**COSC262 Assignment  
 Convex Hulls Report**

**Sarah Jackson (47106810)**

**May/June 2015**

**Algorithm Implementation:**

I found tuples essential for implementing both programs, as they were able to store the co-ordinates of interest collectively. This made it much easier to assess the x and y values of a co-ordinate if it was contained in a tuple. It also made it very easy to store large amounts of points together without their values being mixed up, as a certain x value would always be associated with a certain y value.

A data structure I found particularly useful was a stack for Graham scan. I was able to use it to pop and push relevant points of interest. What I found most desirable about using a stack is that it has an asymptotic complexity of O(1) for the pop() and push() methods, so it did not affect the over asymptotic complexity of the program.

A data structure I found particularly useful were lists for Gift Wrap. The lists enabled me to iterate over lists of points in order to sort them accordingly to the specifications of the Gift Wrap.

A problem I encountered was improving the efficiency of the getStartingPoint function, which found the point with the right-most minimum y value. Initially, I iterated through to find all of the y values, and extracted the minimum y values. If there were multiple points with the same value of y, I appended them to a list, iterated through that list and found the point with the largest x value (this was the condition for the starting point). This was not efficient as I had to use two loops, which increased the complexity. I made this more efficient by using one loop and a conditional statement. I had a minimum y value variable that updated whenever a smaller y value was found. The conditional stating that if the minimum y value came across a point with the same y value, compare the x values. Whichever x value was larger, would belong to the point which would become the new minimum.

Another problem I encountered was general confusion. Whenever I had a bug, or had to review some code I had written, I tended to get lost, and would spend a considerable amount of time re-understanding my code. I solved this problem by giving each variable a clear, concise and suggestive name. I also added a lot of comments so I knew what each line was executing. I also divided my program up into various helper functions. This made the program easy to debug, as I was able to test each step of the program to ensure it was all working. Each helper function has a docstring explicitly describing what it does. I then called all of the helper functions into one main() function, that brought the program together and initialized the algorithm.

**Algorithm Analysis:**

This section of the report compares the time that the Graham Scan and the Gift Wrap algorithm took to compute a circular distribution of points and a rectangular distribution of points. To obtain the plots from the graph, I ran each algorithm with randomly generated data plots with 1000-20000 points with a step of 1000 (thus it computed 20 different sets of data), and recorded the results. As each algorithm generated their own random input, the algorithms were not compared by the same points exactly. However, the behaviour observed by each algorithm over time is sufficient for comparison. I ran each algorithm 3 times to ensure consistency of the trend. Any minor differences were due to the random nature of the input. The time recorded (in seconds) was rounded up to a decimal place of 5.

**Circular Distribution:**

|  |  |  |
| --- | --- | --- |
| Points | Gift Wrap Execution Time (s) | Graham Scan Execution Time (s) |
| 1000 | 0.16152 | 0.01348 |
| 2000 | 0.35371 | 0.02736 |
| 3000 | 0.55342 | 0.04311 |
| 4000 | 1.07787 | 0.05062 |
| 5000 | 1.14461 | 3.07529 |
| 6000 | 1.65139 | 0.09084 |
| 7000 | 1.82841 | 0.10723 |
| 8000 | 2.34500 | 0.12104 |
| 9000 | 1.91961 | 0.13850 |
| 10000 | 2.91078 | 0.15628 |
| 11000 | 3.50752 | 0.16620 |
| 12000 | 3.76060 | 0.19111 |
| 13000 | 4.72337 | 0.16042 |
| 14000 | 3.93081 | 0.20002 |
| 15000 | 4.52943 | 0.23719 |
| 16000 | 4.02622 | 0.16309 |
| 17000 | 5.27713 | 0.28376 |
| 18000 | 5.14971 | 0.20549 |
| 19000 | 5.23293 | 0.30058 |
| 20000 | 5.33214 | 0.23137 |

**Table 1** The recorded results of the Gift Wrap algorithm and the Graham Scan algorithm’s relationship with time as the number of points increases with a circular distribution

**Graph 1** The comparison of the Gift Wrap algorithm and the Graham Scan algorithm’s relationship with time as the number of point’s increases with a circular distribution

**Rectangular Distribution:**

|  |  |  |
| --- | --- | --- |
| Points | Gift Wrap Execution Time (s) | Graham Scan Execution Time (s) |
| 1000 | 0.09724 | 0.12577 |
| 2000 | 0.15807 | 0.02566 |
| 3000 | 0.22779 | 0.03930 |
| 4000 | 0.32021 | 0.04135 |
| 5000 | 0.44382 | 0.06534 |
| 6000 | 0.51309 | 0.08139 |
| 7000 | 0.55402 | 0.09201 |
| 8000 | 0.66889 | 0.10634 |
| 9000 | 0.88630 | 0.11909 |
| 10000 | 0.78807 | 0.13598 |
| 11000 | 0.82112 | 0.14793 |
| 12000 | 0.79808 | 0.16123 |
| 13000 | 1.30564 | 0.12002 |
| 14000 | 0.88509 | 0.16995 |
| 15000 | 0.99473 | 0.19938 |
| 16000 | 1.27217 | 0.13417 |
| 17000 | 0.87754 | 0.23062 |
| 18000 | 1.15896 | 0.17426 |
| 19000 | 1.18654 | 0.25062 |
| 20000 | 1.12295 | 0.18948 |

**Table 2** The recorded results of the Gift Wrap algorithm and the Graham Scan algorithm’s relationship with time as the number of points increases with a rectangular distribution

**Graph 2** The comparison of the Gift Wrap algorithm and the Graham Scan algorithm’s relationship with time as the number of point’s increases with a rectangular distribution

**Observations:**

As shown in Graph 1 and Graph 2, the Gift Wrap algorithm is consistently slower when compared against the Graham scan. The spikes observed in the trend are due to the random nature of the input, but overall there is a clear pattern for both algorithms.

Theoretically, the Gift Wrap algorithm has an asymptotic complexity of O(nh) (n being the size of the data set and h being the number of points on the hull). As the number of points increased, so did the number of points of the hull. The Gift Wrap algorithm performed significantly better on the rectangular distribution of points than the circular distribution. This is because the circular distribution tends to have more points on the hull, and thus the time scaled accordingly as expected. The growth rate of the Gift Wrap algorithm matches its theoretical complexity.  
Theoretically, the Graham Scan has an asymptotic complexity of O(n log n). As expected, in both Graph 1 and Graph 2, the Graham scan exhibited a trend of O(n log n) because the points had to be sorted by angle (which has a complexity of O(n log n). The Graham Scan took longer to process the circular distribution of points, but only by a very insignificant amount. This is because the circular distribution tends to have more points on the convex hull, thus it took more time to process (as expected). The growth rate of the Graham scan matches it theoretical complexity.

**Further Improvement:**

The complexity of the Graham Scan can be reduced to O(n) if the list of points given was already sorted by angle. In my version of the Graham Scan, the asymptotic complexity of sorting the list by their angle is O(n log n), because for each point it goes back to see if any of the previous points is making a “right” turn (clock-wise). If the list of points was already sorted by angle, then each point can only appear once as a point if it is making a “left turn” (counter clock-wise) because it will add that point to the hull and increment to the next point. The time to sort dominates the time it takes for the Graham Scan to actually compute the convex hull, therefore if the list is not sorted, the complexity is O(n log n). If I were to pass through a list of points sorted by their angle, my buildSimpleClosedPath function would become redundant, and I could remove it entirely (as it is this function that sorts the list of points accordingly).

**References:**

Dr. R. Mukundun (2015). 1 *Computational Geometry 1.1 Convex Hulls* [Powerpoint Slides]. Retrived from <http://www.learn.canterbury.ac.nz/> on 2015, May 15th

Graham Scan. (n.d.). In *Wikipedia.* Retrieved 2015, May 16th, from <http://en.wikipedia.org/wiki/Graham_scan>

Gift Wrapping Algorithm. (n.d.). In *Wikipedia.* Retrieved 2015, May 16th, from <http://en.wikipedia.org/wiki/Gift_wrapping_algorithm>

Convex Hull, Set 1 (Jarvis’s Algorithm or Wrapping). (n.d.) In *GeeksforGeeks*. Retrieved 2015, May 17th, from http://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/

[Arnaldo Pedro Figuerira Figueia]. (2013, April, 20). 6 Convex Hull. Video from Coursera, Princeton University. [Video file]. Retrieved from <https://www.youtube.com/watch?v=0HZaRu5IupM> 2015, May, 28th

S. Saurabh [saurabhschool]. (2014, August, 12). Programming Interview: Convex Hull Graham’s Scan Algorithm [Video file]. Retrieved from <https://www.youtube.com/watch?v=QYrpHE8iDGg>. 2015, May 28th