3012. Minimize Length of Array Using Operations

Medium | Greedy | Array | Math | Number Theory

Problem Description

The provided problem asks us to perform a series of operations on an array `nums` containing positive integers to minimize its length. The operations can be executed any number of times and consist of the following steps:

1. Choose two different indices `i` and `j` where both `nums[i]` and `nums[j]` are greater than zero.
2. Calculate `nums[i] % nums[j]` (the remainder of the division of `nums[i]` by `nums[j]`) and append the result to the end of the array.
3. Remove the elements at indices `i` and `j` from the array.

We aim to find the minimum possible length of the array after performing these operations repeatedly.

Intuition

To develop the intuition for the solution, we consider the behavior of the modulo operation and how it can influence the array's length:

1. If there is the smallest number in the array, `mi`, that does not evenly divide at least one other element, using `mi` and this other element in an operation would create a new, smaller number than `mi`.
2. The presence of a smaller number means we can keep performing the operation, pairing the new smaller number with others to continue reducing the size of the array until only one number remains.

Keeping these points in mind, we consider two scenarios:

- If there is an element `x` in `nums` such that `x % mi` is not zero, the array can be reduced to a single element, hence the minimum possible length is 1.
- If all elements in `nums` are multiples of `mi`, we can use `mi` to eliminate them by pairing each occurrence with another `mi`. Doing so repeatedly halves the group of `mi` elements (plus one if the count is odd), leading to the final minimum length being the count of `mi` divided by 2, rounded up.

This reasoning leads to the solution approach implemented in the given code.

Solution Approach

The implementation of the solution is concise, leveraging the minimum element in the array and the count of occurrences of this minimum element:

1. **Finding the Minimum Element**: The first step is to find the minimum element `mi` in the list using the Python built-in function `min()`. This element is crucial because it is the smallest possible remainder we can generate (since any number modulo itself is 0).

2. **Checking for Non-multiples of the Minimum Element**: We then check whether every element in the array is a multiple of `mi`. This is done by iterating through each element `x` in `nums` and checking if `x % mi` yields a non-zero value. This can be achieved using the built-in `any()` function in Python which returns `True` if at least one element satisfies the condition. If we find that `x % mi` is not zero for some element `x`, it implies that we can reduce the size of the array to 1 by performing operations using `x` and `mi`. Hence, in this case, the algorithm returns 1.

3. **Counting Instances of the Minimum Element**: If all elements in the array are multiples of `mi`, it means no operations can produce a smaller element than `mi`. Therefore, the next step is to count how many times `mi` occurs in the array using the `count()` method. If the minimum element `mi` occurs, for example, two times, operations can be performed to reduce these to one instance of `mi` and so forth for pairs of `mi`.

4. **Calculating the Final Array Length**: Lastly, we determine the minimum length of the array by dividing the count of `mi` by 2 and rounding up. The rounding up is accomplished by adding 1 to the count and then performing an integer division by 2 (`(nums.count(mi) + 1) // 2`).

Here's the breakdown of the algorithm used in the `Solution` class:

- Get the smallest element `mi` in `nums` as `mi = min(nums)`.
- Check if there's any element in `nums` that is not a multiple of `mi` with `any(x % mi for x in nums)`:
  - If true, return `1`.
  - If false, calculate `(nums.count(mi) + 1) // 2` and return the result.

This process uses constant space (no extra data structures are needed other than variables to store the minimum element and the count) and requires time complexity of O(n) due to the single pass through the list `nums` to count elements and check for non-multiples of `mi`.

Example Walkthrough

Let's illustrate the solution approach with a small example.

Suppose our input array `nums` is `[6, 4, 4, 2]`.

1. **Finding the Minimum Element**: First, we find the smallest item in `nums`. In our example, `mi = min(nums)` would yield `mi = 2`.

2. **Checking for Non-multiples of the Minimum Element**: Next, we check if each element in `nums` is a multiple of `mi`. We iterate over the array and use the modulo operation. In this case, `6 % 2` is `0`, `4 % 2` is `0`, and `4 % 2` is again `0`. Since all results are `0`, every element is a multiple of `mi`.

3. **Counting Instances of the Minimum Element**: Since all elements are multiples of `mi`, we now count how many times `mi` appears in `nums`. With `nums.count(2)`, we get the value `1`, indicating that the element `2` appears once in the array.

4. **Calculating the Final Array Length**: Finally, to find the minimum possible length of the array, we divide the count by 2 and round up. With only one `2` in the array, we apply `(nums.count(mi) + 1) // 2`, which equals `(1 + 1) // 2`, yielding a final result of `1`.

Therefore, for the input array `[6, 4, 4, 2]`, the minimum possible length of the array after performing the operations is `1`.

This example takes us through the intuitive approach to reduce the array's length by focusing on the minimum element and checking if we can produce a smaller number through the modulo operation. As we found that all elements were multiples of the smallest element, we determined that no further reduction is possible other than pairing the like terms, resulting in the smallest possible length of `1`.

Solution Implementation

Python

```python
from typing import List

class Solution:
    def minimumArrayLength(self, nums: List[int]) -> int:
        # Find the minimum value in the nums list
        minimum_value = min(nums)

        # Check if any number in the list is not divisible by the minimum value
        if any(number % minimum_value for number in nums):
            # If there's at least one number not divisible by minimum_value,
            # the minimum array length required is 1
            return 1

        # Count the occurrences of the minimum value in nums
        count_min_value = nums.count(minimum_value)

        # Calculate the minimum array length by dividing the count of minimum_value by 2
        # and rounding up to the nearest integer if necessary
        return (count_min_value + 1) // 2
```

Java

```java
import java.util.Arrays;

class Solution {
    public int minimumArrayLength(int[] nums) {
        // Find the minimum value in the array using a stream
        int minElement = Arrays.stream(nums).min().getAsInt();

        // Initialize a count to track the occurrence of the minimum element
        int minCount = 0;

        // Iterate over each element in the array
        for (int element : nums) {
            // If the element is not divisible by the minimum element,
            // the minimum length needed is 1, so return 1
            if (element % minElement != 0) {
                return 1;
            }
            // If the element is equal to the minimum element,
            // increment the count of the minimum element
            if (element == minElement) {
                ++minCount;
            }
        }

        // Return half of the count of the minimum element plus one, rounded down.
        // This determines the minimum array length required.
        return (minCount + 1) / 2;
    }
}
```

C++

```cpp
#include <vector> // Required for std::vector
#include <algorithm> // Required for std::min_element function

class Solution {
public:
    // Function to find the minimum array length required
    int minimumArrayLength(vector<int>& nums) {
        // Find the smallest element in the array
        int minValue = *std::min_element(nums.begin(), nums.end());
        // Initialize count of elements equal to the smallest element
        int countMinValue = 0;

        // Iterate over all elements in the array
        for (int num : nums) {
            // If current element is not a multiple of the smallest element,
            // return 1, as we cannot equalize array with elements having
            // a common factor with the smallest element
            if (num % minValue) {
                return 1;
            }
            // Increase count if current element is equal to the smallest element
            countMinValue += (num == minValue);
        }

        // Calculate and return the minimum array length required by dividing the count
        // of smallest elements by 2 and adding 1, then taking the ceiling of the result
        // Since countMinValue is integer, add 1 before division to achieve the ceiling effect
        return (countMinValue + 1) / 2;
    }
};
```

TypeScript

```typescript
function minimumArrayLength(nums: number[]): number {
    // Find the minimum element in the array.
    const minimumElement = Math.min(...nums);
    let minimumElementCount = 0;

    // Iterate over each number in the array.
    for (const number of nums) {
        // If any number is not divisible by the minimum element,
        // the minimum array length that satisfies the condition is 1.
        if (number % minimumElement !== 0) {
            return 1;
        }
        // Count how many times the minimum element appears in the array.
        if (number === minimumElement) {
            minimumElementCount++;
        }
    }

    // Return the half of the count of the minimum element (rounded up).
    // This is due to the right shift operation which is equivalent to Math.floor((minimumElementCount + 1) / 2).
    return (minimumElementCount + 1) >> 1;
}
```


```python
from typing import List

class Solution:
    def minimumArrayLength(self, nums: List[int]) -> int:
        # Find the minimum value in the nums list
        minimum_value = min(nums)

        # Check if any number in the list is not divisible by the minimum value
        if any(number % minimum_value for number in nums):
            # If there's at least one number not divisible by minimum_value,
            # the minimum array length required is 1
            return 1

        # Count the occurrences of the minimum value in nums
        count_min_value = nums.count(minimum_value)

        # Calculate the minimum array length by dividing the count of minimum_value by 2
        # and rounding up to the nearest integer if necessary
        return (count_min_value + 1) // 2
```


Time and Space Complexity

The time complexity of the given code can be discussed as follows. The `min(nums)` call iterates through the `nums` list once to find the minimum value, which takes `O(n)` time where `n` is the length of `nums`. The `any(...)` function in the next line also performs another iteration over `nums`, hence taking up to `O(n)` time in the worst case. The call to `nums.count(mi)` is yet another iteration through the list, which is `O(n)` as well.

Adding these up, we see that the function potentially iterates through the list three times independently. However, since we don't multiply independent linear traversals but rather add them, the overall time complexity remains `O(n)`.

The space complexity does not depend on the size of the input as no additional space is allocated based on `nums`. No extra storage scales with the size of the input; only a fixed number of single-value variables (`mi`) are used. Therefore, the space complexity of the code is `O(1)`.