908. Smallest Range I

Easy Array Math Leetcode Link

Problem Description

In this problem, we are given an array of integers, `nums`, and an integer `k`. Our goal is to minimize the score of `nums`, defined as the difference between the maximum and minimum elements in the array.

We have the ability to perform an operation on each element of the array. The operation involves choosing an index `i` and then changing the value of `nums[i]` by adding an integer `x` that can range from `-k` to `k` (inclusive). We are allowed to use this operation at most once for each index.

The objective is to determine what the minimum achievable score is after applying the operation no more than once to each element of the array, if at all.

Intuition

The intuition behind the solution comes from understanding how the score is calculated—the score is simply the difference between the largest and smallest numbers in the array. If we increase the smaller numbers and/or decrease the larger numbers, we can bring them closer to each other, thus reducing the score.

Since we can only apply the operation once per index, our best strategy is to reduce the maximum possible value by `k` and to increase the minimum possible value by `k`. By doing so, we reduce or eliminate the gap between the maximum and minimum values as much as possible.

The solution checks two scenarios:

1. If the original maximum value minus the minimum value is less than or equal to `2 * k`, then we can fully bridge the gap, making the score zero.
2. If the gap is larger than `2 * k`, we do our best by reducing the gap by `2 * k` (subtracting `k` from the maximum value and adding `k` to the minimum value) and the remaining difference is the minimal score possible.

Therefore, the formula `max(0, mx - mi - k * 2)` is used, where `mx` and `mi` are the maximum and minimum values in the array, respectively. We use `max(0, ...)` because the score cannot be negative—if the max-min difference is less than or equal to double of `k`, bridging the gap completely, the score would be zero.

Solution Approach

The solution to this problem is quite straightforward, without involving complex algorithms or data structures. It follows a direct mathematical approach that efficiently arrives at the result.

Here's a breakdown of the solution implementation:

1. **Find Maximum and Minimum Values:** We start by finding the maximum (`mx`) and minimum (`mi`) values in the array `nums`. This can be done efficiently in a linear pass over the array.

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

2. **Calculate Adjusted Range:** We then calculate what the new range (difference between maximum and minimum) would be if we were to reduce the maximum value by `k` and increase the minimum value by `k`.

   ```
   adjusted_range = mx - mi - k * 2
   ```

3. **Determine the Minimum Score:** Finally, we evaluate the minimum score, which would be the adjusted range if it's positive, or `0` if the adjusted range is negative (which means the entire range has been neutralized by the operation).

   ```
   minimum_score = max(0, adjusted_range)
   ```

   The use of `max(0, ...)` ensures that our final score is not negative, adhering to the definition that the score is the difference between two values (which is inherently non-negative).

4. **Return the Result:** The calculated `minimum_score` is returned as the final result.

This approach uses constant extra space and has a time complexity of O(N) due to the need to iterate over the array to find the minimum and maximum values. No additional patterns or complex operations are needed, making this an efficient and clean solution.

Example Walkthrough

Let's take an array `[1, 3, 6]` as a small example to illustrate the solution approach, with `k = 3`.

1. **Find Maximum and Minimum Values:**
   - Maximum value `mx` in the array is `6`.
   - Minimum value `mi` in the array is `1`.
2. **Calculate Adjusted Range:**
   - We can reduce the maximum value (`mx`) by `k` (6 - 3 = 3) and increase the minimum value (`mi`) by `k` (1 + 3 = 4).
   - The new adjusted range would be the new maximum value (`3`) minus the new minimum value (`4`), which is `3 - 4 = -1`.
   - However, since we cannot have a negative score, we consider the adjusted range as `0`.
3. **Determine the Minimum Score:**
   - The minimum score is the maximum between `0` and the adjusted range.
   - Since in our example the adjusted range is `-1`, we use `0` because the score can't be negative.
   - Therefore, the minimum score is `0`.
4. **Return the Result:**
   - With the given `nums` array `[1, 3, 6]` and `k = 3`, the minimum achievable score after applying the operations is `0`.

In this case, we can conclude that after applying `+k` to the minimum value and `-k` to the maximum value, the adjusted values end up overlapping, and the minimum score becomes zero, which is the best-case scenario.

Python Solution

```python
# Import the typing module to use the List type hint
from typing import List

class Solution:
    def smallestRangeI(self, nums: List[int], k: int) -> int:
        # Find the maximum and minimum values in the list of numbers
        max_num, min_num = max(nums), min(nums)

        # Calculate the possible smallest range after performing the operation
        # described in the problem statement, which reduces the difference
        # between the maximum and minimum numbers by 2*k (k from the max and
        # k from the min). If the difference is already less than 2*k,
        # the smallest range can be brought down to 0.
        smallest_range = max(0, max_num - min_num - 2 * k)

        # Return the calculated smallest range
        return smallest_range

```

Java Solution

```java
class Solution {
    public int smallestRangeI(int[] nums, int k) {
        // Initialize maximum and minimum values to be the opposite of their extremes.
        // This ensures that they will be replaced by actual array values.
        int maxNum = nums[0]; // Initially set maxNum to the first element.
        int minNum = nums[0]; // Initially set minNum to the first element.

        // Iterate through the array to find the maximum and minimum values.
        for (int value : nums) {
            maxNum = Math.max(maxNum, value); // Update maxNum if current value is greater.
            minNum = Math.min(minNum, value); // Update minNum if current value is smaller.
        }

        // Calculate the maximum achievable difference by subtracting the range extension
        // from the actual range. Ensure the result is not negative.
        // The range extension is 'k' from both sides of the range, hence 'k * 2'.
        int maxDiff = Math.max(0, maxNum - minNum - k * 2);

        // Return the smallest possible range after the operations.
        return maxDiff;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // For std::max_element and std::min_element

class Solution {
public:
    // This function finds the smallest possible range after adding/subtracting
    // a value k to each element in the array
    int smallestRangeI(vector<int>& nums, int k) {
        // Find the maximum value in the array using std::max_element
        int maxVal = *std::max_element(nums.begin(), nums.end());

        // Find the minimum value in the array using std::min_element
        int minVal = *std::min_element(nums.begin(), nums.end());

        // Calculate the adjusted range after adding/subtracting k
        // The range cannot be negative, hence the use of std::max with 0
        return std::max(0, maxVal - minVal - 2 * k);
    }
};

```

Typescript Solution

```typescript
function smallestRangeI(nums: number[], k: number): number {
    // Find the maximum value in the array using reduce method.
    const maxValue = nums.reduce((currentMax, value) => Math.max(currentMax, value), -Infinity);

    // Find the minimum value in the array using reduce method.
    const minValue = nums.reduce((currentMin, value) => Math.min(currentMin, value), Infinity);

    // Calculate the adjusted range after the operation, ensuring it can't be less than 0.
    // The operation allows for increasing and decreasing each element by up to k.
    // The goal is to find the smallest possible range [minValue+k, maxValue-k]
    // then we calculate the difference between the maximum and minimum of that range
    // if it is less than 0, we return 0 as the range cannot be negative.
    return Math.max(maxValue - minValue - k * 2, 0);
}

```

Time and Space Complexity

Time Complexity

The time complexity of the code is `O(N)`, where `N` is the number of elements in the given `nums` list. This is because the function uses the `max` and `min` functions which each need to iterate over all elements of the list once to determine the maximum and minimum values, respectively.

Space Complexity

The space complexity of the code is `O(1)` as the memory usage is constant and does not depend on the input size `N`. Only a fixed number of variables (`mx`, `mi`, and the return value) are used, which occupy a constant amount of space.