REPORT

Prepare a report detailing your work. The report should include the following sections:

1) Algorithm implementation: In this section, you may discuss any specific data structures or Python functions that you found useful for your implementation. You may also discuss problems encountered while implementing or testing the algorithms, and how you could solve them. Please also give a description of the third algorithm ("amethod").

For the three algorithms I found using a list (array) the most useful. It was useful as it could store the points in the file that contained the data points easily. As they keep the index they are given it was easy to keep track of the order and the point in the specific index. Using a list as a data structure was especially useful in the graham scan algorithm where we had to store the points based on the increasing angles from the first point on the hull we found. As lists already have a built in pop method and append method it made it easier to remove the last point on the list if we determined that it was not in the convex hull and also add a point at the end of the list if the point could possibly be in the convex hull.
Producing the convex hull in the order of the corresponding .out file is made possible by using a list with the index of the points on the convex hull not changing, it made it easier to see if we have the same output that the .out file needed for all the three methods.

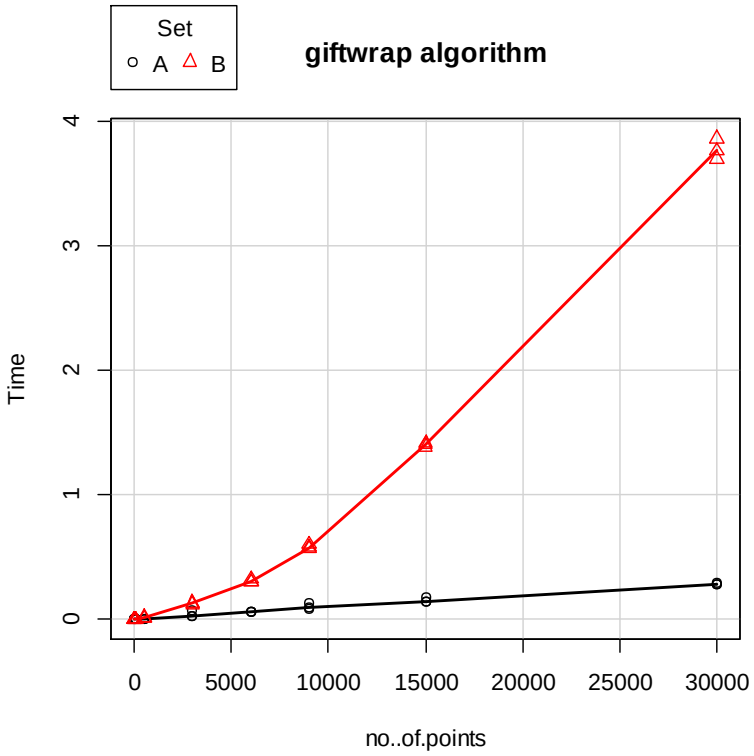
In implementing the gift wrap pseudo code, I found that on some of the files in set A never exited the loop, never finding k to be equal to n. I found that this was because some of the files in set A had a point in the file where it had the same y-value as the first point we determined to be in the convex hull. At this point the point next on the convex hull reached a degenerate case where the angle it gave to the next point on the convex hull was 0.0, so my original code never accepted this point, as at this point the angle is always less than v (angle of the previous point on the convex hull), with the angle always being 0.0 if it had the same y value. The way I fixed this in my code for the gift wrap algorithm is to update k to j if the angle was 0.0 and also if the angle it gave to j and the first point we found on the hull is 0.0 and if the point we are comparing it to is not the same point we previously found. By doing this we finally find that at some point, k will be equal to n, exiting the loop when k is n.

For the "amethod" method I used a variation on the gift wrap algorithm, but instead of updating the next possible point on the hull to the point that has the next minimum angle we update the point, q if it's either counter-clockwise or if it's straight with the point farthest away from p. Once we go through all the points and find the next point on the convex hull we check if it's the first point we found originally found (this is the point with the right-most minimum y value), if it is we are done and have found all the points on the hull, if it's not we go through the data points again finding the next point on the convex hull.

I made tests.py to test all the data files in set A (using test_setA method) and set B (using test_setB method) with the three algorithms to see if the output of all three algorithms matches the corresponding expected output provided by the .out files (I made a list of the numbers from the .out file using expected_output method I created). Both of the methods will print the time it took to get the convex hull for each of the algorithms and print a boolean value depending if it matches the .out file. To make the testing easier I put all the data and output files in the same directory as the convexhull.py and tests.py.

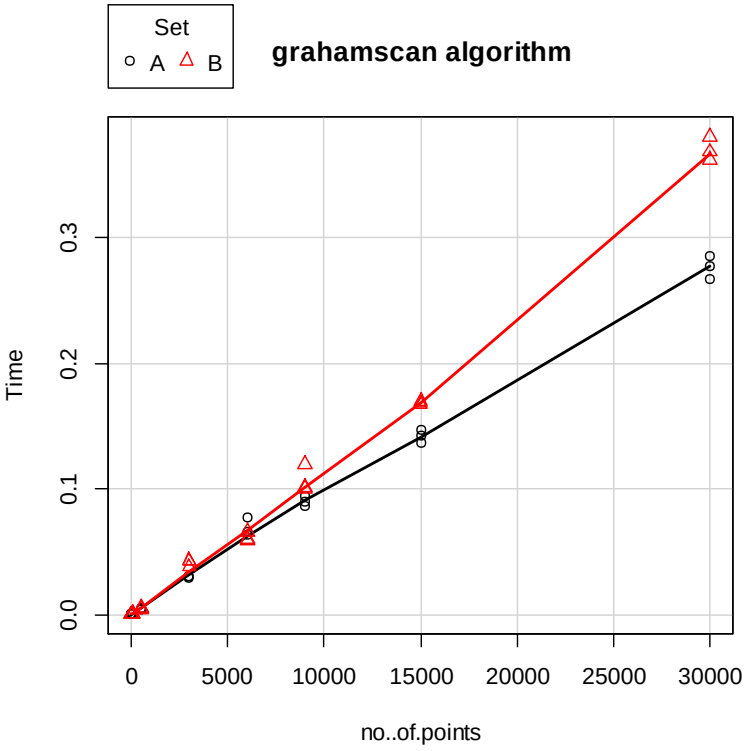
2) Algorithm analysis: Provide the results of the comparative analysis in the form of graphs as described in the previous section, and give your interpretation of the results. If you have proposed an improvement of either the giftwrap algorithm or the Graham-scan algorithm in "amethod", describe how the suggested improvement could be seen in experimental results.

a) How did the results vary for each algorithm? Please give a brief explanation of the trends seen in each graph, unexpected variations if any, and similarities to (or deviations from) the theoretical measures of complexity.

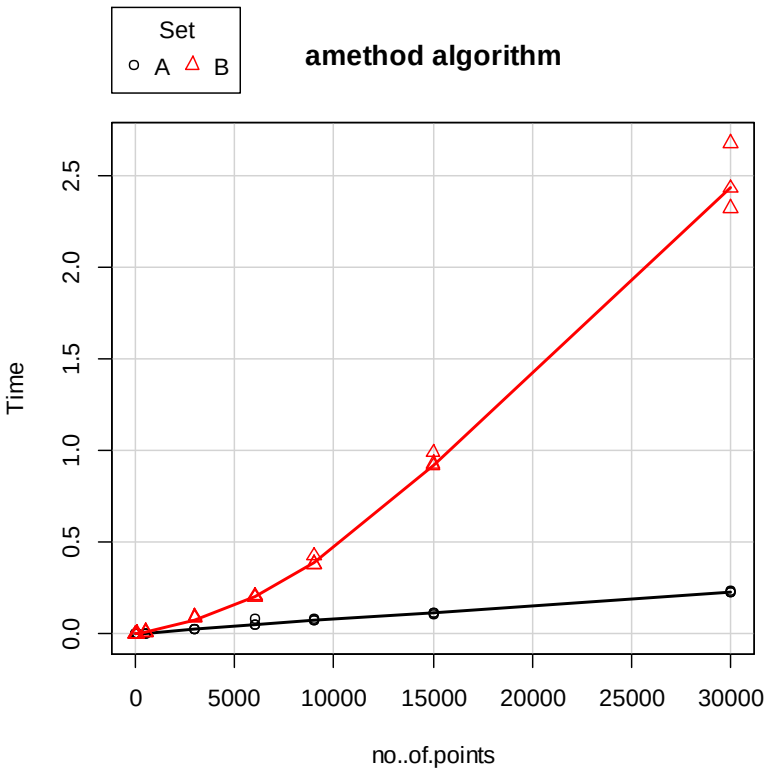


Number of points on the convex hull		
Number of data points	**Set A**	**Set B**
10	3	7
50	3	9
500	4	15
3000	4	25
6000	4	30
9000	4	40
15000	4	60
30000	4	80

From the gift wrap algorithm graph we can see that for set A, the time it takes to get the points of the convex hull is relatively similar as the number of data points increases. For set B we see that the time it takes to get the points of the convex hull drastically increases as the number of data points increases and as the number of points of the convex hull increases. This is mainly because of the number of times we have to repeat the algorithm while we have not found all the points in the convex hull. So if we have a lot of points in the convex hull we are going to loop through the algorithm a lot of times to find the next point in the convex hull, therefore it will take longer if the convex hull has a lot of points. The theoretical complexity for the gift wrap algorithm is O(mn), where n is the number of points and m is the number of points in the convex hull, so the graph produced is similar to the theoretical complexity, as we can see that as the number of points on the convex hull increases the time it takes also massively increases (assuming the number of data points remained constant). We can see this clearly in the 30000 data points, for set A it takes less than half a second but for set B we see that it takes nearly 4 seconds to get the points on the convex hull.



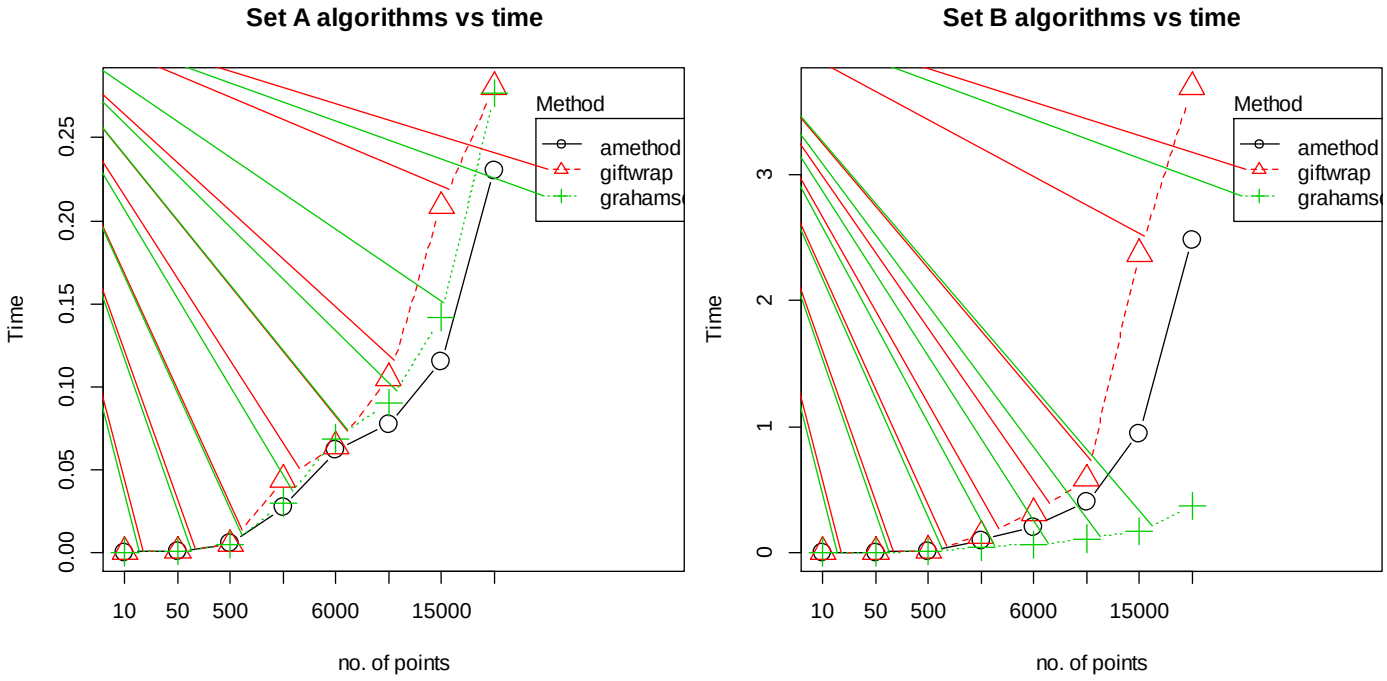
From the graham scan algorithm graph we can see that the time it takes to get the points of the convex hull for set A and set B looks to be almost linear with a slight increase of gradient in the 30000 points in the file (seen more so in set B). We can see that as the number of data points increases the time it takes to get all the points on convex hull increases. The theoretical complexity of the graham scan algorithm is O(nlogn), where n is the number of data points in the file, which is similar to the results of the graph especially from the smaller files (10-6000 data points) we can see that the time it took to get all the points on the convex hull for set A and B are very similar to each other. As we get to bigger files we see that the time increases more so in the set B files compared to the corresponding number of data points set A, which is not a massive increase (~0.1s) especially compared to the other two algorithms implemented.



For the amethod algorithm I chose to use a variation of the gift wrap algorithm, so the theoretical complexity of this algorithm should also be O(mn). We can see from the amethod algorithm graph the same trend is happening with this algorithm as the gift wrap algorithm graph which is that as the number of points on the convex hull increases so does the time (with the same number of data points). We see this by comparing the time it takes to get the convex hull for set A and set B, as the number of points on the convex hull are almost the same for set A we see a relatively small difference in time for the number of points in the files. For set B, as there are more points on the convex hull the time it takes to get all the points on the convex hull has a big difference in time when compared to the corresponding file in set A that has the same amount of data points. But we can see that it is more efficient than the gift wrap method as it takes less time especially for the bigger data points with more points on the convex hull (for 30000 in set B it takes ~2.5 seconds compared to ~4 seconds for the gift wrap algorithm above). The time difference is probably due to the functions used (theta, line_fn, dist), the number of assignments and the number of comparisons to check if the point is in the convex hull (amethod has less comparisons in the loop).

b). Which one of the three algorithms gave the best result in performance analysis? Why did that algorithm perform better than the others?

The algorithm that had the best result in performance analysis overall is the graham scan algorithm. When we performed all the algorithms for the set A data points we see that the time to get all the points on the convex hull is almost the same for all the algorithms, as they have the same trend happening which is seen in the graph Set A algorithms vs time. But when we performed the algorithms on set B we can see clearly that the graham scan algorithm performs the best out of all the three algorithms, this is seen in the graph set B algorithms vs time. This is because for the graham scan algorithm we sort the data points by minimum to maximum angle from the right-most minimum point (sorting takes O(nlogn) finding the minimum point O(n)) and use a stack to store the possible points on the convex hull going through the list of points only once (O(n)), removing points on the stack if we find the points are not on the convex hull and adding on the stack if it could be on the convex hull (both O(1)). Overall the time complexity of the graham scan algorithm is O(nlogn). For the two gift wrap algorithms we perform tests (O(1)) to find the next point on the convex hull for all the data points (O(n)), We repeat this process until we have found all the points in the convex hull (O(mn) with the best case performing the loop three times, worst case performing the loop n times, so that m = n). From this we can see that it is best to use the graham scan algorithm to get the convex hull as it is not output sensitive like the other algorithms tested where the number of points in the convex hull matters to how fast it will perform.



3) References Required only if you found any source of information or resource particularly useful for your understanding of the algorithms, program development, or analysis.

http://tomswitzer.net/2009/12/jarvis-march/
http://tomswitzer.net/2010/03/graham-scan/
http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_cs_07w/Webprojects/Zhaobo_hull/