

Problem A. Data Restoration

To solve the problem, consider the differences between neighboring elements — $a_i - a_{i-1}$.

All negative differences correspond to a decrease, so all elements that form these differences must belong to the segment on which the operation is performed (otherwise, after an element outside the segment there will be a smaller element). Also, the segment must not contain elements that increase — otherwise, after reversing, they will become decreasing.

Therefore, if the array is not already sorted, one should perform one operation from the minimum to the maximum indices of the elements that form negative differences. If there are equal elements on the borders, they must be included in the segment by extending it (otherwise, after reversing the subarray, a smaller element will appear after an equal one). If after that the array remains nonincreasing, then the answer is -1 .

The problem can also be solved in another way: obtain an array b that corresponds to the sorted version of a , and then find all indices i where $a_i \neq b_i$. Then it is enough to take the minimum segment covering all such indices and reverse it — after that, it remains to check that the array became sorted.

Problem B. Necklace

To solve the problem, it is necessary to find the maximum number of beads of the same color that are already consecutive and can become consecutive after removing some color x (the case when there is only one color is considered separately).

For this, compress the array into pairs $(color_i, cnt_i)$ — by color and by count. Then, when removing some color x , it is enough to consider only the neighbors of all pairs for which $color_i = x$. Next, among them, we need to find the maximum new segment of one color. This can be done, for example, by processing them one by one and trying from each of them to build as long a segment of the same pairs as possible, skipping over pairs with $color_i = x$ (and for pairs that have already been processed, there is no need to start from them again).

Since each element has at most 2 neighbors, for color x this will work in $\mathcal{O}(cnt_x)$, where cnt_x is the number of blocks of beads of color x . Since the sum of all cnt_x is at most n , the final solution works in $\mathcal{O}(n)$.

There is also an alternative solution to the problem. Since the optimal segment of color a (for example) will alternate with another color b , it is enough to iterate over the array and maintain with two pointers the current block of the form $abab...ba$, where each letter corresponds to a compressed state with some number of beads. Then it is only necessary to find the maximum length of one of the alternating colors.

Problem C. Rating

We will solve the maximization problem — the minimization problem can be solved analogously after the transformation $x = n + 1 - x$.

We will call readers of type D dynamic, and readers of type S static.

Notice that if the current sum is S and k votes have already been made, then a dynamic reader gives the rating $\min(n, \max(1, x \cdot (k + 1) - S))$.

Let $\text{clamp}(x, l, r) = \min(r, \max(l, x))$. Then after a dynamic reader's rating, the final rating becomes $S + \text{clamp}(x \cdot (k + 1) - S, 1, n) = \text{clamp}(x \cdot (k + 1), 1 + S, n + S)$.

We will show that in an optimal rating procedure, readers vote in order of increasing ratings. For this, it is enough to show that in each of the four possible arrangements of adjacent readers (S-S, S-D, D-S, D-D), it is no worse to place first the one with smaller x . To prove this, we will show that the sum of their ratings is no smaller in this order — since for all transitions the rating is a nondecreasing function of S , it is optimal to rearrange readers so as to increase this rating.

1) In the case of two static readers, it does not matter in which order they give their ratings, so they can be reordered arbitrarily.

2) In the case of two different readers with ratings s for the static one and d for the dynamic one,

for the order S-D we get $\text{clamp}((k+2) \cdot d, S+s+1, S+s+n)$, and for the order D-S we get $\text{clamp}((k+1) \cdot d+s, S+s+1, S+s+n)$.

The bounds are the same, but the value itself is larger when the larger number s or d is added last.

3) For the case of two dynamic readers with values $d_1 \leq d_2$, we get for the order D1-D2 $\text{clamp}((k+2) \cdot d_2, 1+r_1, n+r_1)$, and for the order D2-D1: $\text{clamp}((k+2) \cdot d_1, 1+r_2, n+r_2)$, where $r_1 = \text{clamp}((k+1) \cdot d_1, S+1, S+n)$, $r_2 = \text{clamp}((k+1) \cdot d_2, S+1, S+n)$.

Obviously, $r_1 \leq r_2$.

If $r_1 = r_2$, then the bounds in both expressions are the same, and the first argument in the order D1-D2 is no smaller. Therefore, the order D1-D2 is no worse.

Now let $r_1 < r_2$. We will show that the final sum in the order D1-D2 is no smaller than both $r_2 + 1$ and $(k+2) \cdot d_1$. Then $r_1 + n \geq r_2 + 1$, since both values r_1 and r_2 lie on the segment $[S+1; S+n]$. In addition, r_2 cannot be equal to the lower bound $S+1$, because $r_1 < r_2$. Therefore, $(k+2) \cdot d_2 \geq r_2 + 1$. Hence, after D1-D2 the final sum is no smaller than $r_2 + 1$.

Also, $(k+2) \cdot d_1 \leq r_1 + n$: if $r_1 = (k+1) \cdot d_1$, this follows from $d_1 \leq n$; if $r_1 = S+1$, then $(k+1) \cdot d_1 \leq S+1$, from which $(k+2) \cdot d_1 \leq S+1+n = r_1 + n$.

Therefore, the final sum after D1-D2 is no smaller than $(k+2) \cdot d_1$. On the other hand, the final sum after D2-D1 does not exceed $\max(r_2 + 1, (k+2) \cdot d_1)$. Consequently, the order D1-D2 is always no worse than the order D2-D1.

Thus, to solve the problem, it is enough to sort all values x in increasing order and apply either by adding the rating x for a static reader, or by using the formula for a dynamic reader.

Problem D. Collectible Cards

Let us solve the problem using dynamic programming. Let $dp[i]$ be the number of ways to split the prefix ending at index i inclusive into subsegments satisfying the condition of the problem. Then the answer is $dp[n]$.

To transition to $dp[i]$, we can come from all states j ($j < i$) such that the MEX of the subsegment from $j+1$ to i lies in the required range. As j decreases, the MEX can only increase, so the valid j for the transition form a subsegment themselves (possibly empty). Then, knowing its boundaries, we can compute the next value of $dp[i]$ using prefix sums built on the dynamic programming values.

To find the boundaries of the transition segments, it is enough to maintain them with pointers: one pointer for the maximum segment where MEX is at most R , and another for the minimum segment where MEX is at least L .

To maintain the pointers, for the left pointer it is enough to track that not all numbers from 0 to R appear at least once, and for the right pointer that it cannot be moved further without removing all occurrences of one of the numbers from 0 to $L-1$. The solution works in $\mathcal{O}(n)$.

Problem E. Preparing for the Olympiad

The expected total experience is the sum of the experience for the current problem (this is always exactly n units) and the experience for the problems solved before it — let us compute this second part.

By linearity of expectation, we can separately compute how much experience each pair contributes.

Consider a pair a_i, a_j ($a_i \leq a_j$), appearing in the order i , then j ($i \neq j$), and compute its contribution to the expectation. Let $q_i = 1 - p_i$.

There are exactly $n(n-1)/2$ possible relative positions for them (and this order occurs with probability $1/2$), and in $n-1$ of them they are adjacent — then the contribution to the expectation is $+1$; in $n-2$ cases there is 1 moment between them when the problem may be forgotten — that is, the contribution is $+q_i$. If there are k numbers between them, then there are $n-k-1$ such situations, and the contribution is $+q_i^k$. Therefore, the total contribution is $\frac{1}{n(n-1)} \sum_{k=0}^{n-2} (n-k-1) q_i^k$.

This sum can be computed efficiently using the formulas: $\sum_{k=0}^m q^k = \frac{1-q^{m+1}}{1-q}$ and $\sum_{k=0}^m kq^k = \frac{q-(m+1)q^{m+1}+mq^{m+2}}{(1-q)^2}$.

However, instead of using formulas, one could also use the fact that the p_i take at most $m = 101$ different values, and compute the sum separately for each of them in linear time.

At the same time, for each element a_i , it contributes the same amount with any element not smaller than it if it comes before it — this means that it is enough to count, for each element a_i , the number of elements $a_j \geq a_i$ — and add that contribution that many times.

For this, it is enough to sort the elements and process them carefully in order. The final time complexity of the solution is $\mathcal{O}(n \log n)$ or $\mathcal{O}(n \log n + nm)$ depending on the implementation.

Problem F2. Expedition (hard version)

To solve the problem, consider the following tree DP. Let $dp_0[i]$ be the minimum amount of energy needed to start from vertex i , traverse the entire subtree, and return to vertex i , and let $dp_1[i]$ be the minimum amount of energy needed to start at vertex i , traverse its entire subtree, and stop somewhere. Then the answer to the problem is $dp_1[1]$. (For problem F1, the answer is $dp_0[1]$, and it is enough to compute only that.)

First, let us see how to compute $dp_0[i]$. Also, for each vertex we will maintain $delta[i]$ — the change in energy if we completely traverse this subtree.

Now consider all children u_j of vertex i and their $dp_0[u_j]$ and $delta[u_j]$. Recompute $dp_0[u_j]$ and $delta[u_j]$ taking into account the energy needed on the edges, and obtain an array of pairs $(dp_0[j], delta[j])$ for the DP transition.

If we fix the order in which vertices $j_1, \dots, j_k$ are visited, then the resulting minimum energy will be $\max(dp_0[j_1], dp_0[j_2] - delta[j_1], dp_0[j_3] - delta[j_1] - delta[j_2], \dots, dp_0[j_k] - delta[j_1] - \dots - delta[j_{k-1}])$, since after visiting the next subtree j_r , the current energy increases by the corresponding $delta[j_r]$.

Let us understand which order is optimal. If we start traversing and there are still subtrees with $delta[j] \geq 0$, then it is beneficial to traverse them first, since the current energy does not decrease. Among them, it is beneficial to traverse in increasing order of $dp_0[j]$, because at any moment either there are still subtrees with positive energy change, or there are none — but if there are, then it is never worse to start with the one with the smallest $dp_0[j]$, since if the current energy is not enough to traverse it, then the initial energy was not enough either.

Now it remains to consider the order of those subtrees where $delta[j] < 0$.

For them, it is beneficial to consider all subtrees in decreasing order of $dp_0[j] + delta[j]$. To prove this, let us show that for two consecutive subtrees in the traversal, it is optimal to first enter the one with the larger $dp_0[j] + delta[j]$ — then from the formula for the minimum energy it follows that this is the optimal order for any number of subtrees with negative $delta$.

Suppose we have $d_1, delta_1$ and $d_2, delta_2$, with $d_1 + delta_1 \geq d_2 + delta_2$. Then for order 1, 2 we get energy $\max(d_1, d_2 - delta_1)$, and for order 2, 1 we get $\max(d_2, d_1 - delta_2)$.

At the same time, $d_1 - delta_2 > d_1$ (since $delta_2 < 0$) and $d_1 - delta_2 \geq d_2 - delta_1$ from $d_1 + delta_1 \geq d_2 + delta_2$. This means that the second option (2-1) is no better than the first.

Thus, we can obtain the optimal order $j_1, \dots, j_k$ and compute $dp_0[i]$.

To compute $dp_1[i]$, we need to choose one subtree where the route will end. This vertex can be tried one by one, and for the others we keep the same order and compute their prefix and suffix contributions without one subtree (taking into account that it goes last) — it remains only to choose the minimum and correctly account for a_i of the current vertex — if the currently computed energy is E , then in the end we need at least $\max(0, E - a_i)$ energy.

The overall complexity of the solution is $\mathcal{O}(n \log n)$.