Project 2 Writeup : Kevin DeVocht Section 2

1. Source Code is included as an additional attachment in the email
2. Pseudo-Code with worst case time efficiency :

```
Solve(pointList)
{
    pointList.Sort O(nlogn)
    hull = DivideConquer(pointList)
}

DivideConquer(pointList)
{

    if (pointList.Size > 3)
    {
        left = pointList.firstHalf
        right = pointList.secondFalf
        leftHull = DivideConquer(left)
        rightHull = DivideConquer(right)
    }
    else
    {
        if (pointList.Size == 3)
        {
            return BaseCreateHull(pointList)
        }
        else
        {
            leftMost.next = rightMost
            leftMost.previous = rightMost
            rightMost.next = leftMost
            rightMost.previous = leftMost
            return hull
        }
    }
    return Merge(leftHull, rightHull)
}

BaseCreateHull(pointList) O(1)
{
    if (slope_one > slope_two)
    {
        leftMost.next = middle
        leftMost.previous = rightMost
        middle.next = rightMost
        middle.previous = leftMost
        rightMost.next = leftMost
        rightMost.previous = middle
    }
    else
    {
        leftMost.next = rightMost
        leftMost.previous = middle
        middle.next = leftMost
        middle.previous = rightMost

        rightMost.next = middle
        rightMost.previous = leftMost
    }
```

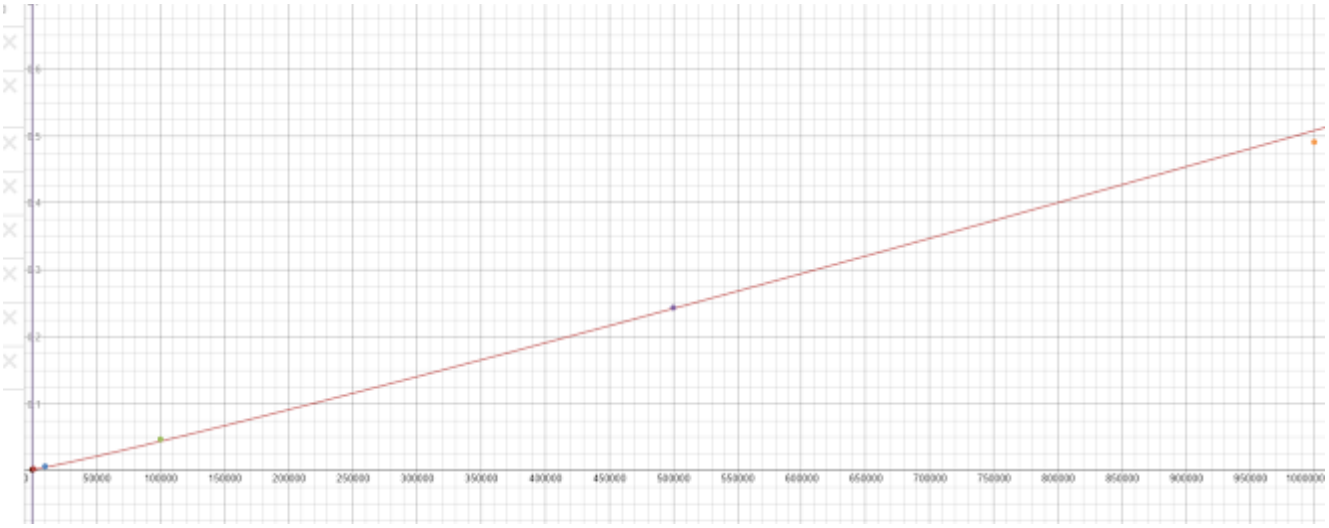
```
    return hull;
}

Merge(left, right) O(n)
{
    mergedHull = new (left.leftMost, right.rightMost)
    leftUpper = left.rightMost
    rightUpper = right.leftMost
    leftLower = left.rightMost
    rightLower = right.leftMost

    upperTangent = true
    while (!upperTangent)
    {
        upperTangent = false
        aMoved = true
        while (aMoved)
        {
            aMoved = false
            double slopeOne = GetSlope(leftUpper, rightUpper);
            double SlopeTwo = GetSlope(leftUpper.previous, rightUpper);
            if (slopeOne > SlopeTwo )
            {
                leftUpper = leftUpper.previous;
                aMoved = true;
            }
        }
        bool bMoved = true;
        while (bMoved)
        {
            bMoved = false;
            slopeOne = GetSlope(leftUpper, rightUpper)
            SlopeTwo = GetSlope(leftUpper, rightUpper.next)

            if (slopeOne < SlopeTwo)
            {
                rightUpper = rightUpper.next
                bMoved = true
                upperTangent = true
            }
        }
    }
    lowerTangent = true
    while (!lowerTangent)
    {
        runAgain = false
        bool aMoved = true
        while (aMoved)
        {
            aMoved = false
            double slopeOne = GetSlope(leftLower, rightLower)
            double SlopeTwo = GetSlope(leftLower.next, rightLower)
            if (slopeOne < SlopeTwo)
            {
                leftLower = leftLower.next
                aMoved = true
            }
        }
        bool bMoved = true
        while (bMoved)
        {
```

```
                bMoved = false
                double slopeOne = GetSlope(leftLower, rightLower)
                double SlopeTwo = GetSlope(leftLower, rightLower.previous)

                if (slopeOne > SlopeTwo)
                {
                    rightLower = rightLower.previous
                    bMoved = true
                    lowerTangent = true
                }
            }
        }
        leftUpper.next = rightUpper
        rightUpper.previous = leftUpper
        rightLower.next = leftLower
        leftLower.previous = rightLower
        mergedHull.leftUpper = leftUpper
        mergedHull.rightUpper = rightUpper
        mergedHull.leftLower = leftLower
        mergedHull.rightLower = rightLower

        return mergedHull
    }


    GetSlope(PointF One, PointF Two)O(1)
    {
        return -((Two.Y - One.Y) / (Two.X - One.X))
    }
```

After reviewing my Pseudo Code and determining the worst case time efficiency of each function(written in green next to the name of each function). We can see that the worst case efficiency time for the entire program is O(nlogn).

3.

	Trial One	Trial Two	Trial Three	Trial Four	Trial Five	Average
10	0.001713	0.0017474	0.0016843	0.0017903	0.0016958	0.00172616
100	0.0019005	0.0018863	0.0018036	0.0017803	0.0018811	0.00185036
1000	0.0023164	0.0049239	0.0021809	0.0021018	0.0022174	0.00274808
10000	0.0065346	0.0062427	0.0067399	0.006582	0.00667	0.00655384
100000	0.0486307	0.0480445	0.0476753	0.0464763	0.044724	0.04711016
500000	0.2466932	0.2380066	0.2433326	0.2462456	0.2433492	0.24352544
1000000	0.4922379	0.4838719	0.4938052	0.4824363	0.5022717	0.4909246



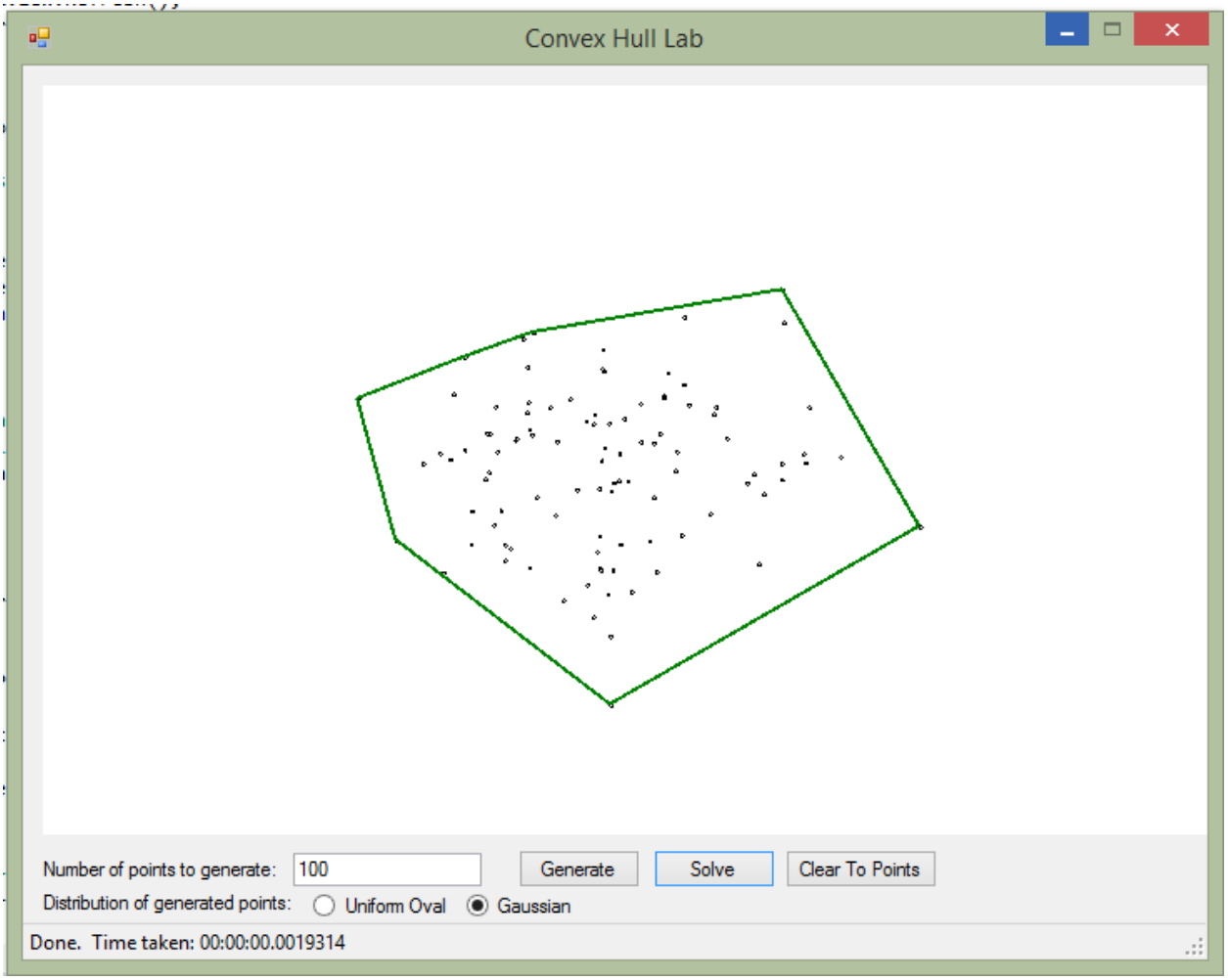
In the graph above you can see a red line and some plotted points(for a more detailed view of the graph please visit https://www.desmos.com/calculator/axqolpj7lw). The red line is the function $y = n(\frac{1}{11851300})$log n +0.00172616. The points are the averaged data points collected from my program. While the red line does not perfectly intersect all of my data points it is very close to all points and indicates that the coefficient used is close to the actual. We must also take into account that we only ran each size group five times. If I had a larger sample size my data would be more accurate and I would probably be able to find a more accurate coefficient as well. The best order of growth is n log n. By adding a constant of proportionality, $\frac{1}{11851300}$, to n log n I am able to very closely match my plotted points.

4. Both my theoretical and my empirical analyses indicates that my program has an efficiency upper bound of O(nlogn). The surprising result from the empirical data was the coefficient . I would not

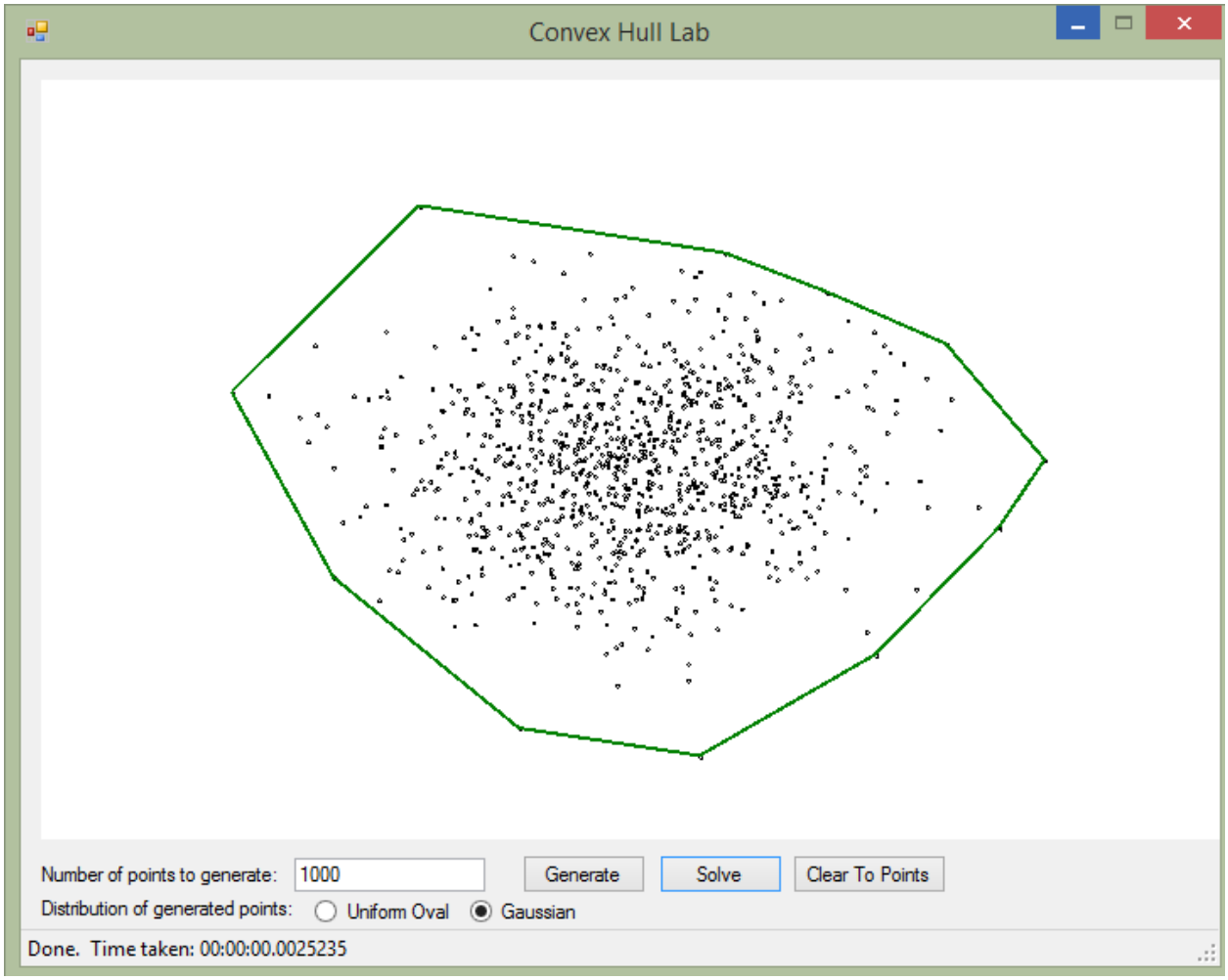
have expected it to be so small. While I knew that there would probably need to be a coefficient, It was interesting to note the significant difference it made on the shape of the graph. Without the coefficient, the normal n log n graph appeared almost like a vertical line when I zoomed out to see the 1,000,000 points data point.

5. Screen captures of working program.

100 point hull:



1000 point hull:


Convex Hull Lab
Number of points to generate:
1000
Generate
Solve
Clear To Points
Distribution of generated points:
Uniform Oval
Gaussian
Done. Time taken: 00:00:00.0025235