Report on Convex Hull algorithms and their Time Complexities

Hong Kai Koh

64276083

HKK18

COSC 262 - Algorithms

Abstract

This report will discuss on the different algorithms used to build convex hulls. The program uses three main functions – giftwrap(listPts), grahamscan(listPts) and amethod(listPts) – to encapsulate each of the algorithms process the .dat files provided into convex hulls. Of the three main functions, there are the Gift Wrap and Graham Scan algorithms, as well as another non-naïve method to be implemented into the amethod(listPts) function; Andrew's Monotone Chain algorithm is chosen. Furthermore, this report will not only analyse and compare the time complexity across all three algorithms, but also between Set A and Set B within each algorithm. Python is used as the language for this program.

1 Algorithm Implementation

1.1 Gift Wrap

The Gift Wrap algorithm, otherwise known as Jarvis march, named after R. A. Jarvis (Jarvis, 1973) starts with finding the coordinates with the minimum y value by calling the get_min_index(listPts) function. In cases where the y values are the same, the helper function will then select the rightmost coordinates as the starting point. The algorithm then forms the hull by computing the minimum angle between the current point and the next, using theta(pointA, pointB). The two helper functions are ironically the core for getting the required minimum y value and angle between two points and these are very helpful in the implementation of the Gift Wrap algorithm. Using tuples to store the coordinates is an efficient way as the x and y values can be easily retrieved by slicing the tuples, index 0 for x-coordinate and 1 for y-coordinate. Also, this is easier and simpler to implement as compared to Graham Scan and the other method, but it's efficiency will be discussed more in the Algorithm Analysis part.

1.2 Graham Scan

This algorithm begins by constructing a simple closed path, by calling the simple_closed_path(listPts) helper function, which also uses get_min_index(listPts) and theta(pointA, pointB) in its computation

of the path. The then proceeds to find and build the hull in a counter-clockwise fashion. This works by calling isCCW(ptA, ptB, ptC), which checks if the directed line of the three points – using lineFn(ptA, ptB, ptC) – is greater than 0. If it is, the current coordinates will be pushed on to the Stack, otherwise the Stack will pop the topmost coordinates. The original algorithm works with a Stack to store the coordinates of the convex hull. However, since the program is coded in Python, using the built-in List data structure is a more convenient way while not compromising any of the effectiveness of using a Stack data structure. This is because Python's list data structure has methods such as list.pop() and list.append() functions which are both O(1) in terms of time complexities, just like a Stack's push and pop. Also, both data structures have a First-In-First-Out way of adding and removing items so using Python's list here is more convenient than having to first import and then use Stack.

1.3 Amethod – Andrew's Monotone Chain

The non-naïve method used in the program is Andrew's Monotone Chain algorithm. This is a variant of the Graham Scan algorithm so the time-complexity is very similar to that of the Graham Scan algorithm. One key difference in this method compared to the former two methods is that the algorithm starts off by sorted the list of coordinates lexicographically by smallest x-coordinate instead of smallest y-coordinate. In case of a tie, the point with the larger y-coordinate is selected. Hence, it only requires one additional helper functions – which is finding the directed line by calling lineFn(ptA, ptB, ptC) – as Python's sorted(list) takes care of the lexicographical sorting. Although there is seemingly another helper function, process_chull(listPts, order=0), used for this algorithm in the program, it is actually just breaking up the main function into two parts to complement the style of the code. After sorting, the algorithm proceeds to form the lower and upper parts of the convex hull. The upper hull is formed in a counter-clockwise fashion (Andrew, 1979) while the lower hull is simply the rest of the convex hull. The code for the algorithm is derived from the pseudocode obtained from the Algorithmist website.

2 Algorithm Analysis

For measurements of time complexities and performance of each algorithm, the program used the time() from Python's time module to keep track of the start time before an algorithm is run and takes the difference between the start time and the time after the algorithm has finished processing the dataset.

2.1 Gift Wrap

Size of Inputs (N)	Time take to process Set A (s)	Time take to process Set B (s)
3000	0.031253099	0.22846508
6000	0.046877861	0.83282423
9000	0.062506199	1.91157341
12000	0.12504077	3.884077787
15000	0.109380722	5.481898546
18000	0.188056946	7.53375864
21000	0.218770742	10.14396358
24000	0.312502146	13.31190276
27000	0.328194857	16.76644564
30000	0.312533379	20.67716885

Figure 1.1 Table for performance comparison for Gift Wrap algorithm between Set A and Set B

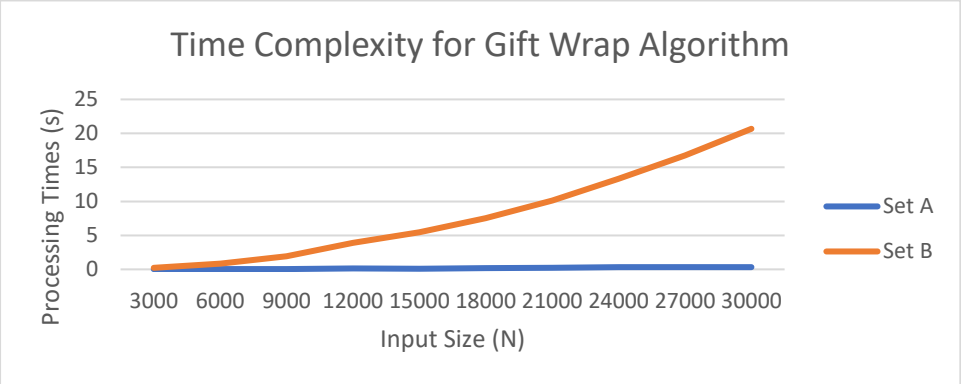

Figure 1.2 Graph for performance comparison for Gift Wrap algorithm between Set A and Set B

Evident from the data provided above, the Gift Wrap algorithm deals well with small datasets; when input size, N, and the number of vertices in the convex hull, h, are small. As the values for N and h increases, the time needed to finish processing each dataset increases exponentially. Since the dataset in A has very few vertices, the expected time complexity will be O(nh), with n being the number of points in the set and h being the number of vertices in the hull. However, in Set B, the number of hull vertices increases to a point where it eventually degrades the time complexity for this algorithm to $O(n^2)$. This is consistent with the time complexity of the algorithm, of O(nh) normally and slowing down to $O(n^2)$ in the worst-case scenario (Jarvis, 1973). Thus, Gift Wrap algorithm is more suitable in handling small datasets, because the running time depends linearly on the number of hull vertices. Therefore, it is more efficient than other O(n log n) algorithms like Graham Scan when the number of hull vertices, h, is smaller than log n (Jarvis, 1973).

2.2 Graham Scan

Size of Inputs (N)	Time take to process Set A (s)	Time take to process Set B (s)
3000	0.032557726	0.03122735
6000	0.046879768	0.06250453
9000	0.093757868	0.093784094
12000	0.124980211	0.124981403
15000	0.156262398	0.156294346
18000	0.187520504	0.187547922
21000	0.21874094	0.210721731
24000	0.23439765	0.265683651
27000	0.281248331	0.281251192
30000	0.296981096	0.312501192

Figure 2.1 Table for performance comparison for Graham Scan algorithm between Set A and Set B

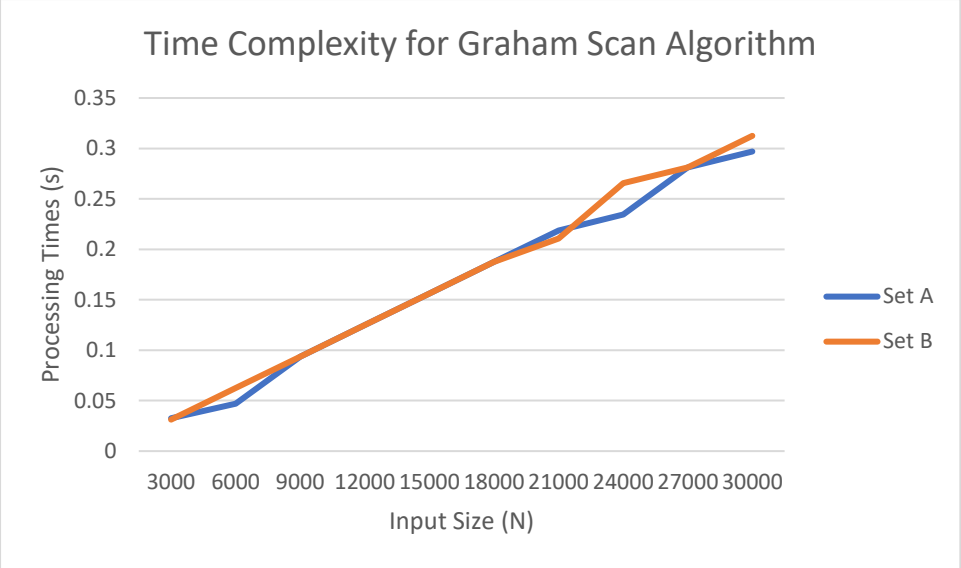

Figure 2.2 Graph for performance comparison for Graham Scan algorithm between Set A and Set B

From the graph and table above, there is a linear increase in processing times as N and h grows larger, unlike the Gift Wrap algorithm which increases exponentially instead. This is supported by the fact that the time complexity of the Graham San Algorithm takes O(n log n) due to the need for sorting the points. That is, the solution for the simple closed path requires O(n log n). It requires the computation of the angle between a line and a horizontal line using the inverse trigonometric function. Moreover, while it may appear to be that the loop in the algorithm takes $O(n^2)$ which would otherwise bring the overall complexity up to $O(n^2)$, it only takes O(n) to check through each point twice (Graham, 1972). Thus, the overall time complexity is still capped at O(n log n) since forming the simple closed path consumes majority of the time needed to build the convex hull. This is why the processing times are more consistent when comparing between Set A and Set B, as expected from the time complexity of the algorithm itself.

2.3 Amethod – Andrew's Monotone Chain

Size of Inputs (N)	Time take to process Set A (s)	Time take to process Set B (s)
3000	0.031253576	0.046883583
6000	0.06306839	0.06253314
9000	0.092037916	0.093762636
12000	0.109416008	0.109385252
15000	0.140606165	0.124984026
18000	0.171891689	0.171921253
21000	0.187521219	0.189185143
24000	0.232931852	0.232190609
27000	0.265652657	0.249995232
30000	0.296937704	0.296872616

Figure 3.1 Table for performance comparison for Monotone Chain algorithm between Set A and Set B

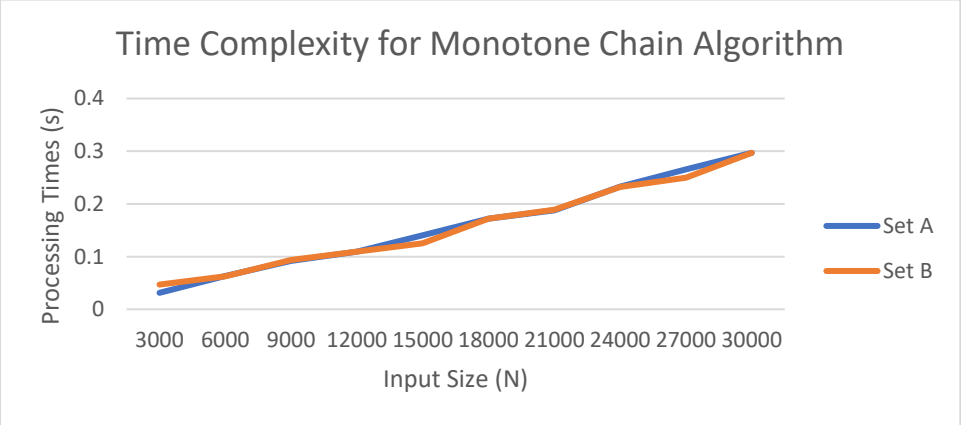

Figure 3.2 Graph for performance comparison for Monotone Chain algorithm between Set A and Set B

For Andrew's Monotone Chain algorithm, not much of a difference can be seen between the processing times of Set A and Set B as this algorithm also takes O(n log n) time as the number of points in the dataset, N, and number of hull vertices, h, grow bigger. This algorithm is a variant, and slight improvement, of the Graham Scan which means that similar processing times can be expected to be observed. The processing times also increase linearly as N and h increases.

2.4 Time Complexities for all three algorithms

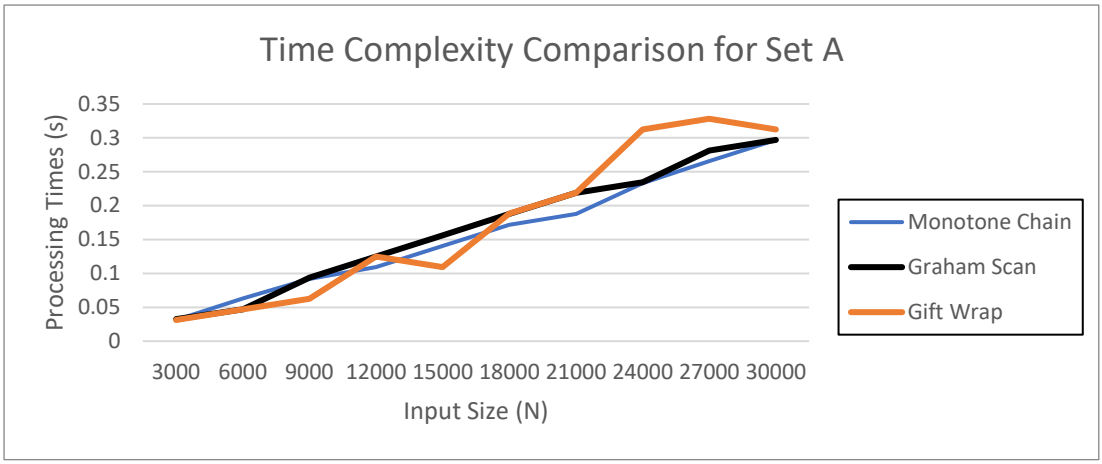

Figure 4.1 Graph for performance comparison among all three algorithms for Set A

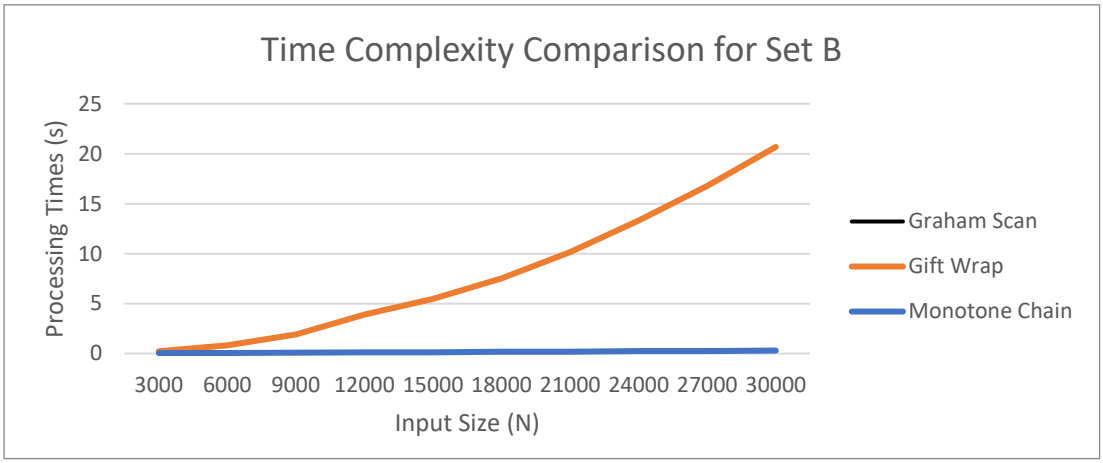

Figure 4.2 Graph for performance comparison among all three algorithms for Set B

As a clarification, the plots for Graham Scan and Monotone Chain algorithms in Figure 4.2 overlap each other due to the large processing times for the Gift Wrap algorithm, such that only the plot for Monotone Chain is represented in the graph, while the plot for Graham Scan is hidden below it. From the graph for the comparison between all three algorithms for Set A, it can be observed that all the algorithms have similar processing times when the number of hull vertices, h is particularly small. For some cases where N is small as well, the Gift Wrap algorithm actually performs better than the other two. This is expected as the running time of the Gift Wrap algorithm, O(nh), is critically dependent on h. When h is smaller than log n, the algorithm is faster than any other O(n log n) convex hull algorithms

and makes the algorithm a more efficient method for building smaller convex hulls as it is easy to implement. However, as h increases, the worst-case time complexity can go up to $O(n^2)$, as seen in the graph for Set B. In such cases, choosing either of the Graham Scan or Andrew's Monotone Chain algorithm is a more viable and efficient way of solving convex hull problems. For the Graham Scan and Andrew's Monotone Chain algorithms, they have O(n log n) time complexities which means that the processing times will not increase as drastically as that of the Gift Wrap algorithm, evident in the graph for Set B where the number of hull vertices is large. Aside their similarities, the Monotone Chain algorithm has a slight advantage over the Graham Scan when the input list has already been sorted as the algorithm now only takes O(n) time complexity to complete whereas it will still take O(n log n) for the Graham Scan algorithm in this case. Moreover, the Monotone Chain may be preferred over the Graham Scan algorithm due to the former having fewer functions to implement than the latter.

3 Conclusion

To sum off, for particularly small numbers of points in the set, N, and hull vertices, h, the Gift Wrap algorithm has the best results compared to Graham Scan and Andrew's Monotone Chain algorithms due to the simple implementation of the Gift Wrap algorithm and the linear dependency on the value of h. But the efficiency of the Graham Scan and Andrew's Monotone Chain algorithms will shine as the values N and h increase, and the Monotone Chain algorithm has the slightest edge over the Graham Scan in terms of performance when the list of points provided has already been sorted accordingly.

Reference List

Andrew, A. M. (1979). Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters, 9(5), 216-219.

Graham, R. L. (1972). An efficient algorith for determining the convex hull of a finite planar set. Information processing letters, 1(4), 132-133.

Jarvis, R. A. (1973). On the identification of the convex hull of a finite set of points in the plane. Information processing letters, 2(1), 18-21.

Pseudocode for Andrew's Monotone Chain Algorithm. http://www.algorithmist.com/index.php/Monotone_Chain_Convex_Hull. Retrieved on 25 May 2018.