When deciding on a strategy to test the correctness of our algorithm we had a few key criteria. We wanted to avoid repeating the steps used in our algorithms, we wanted to be able to use double precision so that we could test for correctness with circular data, and we wanted to use some well tested java libraries. These criteria brought us to the Polygon.contains method in the java.awt library, but the contains method only works on integers, and this would disable our ability to correctly test convex hulls that are in the shape of a circle. We then discovered the Path2D library which supports double precision and has a contains method. The challenge with using the Path2D library comes in building the convex hull. Building the hull requires adding points in the correct order so that each point you add draws the correct line as it would look in the given convex hull. In order to solve this problem we had to sort the convex hull in a circular way so that the path could be drawn clockwise around the convex hull.

```
LinkedList<Point2D> subList = new LinkedList<Point2D>();
int currentPosition = 0;

//top of the array
circularArray[currentPosition] = left;
currentPosition++;
for(Point2D current: cHull) {
    if(current.getX() > left.getX() && current.getX() < top.getX()
            && current.getY() > left.getY() && current.getY() < top.getY())
        subList.add(current);
    else if(current.getX() > top.getX() && current.getX() < right.getX()
            && current.getY() < top.getY() && current.getY() > right.getY())
        subList.add(current);
}
if(left != top && right != top)
    subList.add(top);

while(!subList.isEmpty()) {
    Point2D current = findLeftmostPoint(subList);
    subList.remove(current);
    circularArray[currentPosition] = current;
    currentPosition++;
}

circularArray[currentPosition] = right;
currentPosition++;
```

The code above shows the algorithm for building an array that can be used to draw the convex hull clockwise. First we add the left most point of the convex hull in the first position of the circular array. We start with the left, right, top, and bottom points of the convex hull available, and we use those to add the top left and top right portions of the convex hull to a linked list. We then find the left most element of the list and place it into the next available position of the circular array while removing it from the list. We repeat that step until the list is empty, and add the right point to the next available position. A similar strategy is used to add the bottom portion of the convex hull to the circular array, but the loop portion adds the right most point to the array to continue adding points in the clockwise direction.

The code below shows the construction of the convex hull using the Path2D cHull. The path starts with the left most point contained in the first position of the circular array. The rest of the list is iterated

through and lines are drawn to each consecutive point. A line is then drawn from the final point in the array to the first point added using the closePath method. With the convex hull drawn we are able to iterate through all the points and make sure each point is contained in the convex hull.

```
Point2D [] circularArray = getCircularArray(convexHullArray);

cHull.moveTo(circularArray[0].getX(), circularArray[0].getY());
for(int i = 1; i < circularArray.length; i++) {
    cHull.lineTo(circularArray[i].getX(), circularArray[i].getY());
}
cHull.closePath();


for(Point2D current: points) {
    totalCorrectness = totalCorrectness && cHull.contains(current.getX(), current.getY());
    System.out.println(cHull.contains(current.getX(), current.getY())
            + " " + current.getX() + " " + current.getY());
}
```