Documentation for Minimum Moves to Equal Array Elements

Table of Contents

1. Problem Statement

Given an integer array number of size n, return the minimum number of moves required to make all array elements equal.

Constraints

- $1 \leq nums.length \leq 10^5$
- $-10^9 \leq nums[i] \leq 10^9$
- The answer is guaranteed to fit in a 32-bit integer.

Operations Allowed

- In one move, you can increment n-1 elements by 1.

Example 1

Input:nums = [1,2,3]
Output:3

Explanation:

Only three moves are needed:

- [1,2,3] → [2,3,3]
- [2,3,3] → [3,4,3]
- [3,4,3] → [4,4,4]

Example 2

Input:nums = [1,1,1]
Output:0
Explanation: All elements are already equal, so no moves are needed.

2. **Intuition**

Instead of trying to increase n-1 elements, we can think of the problem in reverse:

- Instead of incrementing n-1 elements, we can decrement one element at a time.
- The optimal way to make all elements equal is to bring all elements down to the minimum value in the array.

3. **Key Observations**

- The number of moves required to equalize all elements is the sum of differences between each element and the minimum element in the array.
- Mathematically, this can be represented as:

$$\text{Moves} = \sum(\text{num} - \min(\text{nums}))$$

4. Approach

- Find the minimum element in nums (min_num).
- Compute the sum of differences between each element and min_num.
- Return the total sum as the answer.

5. Edge Cases

i. All elements are already equal
   a. Example: [5,5,5]
   b. Moves required: 0

ii. Contains negative numbers
   a. Example: [-1,2,3]
   b. Moves required: 3 + 0 + 1 = 4

iii. Single-element array
   a. Example: [7]
   b. Moves required: 0

iv. Large input size (10^5 elements)
   a. The algorithm should run in O(n) time to be efficient.

6. Complexity Analysis

Time Complexity

- Finding the minimum element: O(n)
- Calculating the sum of differences: O(n)
- Total Complexity: O(n)

Space Complexity

- We use only a few extra variables: min_num and a sum accumulator.
- Space Complexity: O(1) (constant space)

7. Alternative Approaches

Brute Force Approach $O(n^2)$

- Increment n-1 elements iteratively until all elements become equal.
- Inefficient for large n, as it requires too many operations.

Sorting Approach O(nlogn)

i. Sort the array.
ii. Choose a target value (e.g., the median).
iii. Calculate the number of moves to make all elements equal to this value.

- Sorting takes O(nlogn), making it less optimal than the O(n) approach.

8. Test Cases

Test Case 1: Basic Input

Input:nums = [1,2,3]

Output: 3

Test Case 2: Already Equal Elements

Input: nums = [1,1,1]

Output: 0

Test Case 3: Negative Numbers

Input: nums = [-1,2,3]
Output: 4

Test Case 4: Large Input

Input:nums = [1000000] * 100000

Output: 0

9. Final Thoughts

- This problem can be efficiently solved in O(n) using a mathematical approach.
- The key insight is that increasing n-1 elements is equivalent to decrementing 1 element.
- The optimal way to solve it is to bring all elements down to the minimum value rather than increasing everything to a maximum.
- The implementation is simple, efficient, and works for large inputs.