Jacob Wright
CS 312 Project 2

which_pyqt.py | convex_hull.py 1, M

convex_hull.py > ConvexHullSolver > merge_hulls

You, 15 hours ago | 2 authors (You and others)

```python
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)


    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 coordinate
        points.sort(key=lambda point:point.x())

        t2 = time.time()
        t3 = time.time()

        hull = self.convex_hull_helper(points)
```

You, 17 hours ago Ln 120, Col 14

Jacob Wright
CS 312 Project 2


```python
    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 coordinate
        points.sort(key=lambda point:point.x())

        t2 = time.time()
        t3 = time.time()

        hull = self.convex_hull_helper(points)

        polygon = [QLineF(hull[i], hull[(i + 1) % len(hull)]) for i in range(0,len(hull))]

        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".format(t4 - t3))


    # recursive helper function for computing the convex hull
    def convex_hull_helper(self, points):
        # base case
        if len(points) <= 2:
            return points

        # get left half of points and right half of points
        left_points = points[:len(points) // 2]
        right_points = points[len(points) // 2:]

        # compute left and right hulls recursively
        left_hull = self.convex_hull_helper(left_points)
        right_hull = self.convex_hull_helper(right_points)

        # merge the two hulls
        return self.merge_hulls(left_hull, right_hull)


    # merge two given hulls and return the merged hull
    def merge_hulls(self, left_hull, right_hull):
        # get leftmost and rightmost points
        leftmost = 0
```

Jacob Wright
CS 312 Project 2

which_pyqt.py | convex_hull.py 1, M

convex_hull.py > ConvexHullSolver > merge_hulls

```python
    # merge two given hulls and return the merged hull
    def merge_hulls(self, left_hull, right_hull):
        # get leftmost and rightmost points
        leftmost = 0
        rightmost = left_hull.index(max(left_hull, key=lambda point:point.x()))

        # get points from upper and lower tangents
        upper_right, upper_left = self.get_upper_tangent(leftmost, rightmost, left_hull, right_
        lower_right, lower_left = self.get_lower_tangent(leftmost, rightmost, left_hull, right_

        # create new hull to return
        merged_hull = []

        # add points from in left_hull from x=0 to upper left point
        for x in range(0, upper_left):
            merged_hull.append(left_hull[x])
        merged_hull.append(left_hull[upper_left])

        i = 0      You, 17 hours ago • FEAT: works, but is keeping all lines? …
        curr = upper_right

        # add points from right_hull from upper right point to lower right point
        while i < len(right_hull):
            merged_hull.append(right_hull[curr])
            if curr == lower_right:
                break
            i += 1
            curr = (curr + 1) % len(right_hull)

        # add points from in left_hull from lower left point to x=0
        if lower_left != upper_left and lower_left != 0:
            for x in range(lower_left, len(left_hull)):
                merged_hull.append(left_hull[x])
        return merged_hull


    # get upper tangent between two hulls and return two points, one from each hull
    def get_upper_tangent(self, leftmost, rightmost, left_hull, right_hull):
        # start upper tangent points at leftmost and rightmost
        upper_left = leftmost
        upper_right = rightmost

        # From pseudocode from slides
        done = False
        while not done:
            next_left = self.max_slope_clockwise(left_hull[upper_right], upper_left, right_hul
            next_right = self.max_slope_counter_clockwise(right_hull[next_left], upper_right,
```

You, 17 hours ago Ln 120, Col 14

Jacob Wright
CS 312 Project 2

Jacob Wright
CS 312 Project 2

which_pyqt.py | convex_hull.py 1, M

convex_hull.py > ConvexHullSolver > merge_hulls

```
180
181
182     #  return the index of the point in hull that has the maximum slope with respect to startin
183     def max_slope_clockwise(self, right_best_point, start, hull):
184         best_slope_index = start
185         prev_slope = None
186         curr = start
187
188         while True:
189             curr_slope = (hull[curr].y() - right_best_point.y()) / (hull[curr].x() - right_best
190             if prev_slope == None or curr_slope > prev_slope:
191                 prev_slope = curr_slope
192                 best_slope_index = curr
193             else:
194                 best_slope_index = (curr - 1) % len(hull)
195                 break
196             curr = (curr + 1) % len(hull)
197         return best_slope_index
198
199
200     #  return the index of the point in hull that has the maximum slope with respect to startin
201     def max_slope_counter_clockwise(self, left_best_point, start, hull):
202         best_slope_index = start
203         prev_slope = None
204         curr = start
205
206         while True:
207             curr_slope = (hull[curr].y() - left_best_point.y()) / (hull[curr].x() - left_best_p
208             if prev_slope == None or curr_slope < prev_slope:
209                 prev_slope = curr_slope
210                 best_slope_index = curr
211             else:
212                 best_slope_index = (curr + 1) % len(hull)
213                 break
214             curr = (curr - 1) % len(hull)
215         return best_slope_index
```

You, 17 hours ago Ln 120, Col 14

Jacob Wright
CS 312 Project 2

2. My solution for convex hull can be broken into the following pieces:

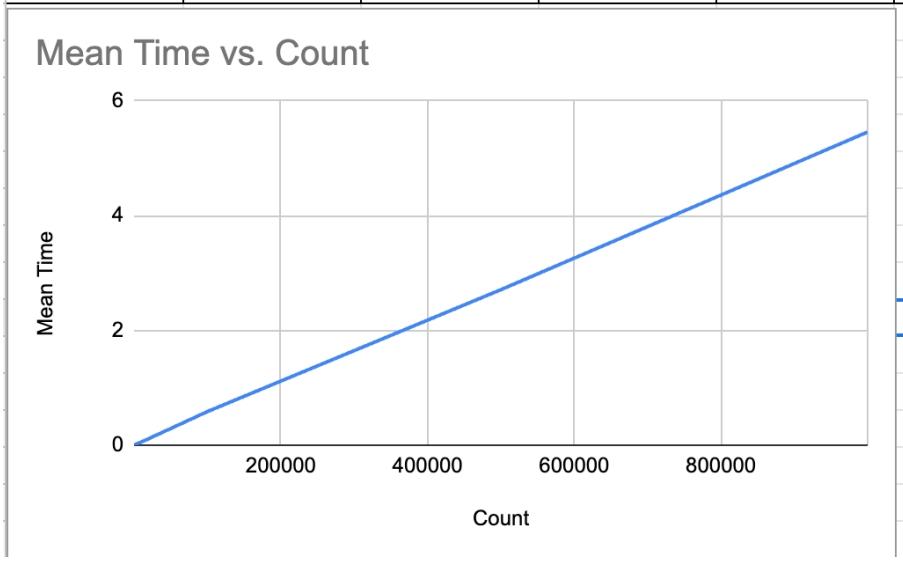
- convex_hull_helper: recursive function call to compute hull
  - Constant time, runs log(n) times
    - Calls merge_hulls
  - O(Log(n)) time complexity
- merge_hulls
  - Calls get_upper_tangent & get_lower_tangent
  - Worst case, iterates over n points in left & right hulls to add to convex hull
  - O(n) time complexity
- get_upper_tangent and get_lower_tangent
  - iterates over points in left and right hull and finds upper/lower tangent respectively
  - calls max_slope
    - gets max/best slope from given starting point for the tangent line
  - Worst case, iterates over N points in left or right hull and calls max_slope each iteration
  - Runs O(n) times and calls max_slope which at worst case has a time complexity of O(n)
  - Thus, get_upper_tangent and get_lower_tangent run in O(n^2) worst case (very very rare).
- Max_slope (clockwise and counter clockwise)
  - Iterates over points in given hull and finds best slope (highest or lowest) for respective tangent using a given point
  - Absolute worst case, runs in O(N) time complexity if it iterated over every single point in the hull. (however, since we are using the divide and conquer algorithm we end up iterating over just a few nodes, making it much less than O(n) in practice.

Overall, using the D&C algorithm, we get O(nlog(n)), however, using the smaller numbers we are using (7 digits and less), we can expect nearly linear times.

Jacob Wright
CS 312 Project 2

3.

Count	Time 1	Time 2	Time 3	Time 4	Time 5	Mean Time
10	0.001	0	0	0	0	0.0002
100	0.002	0.002	0.002	0.002	0.002	0.002
1000	0.011	0.011	0.011	0.011	0.011	0.011
10000	0.055	0.054	0.055	0.055	0.055	0.0548
100000	0.585	0.587	0.582	0.591	0.583	0.5856
500000	2.707	2.698	2.718	2.709	2.714	2.7092
1000000	5.46	5.449	5.463	5.458	5.457	5.4574

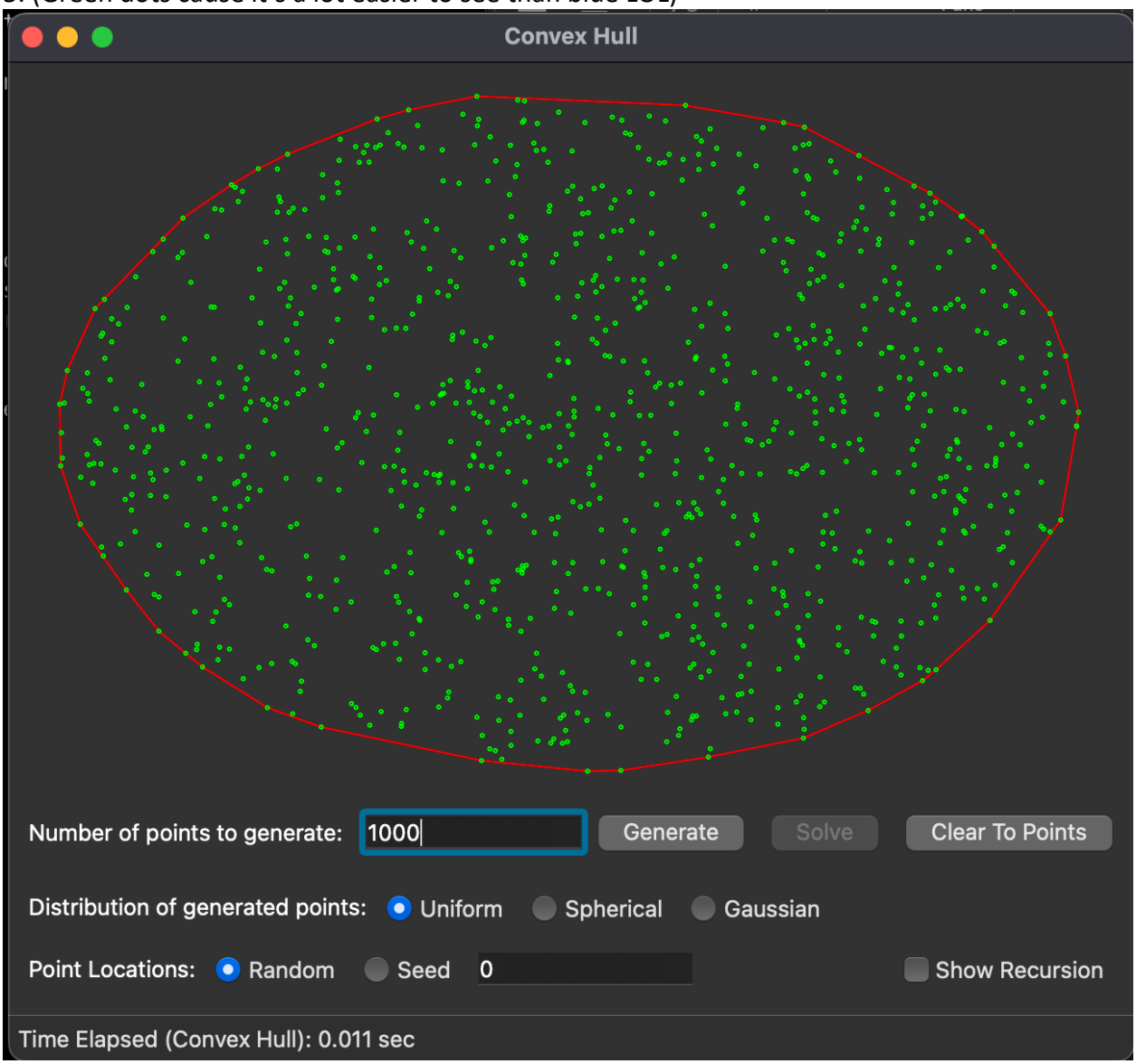


I did not use a logarithmic graph, but if I did then it would make the linear relationship look like a logarithmic relationship. However, the best relationship is linear. This can be seen by inspection, and also calculated the ratio from 500,000 -> 100,000 = 2.7s -> 5.4s (both growth 2x).

4. As mentioned above, there are some discrepancies from the theoretical and empirical analysis. Theoretically, it should be O(nlog(n)), however, in practice, with numbers 7 digits and less, it is essentially linear.

Jacob Wright
CS 312 Project 2

5. (Green dots cause it's a lot easier to see than blue LOL)



Jacob Wright
CS 312 Project 2