Problem A. Arrow Array

Let's say that there are k total right-facing arrows, and let c_i be the number of right-facing arrows among the first i arrows. If $a_i > 0$, then the number of arrows that face that position is $c_i + (n - i - (k - c_i))$, so the constraint here is $c_i \geq \frac{a_i+i-n+k}{2}$, and similarly the constraint if $a_i < 0$ is $c_i \leq \frac{a_i+i+k}{2}$. In both cases, we can say that $l_i \leq c_i \leq r_i$ for some l_i and r_i that increase by 1 every time k increases by 2(if $a_i > 0$ then r_i can be set to ∞, and if $a_i < 0$ then l_i can be set to $-\infty$).

Let's say that we know that the range of possible c_i is $[u_i, v_i]$ for some u_i and v_i. Then, the range of possible c_{i+1} is $[u_{i+1}, v_{i+1}] = [\max(u_i, l_{i+1}), \min(v_i + 1, r_{i+1})]$. We can see that there's a solution iff by starting with $u_0 = v_0 = 0$ we never end up with $u_i > v_i$ and in the end $v_{n-1} \geq k-1$ and $u_{n-1} \leq k$; if that's the case, we can reconstruct it quite easily by working backwards. The former condition is equivalent to checking that starting with $u_0 = -\infty$, $v_0 = \infty$ never ends up with $u_i > v_i$(this isn't affected by increasing all l_i and r_i by the same amount) and checking that the starting position doesn't cause $u_i > v_i$(this happens iff $l_i > i$ or $r_i < 0$ for some i); increasing all l_i and r_i by the same amount doesn't affect the first part and we can easily see if the second part happens by finding the maximum value of $l_i - i$ and minimum value of r_i. To check for the latter condition, notice that starting with $u_0 = 0$, $v_0 = 0$ only changes something compared to starting with $u_0 = -\infty$, $v_0 = \infty$ if $k < 0$ or $k > n$, and starting with $u_0 = -\infty$, $v_0 = \infty$ means that increasing all l_i and r_i by the same amount must also increase all u_i and v_i by the same amount, including u_{n-1} and v_{n-1}.

So, we can check if $k = 2s$ is possible by calculating all l_i, r_i when $k = 0$, calculating all u_i, v_i if $u_0 = -\infty$ and $v_0 = \infty$ using those l_i and r_i, making sure $u_i \leq v_i$ for all i, and then finally checking that $\max(l_i - i) + s \leq 0$, $\min(r_i) + s \geq 0$, $v_{n-1} + s \geq k - 1$ and $u_{n-1} + s \leq k$, and similarly check if $k = 2s + 1$ is possible by starting all the calculations with $k = 1$, and once we find a suitable value of k we can easily reconstruct the answer.

Bonus: try to solve the problem if there can be updates to a_i and after each update you have to say whether a solution exists or not.

Problem D. Dangerous Data

If you maintain the k leading coefficients(apart from the leading coefficient, which is always 1) of a polynomial which is the product of $(x - a_i)$ for all a_i in a set modulo $10^6 + 3$(a prime), it's pretty easy to add or remove elements from the set in $O(k)$. So, in the first run, output the k leading coefficients after adding all $n + k$ numbers into the set, and in the second run remove the n numbers you're given from the set; now you have every coefficient of the product of $(x - a_i)$ since you store k of them and the set now has size k; now you just need to find all roots of that polynomial, which you can do by trying every number from 1 to 10^6. The set of the roots of the polynomial must obviously contain set B, and it's well known that a polynomial of degree k has at most k roots in any field(which the integers modulo $10^6 + 3$ are), so it's impossible for the polynomial to contain any other roots.

Problem E. Exchange Error

Let's divide the array into blocks of size k. Each subarray is either completely contained in one of those blocks, or has a left end in the x-th block and its right end in the y-th block for some $x < y$.

To solve the case of a subarray being completely contained in one of the blocks, we loop over the blocks; for each block, we find the maximum sum of a subarray for each length in $O(k^2)$, then we look at the convex hull of the lines $y = lx + s$ where l is the length and s is the maximum sum of a subarray of that length, and if we sort the queries we can find the maximum subarray sum for all queries in $O(k + q)$, so the complexity of this part is $O(\frac{n}{k}(k^2 + q)) = O(nk + \frac{nq}{k})$.

To solve the case of a subarray having a right end in the k-th block, let's assume we already have the convex hull of all lines $y = ix - ps_i$ where ps_i is the sum of the first i elements of a. That means that we can find the maximum value of -(prefix sum) before this block for each query using 2 pointers in $O(n+q)$, and we can also make a convex hull of all lines $y = ix + ps_i$ inside this block and find the maximum value of prefix sum in this block for all queries in $O(k+q)$, and summing those 2 values together gives us the maximum

subarray sum which ends in this block. Overall, the complexity of this part is $O(\frac{n}{k}(n+q)) = O(\frac{n^2}{k} + \frac{nq}{k})$.

Summing those 2 together, we get a complexity of $O(\frac{n^2}{k} + \frac{nq}{k} + nk)$. By choosing $k = \sqrt{n+q}$, we get a complexity of $O(n\sqrt{n+q})$. In practice, you should probably make your block size larger as the constant factor of finding the maximum sum of a subarray for each length is much smaller than the rest of the solution.

Problem F. Flipping Frenzy

Notice that these operations are equivalent to being able to flip an $a_i \times b_i$ rectangle in the top left corner, being able to flip the first a_i tiles in 2 columns that are b_i apart(the result of doing 2 operations in the first row and consecutive columns) and being able to flip the first b_i tiles in 2 rows that are a_i apart(the result of doing 2 operations in the first column and consecutive rows). This means that every operation that can be done can be represented as doing some operations that are either flipping the first a_j tiles in the i-th column for some i and j(let's call this type of operation $C(i,j)$) or flipping the first b_j tiles in the i-th row for some i and j(let's call this type of operation $R(i,j)$).

However, we can't uniquely define which of these column and row operations we'll have to do; for each $1 \leq i,j \leq k$, doing operations $C(1,i), C(2,i), \ldots, C(b_j,i), R(1,j), R(2,j), \ldots, R(a_i,j)$ is the same as not doing an operation at all; however, if we ignore operations of the form $R(a_i,j)$, we can now uniquely identify which operations we have to do to make the grid into 0s, and then we can handle doing those operations by saying that we're allowed to do an operation that does $C(1,i), C(2,i), \ldots, C(b_j,i), R(1,j), R(2,j), \ldots, R(a_i,j)$(so grid will stay the same after this).

Now, what we're allowed to do is choose $1 \leq i \leq k$ and $1 \leq x \leq m - b_i$ and do operations $C(x,i)$ and $C(x+b_i,i)$, choose $1 \leq i \leq k$ and $1 \leq x \leq n - a_i$ and do operations $R(x,i)$ and $R(x+a_i,i)$, choose $1 \leq i \leq k$ and do operations $C(1,i), C(2,i), \ldots, C(b_i,i)$, or choose $1 \leq i,j \leq k$ and do operations $C(1,i), C(2,i), \ldots, C(b_j,i), R(1,j), R(2,j), \ldots, R(a_i,j)$, we know which row and column operations we have to do an even and odd number of times(not including the k^2 reduntant row operations mentioned in the previous paragraph) and we need to find a way to do that. We have $O(k(n+m))$ options that only change 2 things, and $k(k+1)$ other options. Let's make whether we choose each of the $k(k+1)$ other options a variable. Now, each connected component in a graph where the vertices are the row and column operations and the edges are the options that do 2 of those row and column operations gives us 1 equation: out of the $k(k+1)$ other options which toggle an odd number of the row and column operations in this connected component in total, the parity of the amount that we choose must be the same as the parity of the amount of operations we have to do on this connected component if we ignore operations of the form $R(a_i,j)$ (which we've uniquely identified before). Now we just have to solve a system of $O(k(n+m))$ equations with $k(k+1)$ variables, which we can do in $O(\frac{k^5(n+m)}{w})$ to find which of the other $k(k+1)$ other options we choose.

To find the solution after that, we have a problem of, in a graph where nodes are black and white, with each connected component now having an even number of black nodes, you can choose an edge to flip both endpoints of and you need to turn all nodes into white, which can be solved by making a spanning forest and handling nodes from deepest to the root. Now that we know which edges we need to flip(=doing operations $C(x,i)$ and $C(x+b_i,i)$(which is outputting i 1 x and i 1 $x+1$) or $R(x,i)$ and $R(x+a_i,i)$(which is outputting i x 1 and i $x+1$ 1)) and which rectangles in the top left corner we need to flip(what k of the $k(k+1)$ variables in the system of equations were), we have the answer.

Problem I. Intricate Instrument

It turns out that the condition from the statement is equivalent to an array being strictly increasing and one of the following 2 things being true:

— the array is an arithmetic progression with step $\frac{m}{n}$

— for each $1 \leq k < n$, there are exactly 2 distinct values of $u_{i+k \mod n} - u_i \mod m$

Let's first prove that the condition from the statement implies that one of those 2 things is true. Let

$r \equiv k * b^{-1} \pmod{n}$. Then, for $j < n - r$,

$$u_{a+bj+k \mod n} - u_{a+bj \mod n} = u_{a+b(j+r) \mod n} - u_{a+bj \mod n} \equiv c + d(j+r) - (c+dj) \equiv dr \pmod{m}$$

while for $j \geq n-r$ it can be seen in the same way that the difference is $d(r-n) \mod m$. If $nd \equiv 0 \pmod{m}$, then the array is an arithmetic progression with step $\frac{m}{n}$(all values of $u_{i+k \mod n} - u_i \mod m$ are equal, so applying that to $i = e$ and $i = e+1 \mod n$ we see that $u_{e+1 \mod n} - u_e \equiv u_{e+k+1 \mod n} - u_{e+k \mod n} \pmod{m}$, so since n is prime this means all $u_{i+1 \mod n} - u_i$ are equal modulo m), otherwise these 2 values will always be different.

The case where the array is an arithmetic progression with step $\frac{m}{n}$ can be solved very easily, so let's now prove that, if for each $1 \leq k < n$, there are exactly 2 distinct values of $u_{i+k \mod n} - u_i \mod m$ and u is increasing, there exist a, b, c, and d that generate u.

Let the 2 distinct values for $k = 1$ be x_1 and y_1, $x_1 < y_1$, and let's say that x_1 appears c_1 times. Let $k \equiv c_1^{-1} \pmod{n}$. Let the set of all quantities of adjacent differences equal to x_1 in the "circular intervals" of size k be S. S must be an interval, since changing the starting point by 1 changes the quantity of adjacent differences equal to x_1 by at most 1. Its size obviously can't be 1 since that would mean there would only be 1 distinct value of $u_{i+k \mod n} - u_i \mod m$. Let's say that its size is ≥ 3, this means that there exists x such that $x-1$, x and $x+1$ are all in S, which means that $(x-1)x_1 + (k-(x-1))y_1 \mod m$, $xx_1 + (k-x)y_1 \mod m$ and $(x+1)x_1 + (k-(x+1))y_1 \mod m$ are all values of $u_{i+k \mod n} - u_i \mod m$. Since there are only 2 distinct values, some 2 of these must be equal. 2 "adjacent" ones ($x-1$ and x,x and $x+1$) can't be equal since they differ by $y_1 - x_1$, which is nonzero, so this can only happen if $2(y_1 - x_1) \equiv 0 \pmod{m}$, which means $y_1 = x_1 + \frac{m}{2}$. It's impossible for x_1 to be 0 because that would mean that u isn't strictly increasing, and it's impossible for more than 1 adjacent difference to be equal to y_1 since $m = c_1 x_1 + (n-c_1)y_1$ and $x_1 > 0$ implies $2y_1 > m$, and this means that only k and $k-1$ can be in S.

Now that we know that S is an interval of size 2, let its elements be s and $s+1$. The sum of all the quantities is obviously kc_1, and knowing $kc_1 \equiv 1 \pmod{n}$ we can see that $s = \frac{kc_1 - 1}{n}$, and the quantity is equal to $s+1$ in only 1 interval. From here we can see that, if we start at the only $i+k \mod n$ such that the quantity of adjacent differences equal to x_1 is $s+1$ and circularly move by k each time, we'll have a difference of $sx_1 + (k-s)y_1 \mod m$ every time except when the cycle is finished, which is the exact same thing as generating the array with $a = i + k \mod n$, $b = k$, $c = u_a$ and $d = sx_1 + (k-s)y_1$.

This also gives us an easy way to construct these values if we know the entire array, so the task now is just to figure out a way to fill the gaps in the given array such that it's strictly increasing and for each $1 \leq k < n$, there are exactly 2 distinct values of $u_{i+k \mod n} - u_i \mod m$. Without loss of generality, let's assume $u_0 = 0$(in the original problem, if at most 1 value is known you can always make an arithmetic progression with step 1, and if not you can rotate the array and the values to make some known value equal to 0(and after this you'll still have another known value), but then you'll have to make sure that for some i and v that depend on how much you rotated the array and its values $u_i < v$ and $u_{i+1} \geq v$ so that when you return to the original array $u_{n-1} < m$ and $u_0 \geq 0$)

Let's look at the values of $u_i - \frac{m}{n}i$. These are all multiples of $\frac{y_1 - x_1}{n}$ because $u_i = ix_1 + z(y_1 - x_1)$ for some $0 \leq z \leq i$, so $i \cdot x_1 + z(y_1 - x_1) - \frac{m}{n} \cdot i = i \cdot (x_1 - \frac{nx_1 + (n-c_1)(y_1 - x_1)}{n}) + z(y_1 - x_1) = i \cdot \frac{(c_1 - n)(y_1 - x_1)}{n} + z(y_1 - x_1)$. It's also impossible for any 2 of these values to differ by $\geq y_1 - x_1$; if they did, let's say $(u_j - \frac{m}{n}j) - (u_i - \frac{m}{n}i) \geq y_1 - x_1$ for $j > i$(the case where $j < i$ is handled in the same way), that would mean $u_j - u_i \geq \frac{m(j-i)}{n} + y_1 - x_1$. Let's look at $k = j - i$. The sum of all n differences is clearly mk, so the average difference is $\frac{mk}{n}$. From the previous proof, it's easy to see that the 2 distinct values of differences differ by $y_1 - x_1$, so each of these 2 values must differ by less than that from the average, so $u_j - u_i$ differing by more than $y_1 - x_1$ from the average is impossible.

This means that the values of $u_i - \frac{m}{n}i$ is a range of multiples of $\frac{y_1 - x_1}{n}$ that includes 0(because $u_0 = 0$). Let $v_i = (u_i - \frac{m}{n}i)(\frac{n}{y_1 - x_1})$; the values of v_i are now a range of integers of size n that includes 0, and from the formula in the previous paragraph it's easy to see that $v_i \equiv i \cdot (n - c_1) \pmod{n}$. So, if we somehow knew $y_1 - x_1$ and c_1, since each value of $v_i \mod n$ only depends on i, there are only 2 options for v_i if $i > 0$:

$v_i \mod n$ and $(v_i \mod n) - n$. In fact, the array only depends on which suffix of values of $v_i \mod n$ we subtract n from.

Let's also prove the other direction of everything we've just seen; if we're given some values of v_i such that there exists $1 \leq s < n$ such that $v_i \equiv is \pmod n$ and their values are a range of integers and $v_0 = 0$, then if $r > 0$, $m > r(n-s)$, $m \equiv r(n-s) \pmod n$(this is true in the original direction because $m = (y_1 - x_1)(n - c_1) + nx_1$), by making $u_i = \frac{rv_i + mi}{n}$ we'll always get an increasing array where for each $1 \leq k < n$, there are exactly 2 distinct values of $u_{i+k \mod n} - u_i \mod m$.

All elements of u are integers because $rv_i + mi \equiv ris + r(n-s)i \equiv 0 \pmod n$. It being increasing is easily seen from $n(u_{i+1} - u_i) = m + r(v_{i+1} - v_i) > r((n-s) + (v_{i+1} - v_i))$, so with $v_{i+1} - v_i$ being either s or $s - n$ the difference is obviously positive. For each k, $\frac{rv_{i+k \mod n} + m(i+k \mod n) - rv_i - mi}{n} \equiv \frac{rv_{i+k \mod n} - rv_i + mk}{n} \pmod m$, but $v_{i+k \mod n} - v_i \equiv ks \pmod n$ and due to the values of v being a range of integers of size n that contains 0 the only possible values here are $ks \mod n$ and $(ks \mod n) - n$(every other value would be $\geq n$ or $\leq -n$, both of which are impossible), both of them must be achieved due to the sum of these differences being 0, and the difference they give in the final result is $r \neq 0$ so we have exactly 2 distinct values of $u_{i+k \mod n} - u_i \mod m$.

So, finding a good array is equivalent to finding good values of c_1, $y_1 - x_1$ and v_i. Clearly $y_1 - x_1$ must be divisible by the GCD of all known $u_i n - mi$, but if o values of u other than $u_0 = 0$ are known to us then we only need to check $\frac{n-1}{o}$ values of $y_1 - x_1$ because dividing $y_1 - x_1$ by some number also multiplies all v_i by the same number and we know that the difference between the maximum and minimum value of v_i is $n - 1$. Now that $y_1 - x_1$ is fixed, we can calculate v_i for all known u_i, which along with $v_i \equiv i \cdot (n - c_1) \pmod n$ gives us the value of c_1. Now we need to make sure $m > (y_1 - x_1)(n - c_1)$(we don't need to check that $m \equiv r(n-s) \pmod n$ because dividing r by a value(it has to be $\leq n - 1$ so it's impossible for division by 0 to happen) multiplies all v_i by the same value so it also multiplies $n - c_1$ by the same value modulo n, so $r(n - c_1)$ is a constant modulo n, which means it will always be equal to m modulo n if there's a solution; similarly, if there's a solution, all values of $v_i \cdot i^{-1}$ modulo n will always be equal). Now all that's left to do is to decide how many values of v_i to make negative; all known values of v_i can give us some upper and lower bounds, but we also need to keep in mind that after we rotate to the original solution we need to make sure $u_0 \geq 0$ and $u_{n-1} < m$, which gives us 2 extra bounds we need to keep in mind(or in some cases just makes the current selection of $y_1 - x_1$ impossible); if we end up with the highest lower bound being $\leq$ the lowest upper bound, we have a solution. Overall, all $\frac{n-1}{o}$ values of $y_1 - x_1$ take $O(o)$ time to test, giving a time complexity of $O(n)$.

Problem K. Kempt Kale

Notice that we can formulate this problem in terms of bracket sequences; the days on which Nežmah plants kale will be represented by open brackets, and the days on which kale is harvested will be represented by closed brackets; we want to minimize the sum of the days which correspond to closed brackets.

You can start with a sequence of all brackets being open brackets, try to make each bracket in order from lowest to highest a_i a closed bracket, and if after that there isn't a prefix with more closed than open brackets, then you keep that change; otherwise, you don't. To prove that this results in a regular bracket sequence with the highest tastiness, let's prove that, for all k, this strategy will make the highest amount of brackets out of the k ones with lowest a_i closed.

Let's assume that isn't true for some k. Let's say that there's a different solution that took more closed brackets while maintaining the fact that there isn't a prefix with more closed than open brackets. Let i be the index of the last bracket that the better solution took but our strategy didn't. The only change that taking it would make would be reducing (number of open brackets in prefix)-(number of closed brackets in prefix) for some prefixes near the end of the sequence, and since the better solution has at least as many closed brackets before the i-th position as our solution and it's the same after the i-th position, this number is at least as large in our strategy with the change of the i-th bracket into a closed one as it is in the better strategy, so it's ≥ 0. This means that, whenever we got the opportunity to change the i-th bracket into a closed one, we would've taken it, which is a contradiction with the fact that we didn't take it.

Using this strategy, we'll always make a regular bracket sequence(a bracket sequence of size $2n$ is a regular bracket sequence iff there isn't a prefix with more closed than open brackets and there are n closed brackets; we know we satisfy the former and the latter can be seen to be true using what we've just proven with $k = 2n$), and if there was a bracket sequence with lower sum of closed brackets, let's say it took the closed brackets with the $x_1, x_2, \ldots, x_n$-th lowest tastiness while we took the closed brackets with the $u_1, u_2, \ldots, u_n$-th lowest tastiness, that means that there must be i such that $x_i < u_i$, which means that, among the x_i positions with the lowest value of a, there's a strategy that took i of them while we took less than i, which contradicts what we've just proven.

To efficiently handle checking if making a bracket closed would make it so there's a prefix with more closed than open brackets, we can use a segment tree to find the minimum prefix sum where open brackets are 1 and closed brackets are -1; there is a prefix with more closed than open brackets iff the minimum prefix sum is less than 0.