We create a modified segment tree. Instead of keeping a list of segments at each node, that cover the interval of that node, we keep those same segments in a BST, ordered by y coordinate (an BST at every node of the segment tree).

Space: each segment appears in O(log n) nodes' BSTs, and those BSTs use space that is linear in the number of segments they contain. 
\[= O(n \log n) \text{ total space.} \]

Preproc.: each segment is inserted into O(log n) BSTs, at O(log n) per insertion. 
\[= O(n \log n) \text{ total preproc. time.} \]

Query: Given the query \( q = (x, [ly, hy]) \):
Go down from root to leaf in the segment tree, according to \( q, x \).
At every node, find \( y \) and \( p, y \) in the BST, and print all the nodes in between.
\[O(n \log n) \text{ nodes, and } O(\log n) \text{ at every node, not counting the time to print the hit segments.} \]
\[= O(\log^2 n + A) \text{ total query time.} \]

I still believe there should be a \( O(\log n + A) \) solution, but I can't see it.
b) Credits: I packed at the lecture notes.

Given \( S \subseteq \mathbb{R}^2, |S| = n \), for a query \( q = [e_x, h_x] \times [e_y, h_y] \) report \( S \cap q \).

Make a BST of all the points in \( S \), by y coord, call it \( T \).

For each \( v \in T \), let \( S_{\text{below}} \) = \{points stored in left subtree of \( v \)\} (i.e. below \( v \))

\( S_{\text{above}} \) = \{points stored in right subtree of \( v \)\} (i.e. above \( v \))

and at each node \( v \), keep \( T_{\text{below}} \) = treap for \( S_{\text{below}} \), answering \( \uparrow \uparrow \) queries;

\( T_{\text{above}} \) = treap for \( S_{\text{above}} \), answering \( \downarrow \downarrow \) queries.

Space: each point in \( S \) occurs in some treap at every level of \( T \)

\( \Rightarrow \) \( O(n \log n) \)

Preproc: sort the points, build the BST, then build the treaps in linear time (each)

since the points are already sorted,

\( \Rightarrow \) \( O(n \log n) \)

Query: start at the root.

At node \( v \):

if \( v.y < q.y \), recurse into \( v \), right

if \( v.y > q.y \), recurse into \( v \), left

otherwise use \( T_{\text{below}} \) and \( T_{\text{above}} \) at \( v \) to enumerate \( S \cap q \):

\( \begin{array}{c}
\text{\( q \)} \\
\hline
\text{\( q \)} \\
\hline
\text{\( q \)} \\
\hline
\text{\( q \)} \\
\hline
\text{\( q \)} \\
\hline
\end{array} \)

\( \Rightarrow \) \( 0(\log n) \) to find the node whose \( v.y \) crosses \( q \), then \( 2 \times O(\log n + A) \) to report the pts.

\( \Rightarrow \) \( O(\log n + A) \) total query time.
We sweep from left to right, keeping track of the number of polygons that cover the area immediately above a cut edge. When this number is 2, we know that area is part of the intersection. All intersection regions are thus traced by the sweep line, and we can represent the result as a DCEL or a list of polygons.

Space: O(n) if we only keep the upcoming intersection points of adjacent cut edges only

Time: If the given polygons have O(n) vertices each, there can be as many as O(n^2) intersection points. We spend O(logn) time at each of these, having to update the heap of event points.

=> total time O(n^2 logn).

Since the changes at an event point are local, we should be able to use topological sweep to bring this down to O(n^3).
We normalize y coordinates and build a segment tree with all elementary segments (segments between two adjacent normalized y values). At each node we keep 2 booleans: $CP_i$ (covered by $P_i$) and $CP_j$ (covered by $P_j$). We sweep a vertical line from left to right, stopping at every normalized x coordinate. If we stop at a left vertical edge, we set some $CP$ bits, if it's a right vertical edge, we clear some $CP$ bits. We only have 2 polygons, so we don't bother with PROMOTE and DEMOTE.

At each stopping point we save the vertical edges of the contour of $P_i \cap P_j$, which are segments with both $CP_i$ and $CP_j$ set (here or by an ancestor) to true.

After the sweep, we have all vertical edges of the intersection contour. We find the horizontal edges in linear time just like for the contour-of-union problem.

The sweep takes $O(n \log n)$ to sort the event points, and the time spent to update the segment tree and contour at each event point can be charged to the complexity of the resulting intersection.

Therefore the total time is $O(n \log n + p)$ just like Wood's algorithm.

# points in answer