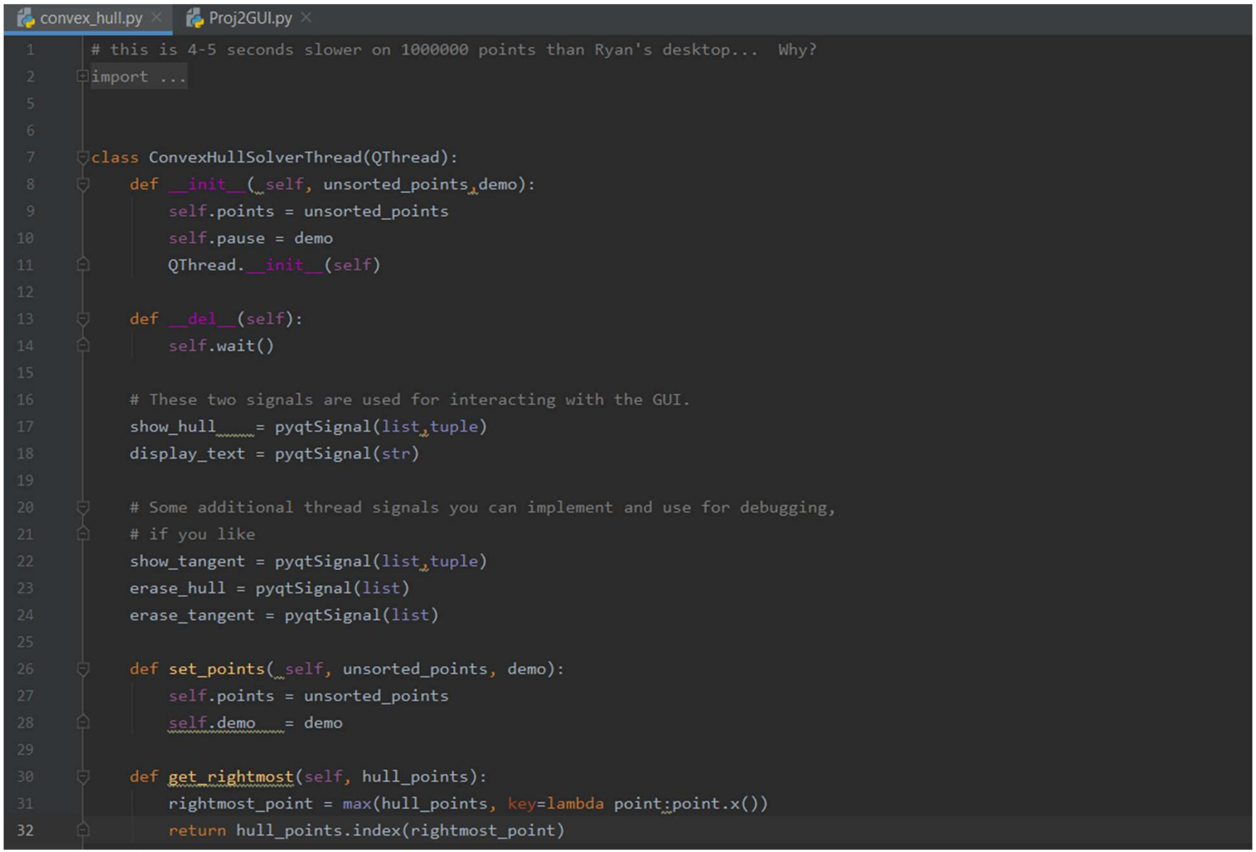
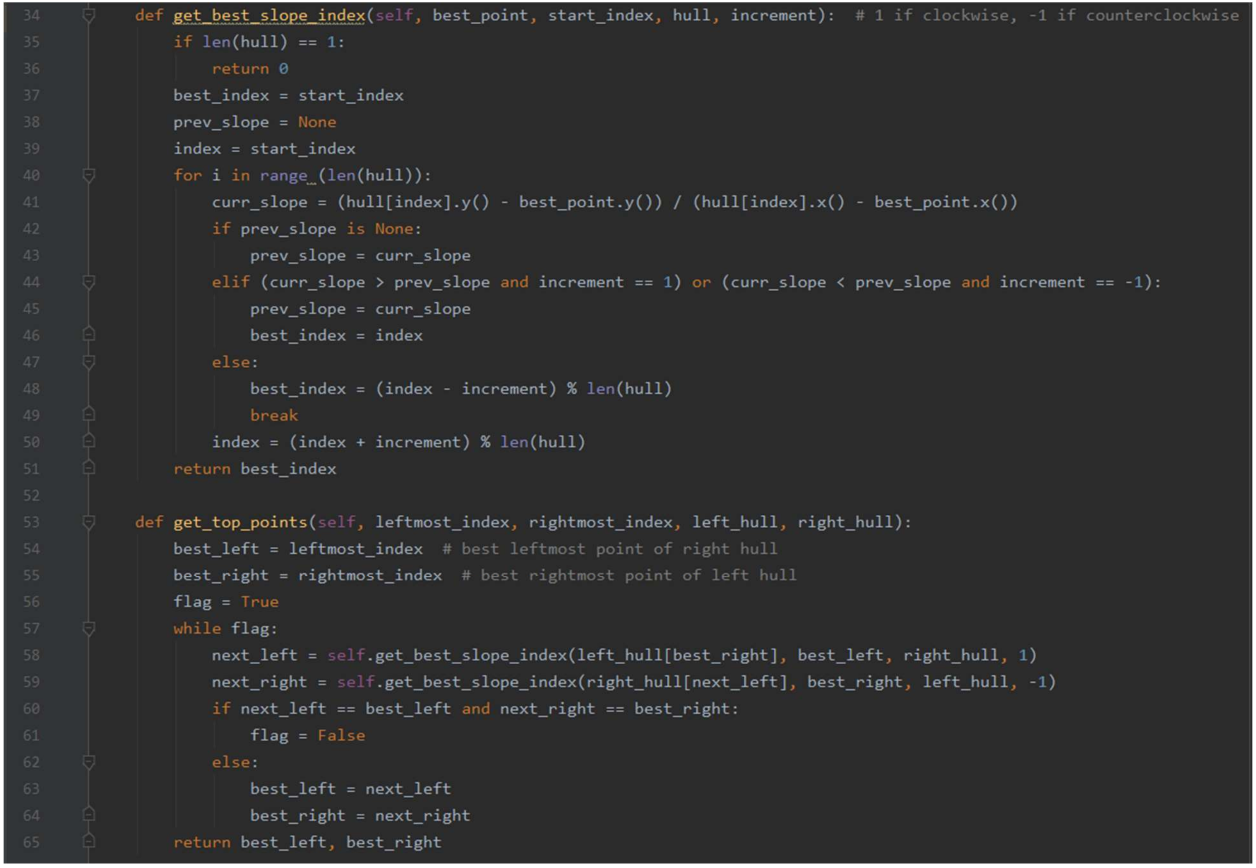
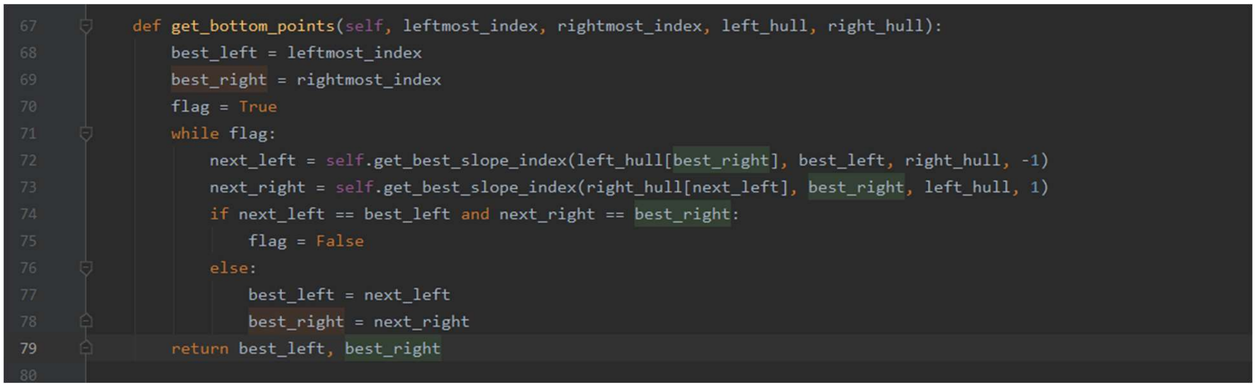


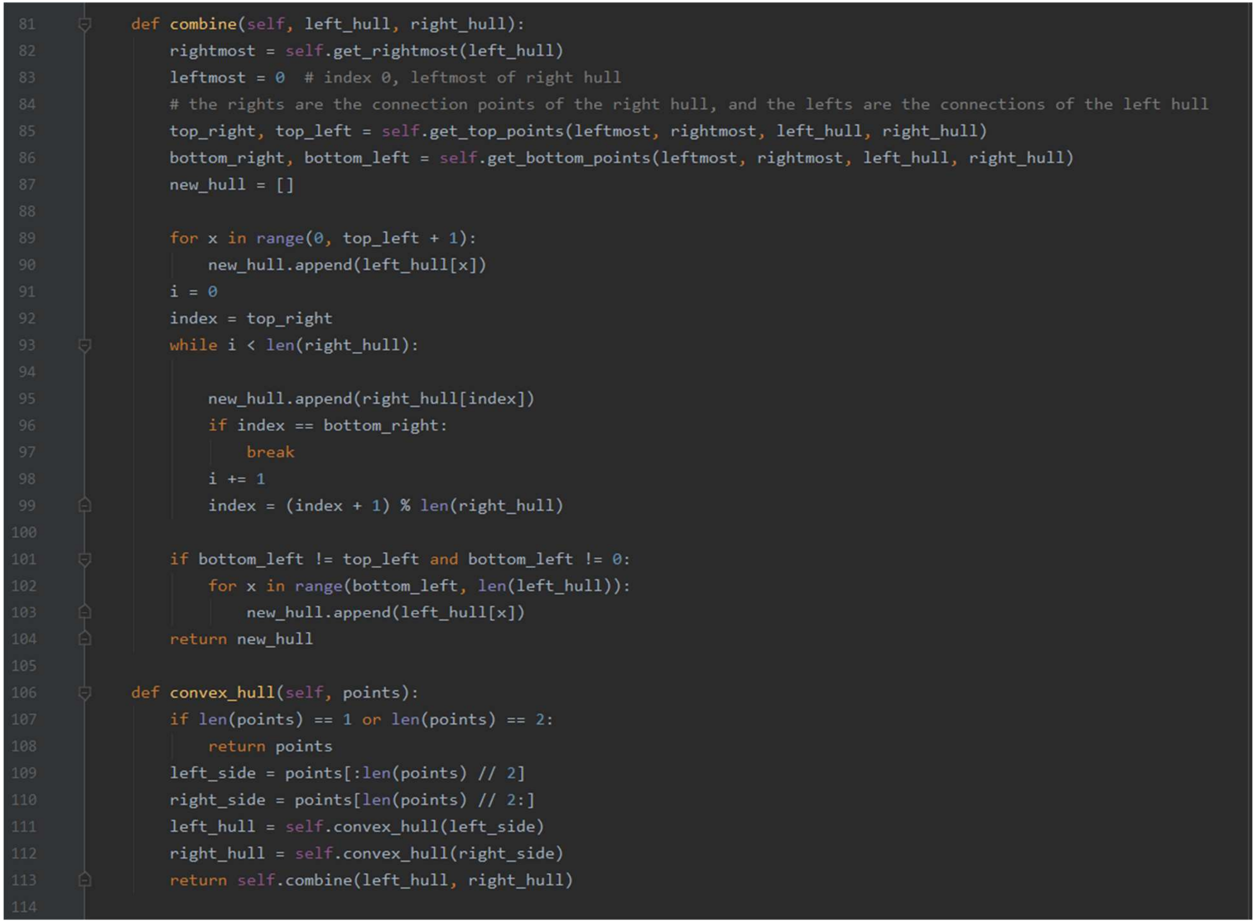
```python
# this is 4-5 seconds slower on 1000000 points than Ryan's desktop...  Why?
import ...


class ConvexHullSolverThread(QThread):
    def __init__( self, unsorted_points,demo):
        self.points = unsorted_points
        self.pause = demo
        QThread.__init__(self)

    def __del__(self):
        self.wait()

    # These two signals are used for interacting with the GUI.
    show_hull    = pyqtSignal(list,tuple)
    display_text = pyqtSignal(str)

    # Some additional thread signals you can implement and use for debugging,
    # if you like
    show_tangent = pyqtSignal(list,tuple)
    erase_hull = pyqtSignal(list)
    erase_tangent = pyqtSignal(list)

    def set_points( self, unsorted_points, demo):
        self.points = unsorted_points
        self.demo   = demo

    def get_rightmost(self, hull_points):
        rightmost_point = max(hull_points, key=lambda point:point.x())
        return hull_points.index(rightmost_point)
```



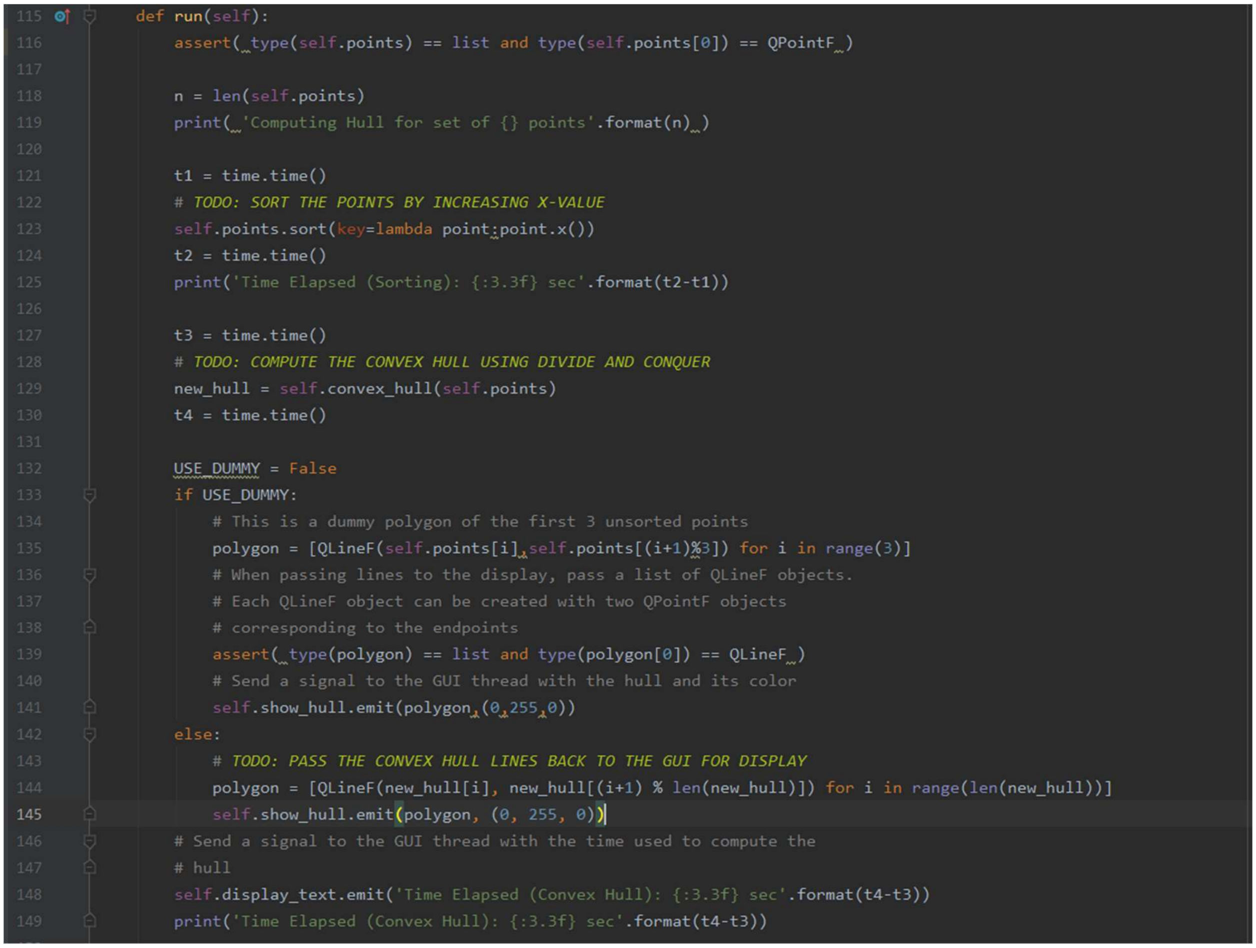
```python
    def get_best_slope_index(self, best_point, start_index, hull, increment):  # 1 if clockwise, -1 if counterclockwise
        if len(hull) == 1:
            return 0
        best_index = start_index
        prev_slope = None
        index = start_index
        for i in range (len(hull)):
            curr_slope = (hull[index].y() - best_point.y()) / (hull[index].x() - best_point.x())
            if prev_slope is None:
                prev_slope = curr_slope
            elif (curr_slope > prev_slope and increment == 1) or (curr_slope < prev_slope and increment == -1):
                prev_slope = curr_slope
                best_index = index
            else:
                best_index = (index - increment) % len(hull)
                break
            index = (index + increment) % len(hull)
        return best_index

    def get_top_points(self, leftmost_index, rightmost_index, left_hull, right_hull):
        best_left = leftmost_index  # best leftmost point of right hull
        best_right = rightmost_index  # best rightmost point of left hull
        flag = True
        while flag:
            next_left = self.get_best_slope_index(left_hull[best_right], best_left, right_hull, 1)
            next_right = self.get_best_slope_index(right_hull[next_left], best_right, left_hull, -1)
            if next_left == best_left and next_right == best_right:
                flag = False
            else:
                best_left = next_left
                best_right = next_right
        return best_left, best_right
```



```python
    def get_bottom_points(self, leftmost_index, rightmost_index, left_hull, right_hull):
        best_left = leftmost_index
        best_right = rightmost_index
        flag = True
        while flag:
            next_left = self.get_best_slope_index(left_hull[best_right], best_left, right_hull, -1)
            next_right = self.get_best_slope_index(right_hull[next_left], best_right, left_hull, 1)
            if next_left == best_left and next_right == best_right:
                flag = False
            else:
                best_left = next_left
                best_right = next_right
        return best_left, best_right

```



```python
    def combine(self, left_hull, right_hull):
        rightmost = self.get_rightmost(left_hull)
        leftmost = 0  # index 0, leftmost of right hull
        # the rights are the connection points of the right hull, and the lefts are the connections of the left hull
        top_right, top_left = self.get_top_points(leftmost, rightmost, left_hull, right_hull)
        bottom_right, bottom_left = self.get_bottom_points(leftmost, rightmost, left_hull, right_hull)
        new_hull = []

        for x in range(0, top_left + 1):
            new_hull.append(left_hull[x])
        i = 0
        index = top_right
        while i < len(right_hull):

            new_hull.append(right_hull[index])
            if index == bottom_right:
                break
            i += 1
            index = (index + 1) % len(right_hull)

        if bottom_left != top_left and bottom_left != 0:
            for x in range(bottom_left, len(left_hull)):
                new_hull.append(left_hull[x])
        return new_hull

    def convex_hull(self, points):
        if len(points) == 1 or len(points) == 2:
            return points
        left_side = points[:len(points) // 2]
        right_side = points[len(points) // 2:]
        left_hull = self.convex_hull(left_side)
        right_hull = self.convex_hull(right_side)
        return self.combine(left_hull, right_hull)

```



```python
    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) )

        t1 = time.time()
        # TODO: SORT THE POINTS BY INCREASING X-VALUE
        self.points.sort(key=lambda point:point.x())
        t2 = time.time()
        print('Time Elapsed (Sorting): {:3.3f} sec'.format(t2-t1))

        t3 = time.time()
        # TODO: COMPUTE THE CONVEX HULL USING DIVIDE AND CONQUER
        new_hull = self.convex_hull(self.points)
        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,(0,255,0))
        else:
            # TODO: PASS THE CONVEX HULL LINES BACK TO THE GUI FOR DISPLAY
            polygon = [QLineF(new_hull[i], new_hull[(i+1) % len(new_hull)]) for i in range(len(new_hull))]
            self.show_hull.emit(polygon, (0, 255, 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))
```

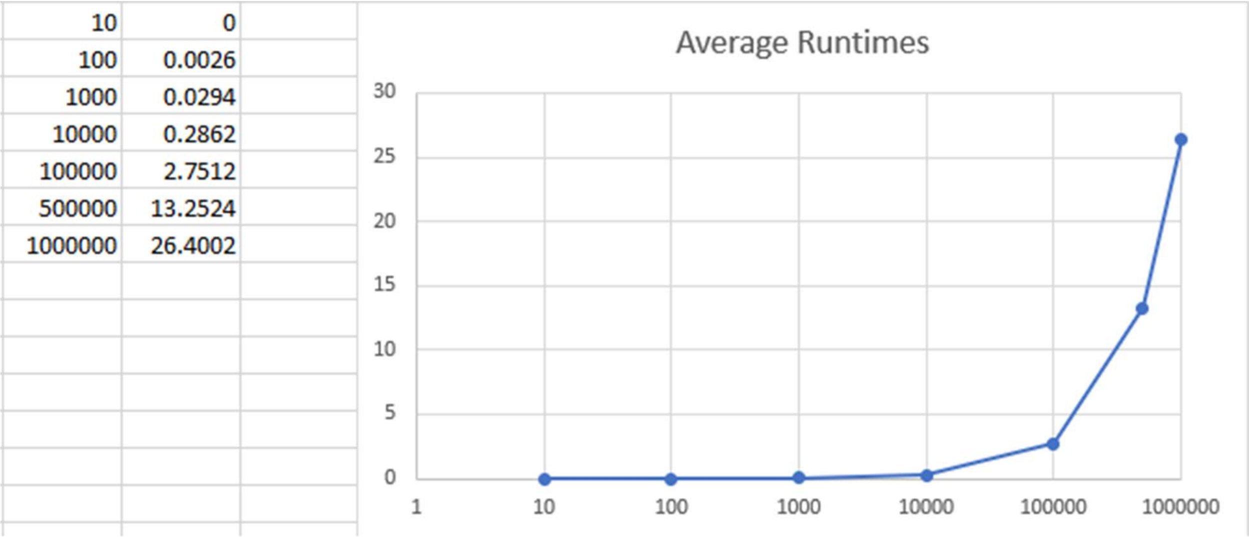
The get_rightmost function will be O(n), since it has to search through the list to find the max.

The get_best_slope_index will be at worst O(n), if it has to go through every point to find the best slope.

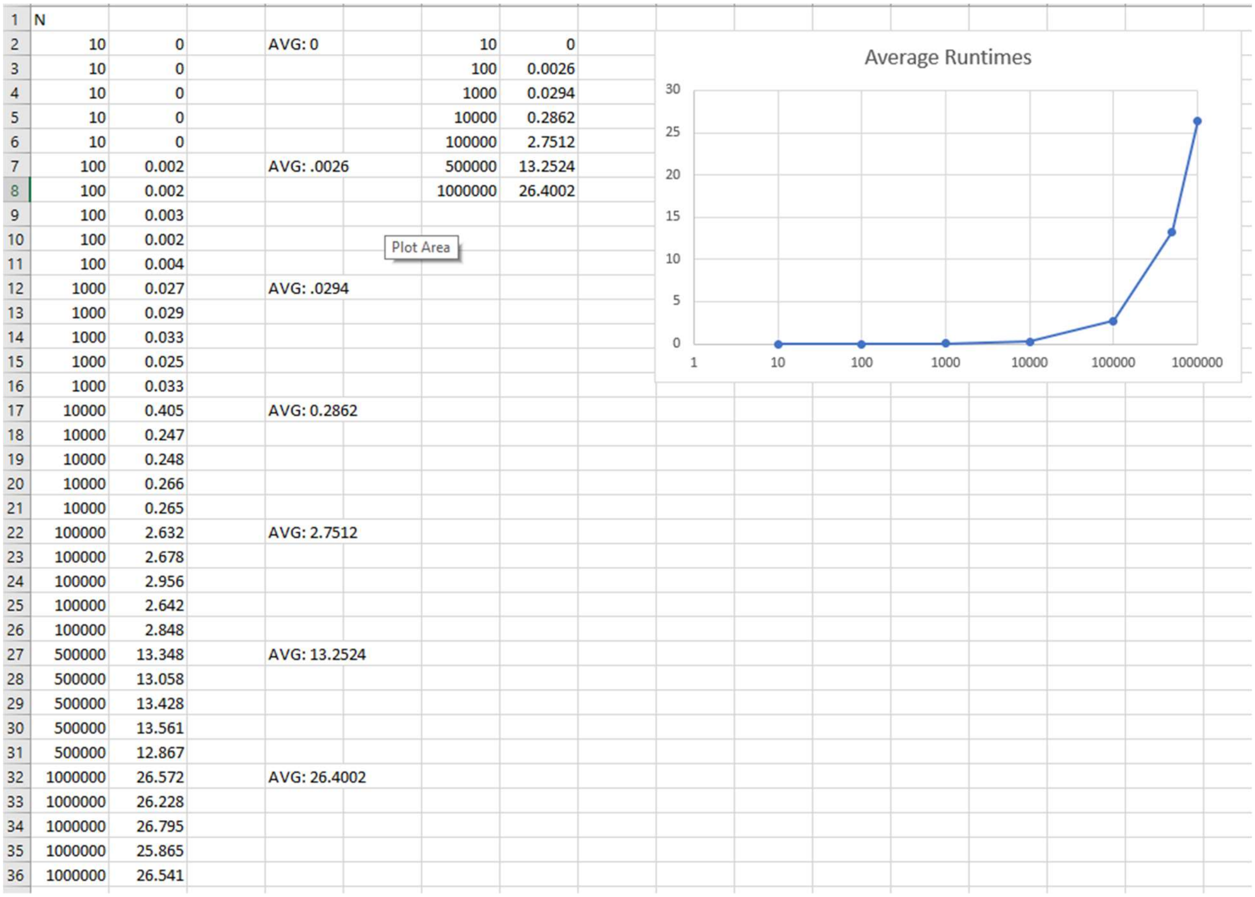
The get_top_points and get_bottom_points will both be O(n), since the biggest time sinks are the calls to get_best_slope_index

Combine will be O(n) since a new hull is created and all the points are added to it.

Overall, the time complexity is O(nlog(n)) according to the Master Theorem; A = 2, B = 2, and D = 1



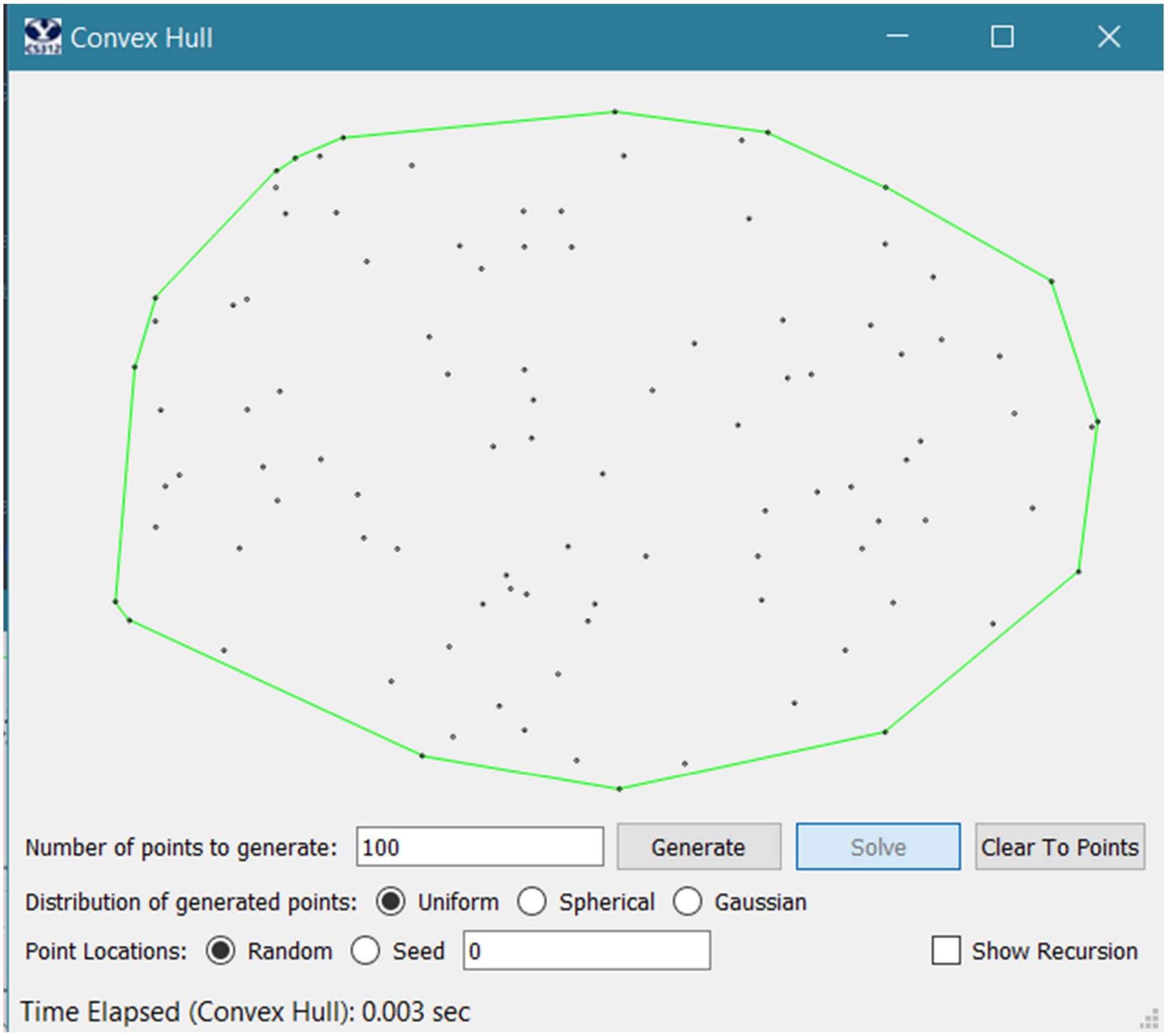
10	0
100	0.0026
1000	0.0294
10000	0.2862
100000	2.7512
500000	13.2524
1000000	26.4002



1	N						
2	10	0		AVG: 0		10	0
3	10	0				100	0.0026
4	10	0				1000	0.0294
5	10	0				10000	0.2862
6	10	0				100000	2.7512
7	100	0.002		AVG: .0026		500000	13.2524
8	100	0.002				1000000	26.4002
9	100	0.003					
10	100	0.002					
11	100	0.004					
12	1000	0.027		AVG: .0294			
13	1000	0.029					
14	1000	0.033					
15	1000	0.025					
16	1000	0.033					
17	10000	0.405		AVG: 0.2862			
18	10000	0.247					
19	10000	0.248					
20	10000	0.266					
21	10000	0.265					
22	100000	2.632		AVG: 2.7512			
23	100000	2.678					
24	100000	2.956					
25	100000	2.642					
26	100000	2.848					
27	500000	13.348		AVG: 13.2524			
28	500000	13.058					
29	500000	13.428					
30	500000	13.561					
31	500000	12.867					
32	1000000	26.572		AVG: 26.4002			
33	1000000	26.228					
34	1000000	26.795					
35	1000000	25.865					
36	1000000	26.541					

For the constant of proportionality, I got about 4.4*10^(-6), but my data fit my time complexity analysis otherwise.

For n = 1000000, k = 26.4002 / (1000000*log(1000000))

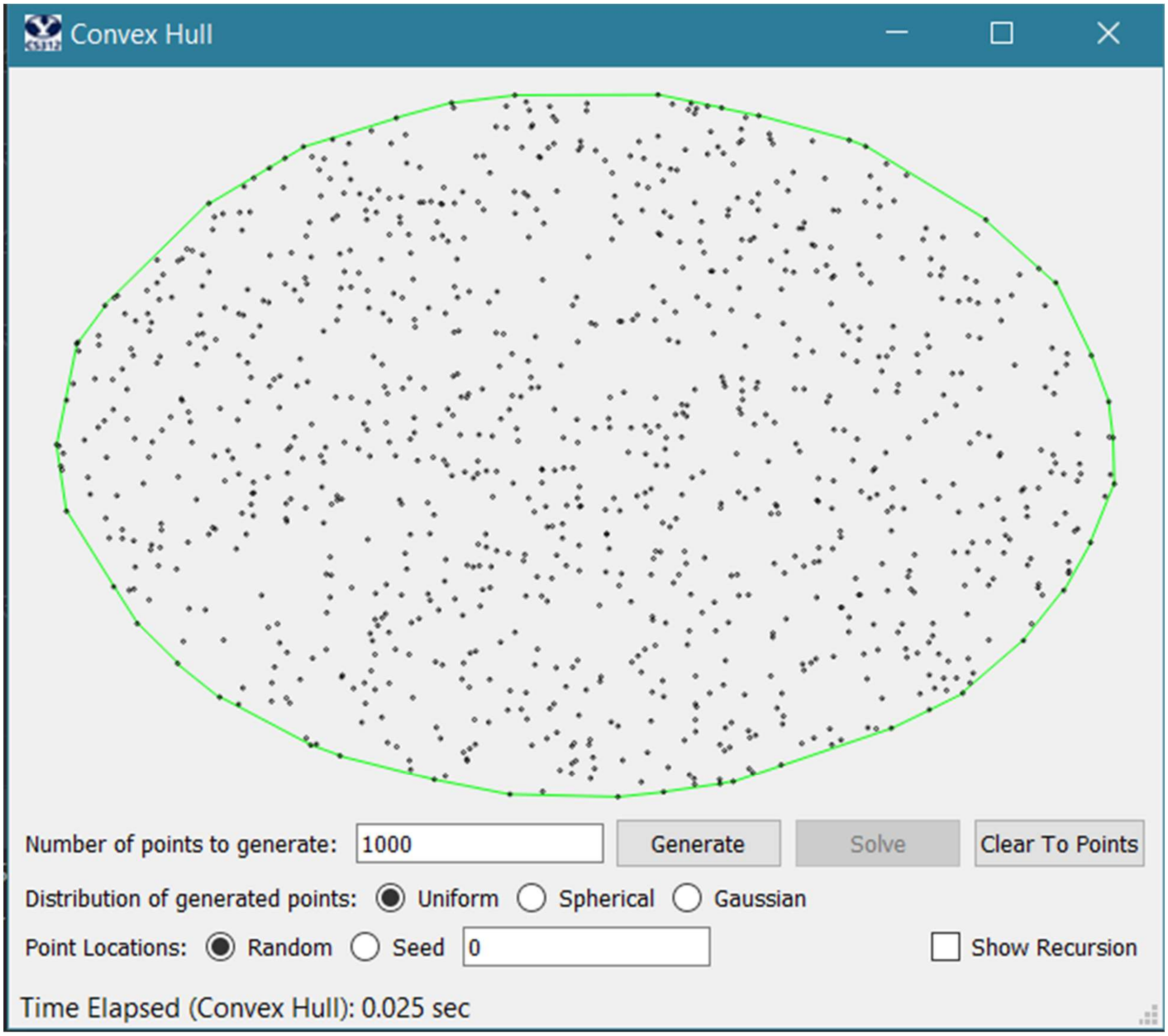


Number of points to generate: 100

Distribution of generated points: ● Uniform ○ Spherical ○ Gaussian

Point Locations: ● Random ○ Seed 0 □ Show Recursion

Time Elapsed (Convex Hull): 0.003 sec



Convex Hull

Number of points to generate: 1000

Distribution of generated points: ● Uniform ○ Spherical ○ Gaussian

Point Locations: ● Random ○ Seed 0

□ Show Recursion

Time Elapsed (Convex Hull): 0.025 sec