Problem 3024: Type of Triangle

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of size

3

which can form the sides of a triangle.

A triangle is called

equilateral

if it has all sides of equal length.

A triangle is called

isosceles

if it has exactly two sides of equal length.

A triangle is called

scalene

if all its sides are of different lengths.

Return

a string representing

the type of triangle that can be formed

or

"none"

if it

cannot

form a triangle.

Example 1:

Input:

nums = [3,3,3]

Output:

"equilateral"

Explanation:

Since all the sides are of equal length, therefore, it will form an equilateral triangle.

Example 2:

Input:

nums = [3,4,5]

Output:

"scalene"

Explanation:

nums[0] + nums[1] = 3 + 4 = 7, which is greater than nums[2] = 5. nums[0] + nums[2] = 3 + 5 = 8, which is greater than nums[1] = 4. nums[1] + nums[2] = 4 + 5 = 9, which is greater than nums[0] = 3. Since the sum of the two sides is greater than the third side for all three cases, therefore, it can form a triangle. As all the sides are of different lengths, it will form a scalene triangle.

Constraints:

nums.length == 3

1 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
string triangleType(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public String triangleType(int[] nums) {

}
}
```

Python3:

```
class Solution:
def triangleType(self, nums: List[int]) -> str:
```

Python:

```
class Solution(object):
def triangleType(self, nums):
"""
:type nums: List[int]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {string}
*/
var triangleType = function(nums) {

};
```

TypeScript:

```
function triangleType(nums: number[]): string {

};
```

C#:

```
public class Solution {
public string TriangleType(int[] nums) {

}
}
```

C:

```
char* triangleType(int* nums, int numsSize) {

}
```

Go:

```
func triangleType(nums []int) string {

}
```

Kotlin:

```
class Solution {
fun triangleType(nums: IntArray): String {

}
}
```

Swift:

```
class Solution {
func triangleType(_ nums: [Int]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn triangle_type(nums: Vec<i32>) -> String {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {String}
def triangle_type(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return String
*/
function triangleType($nums) {

}
}
```

Dart:

```
class Solution {
String triangleType(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def triangleType(nums: Array[Int]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec triangle_type(nums :: [integer]) :: String.t
def triangle_type(nums) do

end
end
```

Erlang:

```
-spec triangle_type(Nums :: [integer()]) -> unicode:unicode_binary().
triangle_type(Nums) ->
.
```

Racket:

```
(define/contract (triangle-type nums)
(-> (listof exact-integer?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Type of Triangle
* Difficulty: Easy
* Tags: array, string, math, sort
*
* 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:
string triangleType(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Type of Triangle
* Difficulty: Easy
* Tags: array, string, math, sort
*
* 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 String triangleType(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Type of Triangle
Difficulty: Easy
Tags: array, string, math, sort

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 triangleType(self, nums: List[int]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def triangleType(self, nums):
"""
:type nums: List[int]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Type of Triangle
* Difficulty: Easy
* Tags: array, string, math, sort
*
* 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[]} nums
* @return {string}
*/
var triangleType = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Type of Triangle
* Difficulty: Easy
* Tags: array, string, math, sort
*
* 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 triangleType(nums: number[]): string {

};
```

C# Solution:

```
/*
* Problem: Type of Triangle
* Difficulty: Easy
* Tags: array, string, math, sort
*
* 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 string TriangleType(int[] nums) {

}
}
```

C Solution:

```c
/*
* Problem: Type of Triangle
* Difficulty: Easy
* Tags: array, string, math, sort
*
* 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
*/

char* triangleType(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Type of Triangle
// Difficulty: Easy
// Tags: array, string, math, sort
//
// 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 triangleType(nums []int) string {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun triangleType(nums: IntArray): String {

}
}
```

Swift Solution:

```swift
class Solution {
func triangleType(_ nums: [Int]) -> String {
}
}
```

Rust Solution:

```
// Problem: Type of Triangle
// Difficulty: Easy
// Tags: array, string, math, sort
//
// 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 triangle_type(nums: Vec<i32>) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {String}
def triangle_type(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return String
*/
function triangleType($nums) {

}
}
```

Dart Solution:

```dart
class Solution {
String triangleType(List<int> nums) {

}
}
```

Scala Solution:

```scala
object Solution {
def triangleType(nums: Array[Int]): String = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec triangle_type(nums :: [integer]) :: String.t
def triangle_type(nums) do

end
end
```

Erlang Solution:

```erlang
-spec triangle_type(Nums :: [integer()]) -> unicode:unicode_binary().
triangle_type(Nums) ->
.
```

Racket Solution:

```racket
(define/contract (triangle-type nums)
(-> (listof exact-integer?) string?)
)
```