Thomas Game Editorial

All test data is randomly generated (despite the linked generator not compiling... my bad). The first thing you should strive to find out is how that helps. Well for most important thing here is that the convex hull of n randomly generated points on an infinite plane is rather small. I was under the impression it was around $n^{\frac{1}{3}}$ but it turns out $\log n$ is also pretty close. But with some stresstesting, the size of a convex hull of $5 \cdot 10^5$ points is around 40 at most.

Now comes the important observation. Lets define a ring (I am also informed that they are called onions...) as the convex hull of a set of points. Note that a ring is a set of points. Now let's define the ith ring as the ring of the array with the points from the first $i-1$ rings removed. All points belong to a ring, and all rings are disjoint.

I claim that removing any 2 points from the array will always guarantee that the convex hull of whats left will only be made out of points from the first three rings.

also ch is short for convex hull.

By default the convex hull without any points removed only uses points from the first ring (well it is the first ring). To make the convex hull use points aside of the first ring, we have to remove one of the points from the first ring (I hope it make sense that removing points not on the first ring won't change the convex hull). For a similar reason, you have to remove the first point from the first ring and the second point from the second ring to give a chance of a point showing up on the third ring. So consider the following 4 cases for queries.

1. neither a or b is on the first ring. Output the convex hull area of the entire array
2. a and b are both on the first ring. The answer will only use points from the first and second rings
3. a is on the first ring and b is on the second ring. The answer will only use points from the first three points at worse
4. a is on the first ring and b is not on the first or second rings, the answer will only use

points from the first and second rings, this is similar to case 2.

So you can just precompute the answers for

case 1: just find the ch of the entire array and store it

cases 2 and 3: for all pairs of points a and b such that a is the on the first ring and b is on either the first or second ring, find and store the ch without points a and b.

case 4: find the ch for all cases a is on the first ring, and you don't have to worry about b.

The total complexity is $O(n \log n + \log^3 n \cdot \log \log n + q \log n)$. $n \log n$ from finding the first three rings and the ch of the entire array, $\log^3 n \cdot \log \log n$ since there are $\log n \cdot \log n$ ways to pick one point from the first ring and one from the first/second rings and $\log n \cdot \log \log n$ time to find the ch, and $q \log n$ for the queries (binary search to find whether or not a point is in a ring), q is doable. Colin Galen also noted that with the three rings idea, you can just find the ch each time for a total of q times, and since you dont have to sort the points repeatedly, $O(q \log n)$ is doable.

The implementation is very kactl-heavy. I would not have set this problem if copying code was not allowed :).