def run(self):

assert(type(self.points) == list and type(self.points[0]) == QPointF )

n = len(self.points)

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

convexHullSolver = ConvexHullSolver()

t1 = time.time()

# SORT THE POINTS BY INCREASING X-VALUE

convexHullSolver.sort\_points\_by\_x(self.points)

t2 = time.time()

print('Time Elapsed (Sorting): {:3.3f} sec'.format(t2-t1))

t3 = time.time()

# COMPUTE THE CONVEX HULL USING DIVIDE AND CONQUER

complete\_hull\_and\_points = convexHullSolver.compute\_hull(self.points, self)

t4 = time.time()

USE\_DUMMY = False

if USE\_DUMMY:

# this is a dummy polygon of the first 3 unsorted points

polygon = [QLineF(self.points[i],self.points[(i+1)%3]) for i in range(3)]

# 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

assert( type(polygon) == list and type(polygon[0]) == QLineF )

# send a signal to the GUI thread with the hull and its color

self.show\_hull.emit(polygon,(255,0,0))

else:

# PASS THE CONVEX HULL LINES BACK TO THE GUI FOR DISPLAY

self.show\_hull.emit(complete\_hull\_and\_points[0].getLines(), (255, 0, 0))

# send a signal to the GUI thread with the time used to compute the hull

self.display\_text.emit('Time Elapsed (Convex Hull): {:3.3f} sec'.format(t4-t3))

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

# -----------------Begin ConvexHullSolver.py-----------------

from operator import methodcaller

from enum import Enum, auto

from Hull import Hull

from which\_pyqt import PYQT\_VER

if PYQT\_VER == 'PYQT5':

from PyQt5.QtCore import QLineF, QPointF, QThread, pyqtSignal

elif PYQT\_VER == 'PYQT4':

from PyQt4.QtCore import QLineF, QPointF, QThread, pyqtSignal

else:

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

class Side(Enum):

L = auto()

R = auto()

W = auto()

class ConvexHullSolver:

# O(n log n)

def sort\_points\_by\_x(self, points):

points.sort(key=methodcaller('x'))

# O(1)

def calculate\_slope(self, left\_point, right\_point):

rise = right\_point.y() - left\_point.y()

run = right\_point.x() - left\_point.x()

return rise / run

# O(n/2) + O(3n/4) = O(3n/4) = O(n)

def delete\_points\_and\_lines(self, points, top\_index, bottom\_index, lines, convex\_hull):

bottom\_point = points[bottom\_index]

points\_to\_delete = list()

index = (top\_index - 1) % len(points)

# O(n/2) = O(n)

while points[index] != bottom\_point:

points\_to\_delete.append(points.pop(index))

index = (index - 1) % len(points)

i = 0

# O(3n/4) = O(n)

while i < len(lines):

if lines[i].p1() in points\_to\_delete or lines[i].p2() in points\_to\_delete:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([lines[i]])

lines.pop(i)

else:

i += 1

# O(n^2)

def combine\_hulls(self, left\_points, left\_hull, right\_points, right\_hull, side, convex\_hull): # returns combined hull, points

left\_changed = True

right\_changed = True

left\_index\_top, right\_index\_top = 0, 0

# O(1)

current\_new\_line = QLineF(left\_points[left\_index\_top], right\_points[right\_index\_top])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([current\_new\_line], (0, 255, 0))

slope\_current = self.calculate\_slope(current\_new\_line.p1(), current\_new\_line.p2())

# O(n)

# find new top line

while left\_changed or right\_changed:

potential\_new\_line = QLineF(left\_points[(left\_index\_top + 1) % len(left\_points)], right\_points[right\_index\_top])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([potential\_new\_line], (0, 0, 255))

slope\_new = self.calculate\_slope(potential\_new\_line.p1(), potential\_new\_line.p2())

if slope\_new < slope\_current:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([current\_new\_line, potential\_new\_line])

current\_new\_line = potential\_new\_line

if convex\_hull.pause:

convex\_hull.show\_hull.emit([current\_new\_line], (0, 255, 0))

slope\_current = slope\_new

left\_index\_top += 1

left\_changed = True

else:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([potential\_new\_line])

left\_changed = False

potential\_new\_line = QLineF(left\_points[left\_index\_top], right\_points[(right\_index\_top + 1) % len(right\_points)])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([potential\_new\_line], (0, 0, 255))

slope\_new = self.calculate\_slope(potential\_new\_line.p1(), potential\_new\_line.p2())

if slope\_new > slope\_current:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([current\_new\_line, potential\_new\_line])

current\_new\_line = potential\_new\_line

if convex\_hull.pause:

convex\_hull.show\_hull.emit([current\_new\_line], (0, 255, 0))

slope\_current = slope\_new

right\_index\_top += 1

right\_changed = True

else:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([potential\_new\_line])

right\_changed = False

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([current\_new\_line])

new\_top\_edge = current\_new\_line

if convex\_hull.pause:

convex\_hull.show\_hull.emit([new\_top\_edge], (255, 0, 0))

# Find bottom line

left\_changed = True

right\_changed = True

left\_index\_bottom, right\_index\_bottom = 0, 0

# O(n)

current\_new\_line = QLineF(left\_points[left\_index\_bottom], right\_points[right\_index\_bottom])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([current\_new\_line], (0, 255, 0))

slope\_current = self.calculate\_slope(current\_new\_line.p1(), current\_new\_line.p2())

while left\_changed or right\_changed:

potential\_new\_line = QLineF(left\_points[(left\_index\_bottom - 1) % len(left\_points)], right\_points[right\_index\_bottom])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([potential\_new\_line], (0, 0, 255))

slope\_new = self.calculate\_slope(potential\_new\_line.p1(), potential\_new\_line.p2())

if slope\_new > slope\_current:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([current\_new\_line, potential\_new\_line])

current\_new\_line = potential\_new\_line

if convex\_hull.pause:

convex\_hull.show\_hull.emit([current\_new\_line], (0, 255, 0))

slope\_current = slope\_new

left\_index\_bottom = ((left\_index\_bottom - 1) % len(left\_points))

left\_changed = True

else:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([potential\_new\_line])

left\_changed = False

potential\_new\_line = QLineF(left\_points[left\_index\_bottom], right\_points[(right\_index\_bottom - 1) % len(right\_points)])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([potential\_new\_line], (0, 0, 255))

slope\_new = self.calculate\_slope(potential\_new\_line.p1(), potential\_new\_line.p2())

if slope\_new < slope\_current:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([current\_new\_line, potential\_new\_line])

current\_new\_line = potential\_new\_line

if convex\_hull.pause:

convex\_hull.show\_hull.emit([current\_new\_line], (0, 255, 0))

slope\_current = slope\_new

right\_index\_bottom = ((right\_index\_bottom - 1) % len(right\_points))

right\_changed = True

else:

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([potential\_new\_line])

right\_changed = False

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([current\_new\_line])

new\_bottom\_edge = current\_new\_line

if convex\_hull.pause:

convex\_hull.show\_hull.emit([new\_bottom\_edge], (255, 0, 0))

points\_to\_check = [left\_points[left\_index\_bottom], left\_points[left\_index\_top], right\_points[right\_index\_top], right\_points[right\_index\_bottom]]

# Trim point arrays and delete unneeded points & associated lines.

# O(n)

self.delete\_points\_and\_lines(left\_points, left\_index\_top, left\_index\_bottom, left\_hull.getLines(), convex\_hull)

# O(n)

self.delete\_points\_and\_lines(right\_points, right\_index\_top, right\_index\_bottom, right\_hull.getLines(), convex\_hull)

# Arrange trimmed arrays and fuse together

combined\_points = list()

# O(n log n)

if side == side.L:

# find rightmost point and pop it off the list. Combine lists and sort for CC order based on slope

i = 1

right\_most\_index = 0

# O(n)

while i < len(right\_points):

if right\_points[i].x() > right\_points[right\_most\_index].x():

right\_most\_index = i

i += 1

# O(n)

right\_most\_point = right\_points.pop(right\_most\_index)

right\_points.extend(left\_points)

# O(n log n)

right\_points.sort(key=lambda point: self.calculate\_slope(point, right\_most\_point))

right\_points.insert(0, right\_most\_point)

combined\_points = right\_points

# O(n log n)

else:

# find leftmost point and pop it off the list. Combine lists and sort for C order based on slope

i = 1

left\_most\_index = 0

# O(n)

while i < len(left\_points):

if left\_points[i].x() < left\_points[left\_most\_index].x():

left\_most\_index = i

i += 1

left\_most\_point = left\_points.pop(left\_most\_index)

left\_points.extend(right\_points)

# O(n log n)

left\_points.sort(key=lambda point: self.calculate\_slope(left\_most\_point, point), reverse=True)

left\_points.insert(0, left\_most\_point)

combined\_points = left\_points

combined\_lines = list()

# O(n)

combined\_lines.extend(left\_hull.getLines())

# O(n)

combined\_lines.extend(right\_hull.getLines())

# Check points w/ newly created lines to see if there is a third middle line to delete

# O(n^2)

# O(n)

for line in combined\_lines:

if line.p1() in points\_to\_check or line.p2() in points\_to\_check:

# O(n)

index\_p1 = combined\_points.index(line.p1())

# O(n)

index\_p2 = combined\_points.index(line.p2())

distance = abs(index\_p1 – index\_p2)

if distance != 1 and distance != (len(combined\_points) - 1):

if convex\_hull.pause:

convex\_hull.erase\_hull.emit([line])

# O(n)

combined\_lines.remove(line)

combined\_lines.append(new\_top\_edge)

combined\_lines.append(new\_bottom\_edge)

return [Hull(combined\_lines), combined\_points]

# Master Theorem applies here: a = 2 subproblems of size n/(b = 2) and combines answers in O(n^(d = 2)). Log2(2) = 1 < d = 2, so overall O(n^(d = 2))

def compute\_hull(self, points, convex\_hull, side=Side.W): # returns hull, points

lines = []

completed\_hull\_and\_points = None

# Work at bottom of tree = O(1)

# O(1)

if len(points) == 2:

new\_line = QLineF(points[0], points[1])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([new\_line], (255, 0, 0))

lines.append(new\_line)

if side == Side.L:

points.reverse()

completed\_hull\_and\_points = [Hull(lines), points]

# O(1)

elif len(points) == 3:

# O(1)

for i in range(3):

new\_line = QLineF(points[i], points[(i + 1) % 3])

if convex\_hull.pause:

convex\_hull.show\_hull.emit([new\_line], (255, 0, 0))

lines.append(new\_line)

# O(1)

if side == Side.L:

right\_most\_point = points.pop(2)

# O(1) because there's only ever 2 points

points.sort(key=lambda point: self.calculate\_slope(point, right\_most\_point))

points.insert(0, right\_most\_point)

# O(1)

elif side == Side.R:

left\_most\_point = points.pop(0)

points.sort(key=lambda point: self.calculate\_slope(left\_most\_point, point), reverse=True)

points.insert(0, left\_most\_point)

completed\_hull\_and\_points = [Hull(lines), points]

# Split and combine parts = O(n^2)

elif len(points) > 3:

# Split parts = O(n)

# O(n)

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

# O(n)

left\_hull\_and\_points = self.compute\_hull(left\_points, convex\_hull, Side.L)

right\_hull\_and\_points = self.compute\_hull(right\_points, convex\_hull, Side.R)

# Combine parts = O(n^2)

completed\_hull\_and\_points = self.combine\_hulls(left\_hull\_and\_points[1], left\_hull\_and\_points[0], right\_hull\_and\_points[1], right\_hull\_and\_points[0], side, convex\_hull)

return completed\_hull\_and\_points