Greater Than the One Directly Before It

Problem Description

You are given a sequence of integers. Your task is to determine how many elements in the sequence are greater than the element that directly precedes them.

Input Format

- The first line contains an integer `t` — the number of test cases
- For each test case:
    - The first line contains an integer `n` — the number of elements in the sequence
    - The second line contains `n` space-separated integers representing the sequence

Output Format

For each test case, output a single integer representing the count of elements that are greater than their immediate predecessor.

Constraints

- Elements are compared only with their direct predecessor
- The first element has no predecessor, so it's never counted

Example

Input

```
2
5
1 2 3 2 5
4
10 20 10 30
```

Output

```
3
2
```

Explanation

Test Case 1: `[1, 2, 3, 2, 5]`

- `2 > 1` ✓ (position 2)
- `3 > 2` ✓ (position 3)
- `2 < 3` ✗ (position 4)
- `5 > 2` ✓ (position 5)
- **Answer: 3**

Test Case 2: `[10, 20, 10, 30]`

- `20 > 10` ✓ (position 2)
- `10 < 20` ✗ (position 3)
- `30 > 10` ✓ (position 4)
- **Answer: 2**

Solution Approach

1. For each test case, read the sequence
2. Iterate through the sequence starting from the second element (index 1)
3. Compare each element with its predecessor
4. Count how many elements satisfy the condition
5. Output the count

Multi-Language Implementations

C Implementation

```
c
```

```
#include <stdio.h>

int main() {
    int t;
    scanf("%d", &t);

    while (t--) {
        int n;
        scanf("%d", &n);

        int arr[n];  // Variable-length array (C99+)
        for (int i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
        }

        int count = 0;
        for (int i = 1; i < n; i++) {
            if (arr[i] > arr[i-1]) {
                count++;
            }
        }

        printf("%d\n", count);
    }

    return 0;
}
```

C Explanation:

- Uses variable-length arrays (VLA) supported in C99
- `scanf` for input, `printf` for output
- Simple loop-based comparison logic

C++ Implementation

cpp

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int t;
    cin >> t;

    while (t--) {
        int n;
        cin >> n;

        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }

        int count = 0;
        for (int i = 1; i < n; i++) {
            if (arr[i] > arr[i-1]) {
                count++;
            }
        }

        cout << count << endl;
    }

    return 0;
}
```

C++ Explanation:

- Uses `vector<int>` for dynamic array allocation
- `cin`/`cout` for cleaner I/O operations
- STL containers provide memory safety

C# Implementation

```
csharp
```

```
using System;

class Program {
    static void Main() {
        int t = int.Parse(Console.ReadLine());

        while (t-- > 0) {
            int n = int.Parse(Console.ReadLine());
            string[] input = Console.ReadLine().Split();
            int[] arr = new int[n];

            for (int i = 0; i < n; i++) {
                arr[i] = int.Parse(input[i]);
            }

            int count = 0;
            for (int i = 1; i < n; i++) {
                if (arr[i] > arr[i-1]) {
                    count++;
                }
            }

            Console.WriteLine(count);
        }
    }
}
```

C# Explanation:

- Uses `Console.ReadLine().Split()` to parse space-separated input
- `int.Parse()` for string-to-integer conversion
- Managed memory allocation with `new int[n]`

Rust Implementation

```
rust
```

```
use std::io;

fn main() {
    let mut input = String::new();
    io::stdin().read_line(&mut input).unwrap();
    let t: usize = input.trim().parse().unwrap();

    for _ in 0..t {
        input.clear();
        io::stdin().read_line(&mut input).unwrap();
        let n: usize = input.trim().parse().unwrap();

        input.clear();
        io::stdin().read_line(&mut input).unwrap();
        let arr: Vec<i32> = input
            .trim()
            .split_whitespace()
            .map(|s| s.parse().unwrap())
            .collect();

        let count = arr
            .windows(2)
            .filter(|pair| pair[1] > pair[0])
            .count();

        println!("{}", count);
    }
}
```

Rust Explanation:

- Uses `Vec<i32>` for dynamic arrays
- `windows(2)` creates sliding windows of consecutive pairs
- Functional approach with `filter` and `count`
- Memory-safe with ownership system

Prime Number Checker

Multi-Language Implementations

C Implementation

c

```c
#include <stdio.h>
#include <math.h>

int main() {
    int t;
    scanf("%d", &t);

    while (t--) {
        int n;
        scanf("%d", &n);

        if (n < 2) {
            printf("NO\n");
            continue;
        }

        int isPrime = 1;
        for (int i = 2; i <= sqrt(n); i++) {
            if (n % i == 0) {
                isPrime = 0;
                break;
            }
        }

        if (isPrime)
            printf("YES\n");
        else
            printf("NO\n");
    }

    return 0;
}
```

C++ Implementation

cpp

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  int t;
  cin >> t;

  while (t--) {
    int n;
    cin >> n;

    if (n < 2) {
      cout << "NO" << endl;
      continue;
    }

    bool isPrime = true;
    for (int i = 2; i <= sqrt(n); i++) {
      if (n % i == 0) {
        isPrime = false;
        break;
      }
    }

    cout << (isPrime ? "YES" : "NO") << endl;
  }

  return 0;
}
```

C# Implementation

```
csharp
```

```
using System;

class Program {
    static void Main() {
        int t = int.Parse(Console.ReadLine());

        while (t-- > 0) {
            int n = int.Parse(Console.ReadLine());

            if (n < 2) {
                Console.WriteLine("NO");
                continue;
            }

            bool isPrime = true;
            for (int i = 2; i <= Math.Sqrt(n); i++) {
                if (n % i == 0) {
                    isPrime = false;
                    break;
                }
            }

            Console.WriteLine(isPrime ? "YES" : "NO");
        }
    }
}
```

Rust Implementation

```
rust
```

```
use std::io;

fn main() {
    let mut input = String::new();
    io::stdin().read_line(&mut input).unwrap();
    let t: usize = input.trim().parse().unwrap();

    for _ in 0..t {
        input.clear();
        io::stdin().read_line(&mut input).unwrap();
        let n: i32 = input.trim().parse().unwrap();

        if n < 2 {
            println!("NO");
            continue;
        }

        let is_prime = (2..=((n as f64).sqrt() as i32))
            .all(|i| n % i != 0);

        println!("{}", if is_prime { "YES" } else { "NO" });
    }
}
```

Algorithm Explanation

Prime Checking Algorithm

1. **Input Validation**: Numbers less than 2 are not prime
2. **Optimization**: Only check divisors up to √n
3. **Early Termination**: Break immediately when a divisor is found
4. **Time Complexity**: O(√n) per test case

Key Language Differences

- **C**: Manual memory management, basic I/O
- **C++**: STL containers, cleaner syntax with cin/cout
- **C#**: Managed memory, built-in parsing methods
- **Rust**: Memory safety, functional programming features, ownership system