Scott Christensen

02/04/16

C.S. 312

Project 2 Submission

My Code:

```
class PointFComparer : IComparer<PointF>
 {
     public int Compare(PointF firstPoint, PointF secondPoint)
     {
         if (firstPoint.X > secondPoint.X)
         {
             return 1;
         }
         else if (firstPoint.X < secondPoint.X)
         {
             return -1;
         }
         else
         {
             return 0;
         }
     }
 }

 //basically just the wrapper function for Divider and Combiner
 /*Solve(The entire alorithm)
  * Master Theorem 2/(2^1) * There are two split branches of size n/2
  * and it costs O(n) to reutrn(i.e. combineHulls function)
  * So according to the Master Theorem, The whole algorithm is O(nlogn)
  *Time Complexity: O(nlogn)
  *Space Complexity: O(nlogn)
  */
 public void Solve(List<System.Drawing.PointF> pointList)
 {

     /*Sort
      * Time Complexity: O(nlog(n))
      * Space Complexity: O(n)
      * */
     pointList.Sort(new PointFComparer());//clockwise order

     /*Convex Hull using Divide and Conquer
      *  Time Complexity: O(nlog(n))
      *  Space Complexity: O(n)
      * */
     List<System.Drawing.PointF> convexHull = divider(pointList);

     //draw the complete hull
```

```
            drawAll(convexHull);

        }

        //draws everything in the list
        private void drawAll(List<PointF> listy)
        {
            for (int i = 0; i < listy.Count - 1; i++)
            {
                g.DrawLine(greeny, listy[i], listy[i + 1]);
            }
            g.DrawLine(greeny, listy[listy.Count - 1], listy[0]);
        }

        //splits up our list into managable chunks, then calls combiner to put them back
together with a hull around them
        private List<System.Drawing.PointF> divider(List<PointF> pts)
        {
            List<System.Drawing.PointF> output = new List<System.Drawing.PointF>();
            if (pts.Count == 2)
            {
                output.Add(pts[0]);
                output.Add(pts[1]);
                //g.DrawLine(new Pen(Color.Blue, 2), pts[0], pts[1]);
                return output;
            }
            else if (pts.Count == 3)
            {
                //Storing them clock-wise
                output.Add(pts[0]);
                //checks to see if the points are clockwise or not.
                if (-(getSlope(pts[0], pts[1])) > -(getSlope(pts[0], pts[2])))
                {
                    output.Add(pts[1]);
                    output.Add(pts[2]);
                    //g.DrawLine(new Pen(Color.Blue, 2), output[0], output[1]);
                }
                else
                {
                    output.Add(pts[2]);
                    output.Add(pts[1]);
                    //g.DrawLine(new Pen(Color.Blue, 2), output[0], output[1]);
                }

                return output;
            }
            else
            {
                /*Split the List into two groups with 1/2 the points in each till we hit
the base cases.
                 *These are both have
                 * Time Complexity: O(logn)
                 * Space Complitity: O(n)
                 *
                 * */
                List<System.Drawing.PointF> left = pts.GetRange(0,
(int)Math.Floor((double)pts.Count / 2));
```

```
                List<System.Drawing.PointF> right =
pts.GetRange((int)Math.Floor((double)pts.Count / 2), (int)Math.Ceiling((double)pts.Count
/ 2));

                /*Since we split to the base case, this is
                * Time Complexity O(nlog(n)) total
                * Space COmplexity O(nlog(n))
                 */
                right = divider(right);
                left = divider(left);

                /*Combine Hulls
                 * Time Complexity: O(n)
                 * Space Complexoty: O(n)
                 *
                 * */
                output = combineHulls(left, right);
                return output;
            }
        }

        //puts the two list of points together by finding the tangents connecting each,
then making a list of all outside points
        /*CombineHulls
         *
         *Time Complexity: O(n)
         *Space Complexity: O(n)
         */
        private List<PointF> combineHulls(List<PointF> left, List<PointF> right)
        {
            PointF leftMostPt = right[0];
            /*Find Right pt
            *Time Complexity: O(n)
            *Space Complexity: O(n)
             */
            PointF rightMostPt = findRightPt(left);

            List<PointF> topPoints = new List<PointF>();
            List<PointF> bottomPoints = new List<PointF>();

            bottomPoints.Add(rightMostPt);
            bottomPoints.Add(leftMostPt);

            topPoints.Add(rightMostPt);
            topPoints.Add(leftMostPt);

            //first get the top tangent
            Boolean changed = true;

            /*First while loop
            *Time Complexity: O(n) since we arn't revisiting any points
            *Space Complexity: O(n)
             */
            while (changed == true)
            {
                PointF TopFirst = topPoints[0];
                PointF TopSecond = topPoints[1];
                topPoints = topLine(left, right, TopSecond, TopFirst);
```

```
                //Did the points change?
                if (!TopFirst.Equals(topPoints[0]) || !TopSecond.Equals(topPoints[1]))
                {
                    changed = true;
                }
                else
                {
                    changed = false;
                }
            }
            //g.DrawLine(new Pen(Color.Blue, 2), bottomTangent[0], bottomTangent[1]);

            //now do the bottom
            /*Second while loop
            *Time Complexity: O(n) since we arn't revisiting any points
            *Space Complexity: O(n)
             */
            changed = true;
            while (changed)
            {
                PointF bottomFirst = bottomPoints[0];
                PointF bottomSecond = bottomPoints[1];
                bottomPoints = bottomLine(left, right, bottomSecond, bottomFirst);

                if (!bottomFirst.Equals(bottomPoints[0]) ||
!bottomSecond.Equals(bottomPoints[1]))
                {
                    changed = true;
                }
                else
                {
                    changed = false;
                }
            }
            //g.DrawLine(new Pen(Color.Blue, 2), bottomTangent[0], bottomTangent[1]);

            //Now we put the two sides back together with the tangent points excluding
the middle points in clockwise order
            List<PointF> result = new List<PointF>();
            int topLeft = left.IndexOf(topPoints[0]);
            int topRight = right.IndexOf(topPoints[1]);
            int bottomLeft = left.IndexOf(bottomPoints[0]);
            int bottomRight = right.IndexOf(bottomPoints[1]);


            //get the points from 0 to the top left...
            /*The three For loops collecting points
            *Time Complexity: O(n) since you simply do one large loop around the left and
right pieces
            *Space Complexity: O(n)
             */
            for (int i = 0; i <= topLeft; i++)
            {
                result.Add(left[i]);
            }
            //...then go across the line to get the points from the topright to the
bottom right...
```

```
            for (int i = topRight; i != bottomRight; i = nextInt(right.Count, i))
            {
                result.Add(right[i]);
            }
            result.Add(right[bottomRight]);

            //..then finally get the points from the bottom left back up to 0
            for (int i = bottomLeft; i != 0; i = nextInt(left.Count, i))
            {
                result.Add(left[i]);
            }

            //return answer
            return result;
        }


        /*nextInt
         *Time Complexity: O(1)
         *Space Complexity: O(1)
          */
        private int nextInt(int count, int i)
        {
            if (i == count - 1)
            {
                return 0;
            }
            else
            {
                return i + 1;
            }
        }

        //returns the top line. Only compares the slope as long as it's moving in the
direction we want
        /*topLine
         *Time Complexity: O(n) Beacuse you could check the slope on every point in both
collections
         *Space Complexity: O(n)
         */
        private List<PointF> topLine(List<PointF> leftList, List<PointF> rightList,
PointF leftMostPt, PointF rightMostPt)
        {
            PointF rightPivot = leftMostPt;
            PointF leftPivot = rightMostPt;

            //move left as long as we're increasing, cclockwise
            Boolean increasing = true;
            PointF saveList = leftPivot;
            double saveSlope = getSlope(rightMostPt, leftMostPt);
            while (increasing)
            {
                //g.DrawLine(new Pen(Color.White, 2), saveList, rightPivot);//erase
                PointF currPoint;
                if (leftList.IndexOf(saveList) == 0)
                {
                    currPoint = leftList[leftList.Count - 1];
```

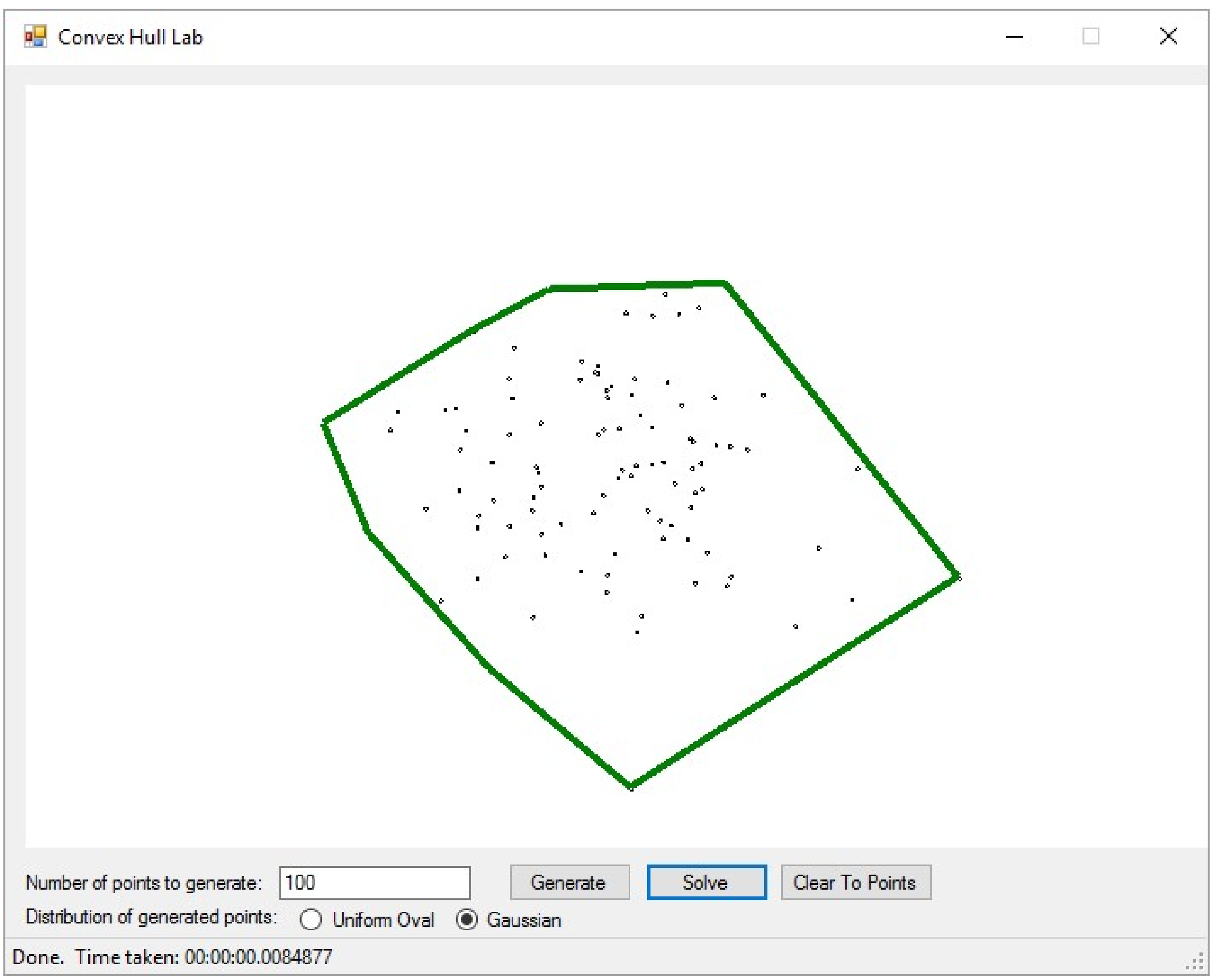
```
                }
                else
                {
                    currPoint = leftList[leftList.IndexOf(saveList) - 1];
                }
                double currSlope = getSlope(currPoint, rightPivot);
                //Console.WriteLine("currSlope is " + currSlope + " and save slope is " +
saveSlope);
                //g.DrawLine(new Pen(Color.Blue, 2), currPoint, rightPivot);
                //g.DrawLine(new Pen(Color.White, 2), currPoint, rightPivot);
                if (currSlope - saveSlope > 0)
                {

                    saveSlope = currSlope;
                    saveList = currPoint;
                    increasing = true;

                }
                else
                {
                    increasing = false;
                }
                //g.DrawLine(new Pen(Color.Blue, 2), saveList, rightPivot);
            }
            leftPivot = saveList;

            //move right as long as we're decreasing, clockwise
            Boolean decreasing = true;
            saveList = rightPivot;
            saveSlope = getSlope(leftPivot, leftMostPt);
            while (decreasing)
            {
                //g.DrawLine(new Pen(Color.White, 2), leftPivot, saveList);
                PointF currPoint;
                if (rightList.IndexOf(saveList) == rightList.Count - 1)
                {
                    currPoint = rightList[0];
                }
                else
                {
                    currPoint = rightList[rightList.IndexOf(saveList) + 1];
                }
                double currSlope = getSlope(leftPivot, currPoint);
                //Console.WriteLine("currSlope is " + currSlope + " and save slope is " +
saveSlope);//for testing purposes
                //g.DrawLine(new Pen(Color.Blue, 2), leftPivot, currPoint);
                //g.DrawLine(new Pen(Color.White, 2), leftPivot, currPoint);//erase
                if (currSlope - saveSlope < 0)
                {

                    saveSlope = currSlope;
                    saveList = currPoint;
                    decreasing = true;

                }
                else
                {
                    decreasing = false;
```

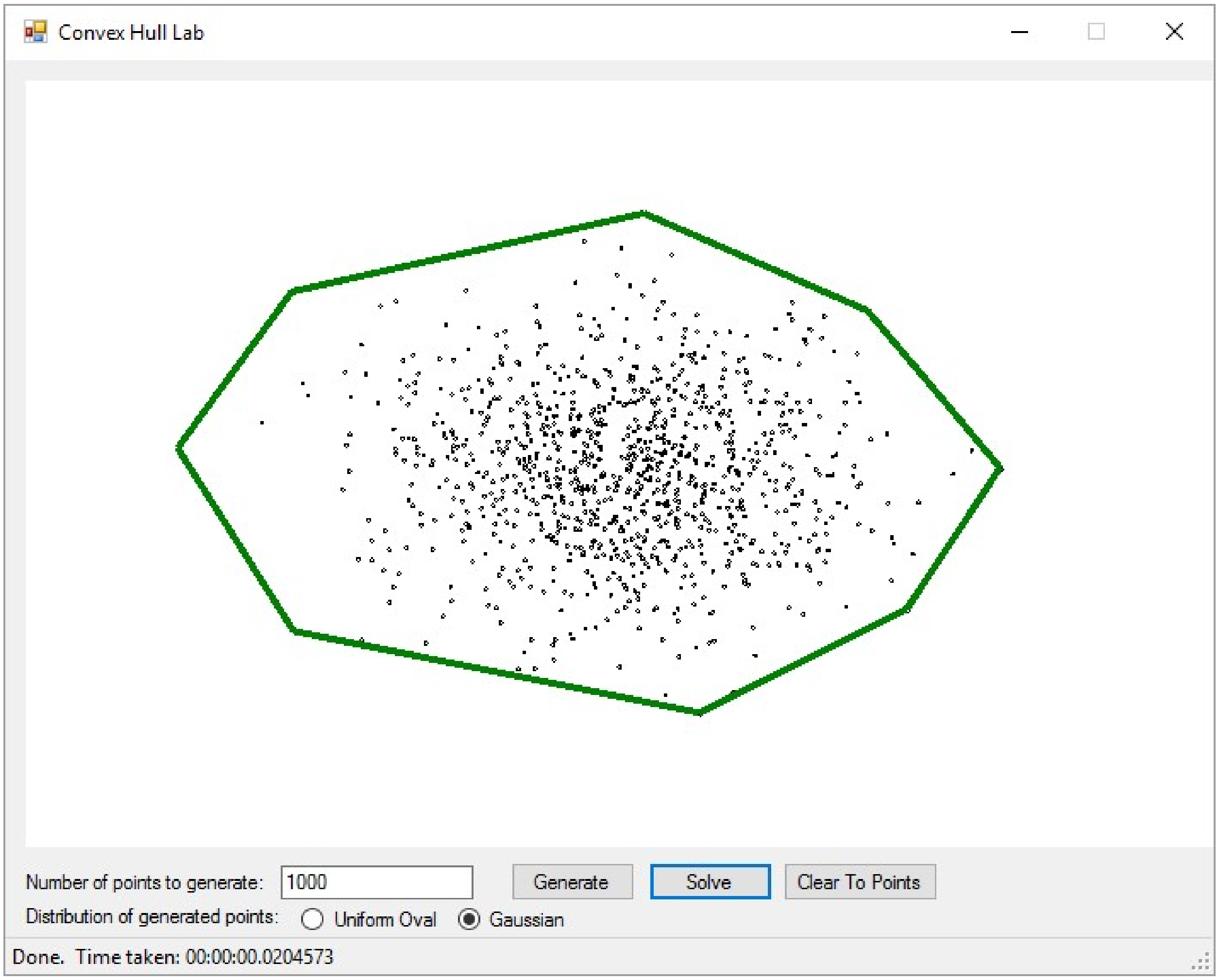
```
                }
                //g.DrawLine(new Pen(Color.Green, 2), leftPivot, saveList);
            }
            rightPivot = saveList;

            List<PointF> result = new List<PointF>();
            result.Add(leftPivot);
            result.Add(rightPivot);
            return result;
        }

        //returns the bottom line. Only compares the slope as long as it's moving in the 
direction we want
        /*bottomLine
         *Time Complexity: O(n) Beacuse you could check the slope on every point in both 
collections
         *Space Complexity: O(n)
         */
        private List<PointF> bottomLine(List<PointF> leftList, List<PointF> rightList, 
PointF leftMostPt, PointF rightMostPt)
        {
            PointF rightPivot = leftMostPt;
            PointF leftPivot = rightMostPt;

            //first mov left as long as we're decreasing, clockwise
            Boolean decreasing = true;
            PointF saveList = leftPivot;
            double saveSlope = getSlope(rightMostPt, leftMostPt);
            while (decreasing)
            {
                //g.DrawLine(new Pen(Color.White, 2), saveList, rightPivot);
                PointF currPoint;
                if (leftList.IndexOf(saveList) == leftList.Count - 1)
                {
                    currPoint = leftList[0];
                }
                else
                {
                    currPoint = leftList[leftList.IndexOf(saveList) + 1];
                }
                double currSlope = getSlope(currPoint, rightPivot);
                //Console.WriteLine("currSlope is " + currSlope + " and save slope is " + 
saveSlope);
                //g.DrawLine(new Pen(Color.Blue, 2), currPoint, rightPivot);
                //g.DrawLine(new Pen(Color.White, 2), currPoint, rightPivot);
                if (currSlope - saveSlope < 0)
                {
                    decreasing = true;
                    saveSlope = currSlope;
                    saveList = currPoint;
                }
                else
                {
                    decreasing = false;
                }
                //g.DrawLine(new Pen(Color.Green, 2), saveList, rightPivot);
            }
            leftPivot = saveList;
```

```
            //move right as long as we're increasing, cclockwise
            Boolean increasing = true;
            saveList = rightPivot;
            saveSlope = getSlope(leftPivot, leftMostPt);
            while (increasing)
            {
                //g.DrawLine(new Pen(Color.White, 2), leftPivot, saveList);//erase
                PointF currPoint;
                if (rightList.IndexOf(saveList) == 0)
                {
                    currPoint = rightList[rightList.Count - 1];
                }
                else
                {
                    currPoint = rightList[rightList.IndexOf(saveList) - 1];
                }
                double currSlope = getSlope(leftPivot, currPoint);
                //Console.WriteLine("currSlope is " + currSlope + " and save slope is " +
saveSlope);
                //g.DrawLine(new Pen(Color.Blue, 2), leftPivot, currPoint);
                //g.DrawLine(new Pen(Color.White, 2), leftPivot, currPoint);
                if (currSlope - saveSlope > 0)
                {
                    increasing = true;
                    saveSlope = currSlope;
                    saveList = currPoint;
                }
                else
                {
                    increasing = false;
                }
                //g.DrawLine(new Pen(Color.Blue, 2), leftPivot, saveList);
            }
            rightPivot = saveList;

            List<PointF> result = new List<PointF>();
            result.Add(leftPivot);
            result.Add(rightPivot);
            return result;
        }

        //returns any slope we put in so we can compare it later
        /*getSlope
         *Time Complexity: O(1)
         *Space Complexity: O(1)
         */
        private double getSlope(PointF one, PointF two)
        {
            double yResult = two.Y - one.Y;
            if (yResult == 0)
            {
                return 0;
            }
            double xResult = two.X - one.X;

            return yResult / xResult;
        }
```

```
        //returns the true rightmost point in a list
        private PointF findRightPt(List<PointF> list)
        {
            PointF mostRight = list[0];
            foreach(PointF p in list )
            {
                if(p.X >= mostRight.X)
                {
                    mostRight = p;
                }
                else
                {
                    break;
                }
            }
            return mostRight;
        }
    }
}
```

Pictures:





Time and space complexity summations:

There is three main parts of the algorithm. This includes Sorting the given points in a clockwise fashion, which takes Time Complexity of O(nlog(n) since I used the standard Library function and Space Complexity of O(n), since that's the size of all the points.

The second part would be dividing the points into groups of two of size n/2. The Time Complexity of doing this division is O(log(n)) by The Master Theorem, as we call the dividing function log(N) times. The Space Complexity is O(log(n)),since were making this many recursive calls so there are only log N things.

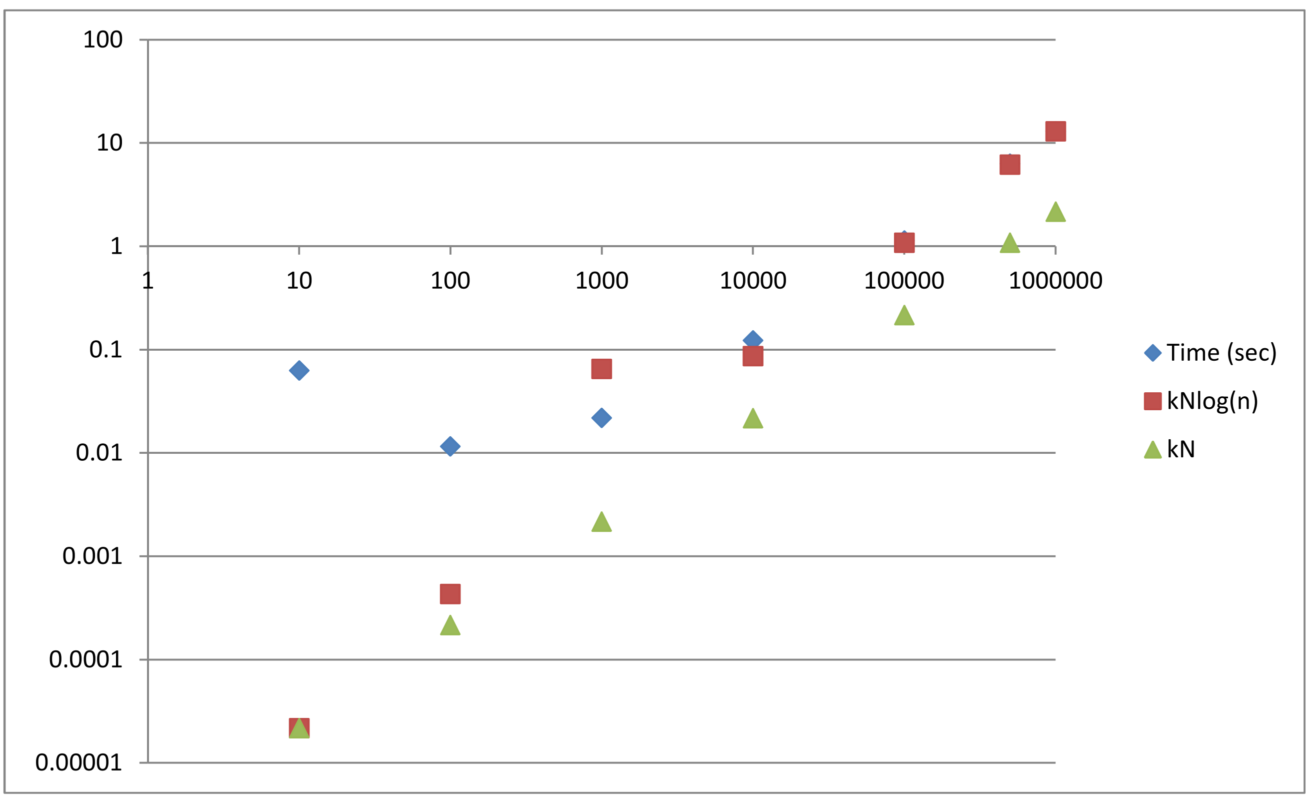
Then, after finding the convex Hull, you combine the points together. The Time Complexity of doing this is O(n) because you iterate over every point in the worst case and the Time Complexity of this is O(n) since we are returning the hull which could contain all points in the worst case.

In conclusion, the Total Time Complexity Big O is O(nlog(n) + n*log(n) + n) = 0(2nlog(n) +n) which, after dropping constants, reduces to O(nlog(n)). This agrees with my Master Theorem Calculation, a/(b^d) = 2/2 = O(nlog(n)). The Total Space Complexity Big O is O(n + n*log(n) + n) = O(2n + nlog(n)) which, after dropping constants, reduces to O(nlog(n)).

Raw and Mean Experimental outcomes:

N	Time (sec)	Nlog(N)	k	kNlog(n)	kN
10	0.0628	10	2.16612E-06	2.16612E-05	2.1661E-05
100	0.0116	200	2.16612E-06	0.000433223	0.00021661
1000	0.0219	30000	2.16612E-06	0.0649835	0.00216612
10000	0.1227	40000	2.16612E-06	0.086644667	0.02166117
100000	1.138	500000	2.16612E-06	1.083058333	0.21661167
500000	6.3171	2849485	2.16612E-06	6.17231695	1.08305833
1000000	12.9967	6000000	2.16612E-06	12.9967	2.16611667

Algorithm Analysis:



Discussion:

- The O(nlgo(n) growth order is indeed the best fit. We can see this is true when we multipy nlogn by the constant of proportionality.
- The constant of proportionality, K, which equals 2.16612E-06. We find this by t/nlog(n) for the biggest case, of dividing time by nlog(n) where n = 1000000.
- The pattern of the plot suggests that the growth follows O(nlog(n)) because the points resembles logarithmic growth. We can confirm this when compared to the linear pattern defined by kN.

Observations with your theoretical and empirical analyses:

Upon viewing the results from the data, it seems that for large values of n, the program does a very good job at matching the logarithmic growth as was expected during the design phase. However, it seems that my program has alot of error when smaller values of n are applied. I believe that this is because my program produces alot of overhead.

In the program we sort and draw all data points, which runs in O(n). For large values of n, this is insignificant when compared with running through so much data points. But for small values of n, these processes take a significant toll of the comparative run time when we consider that we are running so few data points. I believe why that is why there is so much inconsistancy with the respective results for small values of n.