Murasame's Contest 1　Editorial

01.29.2026

Contents

Notes

Some parts of this editorial are translated and polished by Gemini 3.

A Steps to Reach

We are looking for the smallest non-negative integer k such that $A + 3k \geq B$.
Rearranging the inequality for k: $3k \geq B - A \implies k \geq \frac{B-A}{3}$
Since k must be an integer, the minimum k is the ceiling of that division: $k = \lceil \frac{B-A}{3} \rceil$.
In integer arithmetic, $\lceil \frac{x}{y} \rceil$ can be calculated as `(x + y - 1) // y`.

B Single Replacement

Let S be the initial sum of the array. If we replace A_i with X, the new sum S' is $S' = S - A_i + X$.

We require $S' \equiv 0 \pmod{M}$, which implies $S - A_i + X \equiv 0 \pmod{M} \implies X \equiv A_i - S \pmod{M}$.

Let the required remainder be $R = (A_i - S) \pmod{M}$.

Any valid X must satisfy $X = k \cdot M + R$ for some integer k. Additionally, we are constrained by $0 \leq X \leq K$.

Algorithm:

1. Calculate the total sum S.
2. Iterate through every index i from 1 to N.
3. Calculate the target remainder $R = (A_i - S) \pmod{M}$.
4. We need to find $X \in [0, K]$ such that $X \equiv R \pmod{M}$ that minimizes $|X - A_i|$.
5. The candidates for best X are usually:
   - The smallest valid value: R (if $R \leq K$).
   - The value closest to A_i: specifically the multiples of M plus R that are just above or below A_i.
   - The largest valid value $\leq K$.
6. Check all valid candidates for a specific i, update the global minimum difference, and print the result.

C LCM Lollipops

Analysis:

Let $DP[i]$ be the maximum score possible using the first i candies.

$$DP[i] = \max_{0 \le j < i} \{DP[j] + \text{lcm}(A_{j+1}, \ldots, A_i)\}$$

A naive computation takes $O(N^2)$, which is too slow.

The Optimization:

Instead of looking back at all j, we iterate forward from i to $i+1$. We maintain a list of "Active Segments".

An active segment represents a potential last lollipop ending at the current index. Each segment needs to store only two values:

1. **Current LCM:** The LCM of the current block ending at i.
2. **Best Score:** The max total score achieved if we end the partition right before this block starts, plus the LCM of this block.

Let `active_list` be a list of pairs `{current_lcm, current_score}`.

When we move from A_i to A_{i+1}:

1. **Start a new block:** We can always start a fresh lollipop containing just A_{i+1}. The score for this option is `max_score_so_far` $+ A_{i+1}$.
2. **Extend existing blocks:** For every pair `{L, val}` in `active_list`, extending the block changes its LCM to $\text{lcm}(L, A_{i+1})$. The new score becomes `val - L + new_lcm`.

Key Merge Step:

Many different starting points j will eventually converge to the same LCM value as we extend to the right. If multiple segments result in the same LCM, we **only keep the one with the highest score**. This keeps the list size small.

Proof by Gemini:

The efficiency of this solution relies entirely on the size of the `active_list`. If the list stays small, the algorithm is fast.

Theorem:

At any index i, the number of distinct LCM values for all possible subarrays ending at i (i.e., $\text{lcm}(A_j \ldots A_i)$ for $1 \le j \le i$) is $O(\log(\text{MAX\_ANS}))$.

Proof:

1. **Sequence Definition:**
   Consider the sequence of LCMs formed by subarrays ending at i as we extend the left endpoint j backwards from i to 1.
   Let $L_j = \text{lcm}(A_j, A_{j+1}, \ldots, A_i)$.
2. **Divisibility Property:**
   Notice that $L_{j+1} = \text{lcm}(A_{j+1}, \ldots, A_i)$.
   Then $L_j = \text{lcm}(A_j, L_{j+1})$.
   By definition, L_{j+1} must divide L_j.
3. **Growth Rate:**
   Because L_{j+1} divides L_j, there are only two possibilities for step j:
   - **Case A:** $L_j = L_{j+1}$. The LCM value does not change.
   - **Case B:** $L_j > L_{j+1}$. Since L_{j+1} is a divisor of L_j and they are not equal, L_j must be at least twice L_{j+1} (since the smallest integer multiplier > 1 is 2).

   $$L_j \geq 2 \cdot L_{j+1}$$
4. **Bounding the Size:**
   We only store **distinct** LCM values in our list.
   Let the distinct values be $v_1, v_2, \ldots, v_k$ in increasing order.
   From logic above, $v_{m+1} \geq 2 \cdot v_m$.
   The maximum possible LCM value we care about is bounded by the problem constraint on the answer (10^{18}), or realistically the LCM of all elements. Even if we don't bound it by the answer, it is bounded by the product of primes fitting in the integer type.
   The number of times you can double a number before exceeding 10^{18} is $\log_2(10^{18}) \approx 60$.
5. **Conclusion:**
   The size of `active_list` never exceeds ≈ 60.
   In each of the N steps, we iterate through this list, performing GCD/LCM operations.

   $$\text{Total Operations} \approx N \times 60$$

   For $N = 200,000$, this is roughly 1.2×10^7 operations, which easily fits within the 1-2 second time limit (typically handling $\sim 10^8$ operations).

D Pythagorean Triples

A brute force $O(N^2)$ iterating a and b will time out given $N = 10^6$. We need a more mathematical approach.

Rearrange the equation: $a^2 = c^2 - b^2 = (c - b)(c + b)$.

Let $x = c - b$ and $y = c + b$; then $a^2 = x \cdot y$.

Since b, c are integers, x and y must have the same parity (both even or both odd), because their sum $(x + y) = 2c$ and difference $(y - x) = 2b$ must be even.

Algorithm:

1. Iterate a from 1 to N.
2. Factorize a^2 into pairs (x, y) such that $x \cdot y = a^2$ and $x < y$.
3. To factorize efficiently, we can precompute smallest prime factors (Sieve) for numbers up to N. Note that we need factors of a^2, which are just the factors of a with exponents doubled.
4. For each valid pair (x, y) with matching parity: $c = (x + y)/2$, $b = (y - x)/2$
5. Check constraints: $c \leq N$ and $b > a$ (to ensure strict ordering and avoid duplicates).
6. Store valid triples and sort them at the end.

This problem have other solutions, feel free to share yours.

E Color Merge

1. Key Observations

The Target Value

The problem asks us to make all elements identical. Since the only allowed operation is to change a value x to an adjacent value y, we cannot introduce new numbers into the array. The final value (let's call it T) must be one of the values initially present in the array. Since N is small ($N \leq 300$), we can iterate through every distinct value in the array and try solving the problem assuming T is the final value. The answer is the minimum cost over all valid choices of T.

Transforming a Segment

Suppose we have a contiguous segment of numbers and a neighbor immediately outside this segment that already has the value T. We want to convert this entire segment to T.

What is the optimal strategy?

It is never optimal to change a number to a value larger than itself before changing it to T (this only increases the product cost).

The optimal strategy for a segment is:

1. Identify the minimum value, m, within that segment.
2. Convert all other numbers in the segment to m. This effectively "flattens" the segment.
3. Once the entire segment consists of the value m, convert each m to T using the neighbor that is already T.

Why is this optimal?

By converting to the minimum value m first, we minimize the intermediate costs. Any transition involving a value larger than m would incur a higher product cost.

2. Calculating Costs

Let's formalize the cost to convert a range $A[l \dots r]$ into T, assuming we have a helper T adjacent to the range (e.g., at $l-1$ or $r+1$).

1. **Flattening Cost:**

   We change every $A[k]$ in the range ($l \leq k \leq r$) to m, where $m = \min(A[l \dots r])$.

$$\text{Cost}_{\text{flat}} = \sum_{k=l}^{r} \begin{cases} A[k] \times m & \text{if } A[k] \neq m \\ 0 & \text{otherwise} \end{cases}$$

2. **Transformation Cost:**
   After flattening, we have $(r - l + 1)$ elements of value m. We need to change each of them to T.

$$\text{Cost}_{\text{transform}} = (r - l + 1) \times m \times T$$

   (Note: If $m = T$, this cost is 0).

   Total Cost for range $= \text{Cost}_{\text{flat}} + \text{Cost}_{\text{transform}}$.

3. DP

We cannot simply run a DP from 1 to N because we need a valid "source" of T to start the transformations. We use an **Anchor/Pivot** approach.

For a fixed target T, we iterate through every index p such that $A[p] == T$ originally. We treat $A[p]$ as the immutable anchor. We then solve two independent subproblems:

1. Convert the left prefix $A[1 \dots p-1]$ to T.
2. Convert the right suffix $A[p+1 \dots N]$ to T.

DP State:

Let $L[i]$ be the minimum cost to convert the prefix $A[1 \dots i]$ to T, such that $A[i]$ ends up as T (and can act as a helper for the left side).

Transition:

To compute $L[i]$, we iterate over a split point $j < i$. This assumes $A[1 \dots j]$ is already converted. We then convert the segment $A[j+1 \dots i]$ to T using $A[j]$ (which is T) as the helper.

$$L[i] = \min_{0 \le j < i} \{L[j] + \text{Cost}(j+1, i)\}$$

We define $R[i]$ similarly for the suffix, iterating backwards.

4. Algorithm

1. Precompute `FlattenCost[i][j]` and `MinVal[i][j]` for all $1 \le i \le j \le N$. This takes $O(N^3)$.
2. Collect all unique values from input A.
3. For each unique 'target':
   - Compute array L using DP: $O(N^2)$.

- Compute array R using DP: $O(N^2)$.
- Find the best pivot p (where $A[p] == target$): Answer $= \min(L[p-1]+R[p+1])$.

4. Output the global minimum.

5. Complexity

- **Precomputation:** $O(N^3)$.
- **DP Calculation:** There are at most N unique targets. For each target, the DP is $O(N^2)$. Total is $O(N^3)$.
- **Total Complexity:** $O(N^3)$.

Given $N \leq 300$, $N^3 \approx 2.7 \times 10^7$, which fits comfortably within the time limit.