Problem A. Algorithm Was Applied

Consider the final graph (i.e one which doesn't have good tuples). The number of its colorings in n colors is equal to $\prod_{i=1}^{n} n - g_i$, where g_i denotes the number of vertices $j : j > i$ adjacent to i. Proof: color vertices from last to first. i-th vertex will have exactly $n - g_i$ options, because all adjacent vertices with indices greater than i form a clique (otherwise there would be a good tuple).

Vertices a and b are connected by an edge if and only if there exists a path between them, such that all intermediate vertices in the path are less than $\min(a, b)$. Proof: All newly created edges don't break the property, therefore if there is an edge in the final graph there is a path. there. If there exists a path with the given property, there is good tuple with a equal to the minimal vertex on the path. After applying this argument repeatedly we will get a path with no intermediate vertices, which is an edge.

To calculate g_i we will iteratively maintain the following data structure. On i-th step it will contains connected components of vertices less than i (without using vertices greater than i) and for each connected component we will maintain a set of vertices greater than i which are adjacent to it. When adding i-th vertex we merge components of neighbors which are smaller than i to the component of vertex i and add neighbors greater than i to the set of adjacent greater vertices and remove vertex i from this set. The size of this set is equal to g_i. Merging smallest to largest we will obtain $O(n \log n)$ complexity with unordered set and $O(n \log^2 n)$ with ordered set.

Problem B. Balanced Rainbow Sequence

Treat the opening brace as +1 and the closing brace as -1. If the total balance of blue brackets (the common color) is b the total balance of both remaining colors is $-b$.

Assume that we have fixed b. First of all lets bring each color to the required balance. Do this by either changing some suffix of opening braces to closing or some prefix of closing to opening. It can be shown that it is always optimal to do those changes. Now the total balance of each two-color sequence is 0, but it may be less than 0 at some point.

From now on we don't change the total balance of each color, therefore we only basically swap two brackets of different types by changing them both. Yet again it can be shown that if do k swaps of brackets of a given color we swap first k closing brackets with the last k opening. Let l denote the number of swaps of lime brackets, g of green and b of blue.

Consider some point with negative balance t in some two color sequence (WLOG, assume that it is lime-blue one). If there are x closing lime brackets to the left of this point and y opening to the right first $\min(x, y)$ swaps of lime brackets increase the balance at the point by 2 each. Any further swaps do nothing for the balance of this point. The same goes for blue color. In the end we get the inequality $\min(l, x) + \min(b, y) \geq z$, where x, y and z are some constants. For example, z is equal to $\lceil \frac{-b}{2} \rceil$, which is equal to a set of inequalities $l + b \geq z, l + y \geq z, x + b \geq z, b + y \geq z$.

We end up with a set of inequalities: $l \geq c_1, g \geq c_2, b \geq c_3, l + b \geq c_4, g + b \geq c_5$, minimize $l+g+b$. The optimal solution is $l = c_1, g = c_2, b = \max(c_3, c_4 - c_1, c_5 - c_2)$.

For a fixed balance of each color everything is done in $O(n)$, there are $O(n)$ possible balances.

Complexity: $O(n^2)$.

Problem C. Concept Of Probability

Let q denote $1 - p$. Let $pc(x)$ denote the popcount of x.

Rigourous proof

Consider the same game but restricted to integer bets with integer initial amount of money x, no restriction on nondecreasing bet sizes and the threshold of winning 2^k.

Consider the following strategy for the betting: Bet 2^l money where 2^l is the largest power of two such that x is divisible by it. Note that l increases each turn. The probability of winning f such strategy $f(x)$ is equal to $\sum_{i=0}^{x-1} q^{pc(i)} p^{k-pc(i)}$. Note that this strategy does in fact do nondecreasing bets, however the claim that it is optimal even if nondecreasing bets are not required.

On the other hand, $f(x)$ is a supermartingale for any strategy. Consider a position x and an arbitrary move k. We end up at position $x-k$ with probability q and $x + k$ with probability p. The claim is $f(x - k) \cdot q + f(x + k) \cdot p \leq f(x)$. This statement is equal to $(f(x) - f(x - k))q \geq (f(x + k) - f(x))p \iff \sum_{i=x-k}^{x-1} q^{pc(i)+1} p^{k-pc(i)} \geq \sum_{i=x}^{x+k-1} q^{pc(i)} p^{k-pc(i)+1}$. We can make a bijection from $x - k$ to $x - 1$ and from x to $x + k - 1$, such that for each index a from $x - k$ to $x - 1$ and its paired index b from $x - k$ to $x - 1$ $pc(a) + 1 \geq pc(b)$ (how to do that will be described later), therefore $q^{pc(a)+1} p^{k-pc(i)} \geq q^{pc(i)} p^{k-pc(b)+1}$. After summing that over all pairs we will get the required inequality. Due to optional stopping theorem $f(x)$ is the best achievable probability of winning.

How to pair the indices: Let s be the smallest power of 2 greater or equal than k. Pair $x - k$ with $x - k + s$, $x - k + 1, x - k + s + 1$, ..., $x + k - s - 1$ with $x + k - 1$. Note that $s < 2k$, therefore we will make at least one such pair. Run the algorithm recursively with new k equal to $s - k$.

If we have $\frac{x}{2^k}$ and require the bets to be multiples of $\frac{1}{2^k}$ and set the threshold to 1 as in the original game the game is the same as the previous one (everything is simply divided by 2^k). If our number is equal to $0, b_1 b_2 \dots b_k$ in binary the formula for the answer can be rewritten as $\sum_{i=1}^{k} b_i p^{i - \sum_{j=1}^{k-1} b_j} q^{\sum_{j=1}^{k-1} b_j}$.

If x has finite binary representation that means that is is equal to $\frac{y}{2^z}$ for some x and y. We set k to y and apply the strategy mentioned above. If x has infinite binary representation $b_1 b_2 b_3 \dots$ we can do the following: let S_k be the strategy for the x' obtained from x by discarding all the bits from binary representation after the k-th. We can use this strategy for x with at least the same success as for x' because $x > x'$. This strategy has some probability of winning (sum of k terms, described above). As k approaches infinity the probability of winning of S_k will converge to $g(x) = \sum_{i=1}^{+\infty} b_i p^{i - \sum_{j=1}^{k-1} b_j} q^{\sum_{j=1}^{k-1} b_j}$ (the same formula, but k replace with infinity). It will indeed converge to $g(x)$ because the i-th of term of the series is not greater than q^i. It is well known that a geometrical progression converges. Therefore the answer for x is at least $g(x)$ as we have a family of strategies with probability of winning arbitrarily close to $g(x)$.

$g(x)$ is a supermartingale for the continuous game for the same reasons $f(x)$ is

for the discrete game. Lets proof by contradiction. Consider an arbitrary move does conform to the supermartingale property, i.e $g(x) = pg(y) + qg(z) - \epsilon$, where $y = x + m, z = y - m, m, \epsilon > 0$, we can take a big enough k such that there exist x', y', z', which are multiples of $2^{-k}, y' = x' + m', z' = x' - m', |g(x') - g(x)|, |g(y') - g(y)|, |g(z') - g(z)| < \epsilon pq$. The last inequality means that $g(x') < pg(y') + qg(z')$. Consider the discrete version (see the beginning of the proof) of the game with the given k. By definition $g(x') = f(x'2^k)$ and the same holds for y' and z'. This means that $f(x'2^k) < pf(y'2^k) + qf(z'2^k)$. But f is a supermartingale for the discrete version. Contradiction. Therefore g is a supermartingale for the continuous version and is the answer for the problem.
To calculate $g(x)$ we simply obtain x in periodic binary representation. The formulas are easilty derived from there.
Complexity: $O(b)$.

Handwaiving

The intuition goes something like that: we won't to bet as little money as possible on average, because our expected amount of money is equal to x minus $q - p$ multiplied by the total amount of money bet. For $x = \frac{1}{2}$ we obviously should bet once and win or lose instantly. For $x = \frac{1}{4}$ or $x = \frac{3}{4}$ we bet $\frac{1}{4}$ and either instantly lose or win respectively or obtain $\frac{1}{2}$ money. We know what to do from here. The same argument goes to $x = \frac{a}{8}$ and so one. We obtain the same series as in the proof section. The generalization to number with infinite binary representation is done by noticing that the terms in the series start to repeat themselves after P steps with some multiplicative factor, where P is the period of binary representation.

Problem D. Doesn't Contain Loops or Multiple Edges

There exists a monotonic recoloring if and only if there is a vertex which can be recolored without changing colors of the remaining vertices. Proof: if the final colors are not greater/not less than the initial colors a recolored vertex which received the smallest/the greatest color respectively is such vertex (can be recolored, without touching others).
Checking if such vertex exists is trivial. Anything sane works in $O(m + n + k)$ or $O(mlogm + n + k)$.

Problem E. Equal Adjacent Elements

Let $f_{l,r}$ denote the number of ways to remove all elements between l-th and r-th (but without removing l-th and r-th elements themselves), such that the subarray is always good. If $a_l = a_r$ $f_{l,r}$ is zero, otherwise if $l+1 = r$, $f_{l,r} = 1$ and if not $f_{l,r} = \sum_{k=l+1}^{r-1} f_{l,k} f_{k,r} C_{r-l-2}^{k-l-1}$. k represents the index of the last removed element, the combinations part represents the number of way to interleave the removals of elements to the left and to the right of k-th.
To get the answer for the problem add a barrier element on each side (they will have indices 0 and $n+1$). $f_{0,n+1}$ will be the answer.
Complexity: $O(n^3)$.

Problem F. Formally, You Choose Three Integers

Consider the case where a is an identity permutation. a can be transformed into b if and only if the following properties hold:

1. b is an permutation.
2. All elements on odd positions of b are odd and all elements on even positions are even.
3. b has an even number of inversions.

Proof: It can be shown that the operation doesn't change any of those properties. We can build b from a by moving the elements to their required positions one by one (from the first element of b to last) until there are 3 elements left. If the second rule holds there are only 2 possible permutations of those 3 elements left. Exactly one of them is even. We will always get the even one.
Now for the general case (a is not a permutation). The multiset of elements on the even/odd positions of a obviously must be equal to the mutliset of elements on even/odd positions of b. If each of those sets contains distinct elements the case is equal to the permutation case. If there are two equal elements in some set we can always successfully transform, because those two elements can be implicitly swapped without changing the sequence, but changing the parity of the permutation.
Complexity: $O(n)$.

Problem G. Game

Sort the elements in both arrays (however the starting position will be no longer $(1, 1)$). Binary search the answer. Obviously if first player can guarantee than the answer is not greater than the sum of the initial chosen positions. Otherwise, Represent all possible states as a $n \times m$ grid.

The game can be rephrased as follows: There are two types of states: white, which means an acceptable result for the first player and black, which means an acceptable result for the second player. Players take turns moving from the token from a bad position (for example, black position for a first player) to an adjacent good one(there is an edge between two positions if they are in the same row or column). Note that we don't draw edges between vertices of the same colour. The player must move to a good position because otherwise the other player can finish the game on his move. It is not allowed to repeat the position for the 10^{228} time. It can be shown that if we replace 10^{228} with 2 nothing changes. This problem is known, and the answer is: the first player wins if and only if every maximum matching contains the starting vertex.

Now we need to find the maximum matching in our graph and in our graph with the starting vertex removed fast. Our graph is bipartite, therefore the size of the matching is $nm - X$, where X denotes the size of the maximum independent set. Consider this set (or one of them if there are many): let r_w and c_w denote the set of rows in which there is at least one white element in the set and r_b and c_b denote the same thing for black elements. It can be shown that r_w and r_b are disjoint, the same thing holds c_w and c_b. The set consists of all white elements in $r_w \times c_w$ and all black elements in $r_b \times c_b$. Because the elements are sorted the white cells form a Young diagram (i.e the lie below some line from $(n, 0)$ to $(0, m)$ and the black cells lie above that line. It can be shown that is always optimal (not worse than any other way) to make r_w be some prefix of rows and r_b be the remaining ones (i.e we fix a threshold r and all rows $\leq r$ are in r_w and ones $> r$ are in r_b. The same thing holds for c_w and c_b with some threshold c. If we fix one of those thresholds (say r), we can easily find the optimal value of c in constant or logarithmic time (c is equal to the number of columns with at least $n - r$ white cells).

When need to find the size of maximum independent set in a graph with the starting vertex removed we do basically the same thing but bruteforce whether the row of that vertex is in r_w or in r_b and is column in c_w or c_b (4 options in total).

Each iteration of binary search runs in $O(n)$ or $O(n \log n)$. there are $\log(A)$ iterations, where A is 10^8 (the maximal value of a).

Total complexity: $O(n \log n \log A)$ or $O(n \log A)$.

Problem H. Hat With An Integer

Let the distance between two arrays of equal size be the number of positions where elements are different.

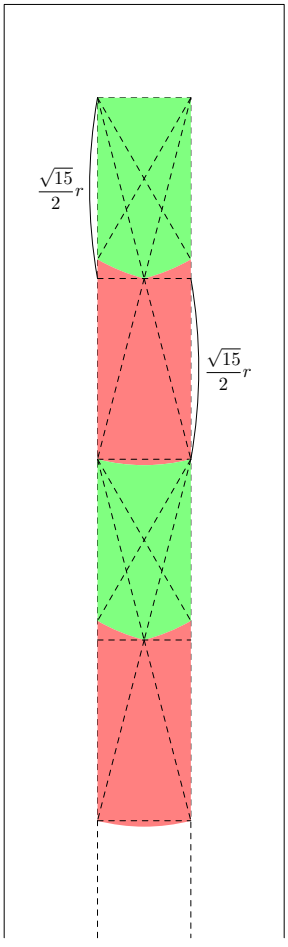
Consider the set of arrays S, such that everyone knows the actual array a is not in S. Initially it is equal to the set of all arrays which are not valid. Let's call this set V. It can be shown that each day if the actual array is at distance 1 from one of the arrays in S at least one of the wise ones announces. (For example, if different i-th element differs (it is x in a, and y in some element s in S and all the remaining elements in a and s are equal) i-th wise one can deduce that the number of his hat is not y. If no one announced each array with distance 1 from some element of S is added to S. Therefore on k-th day someone announces if a is at distance of exactly k from the closest element of V. Now we need to find that distance.

Let B_i be $\sum_{j=1}^{i} b_i$ and C_i be $\sum_{j=1}^{i-1} c_j$. We need to find an element of V with the most possible number of positions where two arrays are the same. Lets fix those position. Consider two adjacent fixed positions i and j (in the sense that there are no fixed positions in between), it can be shown the following is required is sufficient if they hold for all such pairs: $a_i + (B_j - B_i) \leq a_j \iff a_i - B_i \leq a_j - B_j$ and $a_i - C_i \geq a_j - C_j$. This can leads to a dp solution. Let f_i be the maximum number of positions ending at i, such that for each adjacent pair the properties above hold. f_i can be calculated from previous values using 2D queries. However it can be shown that if some subset of positions have nondecreasing values of $a_i - B_i$ and nonincreasing values of $a_i - C_i$ they are automatically ordered by i. Therefore we don't need one dimension of our dp and can reduce the complexity to $O(n \log n)$ with a simply query to a fenwick tree/interval tree. Time limit was big enough for both solutions to pass.

Complexity: $O(n \log n)$ or $O(n \log^2 n)$.

Problem I. Intersect With Other Balls

We can modify the game slightly. It is played it on a rectangle of size $(h-2r)\times r$, players put points instead of circles and each point must be at distance $2r$ from the previous one. Points in the new game correspond to centers of circles in the original one. Therefore the positions is wholly determined by the last move (i.e a point). If there was a finite number of points we would need to do simple win/loss analysis (Process positions from bottom to top, the position is a win if there exists at least one possible move that leads to a losing position and a loss otherwise). But in this game the number of positions is infinite. However if we split the rectangle into connected figure, such that positions inside in each figure are either all wins or all losses), the number of such figures is going to be finite and they will follow a rather simple pattern.



Problem J. J The Attacker Has

The set of ranks is good if the defender can beat all the attacker cards of those ranks using his cards of those ranks. The defender wins if there exists a good set which contains the rank of the starting attack card. Proof: By playing accordingly the defender guarantees the win. In any situation where the attacker loses (i.e can not make a turn) the set of ranks on the board is a good one.
Lets fix the set of ranks. We need to check if it is a good one. The defender strategy is a greedy one. For each trump card of the attacker the defender should beat it with the lowest possible card (i.e of the same suit and greater in rank than the attackers card). In the end either all attackers trumps are beaten and the defender has some excess trumps or they are not and the set is definitely not good. In the former case the defender should use the same strategy for the non trumps, except the fact that if there is a attacker's cards he can't beat he should use his excess trumps.
Checking if the set is good is easily done in $O(mn)$. We iterate over all possible sets of ranks. There are 2^n sets.
Complexity: $O(2^n nm)$.

Problem K. K Space Separated Integers

Do the greedy from right to left. The element should be added if it makes the string lexicographically less or if it is included in the given index sequence.
Complexity: $O(n)$.

Problem L. Labeled Connected Graphs

Calculate the sum of distances from all vertices to the first instead of the distance from second to first. In the end we will need to divide the answer by $n-1$.
Do a BFS state DP. Let $f_{i,j}$ denote the sum of distances to the closest of the first j vertices over all graphs on i vertices, where all vertices are reachable from at least one of the first j and $g_{i,j}$ denote the number of such graphs. Let k denote the number of vertices adjacent to at least one of the first j. The distance from those vertices to the first j is equal to 1 and the distance from any other vertex is equal to the distance to the closest of those k vertices + 1. We can iterate over all k and calculate $f_{i,j}$ and $g_{i,j}$. The exact formulas: $g_{i,j} = 2^{\frac{j(j-1)}{2}} \sum_{k=1}^{i-j} C_{i-j}^{k} (2^j-1)^k g_{i-j,k}$, $f_{i,j} = C_{i-j}^{k} 2^{\frac{j(j-1)}{2}} \sum_{k=1}^{i-j} (2^j-1)^k f_{i-j,k} + g_{i,j}(i-j)$, where $2^{\frac{j(j-1)}{2}}$ represents the number of ways to choose edges between the first j vertices, C_{i-j}^{k} represents the number of ways to choose which k vertices are adjacent to the first j and $(2^j-1)^k$ represents the number of ways to choose edges between the first j and the adjacent k.

Problem M. Moves You Need To Make

Let $(<x, <y)$ denote the set of elements which are on the position with index less than x and their value is less than y.
If we cannot swap the first and the last elements the answer would be simply the number of the inversions. Consider this case as a possible answer.
Otherwise, there will be two elements a and b, which will be swapped. WLOG, assume that a is to the left of b. Therefore $a > b$, otherwise the swap will be useless. This can be treated as follows: elements a and b start swapped but you have to have a at the last position at some point and b at the first. This has the following implications on the number of swaps.

1. Elements which are less than b and to the right of a will have to be swapped twice than it. +2 swaps for each such element.

2. The same thing applies to the elements which are greater than a and to the right of b.

3. The elements which are between a and b by both value and position will have to be swapped with neither a nor b instead of both. -2 for each such element.

The formula for the change in the number of inversions is $2(n-(<a, <pos(a))-(<b, <pos(b))+b+pos(a))$, where $pos(x)$ denotes the position of x-th element. which is equal to $f(a)+g(b)$, where f and g are easily computable in $O(\log n)$ time using fenwick or interval tree. Knowing all f and g getting the answer is trivial.
Complexity: $O(n \log n)$.

Problem N. Number Of Vertices

Consider another graph. Make a duplicate of each vertex v. Denote it as v'. Edge (a, b) in the original, such that $a < b$ aorresponds to an edge (a, b') in the new graph. It is easy to see than any cycle in the new graph is a zigzag cycle in the original graph and vice versa. A graph can be split into a set of cycles if all degrees of its vertices are even. This can be easily maintained in constant time per query.
Complexity: $O(n + q)$.