Project 2

David Olson, CS 312 section 2

PseudoCode with Complexity

Class Hull->I created this class so I could have an object that has a list sorted by slope, with the first element being the leftmost point, and an int value that designates the rightmost point index

```
Function Solve(pointList):
    Sort pointList (O(nlogn))
    Call Convex Hull function (O(convexHull))
    Draw resulting List (O(n))

Function ConvexHull(pointList):    (returns a Hull)   (O(logn + Cost(Combine))
    If pointList size is less than 4
        If size is 2
            Return a hull with pointlist and a right index of one
        If size is 3
            Order appropriately by slope, return a hull with that list and the correct
            right index
    If size is 4 or greater
        Divide pointList in half, recursively call this function on each half, and combine,
        And return that hull
```

```
Function Combine(left hull, right hull):     (returns a hull)     (O(n))  (getNewHull function taken
                                                                              Into account)

    Set up initial values of Booleans and top and bottom edge inde=ices (O(1))

    //find top edge (much more details and fringe cases in actual code)
    Start with an imaginary line from the far right of left hull to far left of right hull
    While the left or right index changes at least once
        While the right index is still changing
            If the next index clockwise makes a “higher” slope
                Change right index
        While the left index is still changing
            If the next index counter clockwise makes a “lower” slope
                Change left index
    //find bottom edge
    Start with same imaginary line
    While the left or right index changes at least once
        While the right index is still changing
            If the next index counter clockwise makes a “lower” slope
                Change right index
        While the left index is still changing
            If the next index clockwise makes a “higher slope
                Change left index
    Call getNewHull(), return its result  (to break up what would be a long, confusing
    function)
```

```
Function getNewHull(indices of top and bottom edge, rightIndex of right Hull, left and right
hull):                    (Returns a Hull Object)
    Make a new list in O(n) time, and in slope order, by:
        adding the left hull list up until the bottom edge
        add the bottom edge
        add the right hull list around to the top edge
        add the top edge
        add from the top edge to the end of the left hull list.
    Return this list and the correct rightmost index
```

Overall Anaylsis

The convex hull problem was solved using divide and conquer. My convexHull function divided the problem, a list of points, into many small but similar problems in log time. Each tiny sub-problem was in constant time, but to solve the big convex hull problem the small hulls needed to be combined in a way that gives us a new and correct hull, and those need to be combined, and so forth until the original set of points has a convex hull. The combine and getNewHull functions operate in O(n) time! Finding the top and bottom edges of the new Hull cannot possibly take more than n steps, since indices are not revisited. When making the combined list, the algorithm never needs to make more than one pass through each old list, because the points are needed in a slope-sorted order, and that's how they were stored. As for the solve function, that takes at most nlogn time due to list sorting.

The divide and conquer complexity can be figured out by the master theorem. In this algorithm, the problem is divided into two equally sized sub-problems each time, and solving the leaves and combining is in linear time, so the recurrence relation is $T(n) = 2T(n/2) + O(n)$. We now calculate $a/(b^d) = 2/2 = 1$. Therefore, we have achieved nlogn complexity.

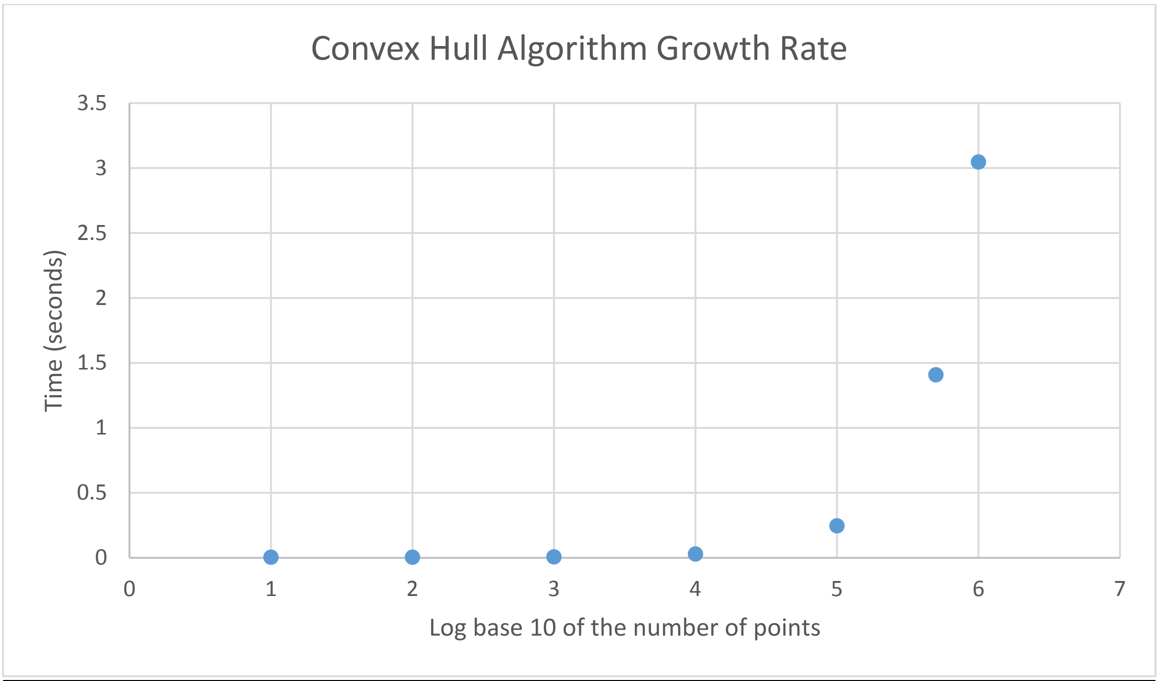
The space complexity for this algorithm is even better. The original list of points is contained in memory, and we recursively look at smaller and smaller parts of it. While combining the hulls, we temporarily create a new list to return. So as the hulls get bigger, we have several lists to keep in memory, but this is still in a linear complexity class.

Real Code

Attached to this same email, also at end of document

Empirical Analysis

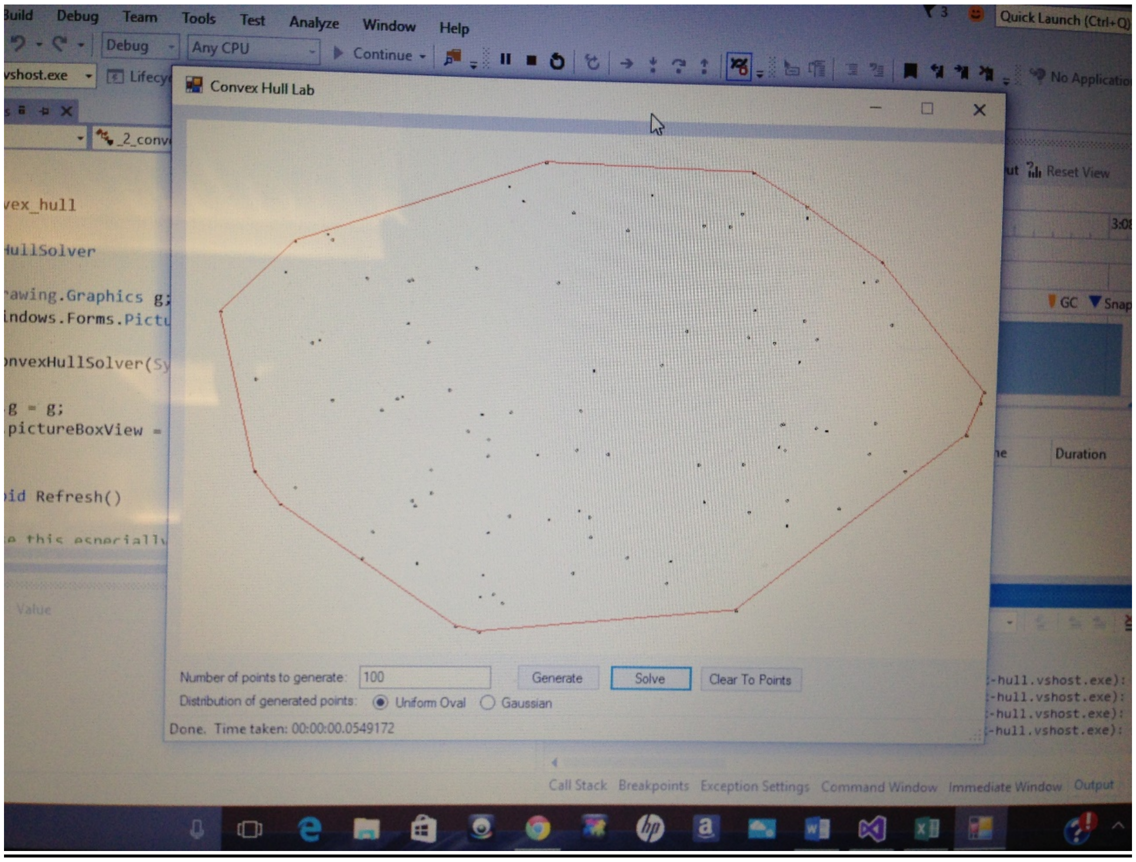
N	1	2	3	4	5	Ave
10	.0066165	.0056066	.0063807	.0059724	.0050947	.0059342
100	.0051420	.0065206	.0057162	.0057974	.0066142	.0059581
1,000	.0069753	.0079706	.0077522	.0086658	.0077839	.0078296
10,000	.0259710	.0305738	.0310162	.0270481	.0328126	.0294843
100,000	.3440276	.2267352	.2275991	.2158211	.2207434	.2469853
500,000	1.4644218	1.3990583	1.4145738	1.3759602	1.3966437	1.4101316
1,000,000	2.9780021	3.0199859	3.0032936	3.2067365	3.0268874	3.0469811



At first glance this graph may look like some sort of exponential function. However, we must first take into account that the beginning of the graph was more affected by constant time factors, and that with the scale it is hard to see. We must also consider that each unit in on the bottom is a 10 fold increase. Upon careful examination of the last few points, and double checking with the table of values, we see that multiplying the input by x will roughly multiply the time by x. So, when we take the log of time, we are left with a roughly linear function. This is good because our prediction for complexity was O(nlogn).

After plotting nlogn on Wolfram Alpha, it looks like my algorithm outgrows that function, so I don't know if I could give a k that will keep my function inside "little o" of nlogn. I think that our estimate was very good, but my only guess is that because I ran this in debug mode, after awhile that catches up to you and slows down the inputs of very large n.

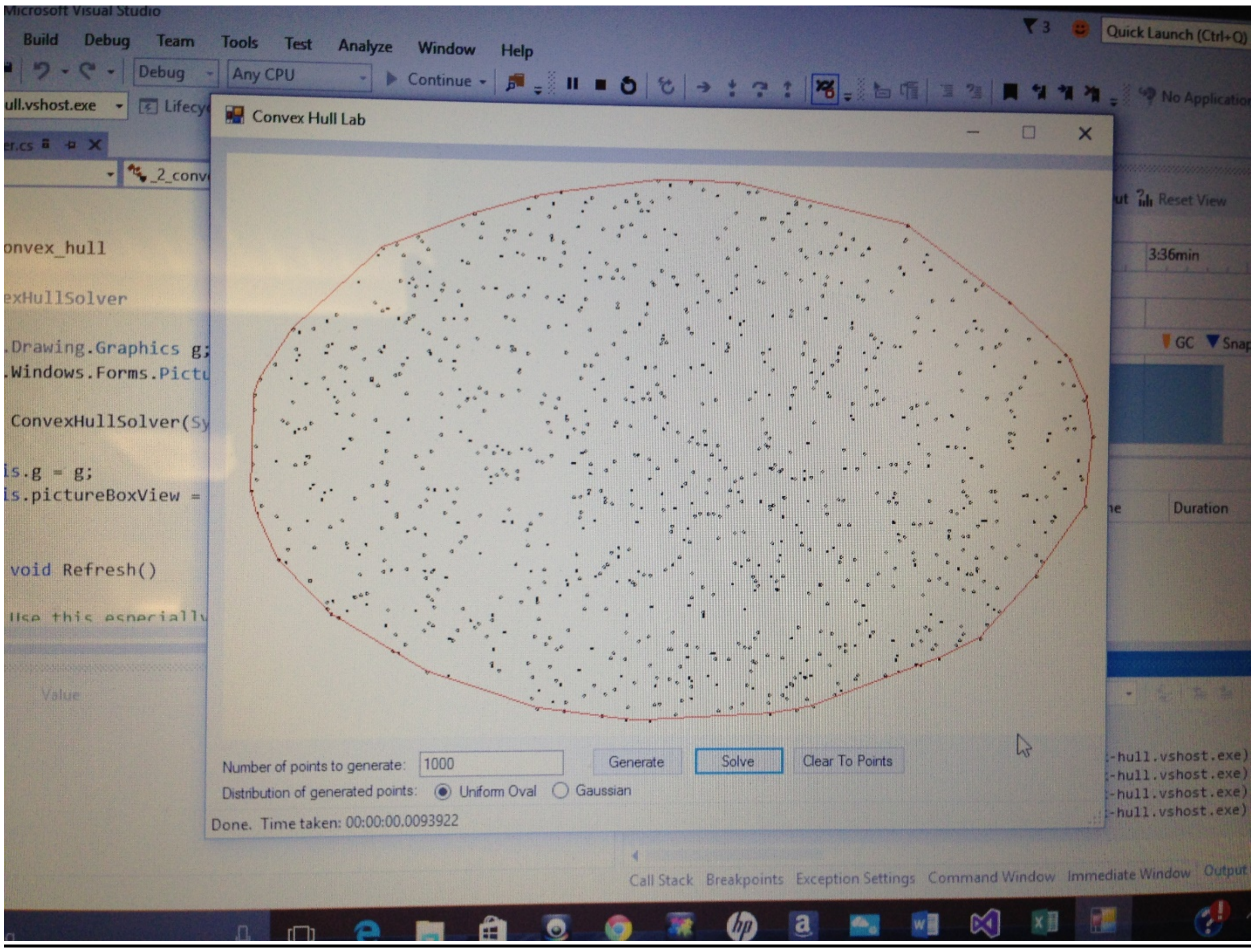
Screenshots



Number of points to generate: 100

Distribution of generated points: [✓ Uniform Oval] Gaussian

Done. Time taken: 00:00:00.0549172



```
onvex_hull

exHullSolver

.Drawing.Graphics g;
.Windows.Forms.Pictu

 ConvexHullSolver(Sy

is.g = g;
is.pictureBoxView =


 void Refresh()

Use this especially
```

Number of points to generate: 1000

Distribution of generated points: [✓ Uniform Oval] Gaussian

Done. Time taken: 00:00:00.0093922

```
:-hull.vshost.exe)
:-hull.vshost.exe)
:-hull.vshost.exe)
:-hull.vshost.exe)
```

Copy and Pasted Code

```
-------------Hull.cs---------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _1_convex_hull
{
    class Hull
    {
        List<System.Drawing.PointF> pointList;
        int rightIndex;

        public Hull(List<System.Drawing.PointF> pointList, int rightIndex)
        {
            this.pointList = pointList;
            this.rightIndex = rightIndex;
        }

        public List<System.Drawing.PointF> getList()
        {
            return pointList;
        }

        public int getRightIndex()
        {
            return rightIndex;
        }
    }
}



--------------ConvexHullSolver.cs----------------

using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Linq;
using _1_convex_hull;


namespace _2_convex_hull
{
    class ConvexHullSolver
    {
        System.Drawing.Graphics g;
        System.Windows.Forms.PictureBox pictureBoxView;
        public ConvexHullSolver(System.Drawing.Graphics g,
System.Windows.Forms.PictureBox pictureBoxView)
        {
            this.g = g;
            this.pictureBoxView = pictureBoxView;
        }

        public void Refresh()
        {
            // Use this especially for debugging and whenever you want to see what you
have drawn so far
            pictureBoxView.Refresh();
        }

        public void Pause(int milliseconds)
        {
            // Use this especially for debugging and to animate your algorithm slowly
            pictureBoxView.Refresh();
            System.Threading.Thread.Sleep(milliseconds);
        }


        //worst case O(nlogn)
        public void Solve(List<System.Drawing.PointF> pointList)
        {
            //sort points
            pointList = pointList.OrderByDescending(p => p.X).ToList();

            //call the recursive convexHull function, retrieve final list
            Hull finalHUll = convexHUll(pointList);
            pointList = finalHUll.getList();

            //prepare list and draw it
            PointF[] pointArray = pointList.ToArray();
            g.DrawPolygon(new Pen(Color.Red), pointArray);

        }

        //worst case O(logn) by itself
        public Hull convexHUll(List<System.Drawing.PointF> pointList)
        {
            //less than four is the size we are done dividing, and we make our small
hulls
            if (pointList.Count() < 4)
            {
                if (pointList.Count == 2)
                {
                    return new Hull(pointList, 1);
                }
                else
                {
                    //order the three points correctly, return the correct rightmost
index
                    if (slope(pointList[0], pointList[2]) > slope(pointList[0],
pointList[1]))
                    {
                        List<PointF> temp = new List<PointF>();
                        temp.Add(pointList[0]);
                        temp.Add(pointList[2]);
                        temp.Add(pointList[1]);

                        return new Hull(temp, 1);
                    }
                    else
                    {
                        return new Hull(pointList, 2);
                    }
                }
            }
            else
            {
                //since size is 4 or bigger, we divide in half, and merge the results
                int secondHalf = pointList.Count() - pointList.Count() / 2;

                return combine(convexHUll(pointList.GetRange(0, pointList.Count() / 2)),
                               convexHUll(pointList.GetRange(pointList.Count() / 2,
secondHalf)));
            }
        }


        //worst case O(n)
        public Hull combine(Hull left, Hull right)
        {
            //extract lists from hulls
            List<PointF> l = left.getList();
            List<PointF> r = right.getList();

            //set booleans, they come in handy
            bool bothDone = false;
            bool rightDone = false;
            bool leftDone = false;

            //those 4 ints abreviate topleftindex,toprightindex,bottomleftindex,and
bottomrightindex
            int tli = left.getRightIndex();
            int tri = 0;
            int bli = left.getRightIndex();
            int bri = 0;


            //find top edge, repeat this loop until no change in left or right index
            while (!bothDone)
            {
                bothDone = true;
                rightDone = false;
                leftDone = false;

                //move right index "up" until no advantage in slope change
                while (!rightDone)
                {
                    int newIndex;

                    if (tri == 0)
                        newIndex = r.Count() - 1;
                    else
                        newIndex = tri - 1;

                    if (slope(l[tli], r[tri]) > slope(l[tli], r[newIndex]))
                    {
                        tri = newIndex;
                        bothDone = false;
                    }
                    else
                    {
                        rightDone = true;
                    }
                }

                //move left index "up" until no advantage in slope change
                while (!leftDone)
                {
                    int newIndex;

                    if (tli == (l.Count() - 1))
                        newIndex = 0;
                    else
                        newIndex = tli + 1;

                    if (slope(l[tli], r[tri]) < slope(l[newIndex], r[tri]))
                    {
                        tli = newIndex;
                        bothDone = false;
                    }
                    else
                    {
                        leftDone = true;
                    }
                }

            }



            bothDone = false;
            //find bottom edge, repeat this loop until no change in left or right index
            while (!bothDone)
            {
                bothDone = true;
                rightDone = false;
                leftDone = false;

                //move right index "down" until no advantage in slope change
                while (!rightDone)
                {
                    if (bri == (r.Count() - 1))
                        rightDone = true;
                    else
                    {
                        if (slope(l[bli], r[bri]) < slope(l[bli], r[bri + 1]))
                        {
                            bri++;
                            bothDone = false;
                        }
```

```
                        else
                        {
                            rightDone = true;
                        }
                    }
                }

                //move left index "down" until no advantage in slope change
                while (!leftDone)
                {
                    if (bli == 0)
                        leftDone = true;
                    else
                    {
                        if (slope(l[bli], r[bri]) > slope(l[bli - 1], r[bri]))
                        {
                            bli--;
                            bothDone = false;
                        }
                        else
                        {
                            leftDone = true;
                        }
                    }
                }
            }

            //pass in the indeces we found
            return getNewHull(bli, bri, tri, tli, right.getRightIndex(), l, r);
        }

        //worst case O(n)
        public Hull getNewHull(int bli, int bri, int tri, int tli, int rightIndex,
List<PointF> l, List<PointF> r)
        {
            List<PointF> combinedList = new List<PointF>();

            //this will be the rightmost index of the combined hull
            int newRightIndex = 0;

            //this keeps track of what index to add to the new list
            int addCount = 0;

            //add from leftmost of left hull until the bottom left index
            while (addCount != (bli + 1))
            {
                combinedList.Add(l[addCount]);
                addCount++;
            }

            addCount = bri;

            //both of these while loops below add from bottom right index around the
right hull
                        //to the top right index.  It is because of a fringe case there
is an if/else
            //also the new rightmost index is discovered in this loop
            if (tri == 0)
            {
                while (addCount != r.Count())
                {
                    combinedList.Add(r[addCount]);
                    if (addCount == rightIndex)
                        newRightIndex = combinedList.Count() - 1;
                    addCount++;
                }

                combinedList.Add(r[0]);

            }
            else
            {
                while (addCount != (tri + 1))
                {
                    combinedList.Add(r[addCount]);
                    if (addCount == rightIndex)
                        newRightIndex = combinedList.Count() - 1;
                    addCount++;
                }
            }

            addCount = tli;

            if (addCount != 0)
            {
                //add from the top left index the rest of the left hull
                while (addCount != l.Count())
                {
                    combinedList.Add(l[addCount]);
                    addCount++;
                }
            }

            return new Hull(combinedList, newRightIndex);
        }


        //pretty self explanatory, this came in handy
        public double slope(PointF p1, PointF p2)
        {
            return (p2.Y - p1.Y) / (p2.X - p1.X);
        }



    }
}
```