Darts

The problem basically is the following:

You are given a list P of n 2D coordinates where $p[i] = (x[i], y[i])$ for $i \in \{1 \dots n\}$ and for each query $q = (x, y)$, you have to answer if there exists an $j \in \{1 \dots n\}$, such that $x[j] \geq x$ and $y[j] \geq y$, and exhibit such a pair if it exists.

We will solve this in the following manner.

First sort array P to P' in increasing order with x coordinate as the primary key and y coordinate as the secondary key. This takes $O(n \log n)$ time. Call the resultant lists of x and y coordinates x' and y' respectively.

Now create another array Q of same length as P, where for $i \in \{1 \dots n\}$, we have $Q[i] = \max_{j=i}^{n} y'[j]$. This step can be done in one go in a reverse iteration on P'. Thus, it takes time $O(n)$.

Now, let's consider a query $q = (x, y)$.

Claim 1: If there is an answer to the problem, it can't lie to the left of the leftmost index which has $x'[k] \geq x$.

Proof: This is fairly straightforward since x' is a sorted array.

Hence, if we use binary search to find the leftmost index k such that $x'[k]$ is greater than or equal to x, then if there exists no such index, the dart misses the board.

Now suppose that we have found such an index.

If $Q[k] < y$, the dart misses the board as it implies that there must be no such element on the right side of index k in P' as well which has a y-coordinate at least equal to y.

If $Q[k] \geq y$, there must be an element $p'[t]$ where $t \in \{k \dots n\}$ such that $x'[t] \geq x'[k] \geq x$ (sorted order and definition of $x'[k]$) and $y'[t] = Q[k] \geq y$ (definition of $Q[k]$, i.e. $t = \texttt{argmax}_{j=k}^{n} y'[j]$).

To find this element, we can find the rightmost index j' such that $Q[j'] \geq y$, since any index j'' to the right will satisfy $Q[j''] < y$, and thus $y \leq Q[j'] = y'[j']$. To find this element, note that by definition of Q, it is sorted in non-increasing order. Hence we can do a binary search to find this element.

Thus, the time taken per query is the same as that of doing binary search i.e. $O(\log n)$.

Total time complexity is $O(n \log n + q \log n)$

A solution that sorts queries and uses two pointers can also be found.

Another solution goes as follows:

We will first focus our attention on the defining sheets first, i.e., pairs (x, y) such that there is no other (x', y') in the queries satisfying $x \geq x', y \geq y'$. This is valid because if there is a sheet S which is punctured by a dart, we can find a defining sheet which covers S (since otherwise there is a contradiction), and hence that is a valid solution too. If there is no solution, well, there is no solution in this reduced set either.

Let's suppose we have actually computed the set of these defining sheets. Now we claim that firstly, no two defining sheets have the same x or y coordinate (which is true since you have one of them covering the other one), and if we sort them by their x coordinates, they will be sorted in the reverse order by their y coordinate.

To see why the second claim is true, suppose there exist two defining sheets with (x, y) and (x', y') such that x comes before x' but $y < y'$. This is a clear contradiction by the definition of defining sheets.

Hence we can simply try to binary search for these sheets according to their $x-$coordinate, and find the least such x in the reduced set of coordinates, and the $y-$coordinate will be the maximum possible. Hence either this sheet is punctured by a dart, or no sheet is.

Now we tackle the problem of how to find this defining set of sheets in the whole set of sheets. We do the following:

Sort the sheets by x coordinate first. Now do the following recursive algorithm.

Split the sheets into two equal halves, and compute the answer to both halves. To merge these two, we take the largest y coordinate on the right half, and to the right half, append all the pairs in the left half which have a strictly greater y coordinate, and return the modified right half. It is easy to see why this is necessary and sufficient using induction.

Note that this algorithm takes $O(n \log n)$ time for preprocessing, then we have a recursive algorithm that satisfies $T(n) = 2T(n/2) + O(n)$, which gives $T(n) = O(n \log n)$. And for each query, we do a binary search, which takes $O(\log S)$ time, where S is the size of the set of the defining sheets.

Hence the total time taken for this algorithm would be $O(n \log n + q \log S)$ which is slightly better than $O(n \log n + q \log n)$.