Cody Kesler
CS 312 Section 2
Lab 2 – Convex Hull

Code:

```
#!/usr/bin/python3
from PyQt5.QtCore import QLineF, QPointF
import time

class ConvexHullSolver:
    def __init__( self, display ):
        self.points = None
        self.gui_display = display

    # Time Complex: O(n) Space Complexity: O(n/2)
    def findRightMostPoint(self, hull):
        rightmost = -10
        index = 0
        for i in range(len(hull)):
            if(hull[i].x() > rightmost):
                rightmost = hull[i].x()
                index = i
        return index

    # Time Complex: O(n) Space Complexity: O(n/2)
    def findLeftMostPoint(self, hull):
        leftmost = 10
        index = 0
        for i in range(len(hull)):
            if (hull[i].x() < leftmost):
                leftmost = hull[i].x()
                index = i
        return index

    # Time Complex: O(constant) Space Complexity: O(2)
    def angleBetween(self, point1, point2):
        return ((point2.y() - point1.y()) / (point2.x() - point1.x()))


    # Time Complex: O(c*n^2 = n^2) Space Complexity: O(n)
    def getUpperTangents(self, leftHull, rightHull, rightmostIndex, leftmostIndex):
        upperTangents = [0,0]
        curUpperLeftTangent = rightmostIndex
        curUpperRightTangent = leftmostIndex
        # Whol loop max O(n) times
        while(True):
            # Time Complex: O(n)
            newUpperRightTangent = self.upperRight(leftHull, rightHull,
                                                   curUpperLeftTangent, curUpperRightTangent)
             # Time Complex: O(n)
            newUpperLeftTangent = self.upperLeft(leftHull, rightHull,
                                                 curUpperLeftTangent, curUpperRightTangent)
            if (curUpperRightTangent == newUpperRightTangent and
                 curUpperLeftTangent == newUpperLeftTangent):
                break
            curUpperRightTangent = newUpperRightTangent
            curUpperLeftTangent = newUpperLeftTangent
        upperTangents[0] = curUpperRightTangent
        upperTangents[1] = curUpperLeftTangent
        return upperTangents
```

Cody Kesler
CS 312 Section 2
Lab 2 – Convex Hull

```
    # Time Complex: O(c*n = n) Space Complexity: O(n)
    def upperRight(self, leftHull, rightHull, rightmostIndex, leftmostIndex):
        i = 0

        # Time Complex: O(c)
        curAngle = self.angleBetween(leftHull[rightmostIndex],
                                 rightHull[leftmostIndex])
        # Repeats n times
        while(True):
             # Time Complex: O(c)
            nextAngle = self.angleBetween(leftHull[rightmostIndex],
                                 rightHull[(leftmostIndex+i+1) % len(rightHull)])
            if(nextAngle < curAngle):
                break
            i = i + 1
            curAngle = nextAngle
        return ((leftmostIndex+i))

    # Time Complex: O(c*n = n) Space Complexity: O(n)
    def upperLeft(self, leftHull, rightHull, rightmostIndex, leftmostIndex):
        # Time Complex: O(c)
        curAngle = self.angleBetween(leftHull[rightmostIndex],
                                 rightHull[leftmostIndex])
        i = 0
        localAngle = curAngle
        # Repeats n times
        while (True):
            # Time Complex: O(c)
            nextAngle = self.angleBetween(leftHull[((rightmostIndex - i - 1) %
                                 len(leftHull))], rightHull[leftmostIndex])
            if (nextAngle > localAngle):
                break
            i = i + 1
            localAngle = nextAngle
        return (rightmostIndex - (i)) % len(leftHull) # Time Complex: O(kn)



    # Time Complex: O(n) Space Complexity: O(5)
    def getLowerTangents(self, leftHull, rightHull, rightmostIndex, leftmostIndex):
        lowerTangents = [0,0]
        curLowerLeftTangent = rightmostIndex
        curLowerRightTangent = leftmostIndex
        # MAX O(n) for loop (Wont execute more than n times)
        while(True):
            # Time Complex: O(c*n)
            newLowerRightTangent = self.lowerRight(leftHull, rightHull,
                                      curLowerLeftTangent, curLowerRightTangent)
             # Time Complex: O(c*n)
            newLowerLeftTangent = self.lowerLeft(leftHull, rightHull,
                                      curLowerLeftTangent, curLowerRightTangent)
            if (curLowerRightTangent == newLowerRightTangent and
                 curLowerLeftTangent == newLowerLeftTangent):
                break
            curLowerRightTangent = newLowerRightTangent
            curLowerLeftTangent = newLowerLeftTangent
        lowerTangents[0] = curLowerRightTangent
        lowerTangents[1] = curLowerLeftTangent
        return lowerTangents
```

```
# Time Complex: O(c*n) Space Complexity: O(n)
def lowerRight(self, leftHull, rightHull, rightmostIndex, leftmostIndex):
     # Time Complex: O(c)
    curAngle = self.angleBetween(leftHull[rightmostIndex],
                     rightHull[leftmostIndex])
    i = 0
    # Repeats n times
    while(True):
         # Time Complex: O(c)
        nextAngle = self.angleBetween(leftHull[rightmostIndex],
                     rightHull[((leftmostIndex -i - 1) % len(rightHull))])
        if(nextAngle > curAngle):
            break
        i = i + 1
        curAngle = nextAngle
    return ((leftmostIndex - i)) % len(rightHull) # Time Complex: O(kn)

# Time Complex: O(c*n) Space Complexity: O(n)
def lowerLeft(self, leftHull, rightHull, rightmostIndex, leftmostIndex):
    # Time Complex: O(c)
    curAngle = self.angleBetween(leftHull[rightmostIndex],
                     rightHull[leftmostIndex])
    i = 0
     # Repeats n times
    while (True):
         # Time Complex: O(c)
        nextAngle = self.angleBetween(leftHull[((rightmostIndex + i + 1) %
                     len(leftHull))], rightHull[leftmostIndex])
        if (nextAngle < curAngle):
            break
        i = i + 1
        curAngle = nextAngle
    return ((rightmostIndex + i) % len(leftHull)) # Time Complex: O(kn)
```

Cody Kesler
CS 312 Section 2
Lab 2 – Convex Hull

```
# Time Complex: O(5n = n) Space Complexity: O(n+6)
    def mergeHulls(self, leftHull, rightHull):
        # Time Complex O(n)
        rightmostIndex = self.findRightMostPoint(leftHull)
        # Time Complex O(n)
        leftmostIndex = self.findLeftMostPoint(rightHull)
        # Time Complex O(n)
        uppers = self.getUpperTangents(leftHull, rightHull, rightmostIndex,
                          leftmostIndex)
        rightUpper = uppers[0]
        leftUpper = uppers[1]
        # Time Complex O(n)
        lowers = self.getLowerTangents(leftHull, rightHull,rightmostIndex,
                          leftmostIndex)
        rightLower = lowers[0]
        leftLower = lowers[1]
        newHullPoints = list()

        # Repeats at most n
        for i in range(len(leftHull)):
            newHullPoints.append(leftHull[i]) O(1)
          # Repeats at most n/2
            if((i % len(leftHull)) == leftUpper):
                for j in range(rightUpper, len(rightHull)+rightUpper):
                    newHullPoints.append(rightHull[j % len(rightHull)]) O(1)
                   # Repeats at most n/2
                    if ((j % len(rightHull)) == rightLower):
                        for k in range(leftLower, len(leftHull)+leftLower):
                            if(k % len(leftHull) == 0):
                                break
                            newHullPoints.append(leftHull[k % len(leftHull)])
                        break
                break
        return newHullPoints

    # Time Complex O(log(n)n) Space Complexity O(n)
    def convexHullRecurse(self, points):
        if (len(points) == 3):
            hull = [QLineF(points[i], points[(i + 1) % len(points)]) for i in
                          range(len(points))]

            assert (type(hull) == list and type(hull[0]) == QLineF)
            self.gui_display.addLines(hull, (255, 0, 0))
            returnList = [points[i] for i in range(len(points))]
            if(self.angleBetween(returnList[0],returnList[1]) <
                 self.angleBetween(returnList[0],returnList[2])):
                temp = returnList[1]
                returnList[1] = returnList[2]
                returnList[2] = temp
            return returnList
        elif (len(points) == 2):
            hull = [QLineF(points[i], points[(i + 1) % len(points)]) for i in
                 range(len(points)-1)]
            assert (type(hull) == list and type(hull[0]) == QLineF)
            self.gui_display.addLines(hull, (255, 0, 0))
            return [points[i] for i in range(len(points))]
```

```
        #Repeats log(n) times
        leftHull = self.convexHullRecurse(points[:(len(points) // 2)])
        rightHull = self.convexHullRecurse(points[(len(points) // 2):])

        #Time Complex: O(n)
        mergedHull = self.mergeHulls(leftHull, rightHull)
        return mergedHull



    # Time Complex O(log(n)n) Space Complexity: O(2n)
    def compute_hull( self, unsorted_points ):
        assert( type(unsorted_points) == list and
                type(unsorted_points[0]) == QPointF )

        n = len(unsorted_points)
        print( 'Computing Hull for set of {} points'.format(n) )

        t1 = time.time()
        unsorted_points.sort(key=lambda p: p.x())
        t2 = time.time()
        print('Time Elapsed (Sorting): {:3.3f} sec'.format(t2-t1))

        t3 = time.time()

        # Time Complex O(log(n)n)
        newHullPoints = self.convexHullRecurse(unsorted_points)
        hull = [QLineF(newHullPoints[i], newHullPoints[(i + 1) % len(newHullPoints)])
                for i in range(len(newHullPoints))]

        t4 = time.time()

        if(hull is None):
            hull = [QLineF(unsorted_points[i], unsorted_points[(i + 1) % 3])
                for i in range(3)]
            assert (type(hull) == list and type(hull[0]) == QLineF)
            assert (type(hull) == list and type(hull[0]) == QLineF)

        self.gui_display.addLines(hull, (0, 0, 255))


        print('Time Elapsed (Convex Hull): {:3.3f} sec'.format(t4-t3))
        self.gui_display.displayStatusText('Time Elapsed (Convex Hull): {:3.3f}
                sec'.format(t4-t3))

        # refresh the gui display
        self.gui_display.update()
```

Cody Kesler
CS 312 Section 2
Lab 2 – Convex Hull

Complexity:

Time:

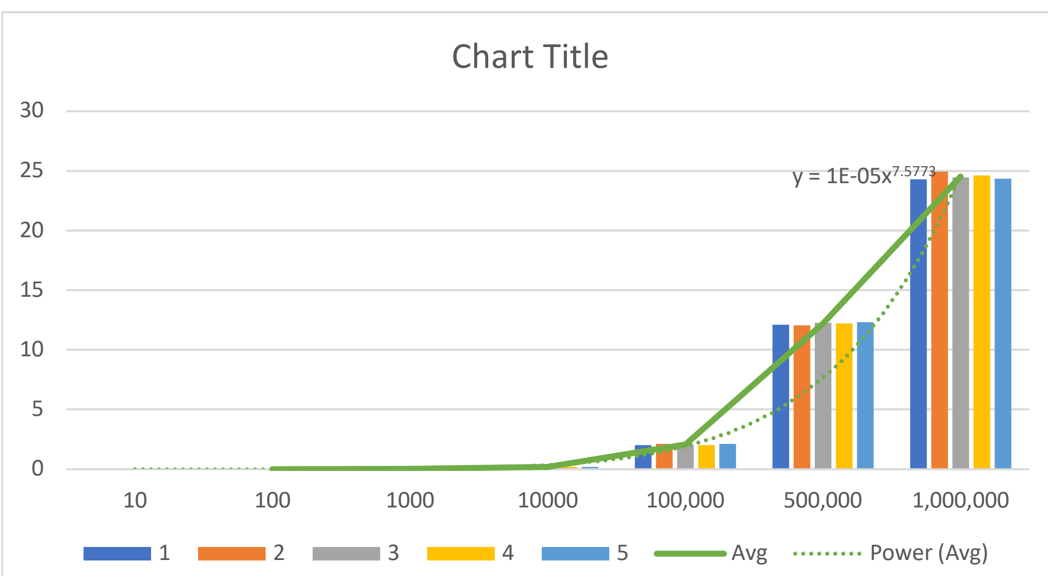
My algorithm divides the n into two different problems of which are size n/2. Thus a =2, b = 2 and d = 1 since combining the hulls is linear. Thus, by the master theorem 2/2^1 = 1 so the theoretical time complexity is O(n*log(n)).

Space:

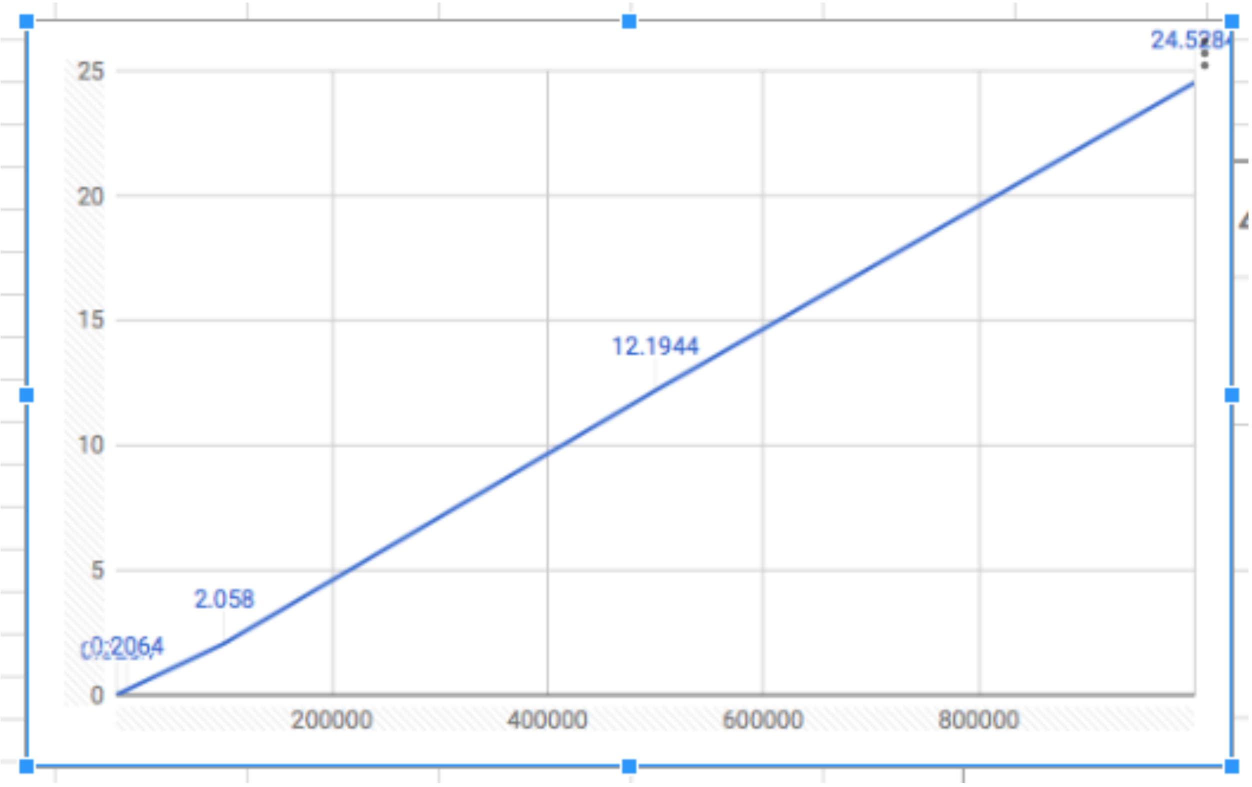
The maximum space that it taken with the program is n number of dots plus the new hull of the border dots made in the merge hull function which cannot be more than n. Thus, the max space complexity is 2n.

Empirical Data:

	Trials					
n	1	2	3	4	5	Avg
10	0	0	0	0	0	0
100	0.003	0.003	0.003	0.003	0.003	0.003
1000	0.025	0.027	0.024	0.024	0.027	0.0254
10000	0.204	0.206	0.201	0.214	0.207	0.2064
100,000	2.018	2.12	2.032	2.015	2.105	2.058
500,000	12.119	12.068	12.25	12.219	12.316	12.1944
1,000,000	24.321	24.925	24.458	24.601	24.337	24.5284



Cody Kesler
CS 312 Section 2
Lab 2 – Convex Hull



Discussion:

The first table and the graph just show the data straight up. The second graph though is determined on a scale of log(n). As is apparent the graph is a linear graph on a log scale showing that it is a n*log(n) time complex.

Real vs Theoretical:

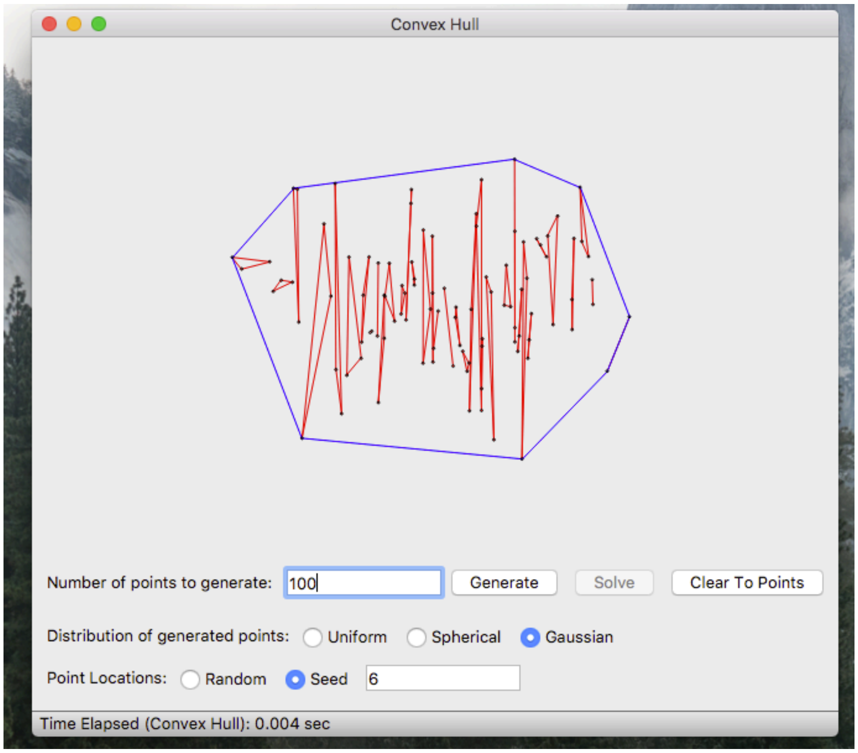
Finding the constant of proportionality:
time = k*nlog(n)
.2064 = k*10,000(log(10,000)) = 40,000
K = 0.00000516
2.058 = k*100,000(log(100,000)) = 100,000
K= 0.00002058
12.1944 = k*500,000(log(500,000)) = 500,000
K= 0.0000243888
24.5284 = k*1,000,000(log(1,000,000)) = 6,000,000
K= 0.000004088066667

Average k = 0.000013554216667 = 1.36 * 10^6
Since the constant of proportionality is so small, the difference of the theoretical and empirical estimates of time complexity is almost negligible.

Cody Kesler
CS 312 Section 2
Lab 2 – Convex Hull

Pictures:

100:



1000: