

A UTPC Valentine's Gala

Author: Zafir

Iterating through the grid, we have a valid couple if $A[i][j] = A[j][i] = 1$. We should consider double counting, either by dividing by 2 or only considering $i < j$.

B Lover's Gift

Author: Nick

Let us split this into two cases:

n is even: We can build the following permutation: $[\frac{n}{2} + 1, 1, \frac{n}{2} + 2, 2, \dots, n, \frac{n}{2} - 1]$, and you will see that we have beauty $\frac{n}{2}$. Also note that we cannot have beauty $> \frac{n}{2}$ as every element in the permutation must be beside at least one element, and if you consider the element $\frac{n}{2}$, the element of greatest absolute difference is n , so placing it next to n results in $\frac{n}{2}$ beauty in the best case. A concrete example of the above permutation would be for $n = 6$, $[4, 1, 5, 2, 6, 3]$.

n is odd: Similar logic can be applied. See the following construction: $[\frac{n+1}{2}, 1, \frac{n+1}{2} + 1, 2, \dots, n]$. The element $\frac{n}{2}$ has a difference of $\frac{n+1}{2} - 1$ from both 1 and n , thus this is the maximum beauty we can attain in the odd case. A concrete example of the above permutation would be for $n = 7$, $[4, 1, 5, 2, 6, 3, 7]$.

You can use these formulas directly for each case or simplify to $\lfloor \frac{n}{2} \rfloor$ in the general case.

C Edward is Sigma

Author: Johnny Liu

The important thing is that the array is increasing. This means that for some fixed size of the subsection, the charisma average is strictly increasing. σ

This means that for each size of subsection, we can binary search to see if there is match in $\log n$.

D Supply and Demand

Author: Michael Xiang

Suppose we focus on a single shop i . The demand is $a_i - b_i \cdot p$, the price is p , and the per-unit cost is c_i . Profit is demand times margin per unit, which is $(a_i - b_i \cdot p)(p - c_i)$. We can expand this out to be:

$$-b_i p^2 + (a_i + b_i \cdot c_i)p - a_i c_i$$

which is a downward-facing parabola.

Taking this idea over a range of shops $[l, r]$, we get a similar formula:

$$-\left(\sum_{i=l}^r b_i\right)p^2 + \left(\sum_{i=l}^r a_i + b_i c_i\right)p - \left(\sum_{i=l}^r a_i c_i\right)$$

Let $A = (\sum_{i=l}^r b_i)$, $B = (\sum_{i=l}^r a_i + b_i c_i)$, and $C = (\sum_{i=l}^r a_i c_i)$. The optimal price is then the vertex of this parabola, $p = \frac{B}{2A}$. Since this could be a non-integer, it suffices to check $\lfloor p \rfloor$ and $\lceil p \rceil$.

Naively, this is $O(n)$ per query, but A , B , and C can be computed with prefix sums for $O(1)$ per query. So, the entire problem can be solved in $O(n + q)$ time.

Note that if p exceeds m , you should plug in m into the equation instead of p . Also, be careful when A is 0. In that case, just return the value after plugging in m .

[Solution by ChatGPT.](#)

E I Wanna Know...

Author: Warith Rahman

This problem is essentially two parts: finding the k th difference, and then finding the best ticket pair with this difference.

Let us first sort the tickets by price. For a given difference d , we can use 2-pointer to find the number of pairs with difference at most d . This gives us a method to binary search for the k th difference.

Now that we have this exact difference, we can run a similar 2-pointer search to find the best ticket pair.

For implementation, you can refer to [Warith's code](#) or [Edward's code](#), both in C++.

P.S. Regarding the announcement sent in-contest, the answer (in order) was: [What is Love](#), [The Feels](#), [Queen of Hearts](#), and [One Spark](#). These are all very good songs that Warith recommends you listen to :)

F The Perfect Gift

Author: Dylan

We can rewrite the MEX function in a way that helps us compute the sums we want. Notice that $\text{MEX}(a) = \sum_{i=0}^{\infty} 1_{j \in a \forall j \leq i}$ where $1_{j \in a \forall j \leq i}$ equals 1 if a contains $1, \dots, i$, and otherwise it equals 0. Since a is a permutation, we know that for some i , a subarray has $1_{j \in a \forall j \leq i} = 1$ iff it contains the leftmost and rightmost values among $1, \dots, i$. For each $0 \leq i \leq n-1$ we will find such values. Let them be l and r respectively. Notice that any subarray containing l and r containing j where $l \leq j \leq r$, so i contributes $(l-1) \cdot (n-r)$ to the answer for such j . Now if $j < l$, the contribution is $(j-1) \cdot (n-r)$, and there is a symmetric case where $j > r$. Notice that the contribution changes at a linear rate, and in general it is a piecewise linear function whose slope changes at l and r . We can store all of the slope changing points in an array, and then reconstruct the function via prefix sums. I've left out the details on exactly which indices to change and by how much, but you can see my code for reference.

Here is my [code](#).

G Four in a Row

Author: Michael Xiang

We can solve this using dp. Let $dp[i][j][k][l]$ be the number of permutations of $[1, i]$ such that j is the last element, there are exactly k length-4 increasing subarrays, and the last l elements are increasing. Note that l is as large as it can be, though it is capped at 3. To transition from i to $i+1$, we can consider "sliding" in a new value, x , into the array, pushing up other elements with a value at least x as needed. So if we have the array $[1, 5, 3, 2, 4]$ and we insert 3 at the end, we get $[1, 6, 4, 2, 5, 3]$.

Let the current array at i be $a_1, a_2, \dots, a_i$. When going from i to $i+1$, let's consider adding a value at the end that is less than a_i . This means we go from some value l to 1. This can be done with the following transition:

$$dp[i+1][j][k][1] = \sum_{l=1}^3 \sum_{j'=j}^i dp[i][j'][k][l]$$

Then we can consider adding a value at the end that is greater than a_i . This means that we will go from l to $l + 1$, unless l was 3, in which case we add 1 to k and keep l the same. This can be done with the following transition: (We will denote $[l = 3]$ to be 1 if $l = 3$ and 0 otherwise.)

$$dp[i + 1][j][k + [l = 3]][\min(l + 1, 3)] = \sum_{j'=1}^{j-1} dp[i][j'][k][l]$$

Naively, this results in a runtime of $O(n^4)$, but we can use prefix sums on j for fixed values of i, k, l to speed this up to $O(1)$ per transition, for a total of $O(n^3)$.

This results in approximately $500^3 \cdot 4 = 5 \cdot 10^8$ operations naively and approximately $500^3 \cdot 3 \cdot 4 = 1.5 \cdot 10^9$ bytes of memory naively. We can reduce the memory by keeping only the arrays corresponding to i and $i + 1$, which reduces the memory to around $8 \cdot 10^6$ bytes. The time constraints allow for unoptimized operations, but this might TLE depending on the implementation. We can optimize the for loop bounds, which reduces the operations to about $\frac{5 \cdot 10^8}{6} \approx 8.33 \cdot 10^7$ operations.

[Michael's code](#)

H Love Triangles

Author: Randy Guo

A naive solution would iterate through every possible love triangle i, j, k in $O(n^3)$, but this wastes time iterating over many invalid triangles. Instead, we can iterate over every i , its neighbors j , and j 's neighbors k in $O(mn)$ time, but this is still too slow. The crucial optimization is achieved by orienting edges in increasing order of degree, so all edges become directed from u to v where $\deg(v) > \deg(u)$. Then, every node can have at most $O(\sqrt{m})$ out-neighbors because otherwise, there would be over $O(\sqrt{m}) \cdot O(\sqrt{m}) = O(m)$ total edges, which is a contradiction. With this optimization, iterating through every i , its neighbors j , and j 's neighbors k takes $O(m\sqrt{m})$ time, which is fast enough.

I Broken Heart

Author: Edward

We will solve the problem via casework on $z = \lfloor y \rfloor$ where y is the vertical displacement of the left piece. In particular, we will find the optimal horizontal displacement for $z \leq y \leq z + 1$ and take the maximum result for all z values between $-n$ and n . Note that we only need to check a finite range of z values due to the fact that the edges of the pieces are flat outside of a fixed vertical range.

Fix z and consider a fixed $y = z + \lambda$ for $\lambda \in [0, 1]$. Then observe that the left piece can be moved toward the right piece until either a vertex on the edge of the left piece intersects with the right edge or a vertex on the right edge intersects with the left edge. Because y is limited to $[z, z + 1]$, for each vertex on one piece's edge, we only need to worry about it colliding with a single line segment in the opposite piece. Thus, for each vertex, we obtain an inequality of the form

$$\text{horizontal displacement} \leq a\lambda + b.$$

We want to maximize the horizontal displacement subject to all of these inequalities, which can be achieved by building a convex hull of the lines $y = a\lambda + b$ and maximizing over the hull on $[0, 1]$. There are $2n$ such inequalities, so the time complexity of finding the maximum is $O(n \log n)$ by sorting the lines by their slopes and building the convex hull.

As we consider $O(n)$ values of z , the total time complexity of the solution is $O(n^2 \log n)$.

My solution can be found [here](#).

Extra: Can you optimize the solution to $O(n^2)$?