levels of branching, and in each of the levels each of the Us is getting considered only for one interval (with the exception of the boundaries, but these sum up to $O(n \log n)$ as well), so the runtime will be $O((m+n)\log n)$.

The other solution is geometric. For any two points U_i, U_j, let us consider the set of points L for which U_i gives a larger rectangle than U_j. The boundary of this set is a straight line, so the set of points L for which U_i is the best choice is an intersection of half-planes, i.e., a convex polygon which is possibly unbounded in some directions. The division of the plane into these polygons contains a total of $O(n)$ vertices, and so we can run a sweep-line algorithm, with the events being a point from L appears, or the set of regions changes. This will run in $O((m+n)\log(m))$ time.

Problem E: Need for Speed

Shortest judge solution: 321 bytes. Shortest team solution (during contest): 413 bytes.
Python solutions by the judges: both Pypy and CPython

This was one of the two easiest problems of the set. Given a guess for the value of c, we can compute the resulting distance that this would result in. If the guess of c was too high, the travelled distance will be too high (because in each segment travelled we're overestimating the speed), and if the guess was too low, the travelled distance will be too low. Thus we can simply binary search for the correct value of c. The potentially tricky part is what lower and upper bounds to use for the binary search. If in some segement the speedometer read v, the value of c needs to be at least $-v$. Thus, c needs to be at least $-\min v$. For the upper bound, a common mistake was to assume that c could never be larger than 10^6. This is almost true, but not quite – the maximum possible value is $10^6 + 1000$ (our true speed can be as large as 10^6, but the reported readings of the speedometer can be -1000).

Problem F: Posterize

Shortest judge solution: 771 bytes. Shortest team solution (during contest): 686 bytes.
Python solutions by the judges: only Pypy (but for no good reason, should be feasible with CPython as well)

This is a fairly straight-forward dynamic programming problem. Let $C(i,j)$ be the minimum squared error cost of posterizing pixels with intensities $r_1, \ldots, r_i$ using j colors. The quantity we are looking for is then $C(d,k)$.

We can formulate the following recurrence for C:

$$C(i,j) = \min_{0 \le i' < i} C(i', j-1) + F(i'+1, i),$$

where we define $F(a,b)$ to be the minimum cost of posterizing pixels with intensities $r_a, \ldots, r_b$ using a single color. Assuming for the moment that we have computed the function F, it is a standard exercise in dynamic programming to turn this recurrence into an algorithm for computing $C(i,j)$ in time $O(i^2 j)$.

Computing $F(a,b)$ can be done in a few different ways. Note that

$$F(a,b) = \min_{x \in \mathbb{Z}} \sum_{i=a}^{b} p_i (x - r_i)^2.$$