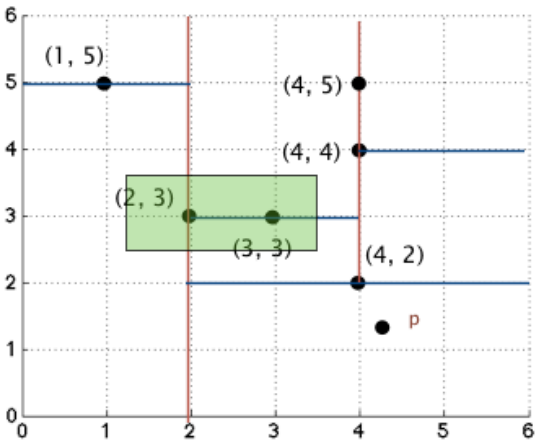
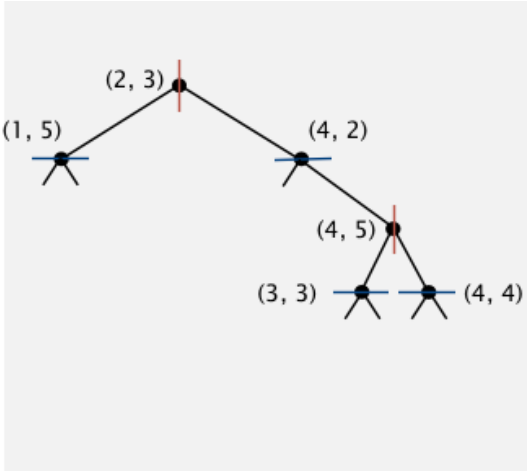
1.

a.




b. Corresponding rectangles:

| | |
|---|---|
| $(2,3)$ | $[(-\infty,-\infty),(+\infty,+\infty)]$ |
| $(1,5)$ | $[(-\infty,-\infty),(2,\infty)]$ |
| $(4,2)$ | $[(2,-\infty),(+\infty,+\infty)]$ |
| $(4,5)$ | $[(2,2),(+\infty,+\infty)]$ |
| $(3,3)$ | $[(2,2),(4,+\infty)]$ |
| $(4,4)$ | $[(4,2),(+\infty,+\infty)]$ |

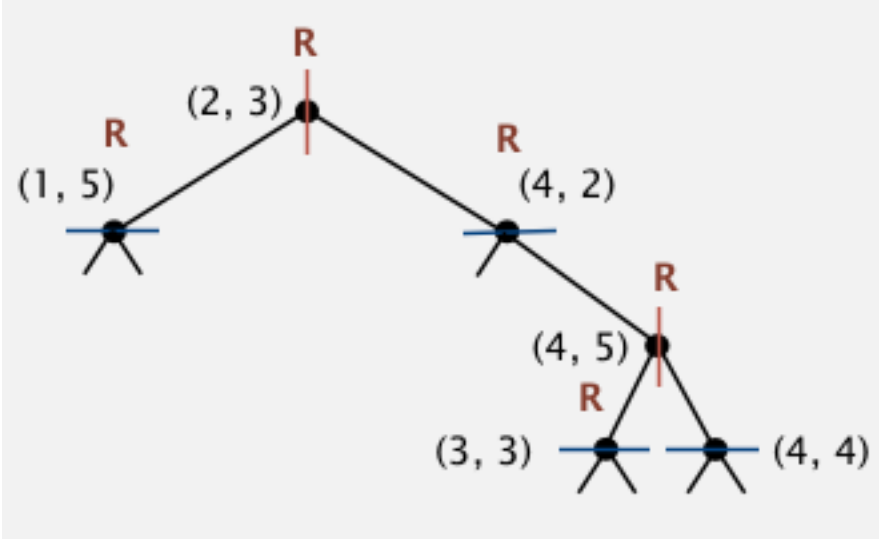
This might get a little tedious, so maybe it's better to simply show them (4, 5) by example, then have them find (3, 3) on their own.

A nice way to think about (3, 3) for example, is that it is:

i. To the right of (**2**, 3)
ii. Above (4, **2**)
iii. To the left of (**4**, 5)

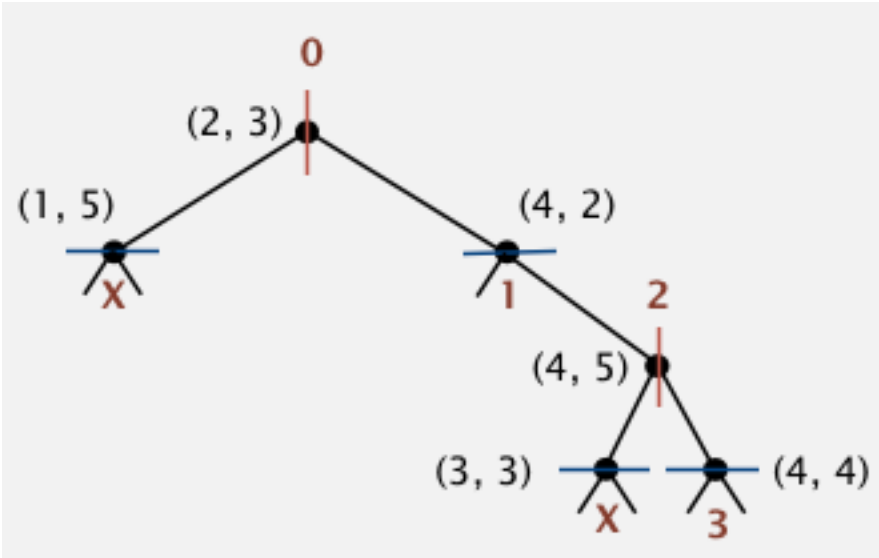
Naturally, that means its corresponding rectangle should be $[(2,2),(4,+\infty)]$

c.




Answer to the mini-question: The corresponding rectangle for (4, 4) rules out the possibility of needing to check (4, 4) or any of its children. In other words, the green rectangle does not intersect $[(4, 2), (+\infty, +\infty)]$

d.




An important point to make here is that the way that we know that (3, 3) and all of its are farther from the query point is that (3, 3)'s corresponding rectangle is farther

from the query point than our champion point (4, 2). Yes, RectHV is smart enough to find distances to rectangles with, for example, infinite far away left edges.

The same idea holds true for (1, 5)'s corresponding rectangle, whose distance to p is greater than (4, 2) to p.

Answer to problem 2. If the current Node is null, abort the search. If the current node's rectangle does not intersect the query rectangle, abort the search.

**Detailed algorithm trace for range search:**

1. (2, 3): Base case: Check to see if (2, 3)'s corresponding rectangle intersects the green rectangle. Since the (2, 3)'s rectangle is the entire universe, of course it intersects.

Check to see if (2, 3) is contained in the rectangle. It is, so add it to the iterable.

Go left first (order doesn't matter). We reach (1,5).

2. (1, 5): Base case: Check to see if (1, 5)'s corresponding rectangle intersects the green rectangle. Yes, the left half of the bisected plane does indeed intersect, so we proceed. We see if (1, 5) is contained in the green rectangle. It is not. We recur left.

3. Null to the left of (1, 5): Base case detects null, returns to (1, 5). This repeats for the right child.

4. (4, 2): Corresponding rectangle intersects, so proceed. Point is not part of rectangle. Recur left, is null, return. Recur right.

5. (4, 5): Rectangle intersects, so proceed. Point is not part of rectangle. Recur left.

6. (3, 3): Rectangle intersects, so proceed. Point is part of rectangle, so add to iterable. Left and right recursive calls result in nulls. Return back to (4, 5), recur right.

7. (4, 4): Rectangle for (4, 4) does NOT intersect green rectangle, so abort search. Return to (4, 5). Return to (4, 2). Return to (2, 3). We are done.

**Detailed algorithm trace for nearest neighbor:**

Champion point starts as (2, 3). Best distance is also recorded for convenience. Program begins.

1. (2, 3): We compute the distance between (2, 3)'s rectangle and the query point, which is 0 since it is contained in the rectangle. 0 is closer than the best distance, so we do not prune.

Now we must decide whether to go left or right. We compare (2, 3)'s x coordinate with p's x coordinate. Since p is to the right, we go right first.

2. (4, 2) We compute the distance between (4, 2)'s rectangle and the query point, which is again 0. 0 is less than the champion distance, so we do not prune. (4, 2) is closer to the query point than (2, 3), so we make (4, 2) the new champion.

We decide to go either up or down. Since p is below (4, 2), we go down first.

3. null to the left of (4, 2): Since this is null, we just return. Then we go up from (4, 2).

4. (4, 5): The distance between (4, 5)'s rectangle and the query point is something slightly less than 1. This distance is clearly a bit less than the champion distance, so we do not prune.

We compare (4, 5)'s distance to p with the champion distance, and it is farther, so (4, 2) remains the champion.

We then decide to go left or right. Since p's x coordinate is greater than (4, 5), we go right first.

5. (4, 4). The distance from (4, 4)'s rectangle is exactly the same as the previous iteration, just a bit less than 1. This distance is less than the champion distance, so we do not prune.

(4, 4)'s distance is farther than the champion distance, so (4, 2) remains the champion.

We then decide whether to go down or up. We go down first, see null. Then go up. See null. Then return to (4, 5). Then we go left to (3, 3).

6. (3, 3). This point's corresponding rectangle is the same distance from the query point than the current champion. Indeed, the champion (4, 2) is exactly the point defining the corresponding rectangle. Thus, we prune this search, since there's no way we can do better.

We return to (4, 5), then to (4, 2), and finally back to (2, 3). From there, we go left to (1, 5).

7. (1, 5). This point's corresponding rectangle is the left half of the plane, with distance of a bit more than 2 from the query point. Since this is greater than the champion distance, we prune the search. We return to (2, 3) and we are done.