(count, up[i] + down[i] + 1);

}

}

return count;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Solution 2: One pass from left to right with O(1) space (30 min)**

**Style 1**

class Solution {

public int longestMountain(int[] arr) {

int count = 0;

int up = 0;

int down = 0;

int len = arr.length;

for(int i = 1; i < len; i++) {

// If we are going down and current element is greater than

// prev means last mountain ended

// And the current mountain began ('up' will be updated below)

if(down > 0 && arr[i - 1] < arr[i] || arr[i - 1] == arr[i]) {

up = 0;

down = 0;

}

// If current element is greater then previous then we are going

// up, else we are going down the mountain

if(arr[i - 1] < arr[i]) {

up++;

}

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

down++;

}

if(up > 0 && down > 0) {

count = Math.max(count, up + down + 1);

}

}

return count;

}

}

**Style 2**

class Solution {

public int longestMountain(int[] arr) {

int count = 0;

int len = arr.length;

int i = 1;

while(i < len) {

while(i < len && arr[i - 1] == arr[i]) {

i++;

}

int up = 0;

while(i < len && arr[i - 1] < arr[i]) {

i++;

up++;

}

int down = 0;

while(i < len && arr[i - 1] > arr[i]) {

i++;

down++;

}

if(up > 0 && down > 0) {

count = Math.max(count, up + down + 1);

}

}

return count;

}

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/longest-mountain-in-array/solutions/135593/c-java-python-1-pass-and-o-1-space/>

**Intuition**: We have already many 2-pass or 3-pass problems, like 821. Shortest Distance to a Character. They have almost the same idea. One forward pass and one backward pass. Maybe another pass to get the final result, or you can merge it in one previous pass.

**Explanation**: In this problem, we take one forward pass to count up hill length (to every point).We take another backward pass to count down hill length (from every point).Finally a pass to find max(up[i] + down[i] + 1) where up[i] and down[i] should be positives.

**Time Complexity**: O(N)

public int longestMountain(int[] A) {

int N = A.length, res = 0;

int[] up = new int[N], down = new int[N];

for (int i = N - 2; i >= 0; --i) if (A[i] > A[i + 1]) down[i] = down[i + 1] + 1;

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

if (i > 0 && A[i] > A[i - 1]) up[i] = up[i - 1] + 1;

if (up[i] > 0 && down[i] > 0) res = Math.max(res, up[i] + down[i] + 1);

}

return res;

}

**Follow up**

Can you solve this problem with only one pass? Can you solve this problem in O(1) space?

In this solution, I count up length and down length. Both up and down length are clear to 0 when A[i - 1] == A[i] or down > 0 && A[i - 1] < A[i].

public int longestMountain(int[] A) {

int res = 0, up = 0, down = 0;

for (int i = 1; i < A.length; ++i) {

if (down > 0 && A[i - 1] < A[i] || A[i - 1] == A[i]) up = down = 0;

if (A[i - 1] < A[i]) up++;

if (A[i - 1] > A[i]) down++;

if (up > 0 && down > 0 && up + down + 1 > res) res = up + down + 1;

}

return res;

}

**Refer to**

<https://leetcode.com/problems/longest-mountain-in-array/solutions/135593/c-java-python-1-pass-and-o-1-space/comments/189334>

int longestMountain(vector<int> A)

{

int res = 0, up = 0, down = 0;

for (int i = 1; i < A.size(); ++i)

{

//If we are going down and current element is greater than prev MEANS last mountain ended

//And the current mountain began (Up will be updated below)

if(down>0 && A[i - 1] < A[i] || A[i - 1] == A[i])

up = down = 0;

//If current element is greater then previous then we are going up

//Else we are going down the mountain

A[i] > A[i -1] ? up++ : down++;

if (up>0 && down>0)

res = max(res, up + down + 1);

}

return res;

}

**Refer to**

<https://leetcode.com/problems/longest-mountain-in-array/solutions/135593/c-java-python-1-pass-and-o-1-space/comments/143182>

Updated list of problems that involved 1 or 2 passes from left to right/right to left:

53 Maximum Subarray

121 Best Time to Buy and Sell Stock

152 Maximum Product Subarray

238 Product of Array Except Self

739 Daily Temperatures

769 Max Chunks to Make Sorted

770 Max Chunks to Make Sorted II

821 Shortest Distance to a Character

845 Longest Mountain in Array

581 Shortest Unsorted Continuous Subarray

42 Trapping Rain Water (3-pass)

896 Monotonic Array