## Problem A. Testing Your Geometry Template

Firstly, let's look at two points with coordinates $(x_1, y_1)$ and $(x_2, y_2)$ with $x_1 \neq x_2$. Where does perpendicular bisector to the segment between them intersect the $x$-axis?

Let's say it's at point $(X, 0)$, then we must have $(x_1 - X)^2 + y_1^2 = (x_2 - X)^2 + y_2^2$, implying $(x_1^2 + y_1^2) - (x_2^2 + y_2^2) = 2X(x_1 - x_2)$, so $X = \frac{(x_1^2+y_1^2)-(x_2^2+y_2^2)}{2(x_1-x_2)}$

Now, we need to find the largest and smallest such values over all pairs $(x_1, y_1), (x_2, y_2)$ of points from the statement with $x_1 \neq x_2$.

How to check if the largest value doesn't exceed $t$?

Note that it's equivalent to the following: $(x_1^2 + y_1^2) - (x_2^2 + y_2^2) \leq 2t(x_1 - x_2)$ whenever $x_1 > x_2$, which is equivalent to $(x_1^2 + y_1^2) - 2tx_1 \leq (x_2^2 + y_2^2) - 2tx_2$, when $x_1 < x_2$.

Note that it's enough to check this inequality only for pairs $(x_1, y_1), (x_2, y_2)$ such that there is no point $(x_3, y_3)$ with $x_1 > x_3 > x_2$. Indeed, if the inequality holds for any two points with "adjacent" $x$-coordinates, it would hold for any two points. So, let's check it only for them.

So we could say just "sort points by $x$ coordinate, find this $X$ for every pair of adjacent points with different $x$ coordinates, and take the largest of them". But everything is not so easy. What if we have many points with $x$ coordinate $x_1$ and many points with $x$ coordinate $x_2$? We would need to find this $X$ for every pair of them to check properly. But... would we?

Note that to maximize $\frac{(x_1^2+y_1^2)-(x_2^2+y_2^2)}{2(x_1-x_2)}$ under fixed $x_1, x_2$ with $x_1 > x_2$, we just have to maximize $|y_1|$ and minimize $|y_2|$. So, the algorithm for finding the largest value of $X = \frac{(x_1^2+y_1^2)-(x_2^2+y_2^2)}{2(x_1-x_2)}$ would go as follows: sort points by the $x$-coordinate, in case of equality put points with bigger absolute value of $y$ coordinate first. Then, find this $X$ for every pair of adjacent points whose $x$ coordinates are different, and take the largest of them.

Similarly, find the smallest such $X$, and take the difference between them.'

## Problem B. Politicians and Competitive Programmers

Let's assign a value of 1 to competitive programmers and 0 to politicians. Then it's easy to see, that the query simply returns bitwise $XOR$ of the values of $X$, $Y$, $Z$.

So, we have the following problem: $n \geq 5$ binary values $x_1, x_2, \ldots, x_n$ are hidden. We can choose 3 distinct indices $(i, j, k)$, where $i$ can't be selected as $i$ more than once, and get $x_i \oplus x_j \oplus x_k$. We need to find all the values.

Let's start with first four values, and ask $y_1 = x_1 \oplus x_2 \oplus x_3, y_2 = x_2 \oplus x_3 \oplus x_4, y_3 = x_3 \oplus x_4 \oplus x_1, y_4 = x_4 \oplus x_1 \oplus x_2$.

From here, we can deduce $y_1 \oplus y_2 \oplus y_3 = x_3, y_2 \oplus y_3 \oplus y_4 = x_4, y_3 \oplus y_4 \oplus y_1 = x_1, y_4 \oplus y_1 \oplus y_2 = x_2$.

After we determined the first 4 values, we can determine the next ones one by one by asking for $x_{i+1} \oplus x_i \oplus x_{i-1}$.

## Problem C. Fast Squarier Transform

Note that there can be at most 6326 different numbers among $a_i$. Indeed, if there are at least 6327 distinct elements there, their sum is at least $0 + 1 + \ldots + 6326 = 20005975 > 20000000$.

So, for every element $x$ of $a$ count $cnt[x]$ — the number of times it appears in $a$, for every element $y$ of $b$ count $cnt[y]$ — the number of times it appears in $b$, and for each such pair $(x, y)$ add to the answer $cnt[x]cnt[y]\lfloor\sqrt{|a_i - b_j|}\rfloor$

This works in $O(n + m + \sqrt{sumA}\sqrt{sumB})$.

## Problem D. LIS of Pairs

First, let's count the number of pairs with $LIS = 4$. Clearly, we should consider only pairs with $a_i < b_i$, and have to count the number of such pairs $(i, j)$ that $b_i < a_j$. For this, count the $cnt[x]$ — the number of pairs with $a_i < b_i \leq x$ for each $x$ with prefix sums in $O(n + m)$, and for each $(a_i, b_i)$ with $a_i < b_i$ add $cnt[a_i - 1]$ to the answer.

We count the number of pairs with $LIS = 1$ similarly.

Let's count the number of pairs $(i, j)$ with $f(i, j) \geq 3$, and then we will subtract the number of pairs with $LIS = 4$ from it, obtaining the number of pairs with $LIS = 3$. We will find the number of pairs with $LIS = 2$ similarly.

Now, if $f(i, j) \geq 3$, at least one of the following must hold:

- $a_i < b_i < max(a_j, b_j)$
- $min(a_i, b_i) < a_j < b_j$

It's easy to count the number of pairs $(i, j)$, which satisfy $a_i < b_i < max(a_j, b_j)$: again, count $cnt[x]$ — the number of pairs $(a_i, b_i)$ with $a_i < b_i \leq x$, we can do this with prefix sums. Similarly we can find the number of pairs $(i, j)$ which satisfy $min(a_i, b_i) < a_j < b_j$. It remains to subtract the number of pairs which satisfy both to avoid double counting.

So, we consider only pairs with $a_i < b_i$, and need to count the number of pairs $(i, j)$ such that $b_i < b_j$ and $a_i < a_j$. This can be done with $2D$ prefix sums: count $cnt[x][y]$— the number of pairs with $a_i \leq x$ and $b_i \leq y$, and add $dp[a_i - 1][b_i - 1]$ for every pair.

This solution works in $O(n + m^2)$.

## Problem E. Patriotic Painting 1

Suppose that point $x$ is counted by some painter. What condition on $k$ must hold so that no matter what $k$ painters are chosen, at least one of them paints $x$? Suppose that $t$ painters would paint $x$. Then, we can choose $n - t$ painters such that none of them would paint $x$, but in any set of $n - t + 1$ painters there would be at least one which would paint $x$. So, the condition is $k \geq n - t + 1$.

So, we just need to find the $min\_painted$ — the smallest of these numbers $t$ over all points which are painted by at least one painter, and output $n - min\_painted + 1$.

We can find this $min\_painted$ by traversing the line from left to right and keeping the number of painters who would paint the current point. When we meet the end of the segment of some painter, we subtract one from this number, when we meet the start, we add one to it, and we just need to find the smallest nonzero value, that's all.

**Note:** It seems that many teams in the start misread the statement, thinking that we only paint integer points. Because of this, we had a weird monitor effect, where teams were scared to try this problem, even though it's one of the easiest in the set. Sad...

## Problem F. Patriotic Painting 2

Suppose that in the final coloring there will be $m$ yellow consecutive segments of lengths $c_1, c_2, \ldots, b_m$. Consider some segment, it was formed from some subset of operations, suppose with length $b_1, b_2, \ldots, b_s$. It's clear that from segments $b_1, b_2, \ldots, b_s$ we can form any yellow segment of length from $max(b_1, b_2, \ldots, b_s)$ to $b_1 + b_2 + \ldots + b_s$.

How would we check if we could arrive to these lengths $c_1, c_2, \ldots, c_m$ then? We would look at all possible partitions of the set $\{1, 2, \ldots, k\}$ into $m$ nonempty parts (where order of these groups matters), and for each of them we would see if corresponding $c_i$ lies between corresponding max and corresponding sum.

This means that for checking we don't care precisely about values of $c_i$, we care only about their position relative to all possible maxes and all sums. So, let's write all these bounds (sums of all subsets and values $a_i - 1$), sort them, obtaining some $d_1, d_2, \ldots, d_t$, and solve the following problem:

How many ways are there to select $m$ yellow segments so that the length of the $j$-th one is in the range $[d_{i_j}+1, d_{i_j+1}]$? If we can solve this problem, then we will iterate through all possible choices of indices $i_j$ (there are less than 20 of them, which gives at most $20^4$ configurations in total), and for each choice, we will check if we can obtain it from some partitioning of the set $\{1, 2, \dots, k\}$ into $m$ nonempty parts.

So, from now on we will be focusing on the following problem:

*Main Problem:* How many ways are there to select $m$ yellow segments so that the length of the $i$-th one is in the range $[l_i, r_i]$?

First, let's solve the following:

*Subproblem:* How many ways are there to select $m$ yellow segments so that the length of the $i$-th one is at least $[l_i]$?

This is an easy problem. Indeed, let's subtract $l_i - 1$ from $n$ for each $i$, reducing the problem to selecting $m$ yellow segments of length at least 1 on the strip of length $n_1$. It's the same as selecting $2m$ distinct ends of the segments, which is $C(n_1+1, 2m)$.

Now, just do inclusion-exclusion, substituting $r_i$ or $l_{i-1}$, and solving the subproblem above $2^m$ times.

## Problem G. If You Are Homeless... Just Build a House

For a given region $h \times w$ with values $b_1 \le b_2 \le \dots \le b_{hw}$, the answer is the sum of $|b_i - b_{med}|$, where $med = \lfloor \frac{hw}{2} \rfloor$. It can also be stated as the following: let's keep the set of smallest $med$ numbers $S_1$, and the set of largest $med$ numbers $S_2$, then the answer is just the sum of numbers from $S_2$ minus the sum of numbers from $S_1$.

Now, let's make some kind of sliding window and support these sets $S_1, S_2$ together with the sums in them in the process. Let's start from the area $h \times w$ in the left upper corner, and move it to the right by 1 every time. We would need to do $O(h)$ insertions/deletions per moving it by 1 to the right.

Next, let's move it down by 1, which takes $O(w)$ insertions/deletions, and start moving it to the left, then again down and to the right, and so on.

Overall we will make $O(mnh+nw)$ insertions/deletions while moving this sliding area, which gives runtime $O((mnh+nw)\log hw)$. The answer is just the smallest value $sum(S_2) - sum(S_1)$ over all possible positions of the area.

## Problem H. Another ICPC String Problem

Let's keep 26 bitsets $bit_a, bit_b, \dots, bit_z$, where corresponding bitset has ones only at positions where his letter is present. Updating these bitsets after queries of type 1 is trivial, let's see how to deal with queries of type 2.

Suppose that we received a string $p_1p_2\dots p_k$ as the query string. Then consider bitset $bit_{p_1}\text{AND}(bit_{p_2} >> 1)\text{AND}(bit_{p_3} >> 2)\dots\text{AND}(bit_{p_k} >> (k-1))$ (here by $>> x$ we denote the shift of bitset to the left by $x$, and by AND — binary AND of the bitsets.

Clearly, in this bitset we will have ones only at such positions $pos$ that $s_{pos} = p_1, s_{pos+1} = p_2, \dots, s_{pos+k-1} = p_k$, which is precisely what we need. To find the number of the ones set, use bitset.count().

Total complexity would be $O(\frac{n(q+sum(|p|))}{64})$, which passes comfortably.

## Problem I. Vovochka

As the order of numbers in the input doesn't matter, let's suppose that they are given in sorted order, and let's say that the median of numbers $a_i, a_j, a_k$ is $a_j$ for $i < j < k$.

Suppose that our triples are $(a_{x_i}, a_{y_i}, a_{z_i})$ with $x_i < y_i < z_i$. Wlog, they are sorted by the position of the middle element, so that $y_1 < y_2 \dots < y_n$. Then, we must have $y_i \ge 2i$ for each $i$, as $max(x_j, y_j) \le y_i$ for $j \le i$. Similarly, we must have $y_i \le 3n + 1 - 2(n+1-i)$, or $y_i \le n - 1 + 2i$.

It can be shown that these conditions on $y_i$ are sufficient: if $2i \le y_i \le n-1+2i$, we can select $x_i, z_i$ in such a way that each number from 1 to $3n$ is in exactly one triple and $x_i < y_i < z_i$. Just take $x_i$ as the $i$-th number which isn't one of $y_i$, and $z_i$ as the $n+i$-th number which isn't one of $y_i$. Indeed, before $y_i$ there are at least $i$ numbers which aren't one of $y$, and after $y_i$ there are at least $n+1-i$ numbers which aren't one of $y$.

Now, how to count the number of distinct multisets $a_{y_1}, a_{y_2}, \ldots, a_{y_n}$ under the conditions above? We will do dp, let's process the groups of equal numbers one by one from left to right. Suppose that we are processing a group of $k$ numbers, that we already processed $l$ numbers, and selected $s$ from them as $y$-s, let's denote $dp[l][s]$ as the number of different ways to select those $s$ $y$-s from first $l$ numbers.

How many numbers can we select as $y$s from our group of $k$ numbers? If we select $s_1$ of them, then the following conditions have to hold:

- $s + s_1 \le n$ (trivial)
- $l + k \ge 2(s + s_1)$ (as $s + s_1$-th $y$ has to be among first $l+k$ elements)
- $l + k + 1 \le n - 1 + 2(s + s_1 + 1)$ if $s + s_1 + 1 \le n$, (as $s + s_1 + 1$st $y$, if it exists, has to be starting from $l+k+1$-th element).

It's easy to show that if these conditions hold, we can choose $s_1$ $y$s among $l+1, l+2, \ldots, l+k$, so that all conditions on them would be true.

So, we just have a dp, where we can transition from $dp[l][s]$ to $dp[l+k][s+s_1]$ when the conditions above hold. It's easy to see that for each $s_1$ we have at most $n$ transitions, and the sum of all $s_1$ is precisely $3n$, so the total complexity is $O(n^2)$.

## Problem J. Typical ICPC Problem

For a tree, it's a pretty well-known problem, but let's remind the solution.

Let's process the nodes in order from the smallest weight to the largest, keeping the current right end $r$. For each node with weight $i$ from 1 to $r$, we will also keep the value $t[i]$, which is $r - i + 1-$ the number of edges whose both ends have weights in range $[i, r]$. Then, the range $[l, r]$ will be good if and only if $t[l]$ will be equal to 1 when we processed all nodes with weight up to $r$.

It's easy to update all the values when moving from $r$ to $r+1$: we just have to add/subtract 1 on some segments. Also note that all numbers are always at least 1, and just have to return the number of ones. This hints to the segment tree solution, which allows queries of adding on subsegment, finding minimum on subsegment, and finding the number of minimums.

But it's not a tree, it's a CACTUS!!! How to deal with this case then?

In the tree case, we used that a forest is connected iff the number of nodes $-$ number of edges is precisely 1. In the cactus case, we have something similar: cactus is connected iff the number of nodes $-$ number of edges $+$ number of cycles is precisely 1. So, the algorithm would be to do the following: determine all edge cycles, which are luckily disjoint, for each of them determine the largest and smallest weight of a node, and when we are processing nodes in the same order of weight, add 1 on prefix $[1, \text{smallest weight on a cycle}]$ when $r$ becomes equal to the largest weight on the cycle.

## Problem K. ICPC and Typos

Let $m$ denote the length of the string $s$.

Note that typos of different types produce words with different number of letters, so it's enough the number of possible words that may occur from each type separately, and then add them.

For type 3 it's easy: each letter can be replaced by $n-1$ other letters, so here we get $m(n-1)$ words.

For type 1: let's consider strings $s_1s_2\ldots s_{i-1}s_{i+1}\ldots s_m$ for each $i$ from 1 to $n$. When are two such strings equal? $s_1s_2\ldots s_{i-1}s_{i+1}\ldots s_m = s_1s_2\ldots s_{j-1}s_{j+1}\ldots s_m$ with $i < j$ if $s_i = s_{i+1}$, $s_{i+1} = s_{i+2}$, ..., $s_{j-1} = s_j$. In other words, when all letters from $s_i$ to $s_j$ are equal. This implies that the number of distinct words is $1 +$ the number of such $i$ that $s_i \neq s_{i+1}$.

For type 2: let's count the number of ways to insert some character. Let the word after inserting it be $t$.

If $t_1 \neq s_1$, the character different from $s_1$ must have been inserted before the first position, which gives $n-1$ words. Otherwise, $s_1 = t_1$, and we safely assume that we inserted after the first position (even if we inserted $s_1$ before $s_1$, we could have inserted it after $s_1$, and nothing would change. So we can consider $t[2 : m+1]$, $s[2 : m]$ from now on.

This iterative process would give us $(n-1)+(n-1)+\ldots+(n-1)$ ($m$ times) $+\ n$ (resulting from inserting any letter in the end) different words.