Problem A. Sequence Conversion

Problem Idea: Eunsoo Choe (gs18115)

Preparation: Jimin Ahn (retro3014)

Let $\{a'_i\}$ be the sequence, which satisfies that a'_i is equal to xor sum of a_1, a_2, $\ldots$, a_i. Define $\{b'_i\}$ similarly. If $a'_n \neq b'_n$, the answer is -1. Otherwise, the answer is the number of index i such that $a'_i \neq b'_i$.

Shortest solution: 325 bytes

Problem B. Histogram

Problem Idea: Eunsoo Choe (gs18115)

Preparation: Eunsoo Choe (gs18115)

We can observe that the optimal way to use cash is using cash as soon as possible. If the amount of cash is already decided, we can decide whether we can buy every items by simulating. Now, the answer can be found with binary search.

Shortest solution: 402 bytes

Problem C. TOO EASY Cookie Run

Problem Idea: Jimin Ahn (retro3014)
Preparation: Jimin Ahn (retro3014)

Subtask 1 (10 points)

Because max available X is small in this subtask, we can check if there are at least K interesting sections for all X, starting from 0 and increasing it.
After adding X to all A_i, we can check if there are at least K interesting sections for given X by calculating $A_i + A_{i+1} + ... + A_j$ for all pairs (i, j). By this, we can count the number of pairs that satisfies $A_i + A_{i+1} + ... + A_j \geq M$.
The total time complexity is $O(XN^2)$.

Subtask 2 (20 points)

We can check if there are at least K interesting sections for all available X, similarly to Subtask 1.
However, as N increased, we can not calculate $A_i + A_{i+1} + ... + A_j$ for all pairs (i, j).
Instead, if we find the smallest j that satisfies $A_i + A_{i+1} + ... + A_j \geq M$ for the given i, we can say that for all $k \geq j$, $A_i + A_{i+1} + ... + A_k \geq M$, because array $\{A_i\}$ consists of only non-negative integers.
Using this condition, we can calculate total number of pair (i, j) that satisfies $A_i + A_{i+1} + ... + A_j \geq M$ by only finding smallest j that satisfies the same condition for each i.

To find the smallest j that satisfies $A_i+A_{i+1}+...+A_j \geq M$ for the given i, we can use Two Pointers algorithm.

Starting from $(i, j) = (1, 1)$, if $A_i+A_{i+1}+...+A_j < M$, we should increase j by 1. If not, $A_i+A_{i+1}+...+A_j \geq M$, and we have found smallest j satisfying the condition for current i, so we should increase i by 1 and continue finding smallest j for new i. In this algorithm i or j increases by 1 in single iteration, so this process takes time complexity of $O(N)$.

Therefore, total time complexity is $O(XN)$.

Also we can use data structures such as Segment Tree, instead of Two Pointer algorithm to find smallest j for all i. In this way total time complexity is $O(XN \log N)$.

Subtask 3 (30 points)

For given X, we can check if there are at least K interesting sections after adding X to all A_i, simply by calculating $A_i + A_{i+1} + ... + A_j$ for all pairs (i, j). This has time complexity of $O(N^2)$.

Also, lets say X_{min} is smallest non-negative integer X that satisfies the given conditions. For all $X \geq X_{min}$, there are at least K interesting sections if we add difficulty X to all stages. Also for all $X < X_{min}$, there are less than K interesting sections if we add difficulty X to all stages. Using this condition, we can use Parametric Search algorithm to find X_{min}.

Say that $s \leq X_{min} \leq e$. Then, if we let $m = \lfloor \frac{s+e}{2} \rfloor$, if there are at least K interesting sections after adding m to all A_i, X_{min} should be at most m, so we can update $e = m$. If there are less than K interesting sections after adding m to all A_i, X_{min} should have bigger value than m, so we can update $s = m + 1$. As initial values are $(s, e) = (0, M)$, time complexity of parametric search algorithm is $O(\log M)$.

Total time complexity is $O(N^2 \log M)$.

Also we should beware of overflow, because $N \times M$ can exceed the maximum *long long* value.

Subtask 3 (40 points)

Using Parametric Search algorithm used in subtask 3, and Two Pointer algorithm used in subtask 2, we can get full score with total time complexity $O(N \log M)$. Also we can use data structures such as Segment Tree instead of Two Pointer algorithm, having total time complexity $O(N \log N \log M)$.

Shortest solution: 383 bytes

Problem D. Sequence Conversion 2

Problem Idea: Jimin Ahn (retro3014)
Preparation: Jimin Ahn (retro3014)

Subtask 1 (20 points)

This subtask can be solved by brute-forcing. Using $(x \oplus y) \oplus z = x \oplus (y \oplus z)$, we can map current state into binary array of length N, so total number of state in brute-force process is at most 2^N.

The total time complexity is $O(N2^N)$.

Beware of maximum number of operations, if we apply $N-2$ operations, length of array is 2 so it is always zig-zag array.

Subtask 2 (30 points)

Lets say we are making zig-zag array using first j integers. Last integer in this array must be $A_i \oplus A_{i+1} \oplus ... \oplus A_j$ for some $i \le j$.
Lets define

- $DP[0][i][j]$ = *minimum operations needed to make zig-zag array using first j integers, ending with $A_i \oplus A_{i+1} \oplus ... \oplus A_j$, and last integer in this array should be bigger than the integer before that.*
- $DP[1][i][j]$ =*minimum operations needed to make zig-zag array using first j integers, ending with $A_i \oplus A_{i+1} \oplus ... \oplus A_j$, and last integer in this array should be smaller than the integer before that.*

We can calculate these values by following ways.

- $DP[0][i][j] = max(DP[1][k][i-1] + (j-i))$ for $k < i$ that $A_k \oplus A_{k+1} \oplus ... \oplus A_{i-1} < A_i \oplus A_{i+1} \oplus ... \oplus A_j$.
- $DP[1][i][j] = max(DP[0][k][i-1] + (j-i))$ for $k < i$ that $A_k \oplus A_{k+1} \oplus ... \oplus A_{i-1} > A_i \oplus A_{i+1} \oplus ... \oplus A_j$.

After calculating $DP[0][i][j]$ and $DP[1][i][j]$ for all (i, j), answer is minimum value of $DP[0][i][N]$ and $DP[1][i][N]$ for all i.

Total time complexity is $O(N^3)$.

Subtask 3 (50 points)

Lets say that we made zig-zag array using first i integers. Only important information here is the value of last integer, whether if last integer smaller or bigger than the one before, and number of operations that are used.

If j operations are used to make first i integers into zig-zag array, and last integer is bigger than the one before, we should only think about maximum value of last integer. As the next element added to the array must have smaller value than the last integer, it is better to last with bigger value.

Lets define

- $DP[0][i][j]$ = *maximum value of last integer in zig-zag array using first* i *integers and having less than* j *integers, and last integer in this array should be bigger than the integer before that.*
- $DP[1][i][j]$ = *minimum value of last integer in zig-zag array using first* i *integers and having less than* j *integers, and last integer in this array should be smaller than the integer before that*

For $k > j$, we can update $DP[0][k][t]$ with $A_{i+1} \oplus A_{i+2} \oplus ... \oplus A_k$ if $DP[1][i][t-1] < A_{i+1} \oplus A_{i+2} \oplus ... \oplus A_k$. We can update $DP[1][k][t]$ using similar idea.

As $DP[0][i][j] \leq DP[0][i][j+1]$ for all (i, j) and $DP[1][i][j] \geq DP[1][i][j+1]$, we can find biggest t using binary search algorithm for every k, and update $DP[0][k][t]$ and $DP[1][k][t]$.

Total time complexity is $O(N^2 \log N)$.

Shortest solution: 1505 bytes

Problem E. Comparing Fractions

Problem Idea: Eunsoo Choe (gs18115)

Preparation: Eunsoo Choe (gs18115)

If both of $A \le C$ and $B \ge D$ are satisfied together, we can decide which fraction is larger. Similarly, we can also compare the fractions when $A \ge C$ and $B \le D$. Remaining case is $A < C$ and $B < D$ has same truth value. Without loss of generality, we can assume that $A < C$ and $B < D$. Let $p = min\{\lceil \frac{C}{A} - 1 \rceil, \lceil \frac{D}{B} - 1 \rceil\}$. $\frac{A}{B} < \frac{C}{D}$ is equivalent to $\frac{A}{B} < \frac{C-pA}{D-pB}$. p, $C - pA$, and $D - pB$ are calculated with $O(\log p)$ operations. Now, we can solve the problem recursively. The number of used operations is $O(\log A + \log B + \log C + \log D)$.

Shortest solution: 2699 bytes

Problem F. Tree GCD

Problem Idea: Sungjae Im
Preparation: Sungjae Im

Subtask 1 (10 points)

For all $1 \le i < j \le N$ compute, all $dist(i, j)$ in $O(N^2)$ time. Then, we can calculate

$$\sum_{1 \le i < j \le N} \gcd(i, j, dist(i, j)$$

in $O(N^2 \log N)$ time.

Subtask 2 (20 points)

$$\sum_{1 \le i < j \le N} \gcd(i, j, dist(i, j))$$

$$= \sum_{d=1}^{N} \sum_{i=1}^{N} \sum_{j=i+1}^{N} d \cdot [\gcd(i, j, dist(i, j)) = d]$$

$$= \sum_{d=1}^{N} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{g \mid d} \phi(g) \cdot [\gcd(i, j, dist(i, j)) = d]$$

$$= \sum_{d=1}^{N} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{g \mid d} \phi(g) \cdot [g \mid \gcd(i, j, dist(i, j))]$$

$$= \sum_{g=1}^{N} \phi(g) \sum_{i=1}^{N} \sum_{j=i+1}^{N} [g \mid \gcd(i, j, dist(i, j))]$$

$$= \sum_{g=1}^{N} \phi(g) \sum_{i=1}^{\lfloor \frac{N}{g} \rfloor} \sum_{j=i+1}^{\lfloor \frac{N}{g} \rfloor} [g \mid dist(gi, gj)]$$

First, we can compute $\phi(g)$ for all $1 \le g \le N$ in $O(N)$ time by using some algorithms such as linear sieve. Thus, we have to get

$$\sum_{i=1}^{\lfloor \frac{N}{g} \rfloor} \sum_{j=i+1}^{\lfloor \frac{N}{g} \rfloor} [g \mid dist(gi, gj)]$$

for all $1 \le g \le N$.

Since the given tree has no vertex having degree more than 2, the given graph is a line. Let's say v is a leaf of the given tree. Let's denote $d(i) = dist(v, i)$. Then, $dist(i, j) = |d(i) - d(j)|$ for any vertex i and j. So,

$$\sum_{1 \le i < j \le N} \gcd(i, j, dist(i, j))$$

$$= \sum_{g=1}^{N} \phi(g) \sum_{i=1}^{\lfloor \frac{N}{g} \rfloor} \sum_{j=i+1}^{\lfloor \frac{N}{g} \rfloor} [g \mid \{d(gi) - d(gj)\}]$$

$$= \sum_{g=1}^{N} \phi(g) \sum_{i=1}^{\left\lfloor \frac{N}{g} \right\rfloor} \sum_{j=i+1}^{\left\lfloor \frac{N}{g} \right\rfloor} [d(gi) \equiv d(gj) \mod g]$$

For all g, compute all $(d(gi) \mod g)$ where $1 \le i \le \left\lfloor \frac{N}{g} \right\rfloor$. After that, we can count the number of pairs such that $d(gi) \equiv d(gj) \mod g$. All these calculation takes $O(\left\lfloor \frac{N}{g} \right\rfloor)$ time.
Therefore, total time complexity is

$$O(N) + \sum_{g=1}^{N} O\left(\left\lfloor \frac{N}{g} \right\rfloor\right) = O(N \log N)$$

Subtask 3 (70 points)

Similar to the previous subtask, Thus, we have to compute

$$\sum_{i=1}^{\left\lfloor \frac{N}{g} \right\rfloor} \sum_{j=i+1}^{\left\lfloor \frac{N}{g} \right\rfloor} [g \mid dist(gi, gj)]$$

for all $1 \le g \le N$.
Let's fix g. Then, what we need to get is the number of pairs (gi, gj) such that $dist(gi, gj)$ is divided by g. Thus, get the virtual tree which contains all vertices with index multiple of g. This takes $O\left(\left\lfloor \frac{N}{g} \right\rfloor \log \left\lfloor \frac{N}{g} \right\rfloor\right)$ time. After that, we cam compute the number of the pairs in $O\left(\left\lfloor \frac{N}{g} \right\rfloor \log \left\lfloor \frac{N}{g} \right\rfloor\right)$ time using the centroid decomposition or small to large technique.
Therefore, we can solve this problem in the time complexity of

$$\sum_{g=1}^{N} O\left(\left\lfloor \frac{N}{g} \right\rfloor \log \left\lfloor \frac{N}{g} \right\rfloor\right) = O(N \log^2 N)$$

In addition, some fast $O(N \log^3 N)$ solution might be accepted.
Shortest solution: 1060 bytes

Problem G. Counting Rectangles

Problem Idea: Jongyoung Lee (moonrabbit2)
Preparation: Jongyoung Lee (moonrabbit2)

Subtask 1 (30 points)

We will calculate the number of newly created rectangles after each query. To do this, we modify the algorithm for calculating the number of black rectangles in a grid. In this algorithm, for each row, we calculate the number of rectangles ending at this row by calculating the number of consecutive black cells ending at this row for each column, then count the number of rectangles inside a histogram made from this sequence using stack. This algorithm already gives a way to solve when you only append an element in A. When you also append elements in B, you can do this algorithm in two directions: do the same thing with columns. The time complexity is $O(NM)$.

Subtask 2 (15 points)

We only need to calculate the answer in the final grid. For a rectangle $(l_1, r_1, l_2, , r_2)$, this rectangle is a valid black rectangle if $\min_{l_1 \le i \le r_1} A_i + \min_{l_2 \le j \le r_2} B_j \ge 0$.

For each i, we can calculate the rightmost point l such that $A_l < A_i$ and leftmost point r such that $A_r < A_i$, with breaking ties in a fixed order using stack. Then, the number of ranges such that minimum in the range is A_i is $(i - l + 1)(r - i + 1)$. Let X be the array of pairs $(A_i, (i - l + 1)(r - i + 1))$ in a sorted order. We can calculate array X in $O(N \log N)$ time complexity.

We apply the same algorithm to B to obtain pairs Y_j. Then, the answer is sum of $X_i.y * Y_j.y$ for all (i, j) such that $X_i.x + Y_j.x \ge 0$. This value can be calculated in $O(N + M)$, since X and Y are sorted. Total time complexity is $O(N \log N + M \log M)$.

Subtask 3 (17 points)

Let the rightmost point l such that $A_l < A_i$ be L_i, and leftmost point r such that $A_r < A_i$ be R_i. We define S_i and E_i similarly with B. If current length of A is n and current length of B is m, the current answer is $\sum_{i=1}^{n} \sum_{j=1}^{m} (A_i + B_j \ge 0)(i - L_i + 1)(\min(R_i, n) - i + 1)(j - S_j + 1)(\min(E_j, m) - j + 1)$ as R_i may be larger than n since it is calculated with the final array.

As in subtask 1, we will calculate the number of newly created rectangles after each query. If n is increased by $1(n - 1$ to $n)$, then the number of newly created rectangles are $\sum_{i=1}^{n} \sum_{j=1}^{m} (R_i \ge n)(A_i + B_j \ge 0)(i - L_i + 1)(j - S_j + 1)(\min(E_j, m) - j + 1)$ Let this value be Add_A.
We also define Add_B as the number of newly created rectangle when m is increased by $1(m - 1$ to $m)$, which is $\sum_{i=1}^{n} \sum_{j=1}^{m} (E_i \ge m)(A_i + B_j \ge 0)(i - L_i + 1)(\min(R_i, n) - i + 1)(j - S_j + 1)$.

Now we should update Add_A and Add_B when n is increased. We can maintain Add_A and Add_B in a following way. The case where m is increased can be done symmetrically.

- Let $Add_C = \sum_{i=1}^{n} \sum_{j=1}^{m} (R_i \ge n)(E_i \ge m)(A_i + B_j \ge 0)(i - L_i + 1)(j - S_j + 1)$.

- For each i such that $R_i = n-1$, subtract $(i-L_i+1)\sum_{j=1}^{m}(A_i+B_j \geq 0)(j-S_j+1)(\min(m,E_j)-j+1)$ from Add_A.
- For each i such that $R_i = n-1$, subtract $(i-L_i+1)\sum_{j=1}^{m}(E_j \geq m)(A_i+B_j \geq 0)(j-S_j+1)$ from Add_C.
- Add $(n-L_n+1)\sum_{j=1}^{m}(A_n+B_j \geq 0)(j-S_j+1)(\min(m,E_j)-j+1)$ to Add_A.
- Add $(n-L_n+1)\sum_{j=1}^{m}(E_j \geq m)(A_n+B_j \geq 0)(j-S_j+1)$ to Add_C.
- Add Add_C to Add_B.

To do this in a fast way, $(j-S_j+1)(\min(E_j,m)-j+1)$ is a piecewise linear function for m, and $A_i+B_j \geq 0$ means that $B_j \geq -A_i$. Therefore, if we maintain these linear functions with a segment tree, each operation can be done in $O(\log N)$.
Shortest solution: 3046 bytes

Problem H. Strange Graph

Problem Idea: Jongyoung Lee (moonrabbit2)
Preparation: Jongyoung Lee (moonrabbit2)

Subtask 1 (7 points)

As there is at most $O(N^2)$ edges, we may use Kruskal's algorithm to calculate the minimum spanning forest.

Subtask 2 (23 points)

Assume that every elements in A are different, by adding different very small values for each element.

Let's fix $x = u \bmod K$ and $y = v \bmod K$, and calculate the minimum spanning tree of the subgraph created by these edges.

For each u, it is enough to only consider edges to vertex v with largest $A_{v,x}$ such that $A_{v,x} < A_{u,y}$ and smallest $A_{v,x}$ such that $A_{u,y} < A_{v,x}$.

For each v, it is enough to only consider edges to vertex u with largest $A_{u,y}$ such that $A_{u,y} < A_{v,x}$ and smallest $A_{u,y}$ such that $A_{v,x} < A_{u,y}$.

To calculate the minimum spanning tree of the whole graph, we calculate the minimum spanning tree out of edges of each subgraphs. There are $O(NK)$ edges in total, so we can use Kruskal's algorithm to calculate answer in $O(NK \log NK)$.

Subtask 3 (46 points)

We use Borvuka's algorithm. In each step, for each vertex u and x, we will find the vertex v such that $v \bmod K = x$, v is in different component with u, and $|A_{u,x} - A_{v,u \bmod K}|$ is minimal among those v.

We solve this subproblem for each components. For current component C, let $S_{i,j}$ be the set of $A_{u,j}$ where $u \bmod K = i$ and $S(C)_{i,j}$ be the set of $A_{u,j}$ where $u \bmod K = i$ and $u \notin C$. We can earn $S(C)_{i,j}$ by removing $A_{u,j}$ where $u \bmod K = i$ and $u \in C$ from $S_{i,j}$. Therefore, $S(C)$ can be calculated in $O(|C| \, K \log \frac{N}{K})$ for each C, if we maintain S with data structures which supports insertion and deletion in $O(\log N)$ like `std::set`.

Finding v for a fixed u and x can be done by finding smallest element larger than $A_{u,x}$ and largest element smaller than $A_{u,x}$ on $S(C)_{x,u \bmod K}$. To take care of edge deletions, if the found edge is deleted, we will "skip" the current edge and try for second smallest/largest element, and so on.

The total number of skips are $O(M)$, so each step can be done in $O((NK + M) \log \frac{N}{K})$, and the total time complexity is $O((NK + M) \log \frac{N}{K} \log N)$.

Subtask 4 (24 points)

We also use another "skip" for same components. Then, we can use the same starting position in $S_{x,u \bmod K}$ in every step, which can be calculated before applying Borvuka's algorithm.

Since naively skipping is too slow, you should implement skipping smartly. One way of doing this is storing the next vertex in different component for each visited vertex in same component.

The total time complexity is $O((NK+M)\log N)$. Fast implementations of $O((NK+M)\log\frac{N}{K}\log N)$ solution may also pass.
Shortest solution: 3645 bytes