1. You are given two sets $A$ and $B$ of points in the plane, both of size $n$. Recall that a Ham-Sandwich cut is a line that bisects both point sets, such that there are at least $\lfloor n/2 \rfloor$ points of $A$ and of $B$ on each side of the cut. We define a connection between $A$ and $B$ as a set of $n$ line segments, each connecting a point of $A$ to some point of $B$ (i.e., the segments define a one-to-one correspondence between $A$ and $B$). The connection is disjoint if no two segments intersect.

Assuming you are given an algorithm for computing the Ham-Sandwich cut in $O(n\log n)$ time. Give an $O(n\log^2 n)$-time algorithm that outputs a disjoint connection between $A$ and $B$.

**Solution:**

```
ComputeDisjointConnection(A,B)
    Compute Ham-Sandwich Cut, C
    if |A| is odd, this will go through
    a point of A and a point of B,
    output the segment connecting those points.
    if |A| = 1, then done.
    else
        Partition A,B into point left of C
        and points right of C
        output ComputeDisjointConnection(Aleft, Bleft)
        output ComputeDisjointConnection(Aright, Bright)
```

**Runtime:**

If the number of points is a power of 2, then there are a total of $n\log n$ calls to `ComputeDisjointConnection`. Everything inside this is linear (partitioning points), so the key is to count how long the ham-sandwhich cuts cost.

We have

1 call that takes $n\log n$ = $n\log n$
2 calls that take $n/2 \log n/2 = 2 * n/2 \log n/2 < n \log n$
4 calls that take $n/4 \log n/4 = 4 * n/4 \log n/4 < n \log n$

There are $\log n$ levels of these calls, so total time is:

$$\log n * n \log n = n * (\log n)^2.$$  

**Correctness:**

We need to prove two things.
1. all points are matched to a point of the other set. This is true because the recursive call is always made with exactly the same number of points, and always partitions the points into equal sets. When —A— is 1, there is 1 point from A and one point from B and these points are matched (and these points aren’t matched to anyone else because they aren’t included in any other recursive call).

2. No segments intersect. Because all segments that are drawn follow the ham-sandwhich cut, the only way that something could intersect that segment is if points from each side of that cut are connected. BUT each recursive call is limited to points on one side of the cut, so this is not possible.

2. Let \( L \) denote a set of \( n \) lines in the plane and let \( A(L) \) denote their arrangement. Let \( S \) be the set of vertices in \( A(L) \). Give an \( O(n \log n) \) time algorithm that computes \( CH(S) \), and prove that it is correct. Note that \( S \) has size \( O(n^2) \) because it is the intersections between every pair of \( n \)-lines, so your algorithm does not have time to explicitly compute all elements of \( S \). The figure below shows an arrangement of 6 lines, the 15 intersection points and the convex hull that you need to compute. Note that points may not be in general position, but you only need to return the extreme points along an edge.

![Image](image.png)

**Algorithm:** Sort the lines by slope. Compute intersections between lines of neighboring slope. Compute the convex hull of those points with reasonable algorithm (e.g. mergehull).

**Runtime:** Sorting is \( O(n \log n) \). Intersections between neighbors is \( O(n) \), convex hull is \( O(n \log n) \).

**Proof:** The algorithm only considers intersections between lines with neighboring slopes. The correctness proof needs to argue that these are the only points that may be on the convex hull.

Let a, b be two lines that are *not* neighboring in slope. We are that their intersection point is not part of the convex hull.

If a,b are not neighboring in slope, there must be a pair of lines (drawn with dashes in the figure below) that are between them in slope. These lines intersect lines a,b also, and all possible ways to draw those lines have intersection points that are on either side of the intersection of a,b. In the case of the figure below, these intersections are on the ”a

3. Consider a collection of \( n \) points \( P \) in the plane. Define a 3-slab to be the region bounded by a pair of non-vertical parallel lines, such that there are at least 3 points in the region (including on the lines). Define the height of a 3-slab to be the vertical distance between its two lines (h in the picture below). Present an \( O(n^2) \)-time algorithm which computes a 3-slab of minimum height.

**Algorithm** Construct the line arrangement defined by the dual of all points. For each vertex of the line arrangement, walk around the face that is immediately above and below and fine the first line that is hit but a vertical bullet path. The intersection point, and the intersection point of the bullet path and the neighboring line are duals to a pair of parallel lines in the original plane. Compute the distance between these two lines, and keep the smallest one.

**Runtime:** Computing the line arrangement is \( O(n^2) \). Now we need to visit every vertex and find the line immediately above and below that line. There are three options:
First, in $O(n^2 \log n)$ time, we could do a plane sweep algorithm that starts by sorting lines by slope, and sorts all $O(n^2)$ events by their x-coordinate, and then keeps track of the sorted order of lines. At each event, you can loop up the lines above and below the current event point, and swap the lines that intersect at that event point. Both of those can be done in $O(\log n)$ time, and there are $O(n^2)$ events for a total of $O(n^2 \log n)$ time.

There is a trick called a "topological plane sweep" that we didn’t talk about that shaves off the extra $\log n$ to give an $O(n^2)$ algorithm.

Second, we can use the DCEL or other representation of the line arrangement, and each time we visit a vertex, we "walk around" the face above that vertex until we find the line that intersects a vertical bullet path. It is easy to create line arrangement configurations where sometimes this "walking around" may take $O(n)$ time, so it is important to bound the total time this may take.

In class we talked about the "Zone Theorem". We proved that incrementally constructing the line arrangement takes $O(n)$ time per line, and we did that by carefully counting all the edges that are part of a face that is touched by a line. The Zone Theorem explicitly states that the complexity of the region of the arrangement that touches one line is $O(n)$. Therefore the time spent "walking around" the faces to find the nearest line above and below a point is $O(n)$ for all the points that arise from one line. Since there are $n$ lines, the total time is $O(n^2)$.

**Correctness:** By searching points that are at the intersection of dual lines, and then looking for the nearest line above and below that, we are assuming:

(a) the thinnest 3-slab must go through a pair of points and touch a 3rd point, and
(b) must not include a fourth point in between.

Point b is obvious (because we could have a narrower slab by just going to that point). Why is it the case that one of the lines must go through 2 points? Suppose that we have the narrowest 3slab, each line goes through one point and there is a third point in the middle. It is always possible to rotate both lines (by the same amount) in such a way to make them narrower — at the limit the two lines converge to one line that goes through the two points. Therefore it is always possible to make a narrower 3slab by rotating these lines until the third point is on the verge of escaping the slab; at that point one of the lines is going through 2 points.