Problem 2951: Find the Peaks

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

0-indexed

array

mountain

. Your task is to find all the

peaks

in the

mountain

array.

Return

an array that consists of

indices

of

peaks

in the given array in

any order

.

Notes:

A

peak

is defined as an element that is

strictly greater

than its neighboring elements.

The first and last elements of the array are

not

a peak.

Example 1:

Input:

mountain = [2,4,4]

Output:

[]

Explanation:

mountain[0] and mountain[2] can not be a peak because they are first and last elements of the array. mountain[1] also can not be a peak because it is not strictly greater than mountain[2]. So the answer is [].

Example 2:

Input:

mountain = [1,4,3,8,5]

Output:

[1,3]

Explanation:

mountain[0] and mountain[4] can not be a peak because they are first and last elements of the array. mountain[2] also can not be a peak because it is not strictly greater than mountain[3] and mountain[1]. But mountain [1] and mountain[3] are strictly greater than their neighboring elements. So the answer is [1,3].

Constraints:

3 <= mountain.length <= 100

1 <= mountain[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<int> findPeaks(vector<int>& mountain) {

}
};
```

Java:

```
class Solution {
public List<Integer> findPeaks(int[] mountain) {

}
}
```

Python3:

```
class Solution:
def findPeaks(self, mountain: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def findPeaks(self, mountain):
"""
:type mountain: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} mountain
* @return {number[]}
*/
var findPeaks = function(mountain) {

};
```

TypeScript:

```
function findPeaks(mountain: number[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindPeaks(int[] mountain) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPeaks(int* mountain, int mountainSize, int* returnSize) {

}
```

Go:

```
func findPeaks(mountain []int) []int {

}
```

Kotlin:

```
class Solution {
fun findPeaks(mountain: IntArray): List<Int> {

}
}
```

Swift:

```
class Solution {
func findPeaks(_ mountain: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_peaks(mountain: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} mountain
# @return {Integer[]}
```

```
def find_peaks(mountain)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $mountain
* @return Integer[]
*/
function findPeaks($mountain) {

}
}
```

Dart:

```
class Solution {
List<int> findPeaks(List<int> mountain) {

}
}
```

Scala:

```
object Solution {
def findPeaks(mountain: Array[Int]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_peaks(mountain :: [integer]) :: [integer]
def find_peaks(mountain) do

end
end
```

Erlang:

```
-spec find_peaks(Mountain :: [integer()]) -> [integer()].
find_peaks(Mountain) ->
.
```

Racket:

```
(define/contract (find-peaks mountain)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Peaks
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> findPeaks(vector<int>& mountain) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Peaks
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<Integer> findPeaks(int[] mountain) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Peaks
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findPeaks(self, mountain: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findPeaks(self, mountain):
"""
:type mountain: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Peaks
* Difficulty: Easy
```

```
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} mountain
* @return {number[]}
*/
var findPeaks = function(mountain) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Peaks
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function findPeaks(mountain: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Peaks
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class Solution {
public IList<int> FindPeaks(int[] mountain) {

}
}
```

C Solution:

```
/*
* Problem: Find the Peaks
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPeaks(int* mountain, int mountainSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find the Peaks
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findPeaks(mountain []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findPeaks(mountain: IntArray): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func findPeaks(_ mountain: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find the Peaks
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_peaks(mountain: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} mountain
# @return {Integer[]}
def find_peaks(mountain)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $mountain
* @return Integer[]
*/
function findPeaks($mountain) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findPeaks(List<int> mountain) {

}
}
```

Scala Solution:

```
object Solution {
def findPeaks(mountain: Array[Int]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_peaks(mountain :: [integer]) :: [integer]
def find_peaks(mountain) do

end
end
```

Erlang Solution:

```
-spec find_peaks(Mountain :: [integer()]) -> [integer()].
find_peaks(Mountain) ->
.
```

Racket Solution:

```
(define/contract (find-peaks mountain)
(-> (listof exact-integer?) (listof exact-integer?))
)
```