Documentation

Table of Contents

1. Problem Statement

The problem requires computing the maximum value of a rotation function F(k) for an array num of length n, where F(k) is defined as the sum of each element multiplied by its index in the rotated array. The goal is to determine the maximum F(k) value across all possible rotations. The constraints ensure that the answer fits within a 32-bit integer.

2. Intuition

Rather than recomputing F(k) from scratch for each rotation, we look for patterns in how the function changes from one rotation to the next. We can efficiently compute the maximum function value by deriving a relationship between consecutive values.

3. Key Observations

- The initial function F(0) can be computed directly from the given array indices.
- The transition from F(k) to F(k+1) follows a specific formula involving the total sum of the array and the last element of the previous rotation.
- Instead of recalculating F(k) for each rotation, we can iteratively compute the next value based on the previous one.

4. Approach

- Compute F(0) using the given array indices.
- Compute the total sum of the array elements.
- Use a formula to compute F(k+1) from F(k) in constant time.
- Track the maximum value across all iterations.
- Return the maximum computed value.

5. Edge Cases

- A single-element array always results in F(k)=0F(k) = 0.
- An array with all identical elements results in the same function value for all rotations.
- Negative values need to be considered carefully to ensure correctness.

6. Complexity Analysis

Time Complexity

- Calculating F(0) takes O(n).
- Computing F(k) iteratively for all rotations takes O(n).
- Overall, the approach runs in O(n).

Space Complexity

- The solution uses only a few extra variables, leading to O(1) additional space complexity.

7. Alternative Approaches

- **Brute Force:** Compute F(k) for each rotation separately, leading to $O(n^2)$ complexity, which is inefficient for large n.
- **Using Prefix Sums:** Precompute prefix sums to reduce redundant calculations but does not offer a significant improvement over the optimized formula.

8. Related Topics

- **Dynamic Programming:** The recurrence relation between F(k) and F(k+1) can be viewed as a DP transition.
- **Mathematical Optimization:** The approach utilizes an algebraic transformation to minimize computations.

9. Test Cases

- **Basic Case:** nums = [4,3,2,6] → Output: 26
- **Single Element:** nums = [100] → Output: 0
- **All Identical Elements:** nums = [1,1,1,1] → Output: 6
- **Negative Values:** nums = [-1,-2,-3,-4] → Output: -6

10. Final Thoughts

This problem highlights the power of mathematical insights in optimizing computations. While a brute-force approach is infeasible for large inputs, recognizing the relationship between consecutive function values allows us to solve the problem efficiently. Understanding such patterns is crucial for tackling optimization problems in competitive programming and real-world applications.