3024. Type of Triangle

Easy Array Math Sorting

Problem Description

In this problem, we're given an array `nums` of size `3` which represents the lengths of the sides of a potential triangle. Our goal is to classify the type of triangle these sides form or determine if they cannot form a triangle at all. According to the rules of geometry:

- An **equilateral** triangle has all sides of equal length.
- An **isosceles** triangle has exactly two sides of equal length.
- A **scalene** triangle has all sides of different lengths.
- Additionally, the sum of the lengths of any two sides must be greater than the length of the remaining side for a valid triangle to be formed.

Given these guidelines, we must return a string indicating the type of triangle (`"equilateral"`, `"isosceles"`, or `"scalene"`) or `"none"` if no triangle can be formed with the given side lengths.

Intuition

The intuition behind the solution is grounded in the basic properties of triangles. Intuitively, we recognize that:

1. To determine if a triangle can be formed, we rely on the triangle inequality theorem which states that the sum of the lengths of any two sides of a triangle must be greater than the length of the third side.
2. Once we know a triangle can be formed, we can classify the triangle based on the equality (or lack thereof) of its sides.

We approach the solution in steps:

1. We start by sorting the array. Sorting ensures that the sides are ordered from smallest to largest, which simplifies our comparisons.
2. Once sorted, we check the triangle inequality using the smallest two sides (now at index 0 and index 1) and compare their sum to the largest side (now at index 2).
3. If the triangle inequality is not satisfied, we cannot form a triangle, and we return `"none"`.
4. If all sides are equal, which is straightforward to check after sorting since all elements would be the same, we return `"equilateral"`.
5. If only two sides are equal (the first and second or the second and third after sorting), we return `"isosceles"`.
6. If none of the above conditions are met, then we have a triangle with all unique side lengths, and we return `"scalene"`.

By proceeding in this logical and systematic way, based on the definitions and properties of triangles, we can accurately categorize the input as forming a specific type of triangle or not being able to form one at all.

Solution Approach

The implementation of the solution employs a simple but effective approach:

1. Sorting: Initially, we sort the array `nums` which is a common and straightforward algorithm known as comparison sort. In Python, the sort operation generally uses an algorithm called Timsort, which is a hybrid sorting algorithm derived from merge sort and insertion sort. By doing this, we guarantee that `nums[0] <= nums[1] <= nums[2]` after sorting.
2. Checking the Triangle Inequality: We use the triangle inequality theorem, which is a fundamental concept in geometry. According to this theorem, for any three sides to form a triangle, the sum of the lengths of any two sides must be greater than the length of the third side. In our case, after sorting, we check if `nums[0] + nums[1] > nums[2]`. If not, we immediately know it's impossible to form a triangle and return `"none"`.
3. Classification of Triangle:
   - **Equilateral**: This check is straightforward after sorting. If all three sides are equal (`nums[0] == nums[2]`), we return `"equilateral"`.
   - **Isosceles**: Given that an equilateral triangle (a special case of isosceles where all sides are equal) has been handled already, any other scenario with two equal sides (`nums[0] == nums[1]` or `nums[1] == nums[2]`) indicates an isosceles triangle, and we return `"isosceles"`.
   - **Scalene**: If neither of the above checks return, we're left with a scenario where all three sides are different, which means the triangle is scalene. Following the process of elimination, we return `"scalene"`.
4. Data Structures: We use the list data structure provided in Python, which allows us to store the sides of the potential triangle and sort them.
5. Patterns: The pattern here is essentially decision-making based on conditional checks. We perform a series of if-else statements to compare the sides of the potential triangle according to the requirements for different categories of triangles.

By using the properties of triangles and following a structured algorithm to inspect and classify the side lengths, this solution is simple, efficient, and correct. It elegantly solves the problem with a minimal amount of code and no extra space required beyond the input.

Example Walkthrough

Let's consider an example to illustrate the solution approach:

Suppose we are given the array `nums = [4, 5, 3]`. We want to determine if these values can form a triangle and, if so, categorize it as either "equilateral", "isosceles", or "scalene".

Here are the steps following the solution approach:

1. **Sorting**: We start by sorting the array, so `nums = [3, 4, 5]` after the sort.
2. **Checking the Triangle Inequality**: We then check the triangle inequality with the sorted array. We compare if the sum of the two smaller sides is greater than the largest side: `3 + 4 > 5`. Since this condition is true, it is possible to form a triangle with these side lengths.
3. **Classification of Triangle**:
   - **Equilateral**: We check if all three sides are equal. This is not the case here (`3 != 4 != 5`), so it's not an equilateral triangle.
   - **Isosceles**: Next, we check for two equal sides. Since `3 != 4` and `4 != 5`, no two sides are equal. It's not an isosceles triangle either.
   - **Scalene**: Given that we don't have all sides equal and there aren't just two sides that are equal, it means that all sides are different. Therefore, we classify the triangle as "scalene".

Based on this example, given the side lengths of `3`, `4`, and `5`, our solution would correctly identify and return the string `"scalene"`. The triangle formed by these sides adheres to the rules of geometry and is a valid scalene triangle, where each side is a different length.

Solution Implementation

Python

```python
# Define the Solution class.
class Solution:
    # Define the method that determines the type of a triangle given its side lengths.
    def triangle_type(self, sides: List[int]) -> str:
        # Sort the side lengths for easier comparison.
        sides.sort()

        # Check for triangle inequality theorem - the sum of any two sides must be greater than the third side.
        if sides[0] + sides[1] <= sides[2]:
            # If the condition fails, it's not a triangle.
            return "none"

        # Check if all sides are equal for an equilateral triangle.
        if sides[0] == sides[2]:
            return "equilateral"

        # Check for an isosceles triangle where exactly two sides are equal.
        if sides[0] == sides[1] or sides[1] == sides[2]:
            return "isosceles"

        # If none of the above conditions matched, it's a scalene triangle with all sides of different lengths.
        return "scalene"
```

Java

```java
class Solution {

    // Method to classify the type of a triangle based on its side lengths
    public String triangleType(int[] sides) {
        // Sort the array to have the sides in ascending order
        Arrays.sort(sides);

        // Check for the triangle inequality theorem to determine if a triangle is possible
        if (sides[0] + sides[1] <= sides[2]) {
            // The sum of lengths of any two sides must be greater than the length of the third side
            return "none";
        }

        // Check if all sides are equal
        if (sides[0] == sides[2]) {
            // All sides are equal, therefore it's an equilateral triangle
            return "equilateral";
        }

        // Check if any two sides are equal
        if (sides[0] == sides[1] || sides[1] == sides[2]) {
            // Two sides are equal, therefore it's an isosceles triangle
            return "isosceles";
        }

        // If none of the sides are equal, it's a scalene triangle
        return "scalene";
    }
}
```

C++

```cpp
#include <vector>
#include <string>
#include <algorithm>

class Solution {
public:
    // Function to determine the type of triangle based on side lengths
    std::string triangleType(std::vector<int>& sides) {
        // Sort the sides of the triangle in non-decreasing order for comparison
        std::sort(sides.begin(), sides.end());

        // Check for a triangle inequality theorem violation
        if (sides[0] + sides[1] <= sides[2]) {
            // The sides cannot form a triangle
            return "none";
        }
        // Check if all sides are equal
        if (sides[0] == sides[2]) {
            // All sides are equal, so the triangle is equilateral
            return "equilateral";
        }

        // Check if any two sides are equal
        if (sides[0] == sides[1] || sides[1] == sides[2]) {
            // Two sides are equal, so the triangle is isosceles
            return "isosceles";
        }
        // If none of the above conditions are true, the triangle is scalene
        return "scalene";
    }
};
```

TypeScript

```typescript
function triangleType(sides: number[]): string {
    // Sort the array of side lengths in non-decreasing order
    sides.sort((a, b) => a - b);

    // Check for the non-existence of a triangle first
    if (sides[0] + sides[1] <= sides[2]) {
        // Sum of two smaller sides should be greater than the longest side
        return 'none';
    }

    // Check for an equilateral triangle (all sides equal)
    if (sides[0] === sides[2]) {
        return 'equilateral';
    }

    // Check for an isosceles triangle (at least two sides equal)
    if (sides[0] === sides[1] || sides[1] === sides[2]) {
        return 'isosceles';
    }

    // If none of the above conditions are met, the triangle is scalene (all sides are different)
    return 'scalene';
}


# Define the Solution class.
class Solution:
    # Define the method that determines the type of a triangle given its side lengths.
    def triangle_type(self, sides: List[int]) -> str:
        # Sort the side lengths for easier comparison.
        sides.sort()

        # Check for triangle inequality theorem - the sum of any two sides must be greater than the third side.
        if sides[0] + sides[1] <= sides[2]:
            # If the condition fails, it's not a triangle.
            return "none"

        # Check if all sides are equal for an equilateral triangle.
        if sides[0] == sides[2]:
            return "equilateral"

        # Check for an isosceles triangle where exactly two sides are equal.
        if sides[0] == sides[1] or sides[1] == sides[2]:
            return "isosceles"

        # If none of the above conditions matched, it's a scalene triangle with all sides of different lengths.
        return "scalene"
```

Time and Space Complexity

The time complexity of the given code is primarily determined by the sorting operation performed on the list of three numbers. Although the sorting operation on three elements can be considered a constant-time operation, traditional sorting algorithms have a time complexity of $O(n \log n)$. However, since the list size is fixed at three elements (representing the sides of a triangle), the sort operation will not vary with the size of the input and can be considered $O(1)$ for this specific scenario.

After sorting, each comparison and equality check in the `if` statements also takes constant time, i.e., $O(1)$. It doesn't matter how large the numbers are; the time it takes to compare or check equality between two numbers does not depend on the size of the input.

Consequently, the overall time complexity of the given code is $O(1)$.

In terms of space complexity, the code uses a fixed amount of extra space for the sorted version of the input list, without using any additional structures that grow with the input size. Hence, the space complexity is also $O(1)$ as it does not scale with the size of the input provided to the function.