Time complexity:

The time complexity of this code is O(nlogn). I’ll demonstrate my reasoning with pseudocode. Given the below pseudocode, the largest complexity is O(nlogn), therefore the total complexity is O(nlogn)

1. Sort list of points (when using quicksort or another efficient algorithm O(nlogn))
2. Divide list in half recursively, and at leaves, execute the following code O(nlogn)
   1. Create tiny hull O(1)
   2. Merge hulls together O(1)

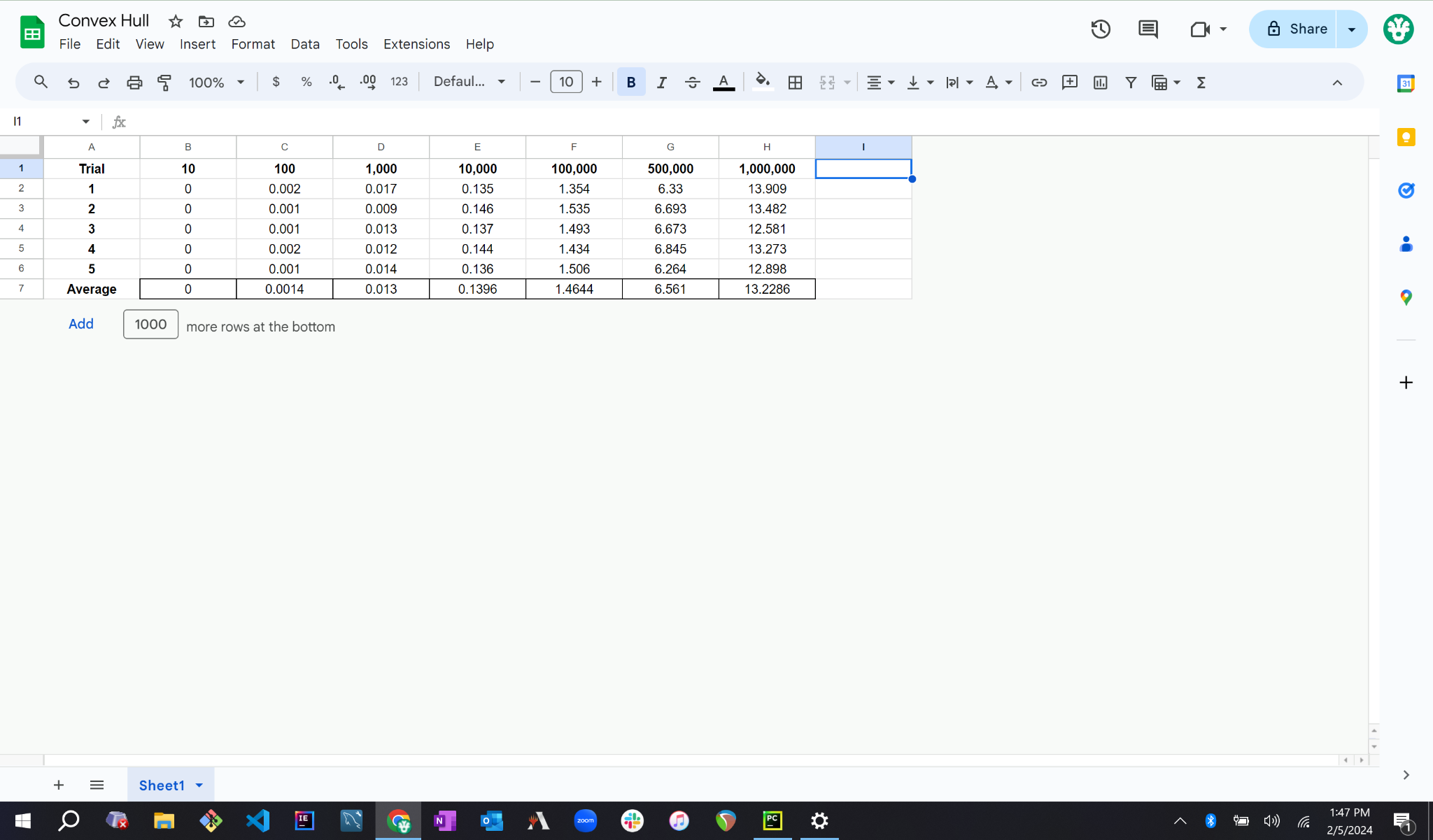
Space complexity:

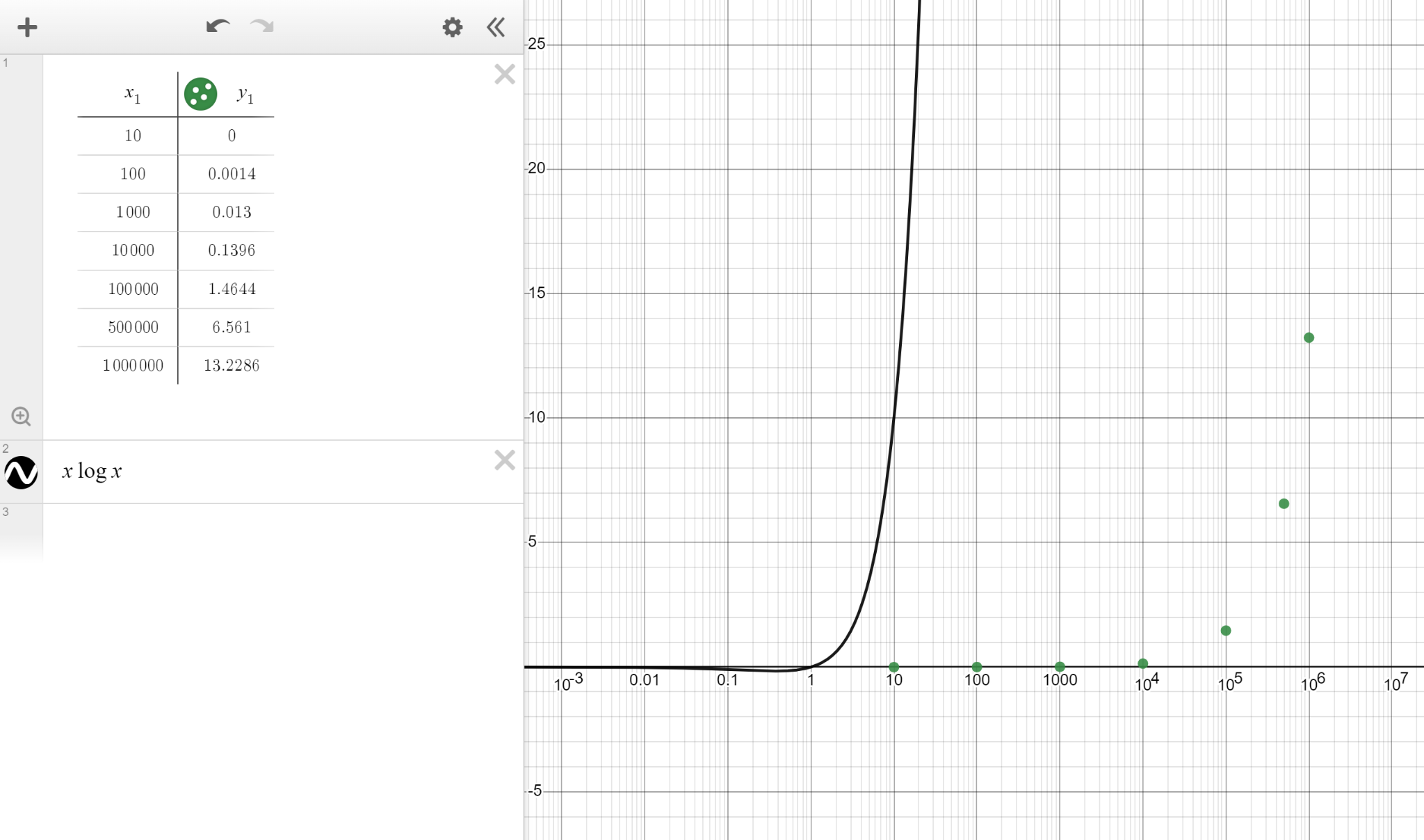
The space complexity of my code is O(nlogn). I’ll demonstrate my reasoning with pseudocode. Given the below pseudocode, the largest complexity is O(nlogn), therefore the total complexity is O(nlogn)

1. List of all points O(n)
2. Store list of 2 halves of the points at each recursive level O(nlogn)
3. Merge back into another list O(n)

Experimental times

1. My experimental times matched up almost exactly with theoretical times. O(nlogn) it is. Probably a teensy slower because my computer is trash.
2. Here are each of my experimental times. I ran the test and recorded the time. It took a really long time. But there it is!

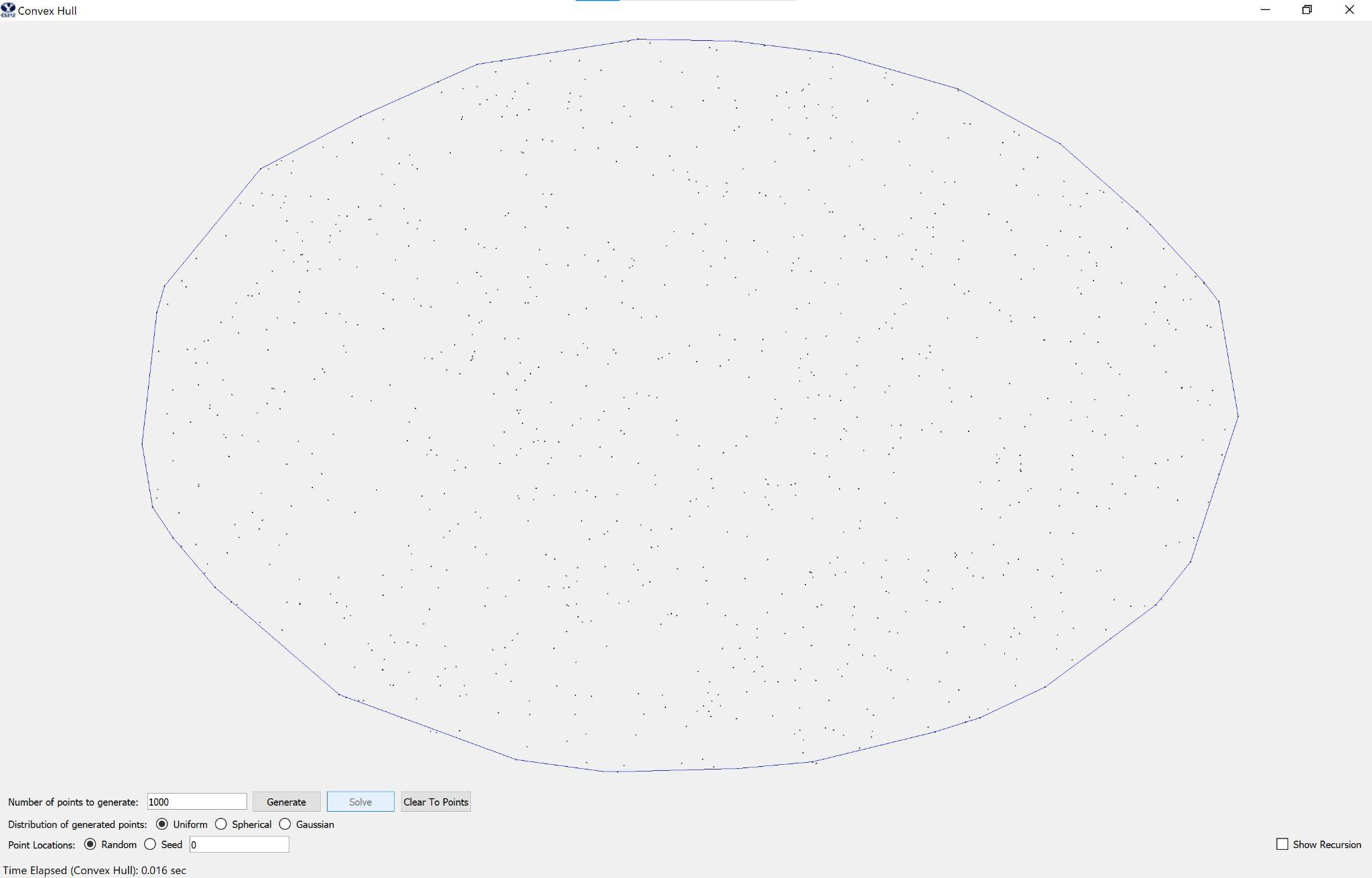


1. The below graph of my times appears to be nlogn as we can see it seems to follow the same shape as xlogx (the black line). I’m not sure exactly what else to say. I mean, if you look at that and tell me it could be anything else, you’re lying

100



1000



Code:

from which\_pyqt import PYQT\_VER

if PYQT\_VER == 'PYQT5':

from PyQt5.QtCore import QLineF, QPointF, QObject

elif PYQT\_VER == 'PYQT4':

from PyQt4.QtCore import QLineF, QPointF, QObject

elif PYQT\_VER == 'PYQT6':

from PyQt6.QtCore import QLineF, QPointF, QObject

else:

raise Exception('Unsupported Version of PyQt: {}'.format(PYQT\_VER))

import time

# Some global color constants that might be useful

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

# Global variable that controls the speed of the recursion automation, in seconds

PAUSE = .25

#

# This is the class you have to complete.

#

class ConvexHullSolver(QObject):

# Class constructor

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.pause = False

# Some helper methods that make calls to the GUI, allowing us to send updates

# to be displayed.

def showTangent(self, line, color):

self.view.addLines(line, color)

if self.pause:

time.sleep(PAUSE)

def eraseTangent(self, line):

self.view.clearLines(line)

def blinkTangent(self, line, color):

self.showTangent(line, color)

self.eraseTangent(line)

def showHull(self, polygon, color):

self.view.addLines(polygon, color)

if self.pause:

time.sleep(PAUSE)

def eraseHull(self, polygon):

self.view.clearLines(polygon)

def showText(self, text):

self.view.displayStatusText(text)

# 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()

points = quicksort(points)

t2 = time.time()

t3 = time.time()

fullHull, L, R = divideAndConq(points)

fullHullPoly = [QLineF(fullHull[i], fullHull[(i + 1) % len(fullHull)]) for i in range(len(fullHull))]

t4 = time.time()

self.showHull(fullHullPoly, BLUE)

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

# quicksort, Big O(nlogn)

def quicksort(points):

if len(points) <= 1:

return points

pivot = points[len(points) // 2]

pivotVal = pivot.x()

left = []

right = []

for point in points:

if point.x() < pivotVal:

left.append(point)

elif point.x() > pivotVal:

right.append(point)

return quicksort(left) + [pivot] + quicksort(right)

# divide and conquer, Big O(nlogn)

def divideAndConq(points):

pointsLength = len(points)

if pointsLength <= 4:

return makeSmallHull(points, pointsLength)

L, Lleft, Lright = divideAndConq(points[0:pointsLength // 2])

R, Rleft, Rright = divideAndConq(points[pointsLength // 2:pointsLength])

newHull, Rright = mergeHulls(L, Lright, R, Rleft, Rright)

return newHull, Lleft, Rright

# create small hulls to be merged, Big O(1) constant time

def makeSmallHull(points, pointsLength):

left = 0

right = None

greatestXVal = None

newPoints = [points[left]]

points.pop(left)

while len(newPoints) < pointsLength:

greatestSlope = None

nextPoint = None

nextPointIndex = None

for index, point in enumerate(points):

slope = ((point.y() - newPoints[left].y()) / (point.x() - newPoints[left].x()))

if greatestSlope is None or slope > greatestSlope:

greatestSlope = slope

nextPoint = point

nextPointIndex = index

if greatestXVal is None or nextPoint.x() > greatestXVal:

greatestXVal = nextPoint.x()

right = len(newPoints)

newPoints.append(nextPoint)

points.pop(nextPointIndex)

return newPoints, left, right

# Merge smaller hulls together, Big O(1) constant time

def mergeHulls(L, Lright, R, Rleft, Rright):

newHull = []

hasChanged = True

topRightIndex = Rleft

topLeftIndex = Lright

bottomRightIndex = Rleft

bottomLeftIndex = Lright

lastSlope = None

while hasChanged:

hasChanged = False

lastSlope = None

# Top index on right hull

for i in range(topRightIndex, len(R)):

if topLeftIndex >= len(L):

topLeftIndex = topLeftIndex % len(L)

slope = ((R[i].y() - L[topLeftIndex].y()) / (R[i].x() - L[topLeftIndex].x()))

if lastSlope is None:

lastSlope = slope

elif slope > lastSlope:

lastSlope = slope

topRightIndex = i

hasChanged = True

lastSlope = None

# Top index on left hull

for i in range(topLeftIndex, -1, -1):

slope = ((L[i].y() - R[topRightIndex].y()) / (L[i].x() - R[topRightIndex].x()))

if lastSlope is None:

lastSlope = slope

elif slope < lastSlope:

lastSlope = slope

topLeftIndex = i

hasChanged = True

lastSlope = None

hasChanged = True

# Bottom indices

while hasChanged:

hasChanged = False

lastSlope = None

# Bottom index on right hull

for i in range(bottomRightIndex, -2, -1):

if i == -1:

i = (len(R) - 1)

if bottomLeftIndex >= len(L):

bottomLeftIndex = bottomLeftIndex % len(L)

slope = ((R[i].y() - L[bottomLeftIndex].y()) / (R[i].x() - L[bottomLeftIndex].x()))

if lastSlope is None:

lastSlope = slope

elif slope < lastSlope:

lastSlope = slope

bottomRightIndex = i

hasChanged = True

lastSlope = None

# Bottom index on left hull

for i in range(bottomLeftIndex, len(L) + 1):

if i == len(L):

i = 0

slope = ((L[i].y() - R[bottomRightIndex].y()) / (L[i].x() - R[bottomRightIndex].x()))

if lastSlope is None:

lastSlope = slope

elif slope > lastSlope:

lastSlope = slope

bottomLeftIndex = i

hasChanged = True

for index in range(topLeftIndex + 1):

newHull.append(L[index])

if bottomRightIndex > 0:

for index in range(topRightIndex, bottomRightIndex + 1):

if index == Rright:

Rright = len(newHull)

newHull.append(R[index])

else:

index = topRightIndex

while True:

if index == len(R):

index = 0

newHull.append(R[index])

break

newHull.append(R[index])

index += 1

if bottomLeftIndex > 0:

for index in range(bottomLeftIndex, len(L)):

newHull.append(L[index])

return newHull, Rright