Cosc 262 Assignment – Convex Hulls

Algorithm Implementation:

Gift-wrap and Graham scan:

The implementation of the Gift-wrap and Graham Scan algorithms were pulled from the pseudocode given in the lecture material. Assistant functions to these, such as the theta function and line function, were also taken from the material.

However, the implementation of finding the lowest Y value from the data sets had to be self-created, and thus uses a sequential search of the data for the lowest Y value. This perhaps is more naïve than needed, but it would have little effect of the results as both algorithms use this function in their implementation. Additionally, the extra method (which is an implementation of quick hull) uses a similar method for finding lowest/highest X valued points.

Quick-Hull:

The implementation of the quick-hull algorithm was taken from pseudocode given on the Wikipedia page. The function takes its name from the similarity to the quicksort algorithm. Firstly, it takes in a set of data points (*listPts*) with at least two points. Then it finds the highest and lowest X points in the set using the *lowestX*/*highestX* functions and adds them to the *chull* list. Then dividing the remaining points into two lists which contains the points on the right or left of the line (*abovePts*, *belowPts*) between the two X points. Then it passes those lists and the *maxX*/*minX* values into the recursive function *findHull*. This function has the base case of whether the inputted *listPts* is empty, returning. Otherwise, it finds the furthest point for the line inputted points A & B then if that point isn't part of the line AB it is added to the *chull* list. Finally, it recursively repeats this process with the points above line A, *maxPt* and the points below the line *maxPt*, B until the base case is reached. At which point the convex hull is returned.

Algorithm Analysis:

The algorithms were all tested using the code contained within the file *convexhull_time.py*. The python built-in module, time, was used to gather timings for each trial for each dataset for each algorithm out of the three. Each dataset file was tested in 30 trials to gain the averages graphed below. The test convex hull results were tested against the results stored in the list *RESULTS* in the file *results.py* for the sake of accuracy.

The testing file is runnable for personal testing and will save the results in the file *Data.txt* (which will need to be deleted to be regenerated, to avoid deleting previous results unintentionally). The program takes in the amount of trials per data file to be run and returns the averages for each file and the total for each data set for all three algorithms. (the average reported is the mean of the values gained in the individual trials, and the total is just the sum of those averages)

Results for dataset A:

Results for the first dataset (which contained low convex hull points, *see fig. 1a*) show mainly the general best-case complexity for the three algorithms. They all display fairly linear complexity for the running times in this set (*see fig. 1b*), increasing positively with the increase in the amount of points in the set. However, the Gift-wrap algorithm starts to show its O(nm) complexity in the larger sets taking a slightly longer time in the sets with 5 convex hull points than the other algorithms. It is also interesting to note the sharp change of performance between quick hull and graham scan in the 24000 and 30000 data sets. Fortunately, the difference is minor enough to be considered insignificant.

	GiftWrap	Graham Scan	Quick Hull	Chull points
3000	0.062	0.0646	0.0495	3
6000	0.124	0.1359	0.1224	3
9000	0.1828	0.1917	0.1812	3
12000	0.2802	0.2411	0.2266	4
15000	0.2958	0.3151	0.2844	4
18000	0.3791	0.3427	0.3528	4
21000	0.4984	0.4302	0.4198	4
24000	0.651	0.413	0.4943	5
27000	0.7354	0.525	0.5127	5
30000	0.788	0.5213	0.6036	5
total	3.9968	3.1807	3.2474	

figure 1a – Data results table for dataset A

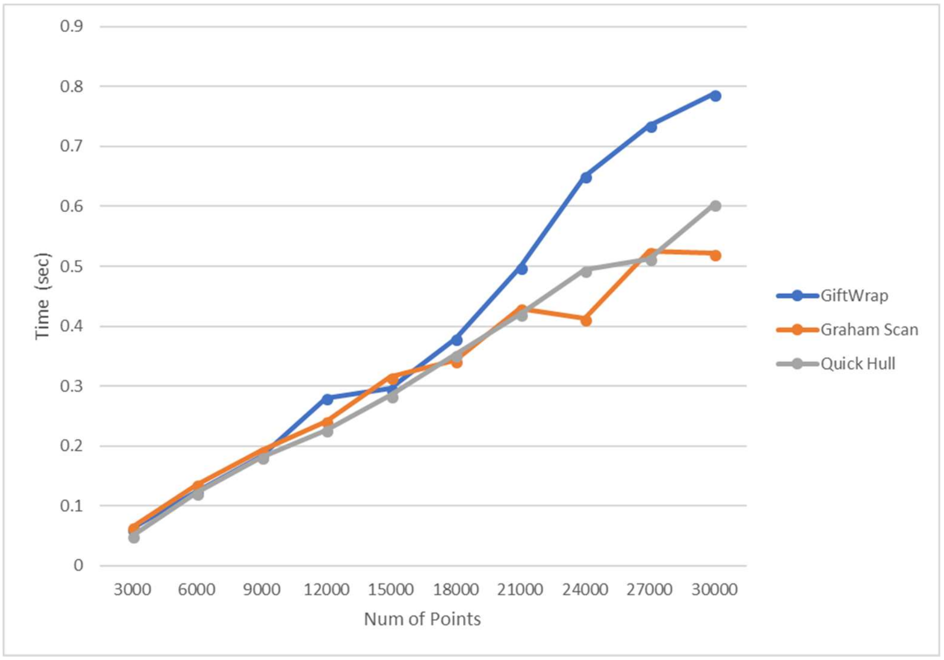

figure 1b – Graph of points vs time for algorithms in dataset A

Results of dataset B:

Results for the second dataset more obviously show the differences between the three algorithms, causing the worst cases to be visible for gift-wrap (but not quick-hull). Data set B contains Convex hulls with far more points than set A, increasing linearly with the points (1% of the total points *see fig. 2a*). Both quick hull and graham scan show their continued pseudolinear increase with the number of points as with data set A, seemingly ignoring the difference in convex hull points.

Gift-wrap however, shows its true failings here. The gift-wrap algorithm being O(nm) complexity (m being the number of convex hull vertices) means that with the linear increase of both the number of points, and the convex hull points, the complexity mimics that of O(n^2) and performance for the B data set exponentially deteriorates to the point of taking half minutes to complete the hull on the higher sets (*see fig. 2b*).

	GiftWrap	Graham Scan	Quick Hull	Chull points
3000	0.3656	0.0672	0.0745	30
6000	1.3672	0.1323	0.1292	60
9000	3.0422	0.1844	0.2052	90
12000	5.1646	0.2646	0.2505	120
15000	8.3202	0.3203	0.3302	150
18000	11.9521	0.3651	0.3505	180
21000	16.2891	0.4109	0.4943	210
24000	21.0516	0.4865	0.5323	240
27000	26.2302	0.55	0.5599	270
30000	32.5474	0.6208	0.6224	300
total	126.33	3.4021	3.549	

figure 2a – Data results table for dataset B

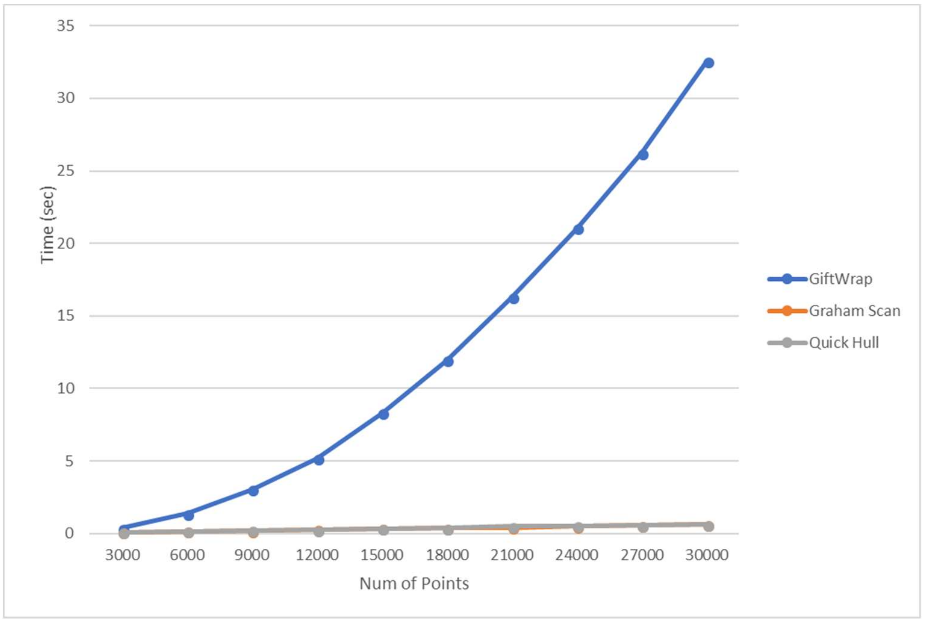

figure 2b – Graph of points vs time for algorithms in dataset B

Comparison:

Conclusively, here the gift-wrap algorithm shows that it isn't the best option for finding convex hulls. With a best-case complexity of O(nlogn) and worst of O(n^2) it under performed when compared to other options such as the quick-hull and graham scan algorithms. This can be attributed to that for every point that is found to be the hull, a comparison with every other point is completed, which proves to be inefficient.

Also, noting that the Graham scan algorithm has a complexity of O(nlogn) in both best and worst case makes it the better option for convex hulls with high numbers of edges, as well as providing a more consistent performance. Mainly being due to that after the O(nlogn) sorting step, there is only a comparison with every point afterwards.

Alternatively, the quick-hull algorithm also proves to be effective here, but still has a worst case of O(n^2), much in the same way that quick sort suffers. Their similarities unfortunately extend to the worst parts too. For example, if the line AB chosen contains all the points, bar a few, on one side of the line, the running time quickly moves to exponential (much like if the pivot for quick sort picks the highest or lowest value in the list). Unfortunately, here we can barely see this issue as the points seem fairly evenly distributed in the datasets, but seeing as this problem exists, the conclusion is that the graham scan algorithm provides the best results for the problem of convex hulls when compared to the other two algorithms tested here.

References:

Aside from the information taken form the lecture material, some information was taken from other locations:

https://en.wikipedia.org/wiki/Quickhull

for the pseudocode for quick-hull

https://stackoverflow.com/

for assistance with finding solutions for timing and printing replacement in the testing program