Problem A. Bags of Candies

If we make k pairs while packing n flavors, and pack remaining $n - 2 * k$ flavors separately, then number of bags is equal to $(n + 2 * k) + k = n - k$. So, we can get the minimum number of bags making maximum number of pairs.

Let $D(n)$ be a set containing every prime number strictly greater than $\frac{n}{2}$ and not greater that n, and additional number 1. We can notice, that no number from the $D(n)$ can be paired, so the upper bound of number of pairs is $\lfloor \frac{n-|D(n)|}{2} \rfloor$.

Let's show, that the mentioned constraint can be reached. We should pair all the numbers from $R(n) = \{1, \ldots, n\} \setminus D(n)$, maybe with the exception of one number (if the $|R(n)|$ is odd).

Let $f(x)$ for $x \geq 2$ be the largest prime divisor of x. Let's split the numbers from $R(n)$ in groups by the value $f(x)$.

We can notice, that for every group of that division, if the $f(x)$ corresponds to the group, then there is a $2 * f(x)$ in that group. Every two numbers from the group can be paired, so let's pair all the numbers, if the size of group is even, or leave number $2 * f(x)$ unpaired if the size of group is odd. So we are left with some $2 * f(x)$ from odd-sized groups, and we can make pairs from them. That was the proof, that we can pair all the numbers from $R(n)$ with the exception of maximum one.

How do we count $|D(n)|$? It can be counted using function $\pi(n)$, which means number of primes not greater that n ($|D(n)| = \pi(n) - \pi(\lfloor \frac{n}{2} \rfloor)$). We can find prime numbers from 1 to $\sqrt{n}$ in $O(\sqrt{n} \log \log n)$. Now we can compute number of prime numbers in segment $[l, r]$ in $O((r - l) \log \log (r - l) + \sqrt{r})$ using modification of Eratosthenes sieve: for every prime number p not greater than r, mark every multiple of p in segment $[l, r]$. In the end, not marked number are prime.

In our problem $n \leq 10^{11}$, so let's do preprocessing. For every $0 \leq d < 10^4$ get the number of prime numbers in segment $[d * 10^7, (d + 1) * 10^7 - 1]$ with mentioned algorithm. Do this computations locally (it takes a few minutes).

So, to compute $\pi(n)$, we already have number of primes in blocks, that are fully inside the segment $[1, n]$, and maybe segment in form of $[d * 10^7, n]$, which we can compute using our algorithm.

Finally, our solution requires time $O(n \log \log n)$ locally, and $O(\frac{n}{b} \log \log n + \sqrt{n})$, where b is number of blocks (in our case 10^4).

Problem B. Binomial

Theorem 0.1 (Lucas) *For non-negative integers m and n and a prime p, the following congruence relation holds:*

$$\binom{n}{k} \equiv \prod_{i=0}^{m} \binom{n_i}{k_i} \, (mod \, p)$$

where:

$$n = n_0 \cdot p^0 + n_1 \cdot p^1 + \ldots + n_m \cdot p^m$$

and

$$k = k_0 \cdot p^0 + k_1 \cdot p^1 + \ldots + k_m \cdot p^m$$

are the base p expansions of m and n respectively.

Using this theorem it's easy to see that $\binom{n}{k}$ is odd if n is a supermask of k. Let's assume that $\max\{a_1, \ldots, a_n\} < 2^w$ for some w. For each a_i we want to find the number of its submasks in the sequence. It's easy to do in $\mathcal{O}(2^w \cdot w)$ using Sum over Subsets(SOS) DP.

The total complexity is $\mathcal{O}(MAX \log MAX)$, where MAX is the maximum number in the array.

Problem C. Bookface

First, let's sort the sequence x_i; from now on we assume $x_1 \leq x_2 \leq \ldots \leq x_n$. We can imagine that the x_i values are points on a line, which we want to move so that no two are closer than d. Note that for any i, the i-th point is to the left of the $(i+1)$-th point, and it is suboptimal for them to "swap", so we can assume the relative order of points will not change, and only enforce the condition that $x_{i+1} - x_i \geq d$ must hold for any i. Since x_i's must remain non-negative, we cannot shift our points below 0. It is convenient to remove this condition; for example, by adding the values $-d, -2d, -3d, \ldots, -(n+1)d$ to the input sequence x_i. Then, it will never be optimal to move any of the original values to negative, and we can forget the $x_i \geq 0$ condition from now on. Now let's define $x'_i = x_i - i \cdot d$; note that x'_i may no longer be a non-decreasing sequence. Moving x_i by ± 1 corresponds to moving x'_i by ± 1. Moreover, $x_{i+1} - x_i \geq d$ translates to $x'_{i+1} \geq x'_i$. While the reduction from this paragraph is not strictly necessary, it allows us to forget the value of d from now on. To solve the final problem, consider the dynamic programming $f[i][t]$, defined as the minimum total cost to make our sequence non-decreasing if we consider only the first i elements, and the value of the last element has to be t. If we define $g[i][t] = min_{t' \leq t} f[i][t']$, then $f[i][t] = |t - x'_i| + g[i-1][t]$. Of course we cannot maintain $f[i][t]$ explicitly, as the range for possible values of t would be prohibitively large. However, we can prove by induction that any $f[i]$ is a bitonic piecewise linear function with respect to t. Transforming $f[i]$ to $g[i]$ corresponds to removing everything after the minimum value of $f[i]$ and replacing that with a constant segment. The last linear piece in the function $g[i]$ will have slope 0, so the last piece in the function $f[i]$ will have slope 1. In order to obtain the optimum solution we don't need to store the values of $f[i]$, only the shape. Then, having restored the optimal sequence x'_i we can compute the cost. We see we only need to store the "breakpoints" of f; i.e. the points where the slope increases by 1. These points can be stored in a multiset. We process element of the sequence x'_i from left to right. We maintain the multiset of breakpoints S, which corresponds to the current function g. To process an element x, we need to insert x into S twice (since in the point x the slope will change by 2). Now, S represents the new f. To transform it into g, we need to remove the rightmost breakpoint. The rightmost breakpoint in g at a given moment is the optimum (smallest value of the cost) for a given prefix. If we save these breakpoints in a sequence p_i, then it's easy to use p_i to compute the optimal solution (hint: go from right to left through p_i and greedily fix it so that it's non-decreasing).

Problem D. Clique

Let's fix the interval p, and let's assume that it is the shortest in the solution. Therefore, among the segments from which we choose a solution, there are none of those that are inside the p. In addition, segments that do not intersect p can skip immediately, and the segments that contain both ends of p (that is, either contain the entire p, or contain both ends of p) can be taken without loss of generality.

We are left with segments that contain exactly one of the ends of p. We will call the ends of p as A and B. We want to choose such a subset of segments that each selected segment containing A intersects with each selected segment containing B

These ranges can be converted to points on a plane; we will change those containing A into black, and those containing B into white. Perform the transformation as follows.

Consider the segment containing A, which protrudes(stare, stick out) on one side of this point by x_a, and on the other by y_a, and another segment that contains B and protrudes on one side by x_b, and on the other by y_b. If the distance between A and B is len_{AB} on the one hand, and len_{BA} on the other, these intervals will not intersect if $x_a + x_b < len_{AB}$ and $y_a + y_b < len_{BA}$ equivalently, $x_a < len_{AB} - x_b$ and $y_a < len_{BA} - y_b$. We transform the first interval into a black point (x_a, y_a), and the second interval into a white point $(len_{AB} - x_b, len_{BA} - y_b)$.

After transformation, our problem is reduced to: select the largest subset of points, so that no white is larger at both coordinates than any black.

Equivalently, if we think about the "extremely top chain" of selected white points, then under this chain we want to take all white points, and above it – black.

We calculate dynamic programming dp_y - the result for currently processed points, assuming that our chain of white points ends at a height of at most y. To define it, it is convenient to remap (rescale) all y coordinates of points to value in the range $[0 \dots n-1]$ first. It can be one with map or sorting. Note that dp is non-decreasing by definition. We process points from right to left (ascending of the x coordinate), and update dp. When we process a white point at the height of y, we want to increase by 1 the value of $dp_{y'}$ for $y' \geq y$. When we process a black point at the height of y, it turns out that it is enough to increase by 1 the value of $dp_{y'}$ for $y' \leq y$. Also, we must also keep the monotonicity of dp, when we process a black point (find last position i that $dp_i \leq dp_y$ and increase only the prefix i by $+1$). Finally, our result is in dp_{n-1}. It remains to be seen what data structure we use to represent dp. We need to add 1 on the interval, find the value at a point, and find for the given v the last i such that $dp_i \leq v$. For this a simple segment tree is sufficient: at each node we keep the added value on the base range and its minimum. Thanks to the minimum, we are able to find the desired position in $\mathcal{O}(\log n)$ going down from the top tree. To sum up, it takes $\mathcal{O}(n \log n)$ to solve the sub-problem with white and black points. As we do this n times, the final complexity is $\mathcal{O}(n^2 \log n)$.

Problem E. Contamination

The task is that for a fixed set of circles on the plane we should answer for the queries of form (p, q, y_{min}, y_{max}), where $p = (p_x, p_y)$ and $q = (q_x, q_y)$ is two points none of which lie in one of the circles, and y_{min} and y_{max} are two such numbers, that $y_{min} \leq y_{max}$. For each such query, it is necessary to determine whether there is a curve at ends in p and q, which entirely lies in the band $Y = \{(x, y) \in \mathbb{R}^2 \mid y_{min} \leq y \leq y_{max}\}$.

It's easy to see that such a curve exists if and only if there is no circle, whose vertical diameter "separates"points p and q in the Y band, that is, when it does not exist such a circle with the center (c_x, c_y) and radius r that $min\{p_x, q_x\} \leq x \leq max\{p_x, q_x\}$ and $c_y - r \leq y_{min} \leq y_{max} \leq c_y + r$.

We can solve this problem using scanline the plane with a horizontal line from smallest y coordinates to highest. We keep a set of active vertical diameters on the segment tree. Vertical diameter of the circle with the cnehter (c_x, c_y) and radius r becomes active at the height $c_y - r$, and stops to be active at the height of $c_y + r$. The answer to the query (p, q, y_{min}, y_{max}) is found at the height of y_{min}, simply checking the segment tree maximum on the range $[min\{p_x, q_x\}, max\{p_x, q_x\}]$. If this maximum is at least y_{max}, then answer to the query is «NO», otherwise «YES». The complexity of the above algorithm is $\mathcal{O}(n \log n)$

Problem F. The Halfwitters

Consider the number of inversions in a permutation. It's easy to see that only in a sorted permutation ther are zero inversions. Ehe first action can increase or decrease the number of inversions by one. For permutation of size n, the second action changes the number of inversions from k to $\frac{n*(n-1)}{2} - k$. Let X be the number of seconds to spend on average to sort a random permutation. Then after the third action we must spend X seconds in average. Therefore, the answer depends only on the number of inversions in the permutation.

At first, let's solve the problem, so if there were no actions of the third type. Let's maintain dp_i — the number of seconds that we need to spend to sort the permutation that originally has exactly i inversions. Now the answer for the permutation with i inversions will be $ans_i = min\{dp_i, X + c\}$. But we still do not know the value of X. Let L_i be the number of permutations, that have exactly i inversions. Then $X = \sum L_i * ans_i$, where i — the number of inversions. Now if we knew for what i $ans_i = dp_i$ and for what $ans_i = X + c$, we could count X. Let's sort the dp_i and for every j in sorted array save num_j — initial index of dp_j. Now fix some position j and say that for prefix j: $ans_{num_j} = dp_j$ ($dp_j \leq X + c$) and for the suffix $j + 1$: $ans_{num_j} = X + c$ ($dp_{j+1} \geq X + c$). Let cnt_j be the number of permutations that have the number of invertions equal to the num_j, in other words $cnt_j = L_{num_j}$. Now let $S = \sum_{l=0}^{j} cnt_l \cdot dp_l$ and $M = \sum_{l=j+1}^{k} cnt_l$, where $k = \frac{n*(n-1)}{2}$ — maximum number of inversions. We now have the following equation:

$$X \cdot n! = S + M \cdot (X + c)$$

with one unknown X. Solve this equation and check if $dp_j \le X + c$ and $dp_{j+1} \ge X + c$. If the conditions are met, then we found the value of X.

We can precalculate L_i for each possible n, using any dynamic progamming. Author's dp works in time $\mathcal{O}(n^4)$. After precalc for each test case we should find X in time $\mathcal{O}(n^2 \log n^2)$, and for each day we should find only the number of inversions in permutation in $\mathcal{O}(n^2)$. So the total complexity is $\mathcal{O}(n^4 + n^2 \log n^2 + d \cdot n^2)$

Problem G. Invited Speakers

Let's sort our arrays a and b of pairs in descending order($(a, b) < (c, d)$ if $a < c$ or $(a = c \wedge b < d)$). Now let's take a positive $C > \max\{|x|, |y|\}$, where (x, y) is an arbitrary element of a or b. Now let's connect point a_i with point b_i through points $(a_i.x, C + i)$, $(C + i, C + i)$, $(C + i, -C - i)$, $(b_i.x, -C - i)$.

Problem H. Lighthouses

The main observation is that if we went from the vertex x to y and then to the vertex z that lies between these vertices, then we will never be able to go to the vertex which does not lie between x and y.

Let's maintain $dp_{i,j,0..1}$.

1. $dp_{i,j,0}$ — the answer if the last path we went was ended in i and we still can visit all the vertices with numbers from $i + 1$ to j
2. $dp_{i,j,1}$ — the answer if the last path we went was ended in j and we still can visit all the vertices with numbers from i to $j - 1$.

Here we mean that the next vertex after vertex n is vertex 1.

Now if we want to make a transition to a state $dp_{i,j,0..1}$, we just go over the next vertex *to* to which we will go. This vertex must lie between i and j and there must be a tram line between i or j and *to*.

The total complexity is $\mathcal{O}(n^3)$, because we have $\mathcal{O}(n^2)$ states and $\mathcal{O}(n)$ transitions in dp.

Problem I. Sum of palindromes

Let A be the input number, and let it have n digits (which means that $A \le 10^n$). Suppose that $n \ge 3$. We can express A as $A_H A_L$ — concatenation of two numbers, where $|A_H| = \lceil \frac{n}{2} \rceil$ and $|A_L| = \lfloor \frac{n}{2} \rfloor$. We substract 1 from A_H, obtaining $A'_H = A_H - 1$ and then append $|A_L|$ digits to A'_H such that the result is a palindrome B. Now $A - B \le 10^{n/2+1}$, so we keep B as one of the desired palindromes, reducing the length of A by a factor of 2. In $\log_2 n + 1 \sim 18$ iterations we reach a 2 digit number, which we deal with as a special case.

Problem J. Space Gophers

Let's consider two tunnels as connected if one can travel directly from one to the other (in other words, if there exists a microcube in one tunnel and another microcube in the other, such that these microcubes touch or coincide). This gives a graph G with tunnels as vertices. In order to see if we can travel between two microcubes c_1 and c_2, we can take any tunnel t_1 that contains c_1, and t_2 that contains c_2, and ask if tunnels t_1 and t_2 are connected in G. Therefore, to solve the problem it is enough to compute the connected components of G. Unfortunately, there can be $\mathcal{O}(n^2)$ edges in G, so we cannot store G directly. However, we can maintain a disjoint-set-union structure on tunnels, and join only some pairs of tunnels that will span the same set of connected components as the full set of edges in G. Let's consider which pairs of tunnels are connected in G. There are two cases: either the connected tunnels are parallel or perpendicular. The first case is easy, since for any tunnel there are just four possible candidate tunnels. Let's now focus on the second case. Assume that we want to consider connected pairs of tunnels a and b,

where a is parallel to the x-axis, and b to the y-axis. If we have a function to perform this, we can then call it for the two other pairs of directions. Denote the z coordinate of a and b as z_a and z_b respectively. Notice that, since a and b are perpendicular, they will be connected if and only if $|z_a - z_b| \leq 1$, the other coordinates of the tunnels don't matter. Let's now group all tunnels parallel to the x-axis by their z coordinate into groups $X[z]$. Similarly, group all tunnels parallel to the y-axis into groups $Y[z]$. Consider a group $X[z]$. We would like to join all tunnels in that group with all tunnels in groups $Y[z-1]$, $Y[z]$ and $Y[z+1]$. If these three groups are empty, then there is nothing to be done. Otherwise, we can join all these tunnels into one component. After that, we can reduce groups $X[z]$, $Y[z-1]$, $Y[z]$ and $Y[z+1]$ into just one tunnel each, since all tunnels in each of these groups are in the same component. By doing so we will join two tunnels only $\mathcal{O}(n)$ times. The complexity is $\mathcal{O}(n \log n)$, since we need to group the tunnels by coordinates.

Problem K. To argue, or not to argue

If number of available seats is less than $2 * k$ then the answer is 0. Let $R(j)$ be the number of arrangements that there are j pairs that share a common side. Then the answer is $(\prod_{j=0}^{k-1} \binom{availableSeats-2*j}{2} + \sum_{j=1}^{k} (-1)^j * (R_j * \binom{k}{j} * j! * \prod_{r=0}^{k-j-1} \binom{availableSeats-2*j-2*r}{2})) * 2^k$.

We can easily compute $R(j)$ using dynamic programming on "broken profile". Consider that $m < n$, if it is not, just transpose the matrix. Let's assign number to each cell of matrix, the number of cell (x, y) is $(x-1) * m + y$. $dp_{i,j,mask}$ is number of ways to arrange first i cells, so that j pairs are sharing a common side and the $mask$ is a binary representation of last $m+1$ cells we have looked at (bit is 0 if the cell is occupied and 1 otherwise).

$\forall j \ R(j) = \sum_{mask=0}^{2^{m+1}-1} dp_{n*m,j,mask}$.

Problem L. Wizards Unite

Let $t := \sum_{i=1}^{n} t_i$. Let's say that we use silver keys at moment 0 to open first k chests with opening times $t_1, \ldots, t_k$. Then the total time to open all the chests equals to $\max\{t_1, t_2, \ldots, t_k, t - (t_1 + t_2 + \cdots + t + k)\}$. On the other hand, it's obvious that the minimum possible total time to open all the chests cannot be less than $\max\{t_1, \ldots, t_n\}$. So it's easy to see that to minimize the total time we should use silver keys to open k chests with the highest opening times and use the golden key to open other chests.

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