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 \([(x_{min}, y_{min}), (x_{max}, y_{max})]\).

<table>
<thead>
<tr>
<th>Point</th>
<th>Bounding Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(2, 3)</td>
<td>([-\infty, -\infty), (+\infty, +\infty)] ]</td>
</tr>
<tr>
<td>B(4, 2)</td>
<td>[ ]</td>
</tr>
<tr>
<td>C(4, 5)</td>
<td>[ ]</td>
</tr>
<tr>
<td>D(3, 3)</td>
<td>[ ]</td>
</tr>
<tr>
<td>F(4, 4)</td>
<td>[ 4, 2), (+\infty, +\infty)] ]</td>
</tr>
<tr>
<td>E(1, 5)</td>
<td>[ -\infty, -\infty), (2, +\infty)] ]</td>
</tr>
<tr>
<td>G(1, 1)</td>
<td>[ ]</td>
</tr>
</tbody>
</table>
(c) Number the tree nodes according to the visiting order when performing a range query using the rectangle shown below. Label pruned subtrees with $\times$.

**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 $p$ and the Kd-Tree shown below. Label pruned subtrees with $\times$.

**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 Search Tree class for storing integers.

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

        private int size; // # of nodes in subtree rooted here

        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. */ }
    // ... other public and private methods
}
```

(a) What modifications should be made to the `put()` method to update the subtree counts?

```java
public void put(Key key) {
    root = put(root, key);
}
private Node put(Node x, Key key) {
    if (x == null) return new Node(key, 1);
    if (key < x.key) x.left = put(x.left, key);
    else if (key > x.key) x.right = put(x.right, key);

    return x;
}
```

(b) Assume the tree is a left-leaning red-black BST. Is the modification you have introduced to `put()` in part (a) enough to maintain the subtree counts? Why?
(c) 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) {
    return rank(key, root);
}
private int rank(int key, Node x) {
}
```

**Extra (Optional) Exercise:**

(d) 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).

```java
// Returns the number of keys in the symbol table in the given range.
public int rangeCount(int lo, int hi)
```

**More Ordered Operations on BSTs:**

Implement method `Node select(int rank)`, which returns the node in the tree with the key of a given rank.

(See pages 406-409 in the textbook)

```java
private Node select(Node x, int rank) {
    if (x == null)
        return null;

    int leftSize = size(x.left);

    if (leftSize > rank) return select(x.left, rank);
    else if (leftSize < rank) return select(x.right, rank - leftSize - 1);
    else return x;
}
Java References Reminder. While updating the champion in the NN method, many students face bugs that are avoidable if the following is understood:

- If the recursive method returns a value, make sure to catch the return value and use it whenever you make a recursive call.

- If the recursive method receives an argument that needs to be updated, note that the following works:

```java
void myMethod(Type1 arg1, Type2 result) {
    result.setX(someValue);
    ...
}
```

But the following does not work:

```java
void myMethod(Type1 arg1, Type2 result) {
    result = new Type2(someValue);
    ...
}
```

In the first example, `result` is a reference, and the method is changing X in the object referenced by `result`, so the effect will be applied to the object passed to the method as an argument.

In the second example, `result` is a reference that is pointing to the object passed to the method as an argument. By doing `result = . . .`, we make the reference `result` point to a new object other than the one passed as an argument. In other words, the object passed as an argument will not be affected by the operation.