Emily Morrow
Section 2
Convex Hull

1. [40] Correct functioning code to accomplish Convex Hull using a divide and conquer scheme. Include your documented source code.

```
# finds slope between two lines
def slope(point1, point2):
   m = (point2.y() - point1.y()) / (point2.x() - point1.x())
   return m

# returns the right-most point in a hull
def getRightPoint(hull):
   right = None

   for curr in hull:
      right = curr
      if curr.x() > right.x():
         right = curr
   return right

# returns the left-most point in a hull
def getLeftPoint(hull):
   left = None
   for curr in hull:
      left = curr
      if curr.x() < left.x():
         left = curr
   return left

def rotateList(l, n):
   return l[n:] + l[:n]

# helper function to find the highest left point, highest right point
# and the lowest left point and lowest right point
def findConnectingPoints(hull, fixedPoint, isLessThan):
   lowestPoint = None
   if isLessThan:
      slope = 10000000000000000
      for y in hull:
         currSlope = ConvexHullSolverThread.slope(y, fixedPoint)
         if currSlope < slope:
            slope = currSlope
            lowestPoint = y

   else:
      slope = -1000000000000000
      for y in hull:
         currSlope = ConvexHullSolverThread.slope(y, fixedPoint)
         if currSlope > slope:
            slope = currSlope
            lowestPoint = y
   return lowestPoint


# removes the points in between the upper common tangent and
# lower common tangent so two hulls can be connected
def removePoints(highPoint, lowPoint, hull, isLeft):
   i = hull.index(highPoint)
   j = hull.index(lowPoint)
```

Emily Morrow
Section 2
Convex Hull

```
    if isLeft:
        # rotates the list representation of the hull so the low point
        # is the beginning of the list
        rotatedHull = ConvexHullSolverThread.rotateList(hull,j)
        if len(rotatedHull) == 3:
            p1 = rotatedHull[1]
            p2 = rotatedHull[2]
            if p1.y() > p2.y():
                rotatedHull[1] = p2
                rotatedHull[2] = p1


        i = rotatedHull.index(highPoint)
        return rotatedHull[:i+1]

    else:
        # rotates the list representation of the hull so the high point
        # is the beginning of the list
        rotatedHull = ConvexHullSolverThread.rotateList(hull,i)
        if len(rotatedHull) == 3:
            p1 = rotatedHull[1]
            p2 = rotatedHull[2]
            if p1.y() < p2.y():
                rotatedHull[1] = p2
                rotatedHull[2] = p1

        j = rotatedHull.index(lowPoint)
        return rotatedHull[:j+1]


# splits the list of points in half until there are only single
# points then it calls connect to recursively connect the points
def split(points, self):
    if len(points) == 1:
        return points

    left = points[:len(points)//2]
    right = points[len(points)//2:]

    lHull = ConvexHullSolverThread.split(left,self)
    rHull = ConvexHullSolverThread.split(right,self)

    return ConvexHullSolverThread.connect(lHull, rHull, self)


# connects the left hull to the right hull
def connect(lHull, rHull, self):
    if len(lHull) == 1:
        combined = lHull + rHull

    else:
        # get right-most point of left hull
        # get left-most point of right hull
        lPoint = ConvexHullSolverThread.getRightPoint(lHull)

        rPoint = ConvexHullSolverThread.getLeftPoint(rHull)

        # fix right point and check it w points on lHull going counter-clockwise
        # to find lowest slope
        highestLPoint =
ConvexHullSolverThread.findConnectingPoints(lHull,rPoint,True)
```

Emily Morrow
Section 2
Convex Hull

```
      # fix the point found on left and check it w points on rHull going clockwise
      # to find highest slope
      highestRPoint =
ConvexHullSolverThread.findConnectingPoints(rHull,highestLPoint,False)

      # fix right point and check it w points on lHull going clockwise
      # to find highest slope
      lowestLPoint =
ConvexHullSolverThread.findConnectingPoints(lHull,rPoint,False)

      # fix the point found on left and check it w points on rHull going counter-
clockwise
      # to find lowest slope
      lowestRPoint =
ConvexHullSolverThread.findConnectingPoints(rHull,lowestLPoint,True)


      # get rid of points in-between the upper common tangent and the lower common
tangent
      lHull = ConvexHullSolverThread.removePoints(highestLPoint, lowestLPoint,
lHull, True)
      rHull = ConvexHullSolverThread.removePoints(highestRPoint, lowestRPoint,
rHull, False)

      indexLTop = lHull.index(highestLPoint)
      indexLBottom = lHull.index(lowestLPoint)

      # since the lHull and rHull got rotated in removePoints() so the lHull starts
with
      # the low point and ends with the high point and the rHull starts with the
high point and ends
      # with the low point they can be added together to create the new hull
      combined = lHull + rHull

      # some concave lines would still show up so this method
      # double checks and gets rid of concave lines
      topIndex = len(combined)
      pointsToRemove = []
      # go through the list and draw a line between one point and the next next
point
      # and if the point in between those two points is on the inside of the line
      # then remove the intermediate point from the hull
      for y in range(0,topIndex):
         if y == len(combined)-2:
            A = combined[y]
            P = combined[y+1]
            B = combined[0]
         elif y == len(combined)-1:
            A = combined[y]
            P = combined[0]
            B = combined[1]
         else:
            A = combined[y]
            P = combined[y+1]
            B = combined[y+2]

         d = (P.x()-A.x())*(B.y()-A.y())-(P.y()-A.y())*(B.x()-A.x())
         if d > 0:
            pointsToRemove.append(P)
```

```
    for y in pointsToRemove:
        combined.remove(y)

return combined
```

2. [15] Explain the time and space complexity of your algorithm by showing and summing up the complexity of each subsection of your code Also, include your theoretical analysis for the entire algorithm including discussion of the recurrence relation.

   In the Split function the points get recursively split in half until each hull is only 1 point. This function will run in O(logn) time but in each Split function the connect function gets called as well so we'll need to take that into account. In the connect function we'll look at the time complexity of every function included in it.

   The getRightPoint function runs y times where y is the number of points in the left hull. In the worst case scenario y can approach n (total number of points) when we're working with bigger hulls. The same will be true for getLeftPoint so each of these run in about O(n) time.

   The findConnectingPoint function will run a for loop on every point on a hull which like the above function will run in O(n) time worst case scenario.

   The removePoints function will run in O(1) time because it's just some if statements.

   The last function in the connect function will also run in O(n) time since it goes through each point in the hull.

   This makes connect O(n) + O(n) + O(n) + O(n) + O(n) + O(n) + O(1) + O(n) = O(n). Since connect gets run in each call to Split then the total complexity of the algorithm is O(nlogn)

   Recurrence relation is $T(n) = 2T(n/2) + n$

   Using Master's Theorem where $a = 2$, $b = 2$, $d = 1$ we get O(nlogn)

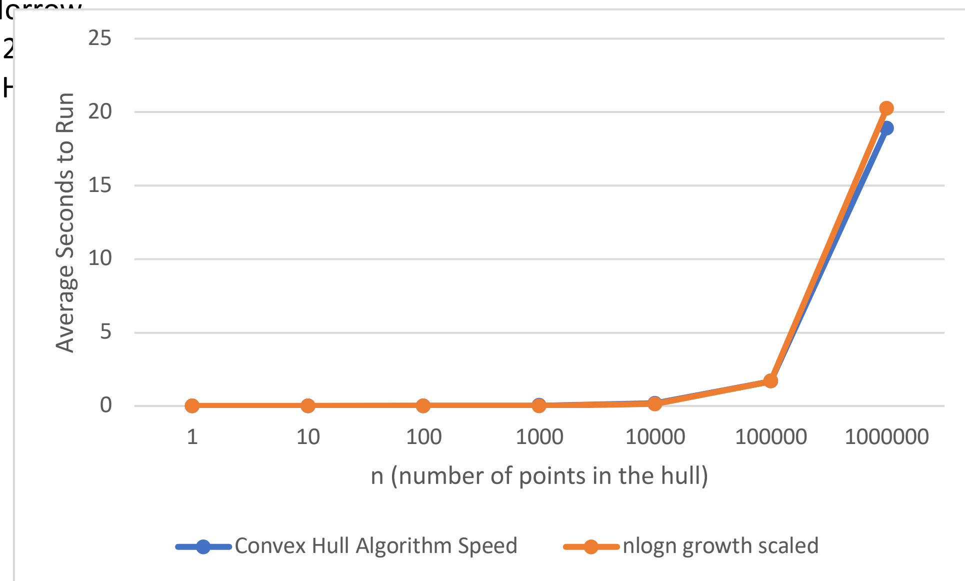
3. [15] Include your raw and mean experimental outcomes, plot, and your discussion of the pattern in your plot. Which order of growth fits best? Give an estimate of the constant of proportionality. Include all work and explain your assumptions.

   To scale the graph of nlogn I multiplied the values by 0.00000146618 which I got by solving 1151292x = 1.688 where 1151292 is the value of nlogn for 100000 points and 1.688 is the value of my algorithm for 100000 points.

   The following data is the data that is included in my line graph. The convex hull algorithm speed was found by taking the average of 3 runs with each number of points.

	Convex Hull Algorithm Speed	nlogn growth scaled
1	0	0
10	0	0
100	0.002	0.001
1000	0.021333333	0.01
10000	0.179	0.135
100000	1.688	1.688
1000000	18.888	20.256

Emily Morrow
Section 2
Convex H

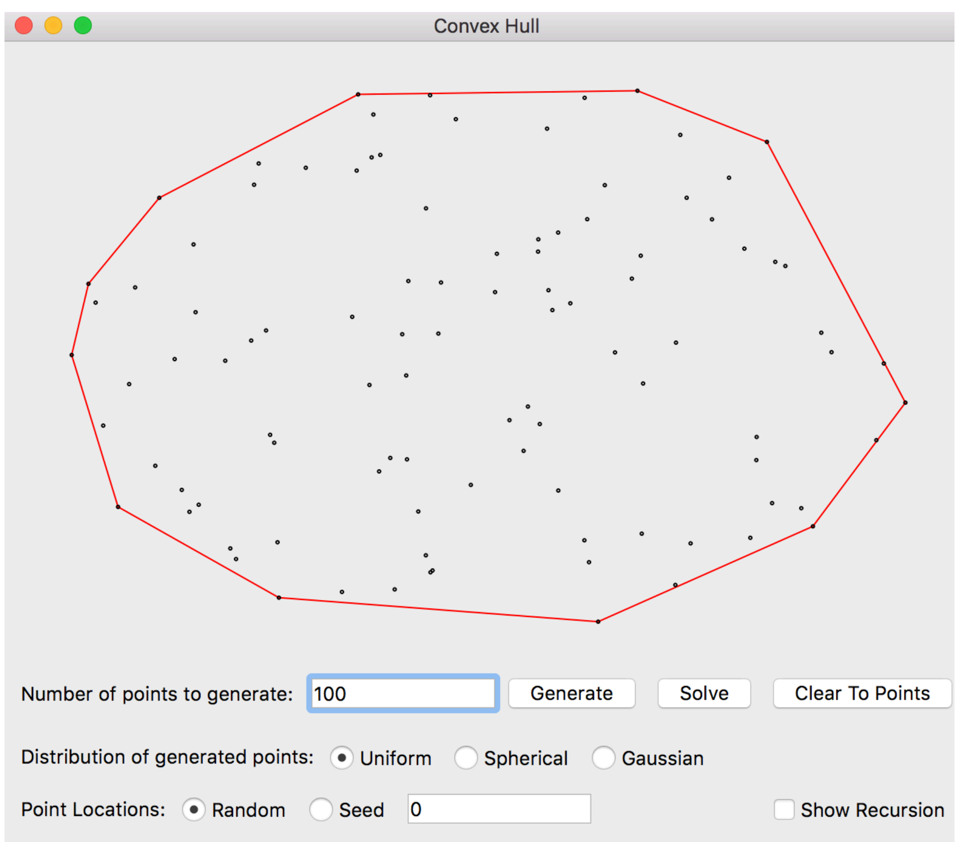


The convex hull algorithm line is close to the line of nlogn growth.

4. [10] Discuss and explain your observations with your theoretical and empirical analyses, including any differences seen.

   The theoretical complexity I found was nlogn and when I looked through my code and analyzed the time complexity I found that it was also nlogn. There is a slight difference (in above graph) between my convex hull algorithm speed and my scaled nlogn but the difference is minimal so it could just be due to constants that weren't taken into account when talking about big-O.

5. [10] Include a correct screenshot of an example with 100 points and a screenshot of an example with 1000 points. You can capture the image of the window using the **ctrl-alt-shift-PrtSc** facility for capturing an image of the window in focus.



Emily Morrow
Section 2
Convex Hull