2148. Count Elements With Strictly Smaller and Greater Elements

Easy Array Sorting

Leetcode Link

Problem Description

The goal of the problem is to count how many elements in an integer array `nums` have at least one other element in the array that is strictly smaller and at least one other element that is strictly greater. In other words, for an element to be counted, it cannot be the smallest or the largest element in the array. The task is to find the number of such elements.

Intuition

To solve this problem efficiently, what comes to mind is that if we know the smallest and the largest elements of the array, we can simply iterate through the array and count the elements that fall strictly between these two extremes. That is because the elements that are equal to the minimum or maximum can't possibly satisfy the condition of having both a strictly smaller and a strictly greater element in the array.

Thus, the solution involves the following steps:

1. Find the minimum value in `nums`, denoted as `mi`.
2. Find the maximum value in `nums`, denoted as `mx`.
3. Iterate through each element `num` in `nums` and
    - Increment a counter each time `mi < num < mx` is `True`.
4. The counter value gives us the desired number of elements.

In the provided solution, a simple yet elegant approach is taken where Python's built-in `min` and `max` functions are used to find `mi` and `mx`. Then a generator expression is used within the `sum` function to add up all the boolean values that are `True` for elements that fall strictly between the minimum and maximum, which inherently gives us the count.

Solution Approach

The implementation of the solution is straightforward and leverages Python's concise and powerful syntax. We use two built-in functions and a very common Python pattern, list comprehension with a condition (which, in this context, creates a generator expression).

Here's how the solution is implemented:

1. **Finding the Minimum and Maximum:** We begin by finding the minimum and maximum value within the `nums` list using Python's `min` and `max` functions.

    ```
    mi, mx = min(nums), max(nums)
    ```

    This step is done in constant time with respect to the input array size, assuming the `min` and `max` functions are implemented efficiently (which they are in Python).

2. **Counting the Elements:** We then count the elements that are strictly greater than `mi` and strictly less than `mx` using a generator expression within the `sum` function.

    ```
    return sum(mi < num < mx for num in nums)
    ```

    The expression `mi < num < mx` evaluates to `True` if `num` lies strictly between `mi` and `mx`. In Python, when `True` is passed to the `sum` function, it is treated as `1`, and `False` is treated as `0`. Thus we effectively count the number of `True` cases which correspond to valid elements.

    This step has a time complexity of O(n), where n is the size of the input list, because it involves iterating over all the elements in the list once.

The solution does not explicitly use any additional data structures, relying on Python's list and integer types. The pattern used here is commonly referred to as a comprehension. It's a concise way to create a new list or in this case, generate values on the fly for the `sum` function, while applying a condition or mapping each item to a new value.

This two-step approach is very efficient because it minimizes the amount of work needed to be done on the input array. By first determining the bounds with `min` and `max`, and then using a simple linear scan with a generator expression to count the qualifying elements, we arrive at an elegant and efficient solution.

Example Walkthrough

Let's suppose our input `nums` array is `[3, 7, 2, 5, 6]`.

1. The first step is finding the minimum (`mi`) and maximum (`mx`) value in the array. We use the `min()` and `max()` functions of Python.
    - `mi = min(nums)` would evaluate to 2.
    - `mx = max(nums)` would evaluate to 7.
2. The second step involves iterating over the array and counting the elements that are strictly greater than the minimum (2) and strictly less than the maximum (7).
    - The first element, `3`, is greater than `2` and less than `7`, so it meets the condition.
    - The second element, `7`, is equal to `mx` and hence does not meet the condition.
    - The third element, `2`, is equal to `mi` and also does not meet the condition.
    - The fourth element, `5`, meets the condition as it is greater than `2` and less than `7`.
    - The fifth element, `6`, also meets the condition.
3. Counting the elements that passed the condition:
    - We calculate `sum(mi < num < mx for num in nums)`, which is essentially `sum([True, False, False, True, True])`.
    - This is equivalent to `1 + 0 + 0 + 1 + 1`, which sums up to `3`.

Thus, for the array `[3, 7, 2, 5, 6]`, the function would return `3`, indicating there are three elements within the array that have at least one other element in the array strictly smaller and at least one strictly greater.

Python Solution

```python
class Solution:
    def countElements(self, nums: List[int]) -> int:
        # Find the minimum and maximum values in the list of numbers
        min_val, max_val = min(nums), max(nums)

        # Count and return the number of elements that are strictly between the minimum
        # and maximum values using a generator expression within the sum function.
        return sum(min_val < num < max_val for num in nums)

```

Java Solution

```java
class Solution {
    public int countElements(int[] nums) {
        // Initialize the minimum and maximum values possible for the elements in the array.
        int minElement = Integer.MAX_VALUE, maxElement = Integer.MIN_VALUE;

        // Iterate over each element to find the smallest and largest number.
        for (int num : nums) {
            minElement = Math.min(minElement, num);
            maxElement = Math.max(maxElement, num);
        }

        // Initialize the counter for the number of elements that fall strictly between the min and max.
        int count = 0;

        // Count elements that are strictly greater than the minimum and strictly less than the maximum.
        for (int num : nums) {
            if (minElement < num && num < maxElement) {
                count++;
            }
        }

        // Return the total count of elements satisfying the condition.
        return count;
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    // Function to count the elements that are greater than the minimum
    // and less than the maximum elements in the given vector nums
    int countElements(vector<int>& nums) {
        // Initialize minimum and maximum values with extremes
        int minVal = INT_MAX; // Use INT_MAX to represent initially the largest possible integer
        int maxVal = INT_MIN; // Use INT_MIN to represent initially the smallest possible integer

        // Loop to find the smallest and largest values in the vector
        for (int num : nums) {
            minVal = std::min(minVal, num); // Update minVal to the minimum found so far
            maxVal = std::max(maxVal, num); // Update maxVal to the maximum found so far
        }

        // Initialize the answer count
        int ans = 0;

        // Iterate through the elements of nums
        for (int num : nums) {
            // Increment ans if num is greater than minVal and less than maxVal
            if (minVal < num && num < maxVal)
                ans++;
        }

        // Return the count of elements that satisfy the condition
        return ans;
    }
};

```

Typescript Solution

```typescript
function countElements(nums: number[]): number {
    // Find the minimum and maximum elements in the array.
    const minValue = Math.min(...nums),
          maxValue = Math.max(...nums);

    // Initialize the count of elements that are not the min or max value.
    let count = 0;

    // Iterate through the array to count elements that are greater than min and less than max.
    for (let i = 0; i < nums.length; ++i) {
        const currentElement = nums[i];
        if (currentElement < maxValue && currentElement > minValue) {
            // Increment the count for each qualifying element.
            count++;
        }
    }

    // Return the total count of elements that are not the min or max value.
    return count;
}

```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the number of elements in the input list `nums`. This is because the `min(nums)` and `max(nums)` functions each iterate through the list once, resulting in `2 * O(n)` operations, which simplifies to `O(n)` in Big O notation. The `sum` function with the generator expression also iterates through the list once, adding another `O(n)` operation. Therefore, the overall time complexity remains `O(n)`.

The space complexity of the code is `O(1)`. The reason behind this is that only a constant amount of additional space is used. Variables `mi` and `mx` are used to store the minimum and maximum elements of the list, which does not depend on the size of the input list. The generator expression in the `sum` function computes its result in-place and does not allocate additional space that is dependent on the input size. Thus, the space used is constant regardless of `n`.