**PA2: Closest Pair Report  
Jed Pulley & Matthew Starr**

**Part 0:**

Here we are sorting our points array along the X axis so that we can then go on to find the median easily. To do this, we use MergeSort plus our PointComparator.

**Part 1:**

We are defining our base case to be when there are <= 3 points. We stop recursing, sort using InsertionSortPointsByY, and then run our brute for on the points. Sorting by Y allows us to work our way from bottom to top.

**Part 2:**

Creating mid here is different than normal since we add left at the end. This is because we’re parsing through the array based on its X value, so adding left offsets it to match where in the array we’re working. Once we get that mid, we find the X value of the point at said mid.

**Part 3:**

These are our recursion statements to find the closest pair on the left and right sides, respectively.

**Part 4:**

Here we are getting the minimum distance of both the left and right sides and then comparing them to find the minimum of those two. This distance is the basis for how we determine the bounds in our cross-section search.

**Part 5:**

Sort points by Y value so that when we search our cross-section, we can start at the bottom and work our way up.

**Part 6:**

The distance between the right and the left portion of this section of the array is the minimum distance from the median X found from searching the left and right sides. This ensures that we can ignore any point outside of these bounds.

**Part 7:**

Here we are brute forcing the search between the section defined in Part 6, however, we only need to check a maximum of seven points (due to the minimum distance requirements) so this search looks like O(7n) and since 7 is a constant, we get O(n).