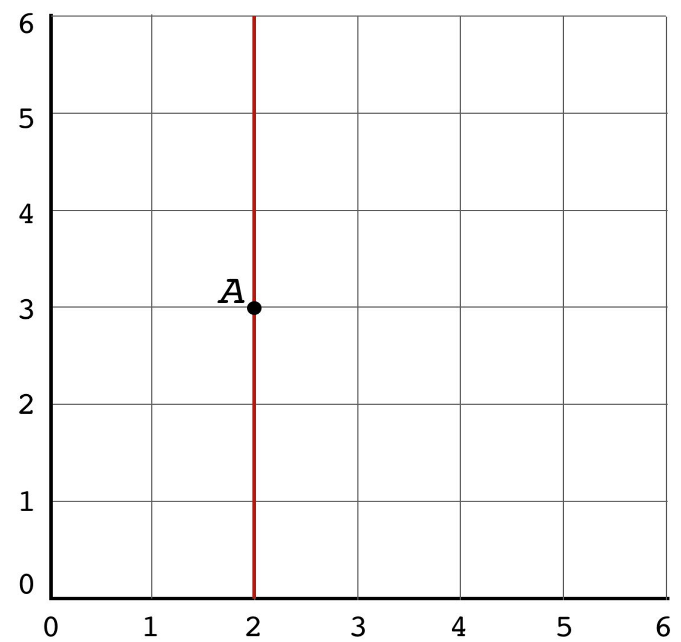
EXERCISE 1: Kd-Trees

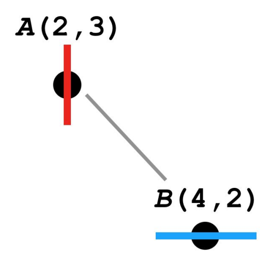
(a) Draw the Kd-tree that results from inserting the following points:

[*A*(2, 3), *B*(4, 2), *C*(4, 5), *D*(3, 3), *E*(1, 5), *F*(4, 4), *G*(1, 1)]

Draw each point on the grid, as well as the vertical or horizontal line that runs through the point and partitions the plane, or a subregion of it.

Note: While inserting, go left if the coordinate of the inserted point is less than the coordinate of the current node. Go right if it is greater than ***or equal***.



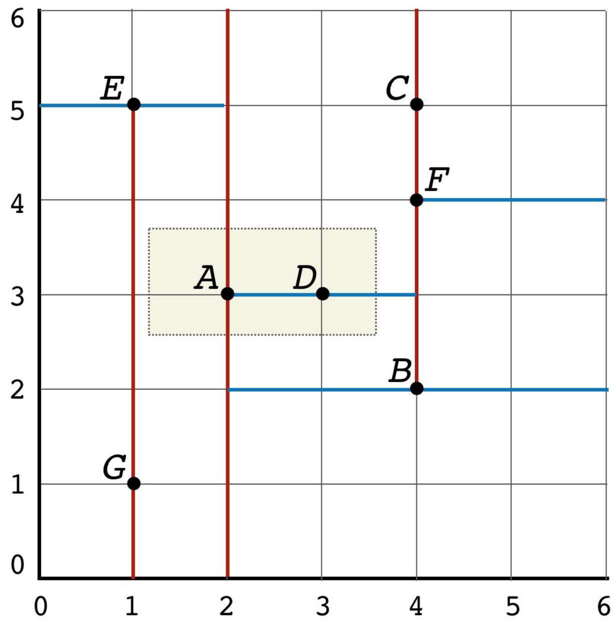


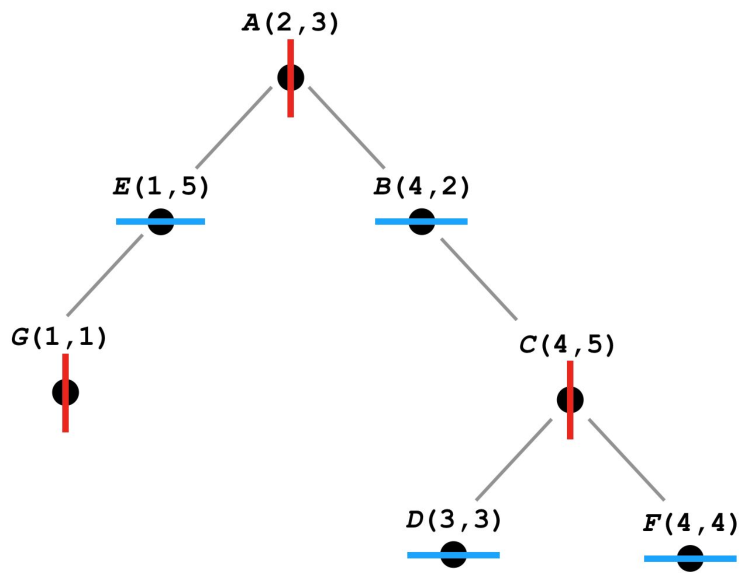
(b) Give each point's bounding rectangle.

A(2, 3)	$[(-\infty, -\infty), (+\infty, +\infty)]$
B(4, 2)	
C(4, 5)	
D(3, 3)	
F(4, 4)	$[(4, 2), (+\infty, +\infty)]$
E(1, 5)	$[(-\infty, -\infty), (2, +\infty)]$
G(1, 1)	

(c) Number the tree nodes according to the visiting order when performing a *range query* using the rectangle shown below. Label pruned subtrees with ✘.

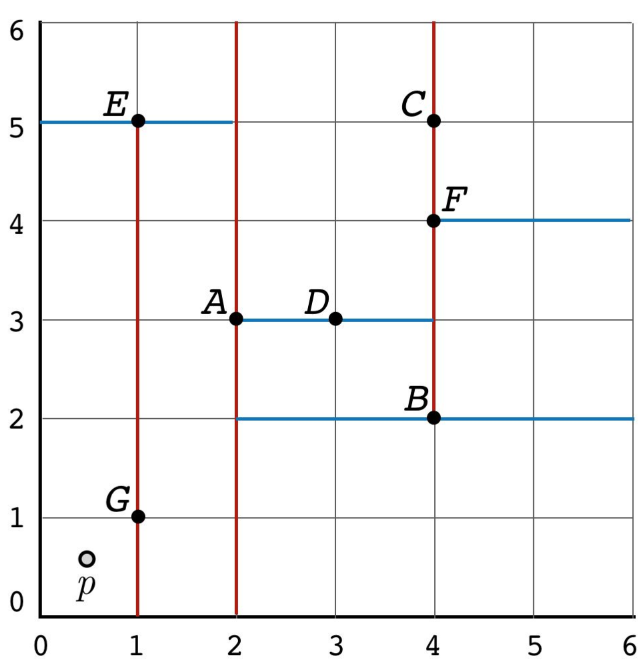
Remember. The range search algorithm recursively searches in both the left and right subtrees unless the bounding rectangle of the *current* node does not intersect the query rectangle.

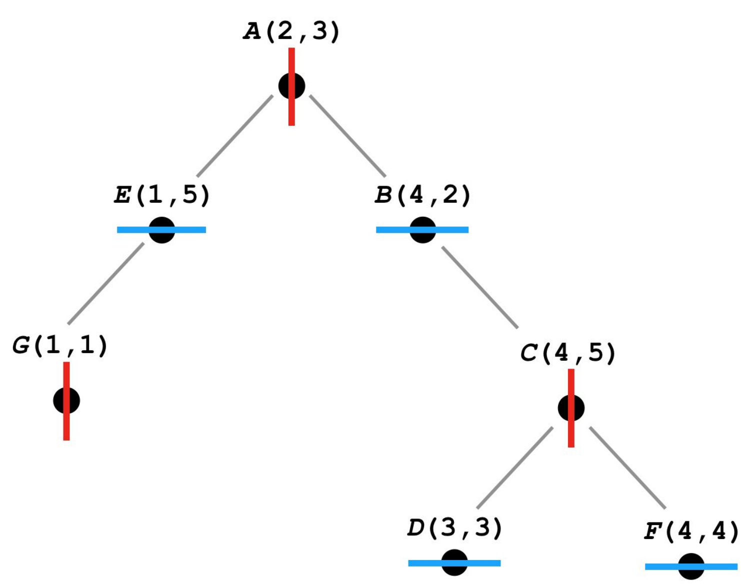




(d) Number the tree nodes according to the visiting order when performing a *nearest neighbor (NN) query* using the point p shown below. Label pruned subtrees with ✘.

Remember. The NN algorithm recursively searches in *both* the left and right subtrees unless the distance between *p* and the bounding rectangle of the *current* node is not less than the distance between *p* and the nearest point found so far.





EXERCISE 2: Operations on Binary Trees

Consider the following Binary Tree class for storing integers.

```java
public class BinaryTree {
      private Node root;
      private class Node {
            private int key;
            private Node left, right;

            private int size;       // # of nodes in subtree rooted here
            private int height;     // maximum number of links between the node
                                    // and any of its children

            public Node(int key, int size) {
                  this.key = key;
                  this.size = size;
            }
      }

      private int size(Node x) { /* returns x.size or 0 if x is null. */ }
      private boolean contains(int key) { /* checks if key is in the tree. */ }
      // ... other public and private methods
}
```

(a) Assume that elements in `BinaryTree` are ordered such that the tree is a *Binary Search Tree* (BST). Implement method `rank`, which returns the number of keys in the BST that are strictly less than the given key.

```java
public int rank(int key) {
     // use the recursive private helper method rank(Key key, Node x)


}

private int rank(int key, Node x) {











}
```

(b) Assume that `BinaryTree` is ***not*** necessarily a *Binary Search Tree*. Implement method `isHeapOrdered` to check if the tree is *heap-ordered* as a Max-Heap, i.e. every node in the tree is not less than its children.

Note. checking if a binary tree is a valid Max-Heap requires also checking if every level is full, except the last level, which could be partially filled left-to-right. In this exercise check *only* if the tree is *heap-ordered*.

```
private boolean isHeapOrdered(Node x) {
      if (x == null) return true;












}
```

Extra (Optional) Exercises:

(a) Assuming `BinaryTree` is a BST, implement `int rangeCount(int lo, int hi)`. This method should return the number of keys in the BST that are between `lo` and `hi` (inclusive).

HINT: Use method `rank`!

(b) Implement the method `boolean allLevelsFull(Node x)`, which returns `true` if all levels in the tree are full.

HINT: This method can be implemented in *constant* time!

(c) Assuming `BinaryTree` is a BST, implement method `Node select(int k)` which returns the node in the tree with the key of rank *k*.

HINT: See pages 406-409 in the textbook.

(d) Implement the method `boolean isComplete(Node x)`, which returns `true` if all levels in the tree are full except the last level, which could be filled from left to right.

HINT: Use Breadth-First Traversal!