# Problem A

For every multiset with at most 2 distinct elements median and mode are the same. Furthermore if the median is either the smallest or the largest element it is the mode too. Otherwise there are two cases:

- Mode is less than the median. Let $a$ denote the smallest element of the given set, $c$ - largest, $b$ - second largest ($b < c$). Obviously the mode is not less than $a$. There are at least 3 distinct elements therefore the median is not the largest and it is not equal to the largest element of the whole set. Thus it is at most $b$. The difference is at most $b - a$. Achievable with the set $\{a, b, c\}$.
- Mode is greater than the median. Let $a$ denote the smallest element, $b$ - second smallest (greater than $a$), $c$ - greatest element which occurs at least twice. Mode can't be greater than $c$ (otherwise the largest number of occurrences is 1 and mode is the smallest element). Median can't be less than $b$ (there are at least 3 distinct elements). The difference is at most $c - b$. Achievable with the set $\{a, b, c, c\}$.

Complexity: $O(n \log n)$.

# Problem B

Let $f_i = c_i - c_{i+1}$. Basically you can freely choose $f_i$. Let $g_{k,p}$ denote the $\sum_{i \in S_p, i<k} a_i$. You have $n$ 4D points $(g_{k,0}, g_{k,1}, g_{k,2}, g_{k,3})$, choose their convex (linear, but WLOG we can divide by the sum of chosen values) combination with maximal $\frac{min_{x_1,x_2,x_3}}{x_0}$, where $x_i$ denotes $i$-th coordinate of the sum.
Binary search the answer, assume are checking whether we can achieve the ratio of at least $R$. Replace $x_i$ with $x_i - x_0 \cdot R$ and remove the 0-th coordinate. Now we are given a set of 3D points and have to check whether their exists their convex combination with all coordinates greater or equal to 0.

## First way

The latter is basically $\frac{1}{8}$ of the plane and is a convex figure. Two convex figures don't intersect if there exists a plane between them. It can be shown that if there exists such plane there exists another one which passes through the origin. I.e all points lie below the plane $ax + by + cz = 0$. We get a system of linear equations on $a$, $b$, $c$. Divide everything by one of those variables you will get the same problem on a plane with a set of half-planes as restrictions. Check the emptiness of the half-plane intersection.

## Second way

Divide everything by the absolute value of the first coordinate.
Add a point of form $(-1, 0, 0)$. Now we choose a convex combination of points

with the first coordinate = 1 (represent them on a plane, ignoring the first coordinate) and a convex combination of points with first coordinate = -1. The former should be above by both coordinates. We can take the upper convex hull of the former and lower convex hull of the latter and combine them as half-planes. Check the emptiness of the half-plane intersection afterwards.

Complexity: $O(n \log^2 n)$ or $O(n \log n)$ (if you use a randomized algorithm to check the emptiness of HPI).

# Problem C

Let $p_i$ denote the number of inversions in the $i$-th (by $i$ positions) cyclic shift. Let $d_i$ denote the $p_{i+1} - p_i$. Basically you are asked to find the smallest prefix sum of $d_i$ (or rather its index). It can be shown that $d_i = n - 1 - 2 * a_i$. Now instead of inversion stuff you have to do a cyclic shift of the array and find its minimal prefix sum. Can be done using an implicit treap.
Complexity: $O(n + q) \log n)$.

# Problem D

If the graph is acyclic the answer is 0 (we can order the people, at process them one by one (starting from the one who sees nothing), get his named color and put the hat of the other color on him).
Otherwise, let $v_0, v_1, \ldots, v_{k-1}$ denote the indices of people in some cycle ($v_i$ sees $v_{(i+1) \bmod k}$). Let $v_i$ name the color different than the color of the $v_{i+1}$s hat for $i < k - 1$. Let $v_{k-1}$ name the same color as the color of the $v_0$-s hat. It can be shown that at least one of them will survive.
Complexity: $O(n + m)$.

# Problem E

$K$ is $\lfloor n \rfloor$.
At any point the guesser knows the remainder of $x$ modulo some number $p$. After he asks about $y$ he knows the remainder modulo $lcm(y, p)$, i.e set $p$ to $lcm(y, p)$. Thus after each query $p$ increases by at least 2.

## Solution

Choose $x = \lfloor \frac{n}{2} \rfloor$. Never adapt. After $K$ queries $x$ will be known ($p \geq 2^K$).

## Interactors strategy

Ask queries of form $2^i$.

Complexity: $O(\log n)$.

# Problem F

Consider two indices $i, j : i < j$. In which subsequences will they act as a pair which adds 1 to its palindromicity? It is obviously none if the letters are different. Otherwise its those where the number of included indices in the first $i$ is equal to the number of included indices after $j$, $i$ and $j$ are included and indices in between don't matter. Thee are $\frac{(i+(n-j-1))!}{i!(n-j-1)!}2^{j-i-1}$ of those. Setting $r = n - j - 1$ this becomes $\frac{(i+r)!}{i!r!}2^{n-2-(i+r)} = f(i) \cdot g(r) \cdot h(i+r)$.
Iterate over all possible letters. For any given letter the sum can be obtained with a convolution.
Complexity: $O(Ln \log n)$, where $L$ is the alphabet size.

# Problem G

## Static case

Let $p_i$ denote the element which is to the right from $i$-th which is equal to $a_{i+1}$-th (in other words, $p_i$ the one we would take next when taking the required sequence greedily after the $i$-th element). To answer the query we have to find the leftmost element in the subinterval equal to $k$ (done using a set) and going from $i$ to $p_i$ until we reach an element outside the subinterval. The value of the last element inside the subinterval is the answer. We can view $p_i$ as parent of $i$ in the tree. Queries can be answered efficiently using binary lifting. Or HLD (worse, but closer to the full solution).

## A Wrong Solution for the Dynamic Case

Simply use link-cut. The problem is that there may be many changes in the tree structure (e.g, change 1 to 2 in $[0, 0, \ldots, 0, 1]$. Every 0 has the last element as its parent. All those edges are cut (or created if we reverse the change). To answer the query we search the BST, representing a path, used for the link-cut structure.

## Fix

Let $p_i$ denote the closest element to the right which is equal to $a_i$ or $a_i+1$. It can be shown that nothing changes in terms of queries. But now each modification of the array leads to a constant amount of modifications to the tree structure.

## The Dark Side

You have to use Splay as the underlying BST or you will get TLE (see below for the reasons behind such a harsh TL).

**SQRT Decomposition instead of Link-Cut**

If you got this far you will easily come up with the way to do it.
If you don't use the Fix you will get either an $O(n\sqrt{n \log n})$ solution with $O(n)$ memory or $O(n\sqrt{n})$ solution with $O(n\sqrt{n})$. Neither of those should pass.
If you use the Fix you can easily get an $n\sqrt{n}$ solution with $O(n)$ memory. It might or might not pass depending on the constant factor. We couldn't reliably make it either always pass or always fail while making sure that the link-cut comfortably passes and the $O(n\sqrt{n \log n})$ fails.
Complexity: $O((n+q)\log n)$ or $O((n+q)\sqrt{n})$

# Problem H

Due to Matroid Theory (or consider how Kruskal's algorithm would behave if we run it instead) we only need to use edges from the spanning tree of some row or column. The spanning tree of some row or column is a set of edges between adjacent elements in sorted order. There is only $O(nm)$ edges of this kind. Run some spanning tree algorithm leaving only those edges.
Complexity: $O(nm \log nm)$

# Problem I

We will describe point in some sequence is described by two numbers: $(x, y)$, where $x$ is the best (greatest) balance before that point and $y$ is the worst (smallest) balance before that point. We will ignore the points inside some RLE block.
Two sequences are unmergeable if one of the following holds:

- There exists a point $x_1, y_1$ in the first sequence and $x_2, y_2$ in the second sequence, such that $x_1 + y_2 < 0, x_2 + y_1 < 0$.
- The condition above holds for the reversed and inverted sequences (note that reversed and inverted balanced sequences is balanced too).
- If the best balance in of the sequences is less by absolute value than the worst balance in the other sequence.

The first two conditions can be checked by passing over all points in the query sequences and doing a binary search over all presorted points in the main sequences. The third condition is trivial to check.
Complexity: $O((n+L)\log n)$, where $L$ is the sum of lengths of the query sequences.

# Problem J

The answer is equal to the halved sum of distances between the vertex where number $v$ is in and the vertex with index $v$. This can be calculated using lifting

sets or LCA.
Complexity: $O(n \log n)$.

### Proof

Let $L$ denote the assumed answer. Lets prove by induction on $L$. For $L = 0$ it is obvious. Assume that we proved the statement for $0, 1, \ldots, L-1$.
Root the tree by some non-leave vertex (if there are none there are at most 2 vertices and the proof is trivial). We claim that it is always possible to make a move which decreases the assumed answer or a move that doesn't change the assumed answer, but misplaces some number (the swap that costs 0). This is finite, because after each move either the answer decreases or the amount of nubmers in its place decreases, neither or those can be negative. Lets prove the claim by contradiction. Lets call a vertex an **up** vertex if direction from the token in it to its destination is up, **down** if the direction is down and **inplace** if the token in it is in its place. Leaves can not be down (there is nowhere to go down). While some leave is inplace remove it. Now all leaves are up. Consider a vertex such that all its children are up. If this vertex is down we can exchange the number in it with the number of the child it needs to go to. This will cost 1 but decrease the assumed answer by1. If this vertex is inplace exchange it with its arbitrary child. This will cost 0 and will decrease the number of inplace vertices. Otherwise this vertex up. Going from the bottom of the tree to the top we can deduce that all vertices are up. But the root can't be up. Contradiction.

## Problem K

Bruteforce the payouts. Let $(C, D)$ denote the payout we are currently bruteforcing. Each subtree can be described by three boolean variables:

- Whether the token arriving at the root of the subtree will a vertex with the payout $(C, D)$.
- Whether Carol can change her strategy to arrive to a vertex with her payout greater than C.
- Whether David can change his strategy to arrive to a vertex with his payout greater than D.

Basically there are 8 possible states of the subtree. Can be calculated with dynamic programming.
Complexity: $O(nk^2)$ with a constant factor of about 128.

## Problem L

Represent the set of matches as a graph.
The following two conditions are required:

- The graph is a tree
- There is no vertex with degree greater than $\frac{n+1}{2}$.

Those conditions are sufficient, but the proof is a bit tedious (it goes something like: centroid of the tree will be the winner, the worst case is when all of the subtrees are bamboos. Enumerate them. You have to construct a sequence such that 0 occurs $a_0$ times, 1 occurs $a_1$ timers, $\ldots, k-1$ occurs $a_{k-1}$ times and so on, such that no two equal numbers are adjacent and no two last numbers of the same value are adjacent, except the last two. There is a construction which works (provided that $k \leq \frac{n+1}{2}$).
To calculate the number of such graphs you take the total number of trees (Cayles formula) and subtract the number of trees with a vertex with degree $> \frac{n+1}{2}$. The latter can be done using the fact that a vertex with degree $d$ occurs exactly $d-1$ times in the Prufer encoding of the tree (i.e. you have to count the number of sequences of numbers from 0 to $n-1$ of length $n-2$, such that no number occurs more than $\frac{n-1}{2}$ times, the latter is trivialy doable in $O(n)$ with basic combinatorics).
Complexity: $O(n)$.

# Problem M

Lets solve the problem for the inverse permutation (you are given sets of number which must be consecutive in the permutation). Split all the elements into connected components (no elements in two different components are in the same set).
Consider the largest set $S$. Lets split the permutation into 3 parts: before the set, inside it and after the set. We want to split the problem into smaller problems. We know the set of elements inside (it is $S$). Consider all the remaining sets. All subsets of our set are simply put into the inside problem and ignored for the purposes of splitting. If there are no sets which intersect $S$, but lie inside we are done, simply solve the inside problem recursively. Otherwise, let $T$ denote the set with the largest intersection with $S$, denote their intersection as $I$. We can assume that WLOG $T$ consists of some elements before the inside elements and some elements (i.e $T$ lies to the left of its intersection with $S$). The other option (it lies to the right is symmetrical). Consider some other set $G$ which intersects with $S$. It can be shown that if the intersection of $G$ is a subset of $I$, then $G$ lies to the left of $S$, otherwise $G$ lies to the right. The remaining sets (the ones which don't intersect of $S$) can be distributed to the left or to the right of $S$ using a DFS.
Now we've split the problem into 3 smaller problems but with a twist. There are some sets which have to be prefixes or suffixes. First of all we replace the requirement of $T$ being a suffix with the requirement of the complement of $T$ being a prefix. Now, knowing some prefixes we can split the permutation into some groups (the smallest prefix, the difference between the second smallest prefix and the smallest, ..., the elements which are not included in any prefix).

We will split the problem into the same problem for each group. Each set which is required to be interval is either inside the group (simply put in the groups problem) or contains a suffix of some group, a prefix of some other group and all the groups in between. Which groups are fully contained is easily deducible and require the element in the group to the left to be a suffix and the group to the right to be a prefix is a requirement we known how to deal with. Complexity: $O(n)$.