Convex Hull Report

1. Pseudocode (See appendix for full code) — Complexity

Pseudocode	Complexity
Sort points by x value	nlogn
Call recursive hull solver on points	
Def hull solver:	
If less than 4 points:	
Make hull by connecting all points	O(1) because max of 3 points
Sort by slope	O(1) because sorting 2-3 points
Return hull	
Else:	
Split points in half	
Lefthull = Recursive call on left half	O(logn) will make log2n calls
rightHull = Recursive call on right half	O(logn) will make log2n calls
find index of right most point in left half	n
find index of left most point in right half	n
create line connecting these points	O(1)
loop while moving line up	worst case n
moving = False	
find next point on right	
update line	
if new line slope > old line slope:	
moving = True	
update index and line	
“repeat for left side”	
“repeat for bottom line”	worst case n
NewHull = topline points	
While toprightIndex != bottomrightIndex:	worst case n
Add point from right hull at index	
NewHull append bottom line points	
While bottmLeftIndex != topLeftIndex:	worst case n
Add point from left hull at index	
NewHull = line from points[i] to points[i+1] for i in range len(points)	Worst case n
Return NewHull	

2. Theoretical analysis of time and space complexity

Time Complexity:

O(nlogn + nlogn*(8n)) simplifies to O(n*logn) by max rule for asymptotic complexity.

The recursion will generate logn smaller problems. Solving each problem will take a worst case of n*constant factor so overall this is a nlogn complexity.

Master Theorem

Recurrence Relation: t(n) = at(n/b) + O(n^d)

For this algo: T(n) = 2t(n/2) + O(n^1), a = 2, b = 2, d = 2

Solve: a/b^d = 2/2 == 1 therefore by the master theorem, the overall complexity will be O(n^d*logn) or **O(n*logn) in this case**

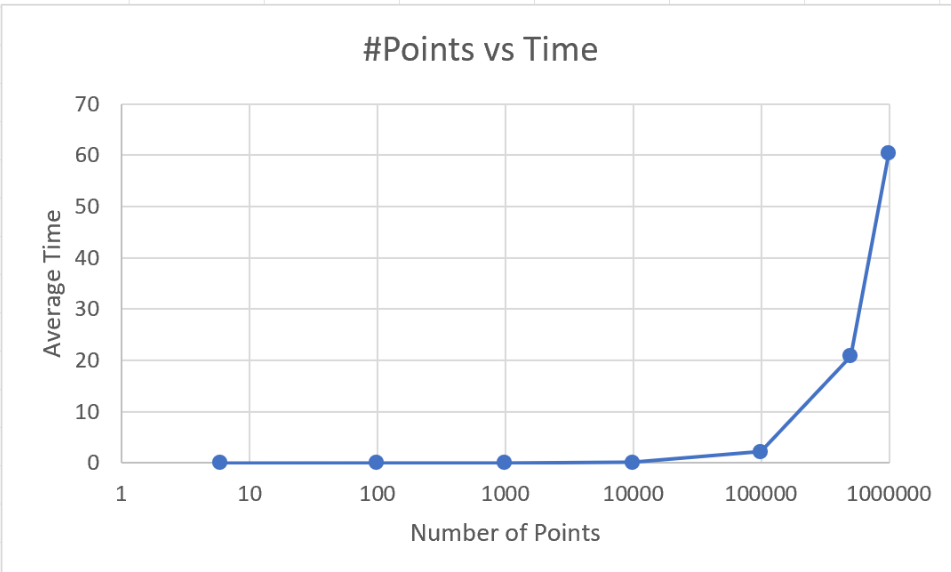
Space Complexity:

O(n) for the number of points. Each recursive call stores a part of the overall list. Logn calls would be dominated by the storing of the total list in the top layer so space complexity is O(n).

3. Experimental Analysis

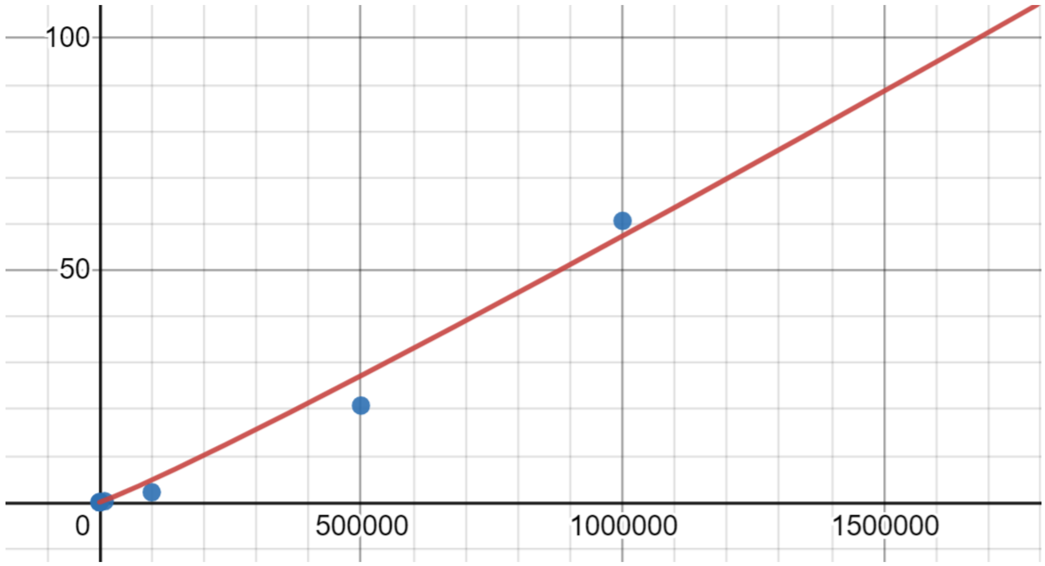
Table of Experimental results and plot of growth rate

# points	test 1	test 2	test 3	test 4	test 5	avg time
6	0.001	0	0.001	0	0	0.00100
100	0.005	0.005	0.005	0.004	0.006	0.00500
1000	0.029	0.025	0.027	0.024	0.026	0.02620
10000	0.213	0.215	0.226	0.231	0.226	0.22220
100000	2.007	2.104	2.222	2.066	2.392	2.15820
500000	17.952	17.427	20.396	23.854	24.505	20.82680
1000000	62.297	60.951	59.943	60.243	59.801	60.64700



This Plot seems to show nLogn growth although there is an increase at higher numbers because of overhead with running the algorithm. It is hard to compare the early numbers with the larger numbers and so this makes me assume that there is something else going on with the larger numbers to make them so much larger. I did analysis using some line fitting software to find the best line fit the following algorithm fit the best:

y = 0.000009562869*x*log(x)



This suggests a constant of proportionality of about 1/105000 = 0.00000952380952381.

I did this analysis with a log base 10 as changing the base is a constant time change.

4. Theoretical vs Empirical Analysis

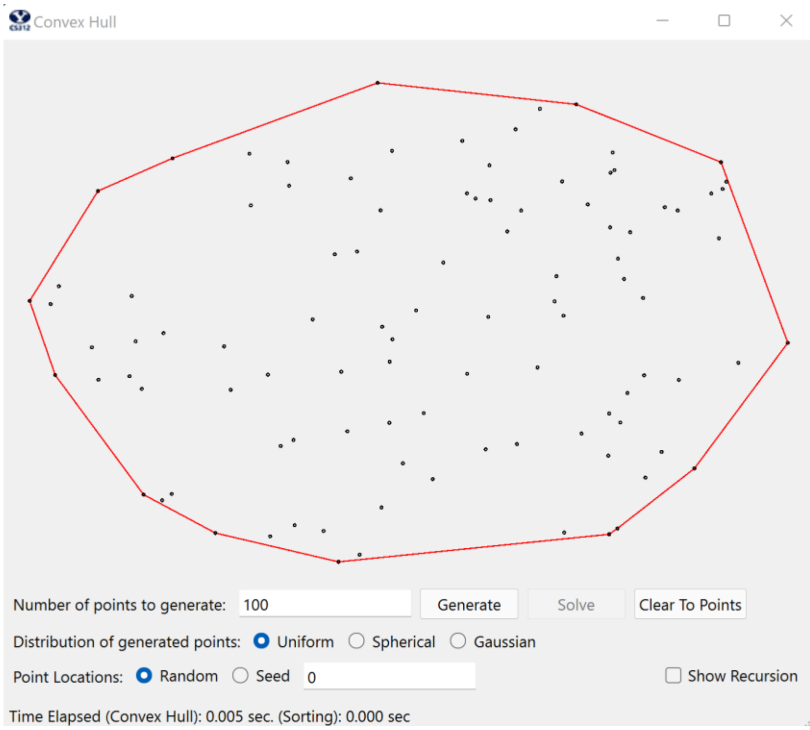
The main difference between the theoretical and empirical analysis is the constant of proportionality. I think this constant comes from being very far off the worst case time for most small numbers. As can be seen in the graph, the empirical analysis shows an increase at higher numbers. I think that at the higher values, the empirical time is starting to approach more the theoretical time. Eventually this curve would straighten out again but still be asymptotically less than the theoretical analysis.

The Theoretical analysis suggests that there would be a rather large constant factor especially as the time to combine the two hulls is more like 8n than just n. This is reflected more in the larger values (>100000) and not seen as much in the smaller n.

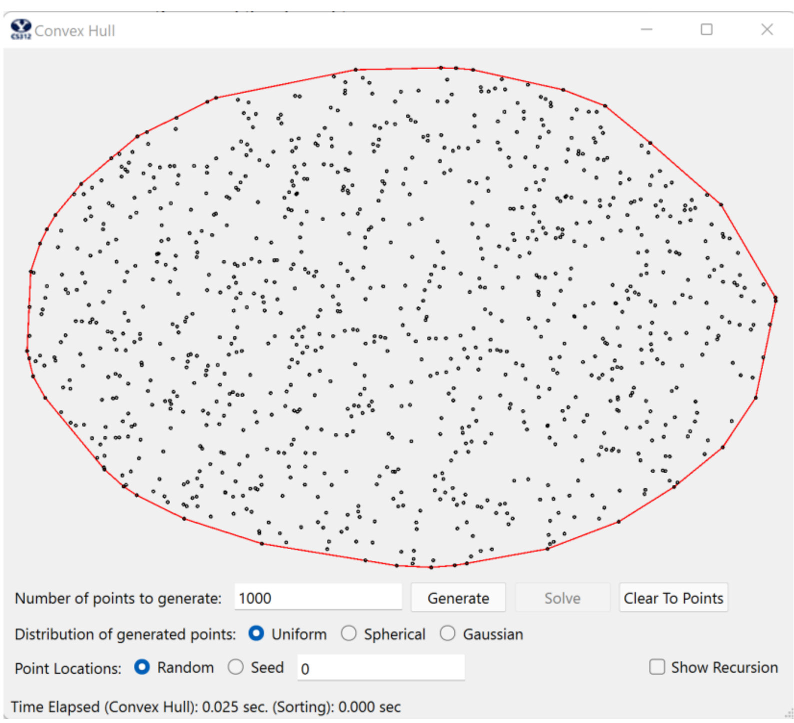
This could also be due to the cpu schedular on my laptop choosing to give the algorithm less time as the size of n increases, or having more interruptions which would account for the increase at higher numbers.

5. Example Screen Shots

100 point example



1000 points example



Appendix: Full Code

```
# This is the method that gets called by the GUI and actually executes
# the finding of the hull
def compute_hull(self, points, pause, view):
    self.pause = pause
    self.view = view
    assert (type(points) == list and type(points[0]) == QPointF)

    t1 = time.time()
    # Sort points by x value
    points.sort(key=lambda x: x.x())
    t2 = time.time()

    t3 = time.time()
    polygon = self.convex_hullDC(points)
    t4 = time.time()

    # when passing lines to the display, pass a list of QLineF objects.  Each
QLineF
    # object can be created with two QPointF objects corresponding to the
endpoints
    self.showHull(polygon, RED)
    self.showText('Time Elapsed (Convex Hull): {:3.3f} sec. (Sorting):
{:3.3f} sec'.format(t4 - t3, t2 - t1))

def convex_hullDC(self, points):
    if len(points) <= 3:
        polygon = [QLineF(points[i], points[(i + 1) % len(points)]) for i in
range(len(points))]
        polygon.sort(key=lambda x: self.getLineSlope(x))  # might not be
working as order of points is not fixed?
        return polygon
    else:
        mid = len(points) // 2
        leftHull = self.convex_hullDC(points[:mid])
        rightHull = self.convex_hullDC(points[mid:])

        # Merge left and right
        # Get starting points and indices
        leftPoint, leftIndex = self.rightMostPoint(leftHull)
        rightPoint, rightIndex = self.leftMostPoint(rightHull)

        topLeftIndex = leftIndex
        bottomLeftIndex = leftIndex
        topRightIndex = rightIndex
        bottomRightIndex = rightIndex

        # Create and show connecting line
        topConnLine = QLineF(leftPoint, rightPoint)
        # self.showHull(leftHull.copy(), RED) # this was changing length of
leftHull....
        # self.showHull(rightHull.copy(), RED)
        # self.showTangent([topConnLine], BLUE)

        moving = True
        while moving:
```

```
            moving = False
            # self.eraseTangent([topConnLine])
            nextRight = rightHull[(topRightIndex + 1) %
len(rightHull)].pointAt(0)
            nextLine = QLineF(topConnLine.pointAt(0), nextRight)
            if self.getLineSlope(nextLine) > self.getLineSlope(topConnLine):
                moving = True
                topRightIndex += 1
                topConnLine = nextLine
            # self.showTangent([topConnLine], BLUE)

            # self.eraseTangent([topConnLine])
            nextLeft = leftHull[(topLeftIndex - 1) %
len(leftHull)].pointAt(0)
            nextLine = QLineF(nextLeft, topConnLine.pointAt(1))
            if self.getLineSlope(nextLine) < self.getLineSlope(topConnLine):
                moving = True
                topLeftIndex -= 1
                topConnLine = nextLine
        # self.showTangent([topConnLine], BLUE)

        bottomConnLine = QLineF(leftPoint, rightPoint)
        moving = True
        while moving:
            moving = False
            # self.eraseTangent([bottomConnLine])
            nextRight = rightHull[(bottomRightIndex - 1) %
len(rightHull)].pointAt(0)
            nextLine = QLineF(bottomConnLine.pointAt(0), nextRight)
            if self.getLineSlope(nextLine) <
self.getLineSlope(bottomConnLine):
                moving = True
                bottomRightIndex -= 1
                bottomConnLine = nextLine
            # self.showTangent([bottomConnLine], BLUE)

            # self.eraseTangent([bottomConnLine])
            nextLeft = leftHull[(bottomLeftIndex + 1) %
len(leftHull)].pointAt(0)
            nextLine = QLineF(nextLeft, bottomConnLine.pointAt(1))
            if self.getLineSlope(nextLine) >
self.getLineSlope(bottomConnLine):
                moving = True
                bottomLeftIndex += 1
                bottomConnLine = nextLine
        # self.showTangent([bottomConnLine], BLUE)

    # Construct new hull
    newHullPoints = [topConnLine.pointAt(0)]
    nextPoint = topConnLine.pointAt(1)
    while nextPoint != bottomConnLine.pointAt(1):
        newHullPoints.append(nextPoint)
        nextPoint = rightHull[(topRightIndex + 1) %
len(rightHull)].pointAt(0)
        topRightIndex += 1
    newHullPoints.append(bottomConnLine.pointAt(1))
    nextPoint = bottomConnLine.pointAt(0)
```

```
    while nextPoint != topConnLine.pointAt(0):
        newHullPoints.append(nextPoint)
        nextPoint = leftHull[(bottomLeftIndex + 1) %
len(leftHull)].pointAt(0)
        bottomLeftIndex += 1
    newHull = [QLineF(newHullPoints[i], newHullPoints[(i + 1) %
len(newHullPoints)]) for i in range(len(newHullPoints))]
    # self.showHull(newHull.copy(), RED)
    return newHull

def getLineSlope(self, line):
    if line.dx() == 0:
        return 0
    return line.dy()/line.dx()

def rightMostPoint(self, hull):
    rightPoint = hull[0].pointAt(0)
    hullIndex = 0
    for i in range(len(hull)):
        if hull[i].x1() > rightPoint.x():
            rightPoint = hull[i].pointAt(0)
            hullIndex = i
    return rightPoint, hullIndex

def leftMostPoint(self, hull):
    leftPoint = hull[0].pointAt(0)
    hullIndex = 0
    for i in range(len(hull)):
        if hull[i].x1() < leftPoint.x():
            leftPoint = hull[i].pointAt(0)
            hullIndex = i
    return leftPoint, hullIndex
```