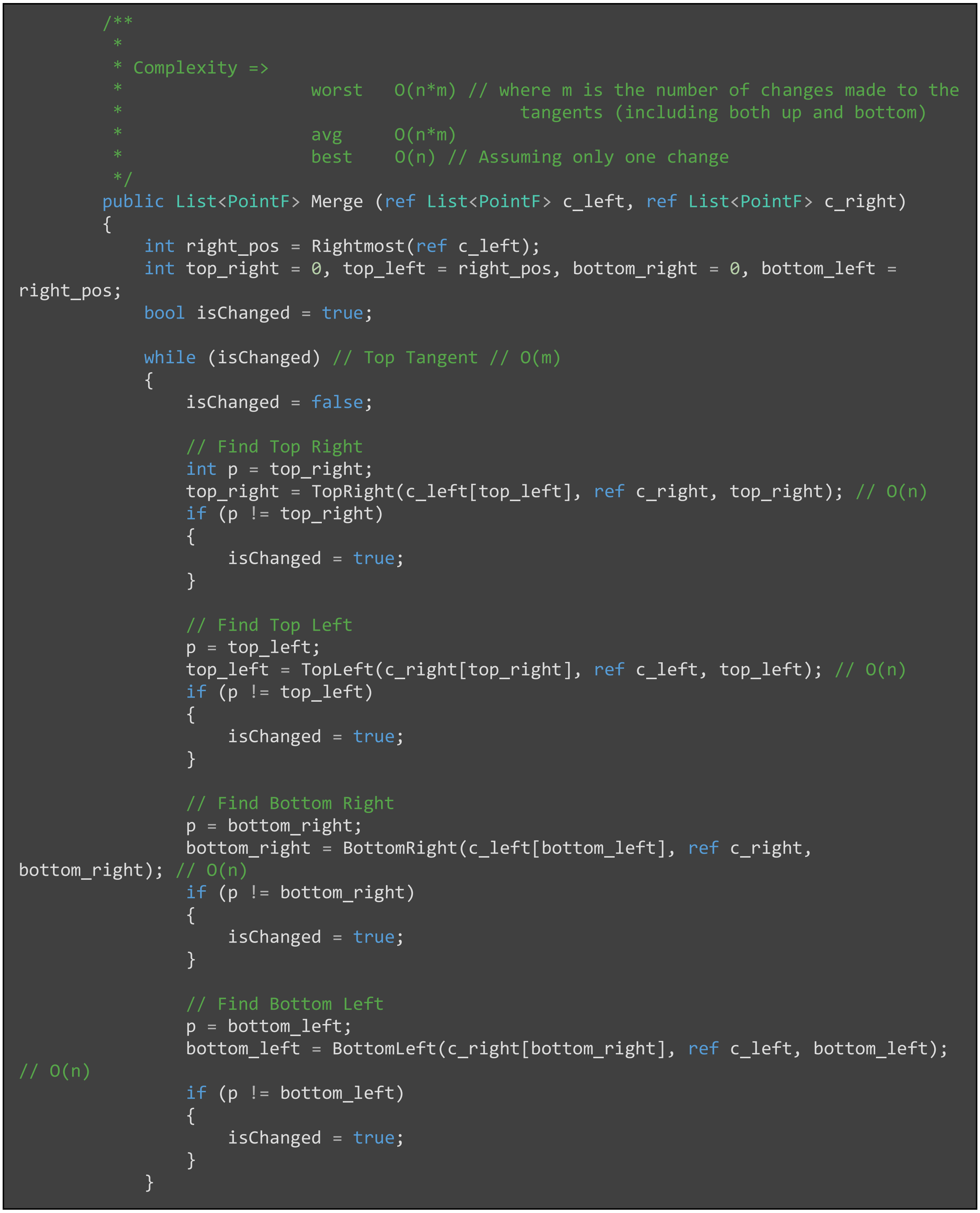
Benjamin Thompson

CS 312 Section 1

Lab 2

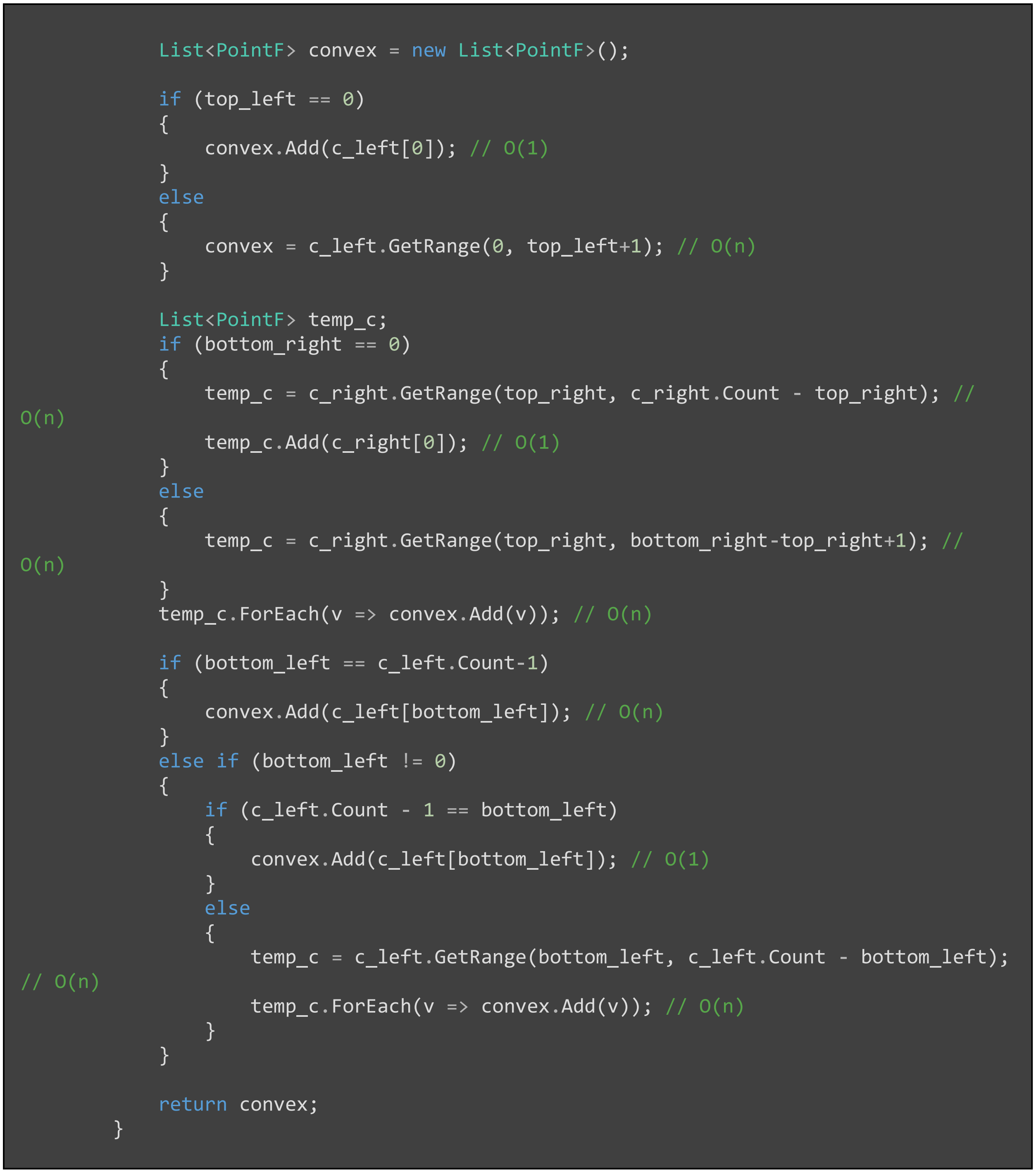
Code commented with complexity:

```
        /**
         * Main function for solving the outer hull problem
         *
         */
        public void Solve(List<System.Drawing.PointF> pointList)
        {
            // Lambda function for sorting by X
            // https://msdn.microsoft.com/en-us/library/b0zbh7b6.aspx -- O(nlogn) average, O(n^2) worst case
            pointList.Sort((a, b) => a.X.CompareTo(b.X));

            // Convex Hull - O()
            pointList = ConvexHull(ref pointList, 0, pointList.Count-1);

            // Draw Convex Hull
            g.DrawPolygon(new Pen(Color.Red), pointList.ToArray());
        }
        /**
         * Convex Hull algorithm. Finds the smallest encompasing hull of a group of 
verticies
         *
         * Complexity =>
         *                  worst   O(n^2)
         *                  avg     O(nlogn)
         *                  best    O(nlogn)
         *
         */
        public List<PointF> ConvexHull(ref List<System.Drawing.PointF> pointList, int 
left, int right)
        {
            List<PointF> convex = new List<PointF>();
            List<PointF> convex1 = new List<PointF>();
            // Check if grouping is too big
            if ((right - left) > 2)
            {
                convex = ConvexHull(ref pointList, left, (right - left + 1) / 2 - 1 + 
left);
                convex1 = ConvexHull(ref pointList, (right - left + 1) / 2 + left, 
right);
                //return convex;
            }

            List<System.Drawing.PointF> points = new List<PointF>();
            for (int i = left; i < right + 1; i++) // O(n)
            {
                points.Add(pointList[i]);
            }

            if (convex.Count > 0)
            {
                convex = Merge(ref convex, ref convex1); // O(n)
            }
            else
            {
                convex = SortCC(points); // O(n^2)
            }
            return convex;
        }
```

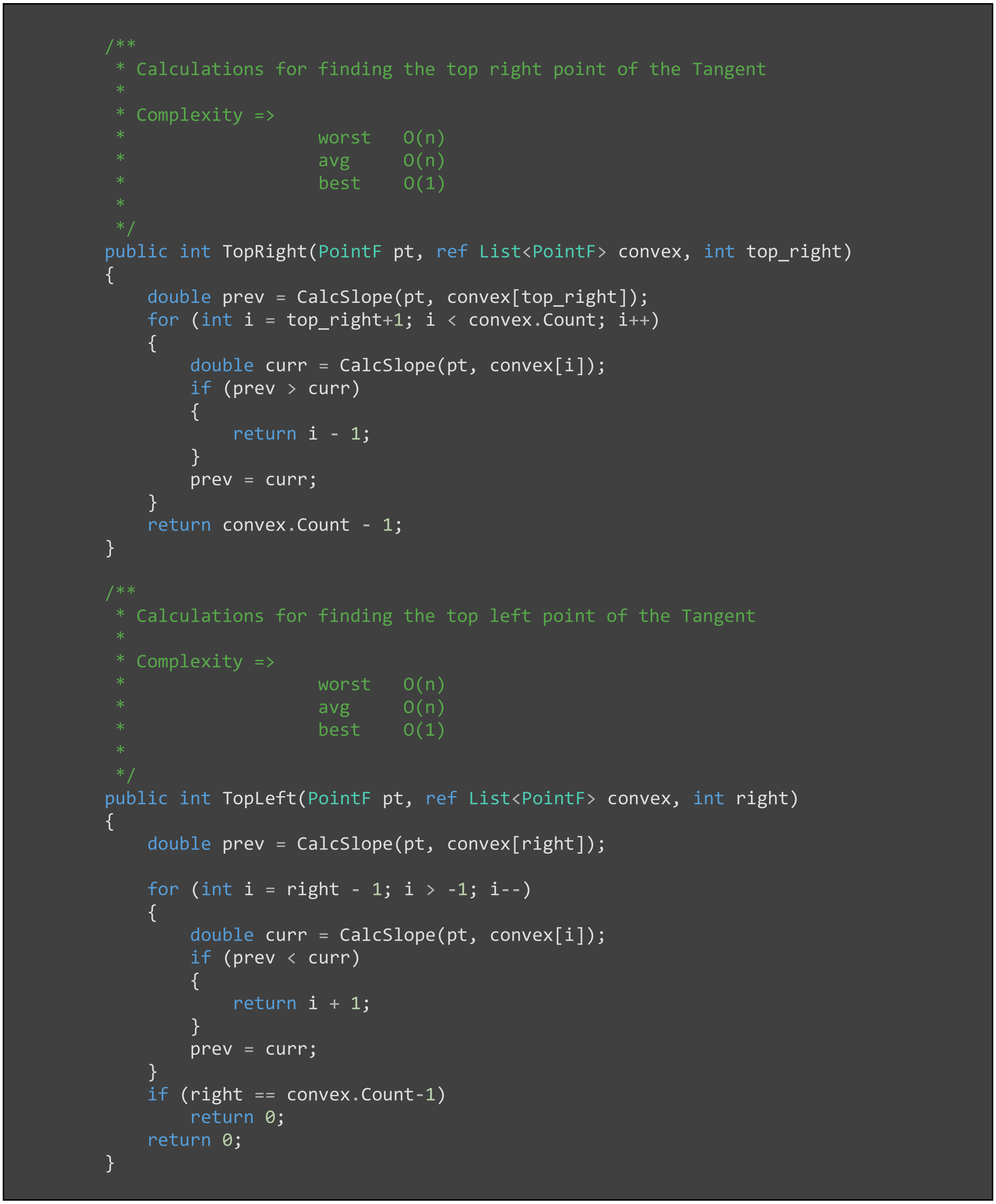


```
        /**
         *
         * Complexity =>
         *                  worst   O(n*m) // where m is the number of changes made to the
         *                                      tangents (including both up and bottom)
         *                  avg     O(n*m)
         *                  best    O(n) // Assuming only one change
         */
        public List<PointF> Merge (ref List<PointF> c_left, ref List<PointF> c_right)
        {
            int right_pos = Rightmost(ref c_left);
            int top_right = 0, top_left = right_pos, bottom_right = 0, bottom_left =
right_pos;
            bool isChanged = true;

            while (isChanged) // Top Tangent // O(m)
            {
                isChanged = false;

                // Find Top Right
                int p = top_right;
                top_right = TopRight(c_left[top_left], ref c_right, top_right); // O(n)
                if (p != top_right)
                {
                    isChanged = true;
                }

                // Find Top Left
                p = top_left;
                top_left = TopLeft(c_right[top_right], ref c_left, top_left); // O(n)
                if (p != top_left)
                {
                    isChanged = true;
                }

                // Find Bottom Right
                p = bottom_right;
                bottom_right = BottomRight(c_left[bottom_left], ref c_right,
bottom_right); // O(n)
                if (p != bottom_right)
                {
                    isChanged = true;
                }

                // Find Bottom Left
                p = bottom_left;
                bottom_left = BottomLeft(c_right[bottom_right], ref c_left, bottom_left);
// O(n)
                if (p != bottom_left)
                {
                    isChanged = true;
                }
            }
```

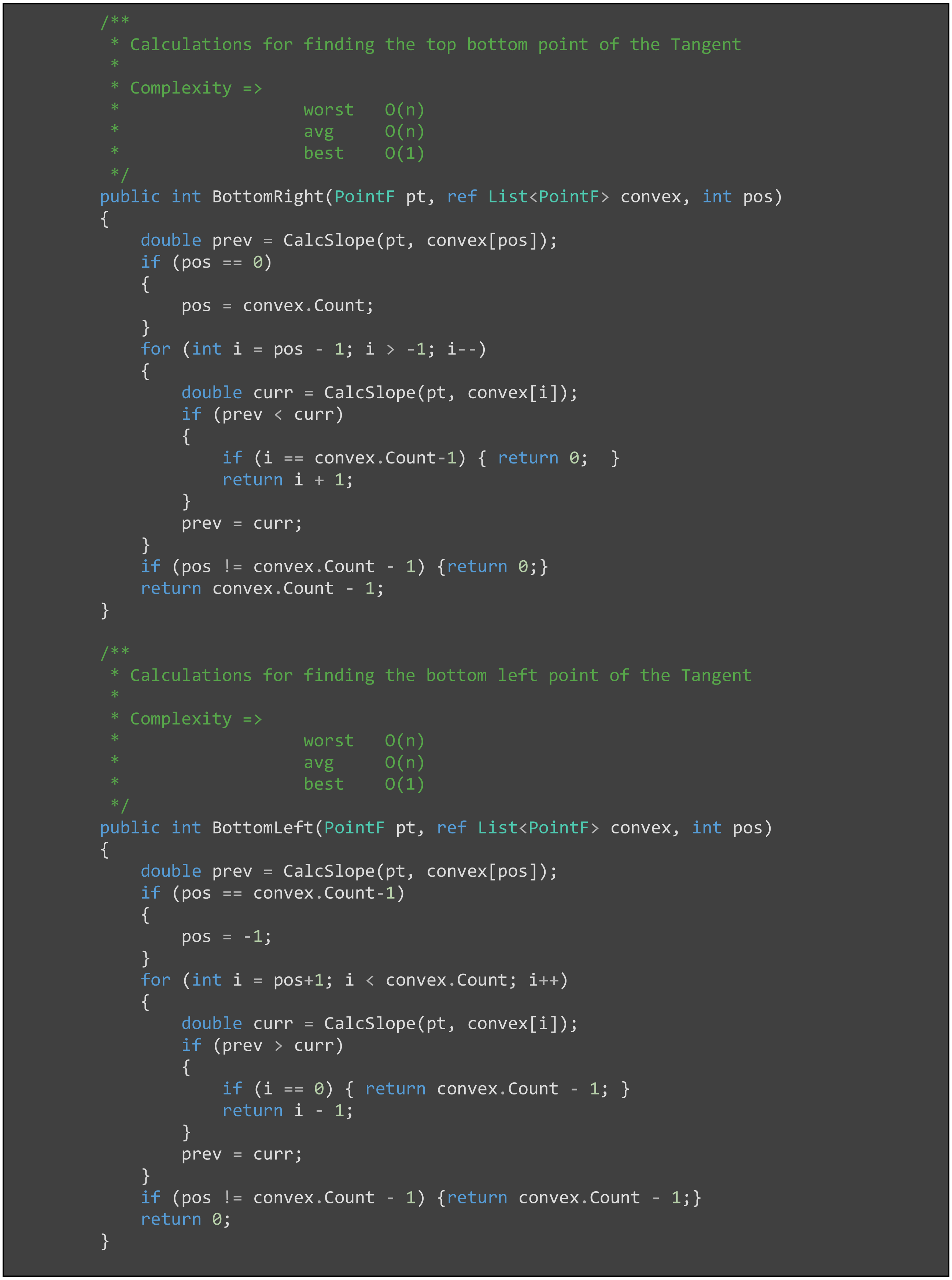
Continued on next page



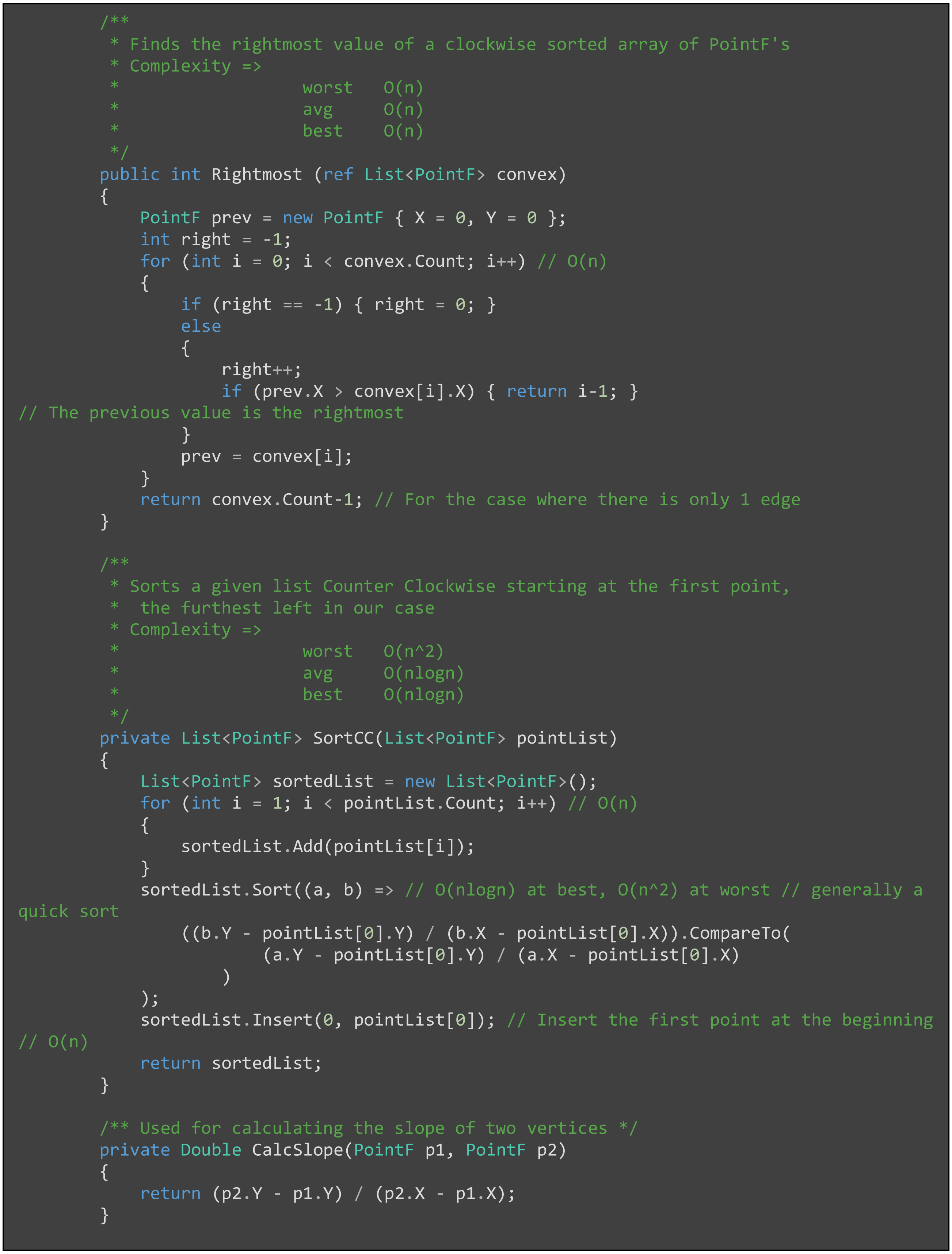
```
            List<PointF> convex = new List<PointF>();

            if (top_left == 0)
            {
                convex.Add(c_left[0]); // O(1)
            }
            else
            {
                convex = c_left.GetRange(0, top_left+1); // O(n)
            }

            List<PointF> temp_c;
            if (bottom_right == 0)
            {
                temp_c = c_right.GetRange(top_right, c_right.Count - top_right); //
O(n)
                temp_c.Add(c_right[0]); // O(1)
            }
            else
            {
                temp_c = c_right.GetRange(top_right, bottom_right-top_right+1); //
O(n)
            }
            temp_c.ForEach(v => convex.Add(v)); // O(n)

            if (bottom_left == c_left.Count-1)
            {
                convex.Add(c_left[bottom_left]); // O(n)
            }
            else if (bottom_left != 0)
            {
                if (c_left.Count - 1 == bottom_left)
                {
                    convex.Add(c_left[bottom_left]); // O(1)
                }
                else
                {
                    temp_c = c_left.GetRange(bottom_left, c_left.Count - bottom_left);
// O(n)
                    temp_c.ForEach(v => convex.Add(v)); // O(n)
                }
            }

            return convex;
        }
```



```
/**
 * Calculations for finding the top right point of the Tangent
 *
 * Complexity =>
 *                  worst   O(n)
 *                  avg     O(n)
 *                  best    O(1)
 *
 */
public int TopRight(PointF pt, ref List<PointF> convex, int top_right)
{
    double prev = CalcSlope(pt, convex[top_right]);
    for (int i = top_right+1; i < convex.Count; i++)
    {
        double curr = CalcSlope(pt, convex[i]);
        if (prev > curr)
        {
            return i - 1;
        }
        prev = curr;
    }
    return convex.Count - 1;
}

/**
 * Calculations for finding the top left point of the Tangent
 *
 * Complexity =>
 *                  worst   O(n)
 *                  avg     O(n)
 *                  best    O(1)
 *
 */
public int TopLeft(PointF pt, ref List<PointF> convex, int right)
{
    double prev = CalcSlope(pt, convex[right]);

    for (int i = right - 1; i > -1; i--)
    {
        double curr = CalcSlope(pt, convex[i]);
        if (prev < curr)
        {
            return i + 1;
        }
        prev = curr;
    }
    if (right == convex.Count-1)
        return 0;
    return 0;
}
```



```
/**
 * Calculations for finding the top bottom point of the Tangent
 *
 * Complexity =>
 *                  worst   O(n)
 *                  avg     O(n)
 *                  best    O(1)
 */
public int BottomRight(PointF pt, ref List<PointF> convex, int pos)
{
    double prev = CalcSlope(pt, convex[pos]);
    if (pos == 0)
    {
        pos = convex.Count;
    }
    for (int i = pos - 1; i > -1; i--)
    {
        double curr = CalcSlope(pt, convex[i]);
        if (prev < curr)
        {
            if (i == convex.Count-1) { return 0;  }
            return i + 1;
        }
        prev = curr;
    }
    if (pos != convex.Count - 1) {return 0;}
    return convex.Count - 1;
}

/**
 * Calculations for finding the bottom left point of the Tangent
 *
 * Complexity =>
 *                  worst   O(n)
 *                  avg     O(n)
 *                  best    O(1)
 */
public int BottomLeft(PointF pt, ref List<PointF> convex, int pos)
{
    double prev = CalcSlope(pt, convex[pos]);
    if (pos == convex.Count-1)
    {
        pos = -1;
    }
    for (int i = pos+1; i < convex.Count; i++)
    {
        double curr = CalcSlope(pt, convex[i]);
        if (prev > curr)
        {
            if (i == 0) { return convex.Count - 1; }
            return i - 1;
        }
        prev = curr;
    }
    if (pos != convex.Count - 1) {return convex.Count - 1;}
    return 0;
}
```



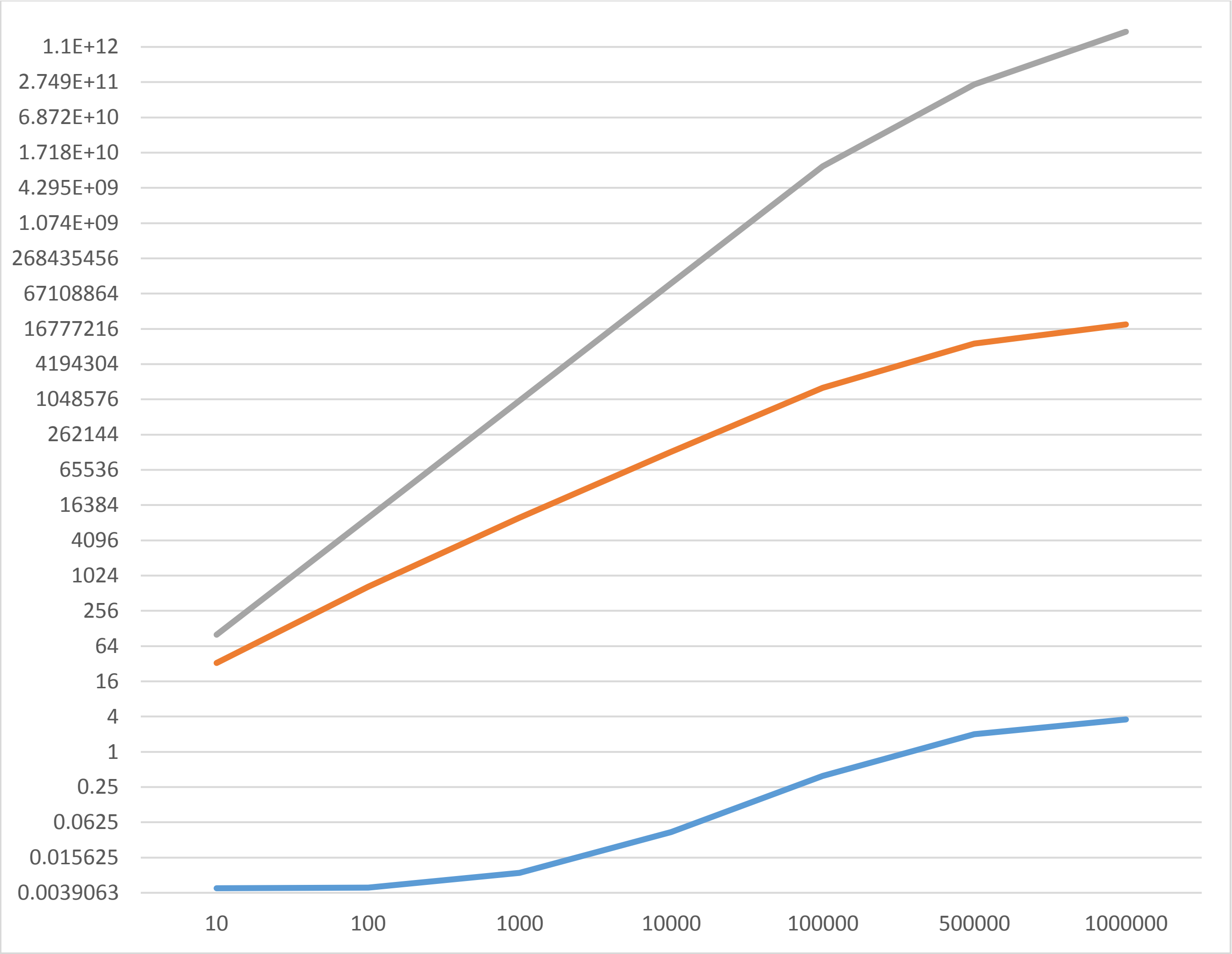
```
        /**
         * Finds the rightmost value of a clockwise sorted array of PointF's
         * Complexity =>
         *                  worst   O(n)
         *                  avg     O(n)
         *                  best    O(n)
         */
        public int Rightmost (ref List<PointF> convex)
        {
            PointF prev = new PointF { X = 0, Y = 0 };
            int right = -1;
            for (int i = 0; i < convex.Count; i++) // O(n)
            {
                if (right == -1) { right = 0; }
                else
                {
                    right++;
                    if (prev.X > convex[i].X) { return i-1; }
// The previous value is the rightmost
                }
                prev = convex[i];
            }
            return convex.Count-1; // For the case where there is only 1 edge
        }

        /**
         * Sorts a given list Counter Clockwise starting at the first point,
         *  the furthest left in our case
         * Complexity =>
         *                  worst   O(n^2)
         *                  avg     O(nlogn)
         *                  best    O(nlogn)
         */
        private List<PointF> SortCC(List<PointF> pointList)
        {
            List<PointF> sortedList = new List<PointF>();
            for (int i = 1; i < pointList.Count; i++) // O(n)
            {
                sortedList.Add(pointList[i]);
            }
            sortedList.Sort((a, b) => // O(nlogn) at best, O(n^2) at worst // generally a
quick sort
                ((b.Y - pointList[0].Y) / (b.X - pointList[0].X)).CompareTo(
                        (a.Y - pointList[0].Y) / (a.X - pointList[0].X)
                    )
            );
            sortedList.Insert(0, pointList[0]); // Insert the first point at the beginning
// O(n)
            return sortedList;
        }

        /** Used for calculating the slope of two vertices */
        private Double CalcSlope(PointF p1, PointF p2)
        {
            return (p2.Y - p1.Y) / (p2.X - p1.X);
        }
```

The convex (O(n^2) at worst) hull does seem to fit my data. The data (blue line graph) seems to fit a similar pattern to my thoughts. The graph I used is a logarithmic one, meaning that if the data set is logarithmic, then it will be linear on the plot. As you can see, my data starts out linear (O(logn)), the graph strays from it a bit, showing more of a O(nlogn) pattern (compare to the orange line graph). The orange line graph is an nlogn graphing of the same initial values, the gray is an O(n^2) graphing. Variations between the two graphs (blue and yellow) are slight.

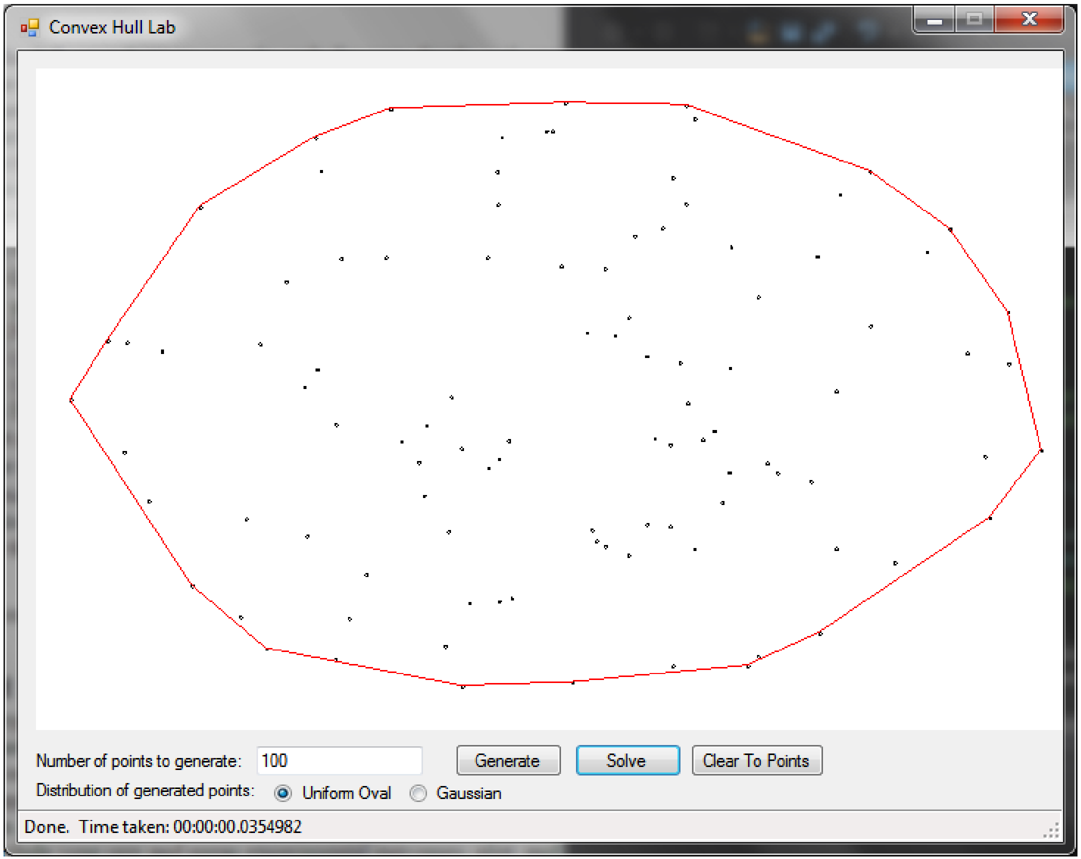
The recurrence relation for this algorithm is T(n) = 2T(n/2) + O(n) for the average case, and T(n)=2T(n/2) + O(n^2) for the worst case, which is extremely unlikely as the data shows. This solves to O(nlogn) and O(n^2).

Data :

10	0.0047104
100	0.0047735
1000	0.0086178
10000	0.0426202
100000	0.3960408
500000	2.0077503
1000000	3.5943062



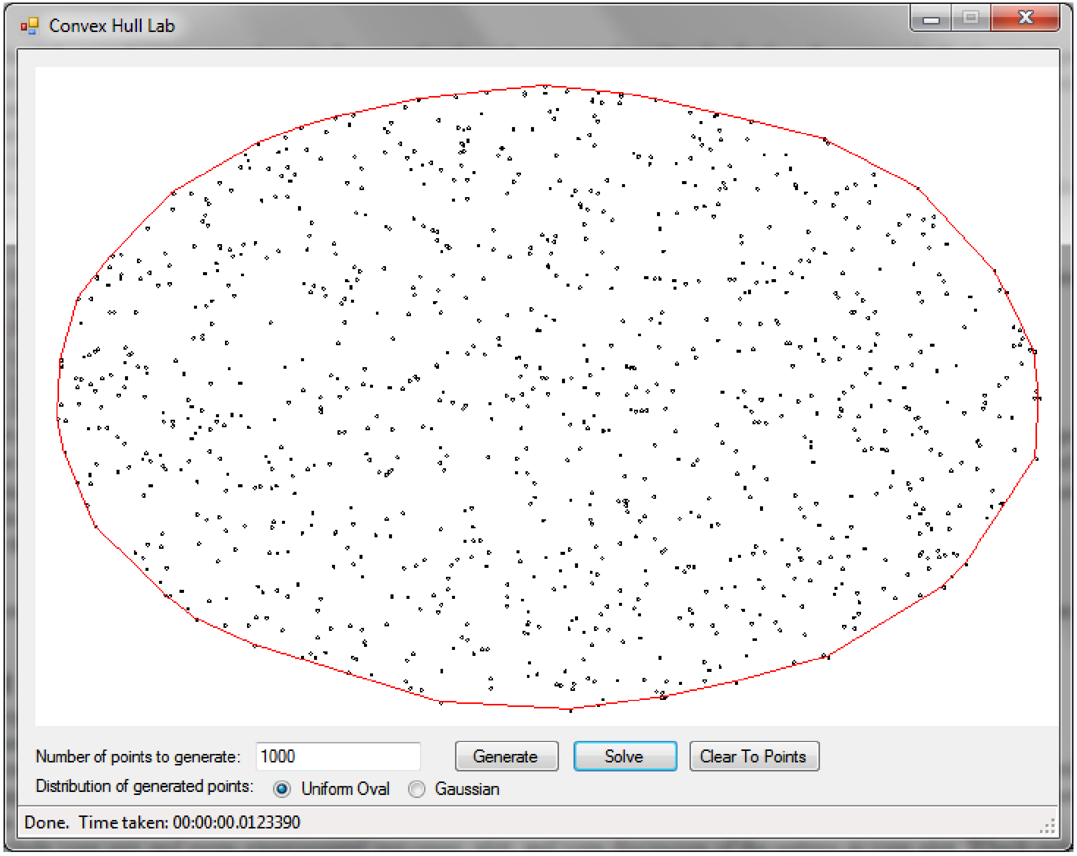
Screenshots



Number of points to generate: 100

Distribution of generated points: ● Uniform Oval ○ Gaussian

Done. Time taken: 00:00:00.0354982



Number of points to generate: 1000

Distribution of generated points: ● Uniform Oval ○ Gaussian

Done. Time taken: 00:00:00.0123390