1. Basic description of algorithm (just bullet phrases)
2. Mostly complete description (steps with some detail, maybe few missing elements)
3. Complete description (sufficient to completely understand what the algorithm does)
4. Correctness attempt (high-level or hand-wavy justification)
5. Complete correctness proof (convincing)
6. Has a complexity
7. Has a justified complexity
8. Has optimal complexity

DISTRIBUTION
30 27 24 24 24 23 21 21 20 20 19 19 18 16
Claim: A separating line for red and blue points exists if and only if their convex hulls do not intersect.

Pf: WLOG Assume their convex hulls intersect and a red point lies in the blue convex hull.

Then any line with all blue points to one side contains a red point as well.

If their convex hulls do not intersect, then these two convex polygons have 4 lines of support, 2 of which separate the polygons. The separating lines of support are separating lines.

Algorithm: +1 approach

1. Compute convex hulls \(CH(R), CH(B)\) of points. \(\Theta(n \log n)\)

2. For each red and each blue point, check whether it is contained in the other set's convex hull. Since \(CH(R), CH(B)\) are monotone, each check takes \(O(\log n)\) time.

3. If an intersection exists, return "this point is in the convex hull of the other set".

4. If no intersection exists, find a separating pair of points between the two convex hulls by walking along the boundaries of the hulls. This is linear in the size of the convex hulls.

5. Return a supporting line that divides the convex \(O(1)\) hulls. +1 finds separating line.
Claim: A separating line for red and blue points exists if and only if their convex hulls do not intersect.

Proof: WLOG Assume their convex hulls intersect and a red point lies in the blue convex hull.

Then any line with all blue points to one side contains a red point as well.

If their convex hulls do not intersect, then these two convex polygons have $4$ lines of support, $2$ of which separate the polygons. The supporting lines of support are supporting lines.

Algorithm:  

1. Compute convex hulls $\text{CH}(R), \text{CH}(B)$ of points. $O(n \log n)$

2. For each red and each blue point, check whether it is contained in the other set's convex hull. Since $\text{CH}(R), \text{CH}(B)$ are monotone, each check takes $O(\log n)$ time.

3. If an intersection exists, return "this point is in the convex hull of the other set". $O(n)$

4. If no intersection exists, find a supporting pair of points $O(n)$ between the two convex hulls by walking along the boundaries of the hulls. This is linear in the size of the convex hulls.

5. Return the separating line that divides the convex hulls. $\star \star$ finds separating line.

6. Running Time $O(n \log n)$
#3. Given a set $R$ of $n$ points colored red and $B$, a set of $n$ points colored blue, the goal is to find a line $l$ that keeps most of $R$ above and most of $B$ below, minimizing the sum of the number of red pts below and the number of blue pts above.

Suppose we have found one such line $l$. What do we know to be true? Let $R_2 = \{x \in R \mid x \text{ is above } l\}$ and let $B_2 = \{x \in B \mid x \text{ is below } l\}$. Then $l$ separates $\text{conv}(R_2)$ and $\text{conv}(B_2)$.

But then we can rotate $l$ to a line $l^*$ such that $l^*$ contains one point of $R_2$ and one point of $B_2$ but $l$ and $l^*$ still have the same value of the sum of the number of pts on the wrong side.

Therefore we can reframe this question into asking for a line through a red point and a blue point with the smallest sum of the number of red pts below and blue pts above.
If we look at the dual of this problem, then we are looking for a point where a red line intersects a blue line and the sum of the number of blue lines above and red lines below is as small as possible.

What if, as in problem #2, it were possible to find a red-blue intersection pt which completely separates the red and blue lines. Then we would be looking for when the lower envelope of the red lines intersects the upper envelope of the blue lines.

But in the case where these two envelopes do not intersect, we have the situation described in problem #3.

So let's move to the solution for problem #3, constructed in the dual, where we have an arrangement of n red and n blue lines which we sort by slope in \(\Theta(n \log n)\) time.
As we prepare to begin the sweep of the arrangement, the first "cut" intersects the \(2n\) lines in decreasing order of slope. In linear time, we can walk up the cut and annotate each cut-edge with the number of red lines that lie below it. Similarly, we can walk down the cut in linear time and annotate each cut-edge with the number of blue lines that lie above it.

As the sweep processes a red-red intersection pt or a blue-blue intersection pt, two red or two blue lines exchange position, but there will be no changes in the "counts".

So all we need to do is to detail the processing that occurs at red-blue or blue-red intersection pts (where the first color belongs to the line coming from the top and the second color to the line that comes from below).

**Case I**

- The number of blue lines above \(c'_{[i]}\) is one greater than the number of blue lines above \(c_{[i]}\).

**After processing pt \(p\), things get worse:**

- The number of red lines below \(c_{[i+1]}\) is one greater than the number of red lines below \(c_{[i]}\).

The value of \(p\) itself is the sum of the number of red lines below \(c_{[i]}\) and the number of blue lines above \(c_{[i+1]}\). If best, save
Case II

After processing $p$, things get better

the number of red lines below $c'[i+1]$ is one
less than the number of red lines below $c[i+1]$

The number of blue lines above $c'[i]$ is one less than the
number of blue lines above $c[i]$.

The value of $u_p$ itself is the sum of the number
of red lines below $(c'[i])$ and the number of blue lines
above $(c'[i+1])$. If best so far, save,

We need to sweep across the entire arrangement,
but since all of our calculations are local,
we do not need vertical line sweep, but can
use topological line sweep instead.

Consequently the time complexity is $\Theta(n^2)$. The space is $\Theta(n)$.
And at the end, we report the best $u_p=(x_p, y_p)$
or rather, the line $y=(-x_p)x + (y_p)$. 