I liked the more in-depth analysis of the results of the experiments.
I liked that you divided the running time by the runtime given by the theoretical analysis. The only minor issue is that it looks like the "theoretical analysis" is wrong for quickhull since the plot for quickhull is clearly not a constant. This discrepancy requires extra investigation. But as I said, this is only a minor issue.
But otherwise, this is a very nice report and with good experiments.

Other small things you could have done:
* Plot the output sizes on log scale to discover the pattern (details in the report)
* Separate sorting time from computation time in GS
* Use counters and other things to confirm or dispute other theoretical claims about various algorithms.

Computatinal Geometry: Theory and Experimentation (2019)
Project 2: Convex Hull Computation

Group 3
Bingsen Wang - au641000
Zhile Jiang - au641092
Fatemeh Zardbani - au640364

October 23, 2019

1 Setup

1.1 Machine

All the tests were run on the same machine, as suggested in the project description. The machine is a MacOS with 32GB ddr4 RAM which means all the test data can be stored in RAM and we don't have to count on disk access time.

1.2 Analysis

We create three input case as required

- all points within a square with size s
- all points within a circle with radis s^2
- all points on the curve $y = x^2$ with $|x| <= s$

and make the number of points n between 10^2 and $2 * 10^7$ with s $=10^3, 10^4, 5 * 10^4$.

2 Part A.

2.1 The Grahams Scan we implemented

Following the textbook and the lecture slides, we implement the Grahams Scan as first sort all the points by their coordinates then compute the upper hull and lower hull, respectively.
As analysis in book, the time complexity of Graham's Scan is mainly depends on the sort which is $O(nlogn)$ if n is the number of points.

2.2 Our Analysis and Hypothesis

Not sure if I follow this ..

For same n, the set of points is completely disordered which means the time to sort this point is equivalent. But for the curve case, it is more likely to exists collision points, which means the true number of points are less, this will cause it a little quick than the square or circle case.
In general, for different input case, performance of Graham Scan will be quite same and accepetable.

I agree

3 Part B.

3.1 The Quickhull we implemented

We implement the quickhull following the pseudocode in slides. It calculate the upper hull and lower hull independently.

3.2 Our Analysis and Hypothesis

Needs to be more precise, because "totally random" is not really mathematically well defined. But I agree with your general argument.

For totally random data, this algorithm should have time complexity of $O(n)$. The number of points in a certain area is proportional to the area in random case. Since we block a triangle each time, we intuitively think that the sum of all calculated areas should be a constant ratio to the initial area. Which means time complexity is $O(n)$.

"totally random" should be "random data follows uniform distribution".

Also for random data, but all the point are on convex hull. The time complexity should be $O(n \log n)$. First, in this situation, we cannot prune any points. Second, the furthest point should divide the point set to two subset where the number of points are very close. So the depth of recursion is $\log n$. In each recursion we need to do $n * 2^{-d}$ times calculation.

Again, I agree with the general idea of why this algorithm should work fine in practice.

When all the points are on convex hull and each time it find a furthest point, this point cannot divide the point set to two subsets but just remove itself from point set, we will meet the worst case to quickhull. Because in this situation, we need to do $(n-2)+(n-3)+\ldots+2 = n(n-3)/2$ times calculation for distance between the point and the segment. The time complexity should be $O(n^2)$ in worst case.

Very nice!

In our test, square input case and circle input case are both consist of some points generated randomly in a limit area. So our program should performs as $O(n)$.

And quadratic curve input case is a perfect example for random data on convex hull. It is because we have $f'(x) = 2x$ and the slope of the line determined by any two points (x_1, x_1^2),(x_2, x_2^2) on this curve is $x1 + x2$. By these two properties, we can always get two subsets with the same size. So our program should perform as $O(n \log n)$.

Seems reasonable, although this needs a mathematical proof but of course it was not required for you to prove such things.

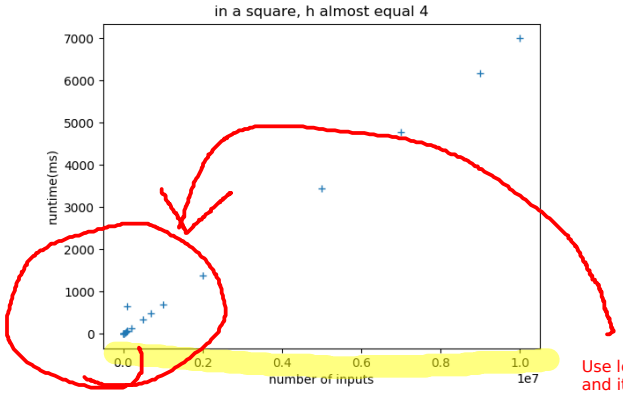
4 Part C.

4.1 The Gift Wrapping we implemented

We implement the Gift-Wrapping algorithm following the pseudo-code given in the project description. The results are those of its compilation and run on the same machine. The code simply initiates the pivot to the left-most point and then iterates until we reach it again. In each iteration, it will find the point with the least angle with the current ray and add it to the convex hull. As the angles are computed via the a-tangent function and for the sake of comparison the result will have to become positive, there may be some floating point inaccuracies, in the case of highly dense workloads. I never faced them, but I imagine they may arise.

4.2 Our Analysis

The analysis of the gift-wrapping method shows that it should follow the complexity of $O(nh)$, h being the number of points on the hull. To test that, we show the results in these two cases; First, we set the points in a square, so that h should roughly be 4 and then increase the number of input points(n), we expect the run time to grow linearly, and it does.



Use log scale. Most of your data is compressed in the corner and it makes it impossible to get any useful information from this plot.

Second, we try to increase h, by increasing the radius of the circle and also the number of input points, and the run time grows almost linearly, which is as expected.

5 Part D.

5.1 The MB_CH we implemented

We implement the MB_CH using linear programming to find the bridge. During coding, we find that using dividing in Computational Geometry is very dangerous. In order to avoid precision errors as much as possible, we represent a line by two points on it instead of by expression $y = k * x + b$ or $Ax + By = C$.

Interesting!

5.2 Our Analysis and Hypothesis

According to the analysis in lecture, the running time of MB_CH is $O(n \log h)$.
But during our analysis, we find it will do much more cross product that other models during linear programming. And we also need to divide the points for three times each recursion. The constant should be very big. So it may not perform very fast in the tests.

You can actually optimize MbC a lot and you can lower the constants considerably, but as we discussed in the Alg-Eng part, the optimizations should not change the general behaviour of the algorithm.

6 Comparative Experiment

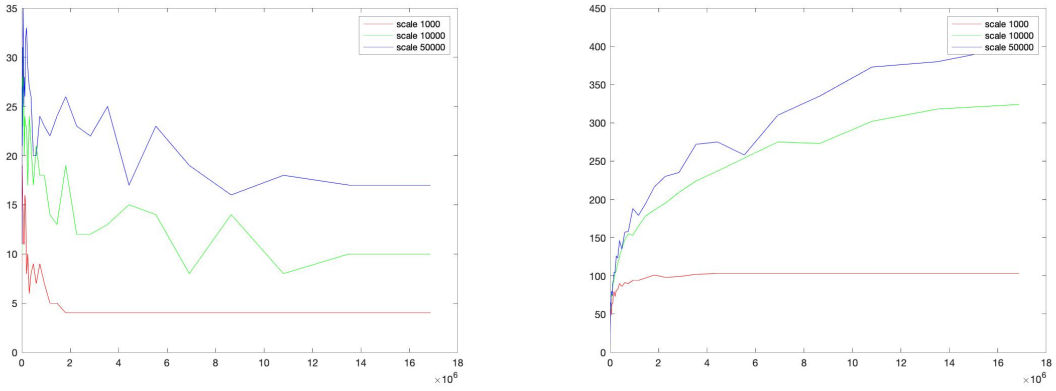
In this section, we first analyze our input test case about the true number on the convex hull, then, we use them to test above 4 convex hull algorithm and compare their difference.

6.1 Number of points on the convex hull

When the shape of the domain in which we randomly choose all the points is given, we can say that when the number of points get larger the shape of convex hull will get close to the shape of domain.
Therefore we can predicate that in square, when n is large enough, there will be only 4 points on the final convex hull, exactly 4 corner of square. But in circle and quadratic curve, it will be larger and larger, and will not converge to some constant number.
We also compare the number of points on convex hull computed by different algorithm to verify the correctness of our models. Of course all the models should get the same number.
Specific numbers are given in figure 1.

FYI, the output size for square is \Theta(log n) and for cicle is \Theta(n^{1/3}). You can discover these by plotting the output size using log scale. As we discussed in the algorithm engineering part of the course, if you have an unknown function y=F(x) and if you plot using log scale for x and y, then if F(x)=log x then the plot looks linear. If you plot log(F(x)), then if F(x) = n^a for some constant a, then the plot looks like a linear function with slope "a". You can actually experimentally measure the slope of 1/3 and conclude that the cicle case has output size n^{1/3}.

Interesting, this means that an O(n log h) algorithm is O(n loglog n) for square and O(n log n) for circle!



(a) square (b) circle

Figure 1: number of points on curve

6.2 Run time analysis

We have done the run time test for n from 10^3 to $6 * 10^7$, with a large domain to choose points.

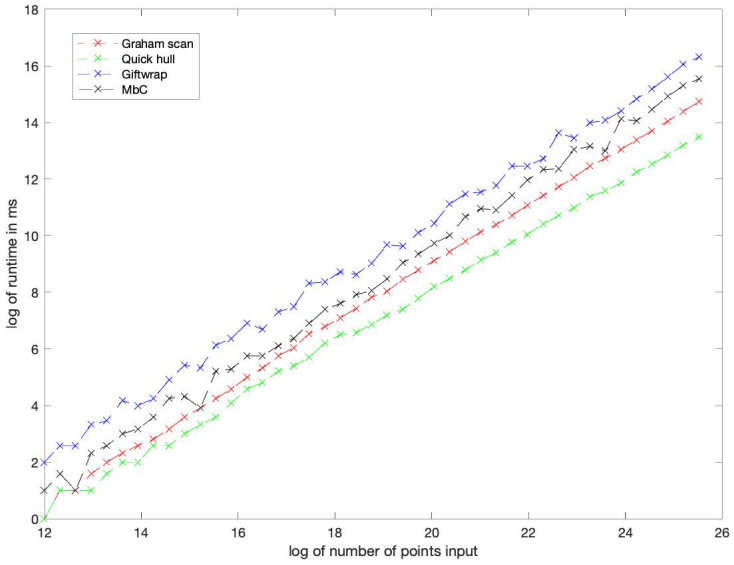


Figure 2: run time on square input case

From figure 2 we can see that all the performance of 4 algorithm are very close while the Quick hull is the most fast on.

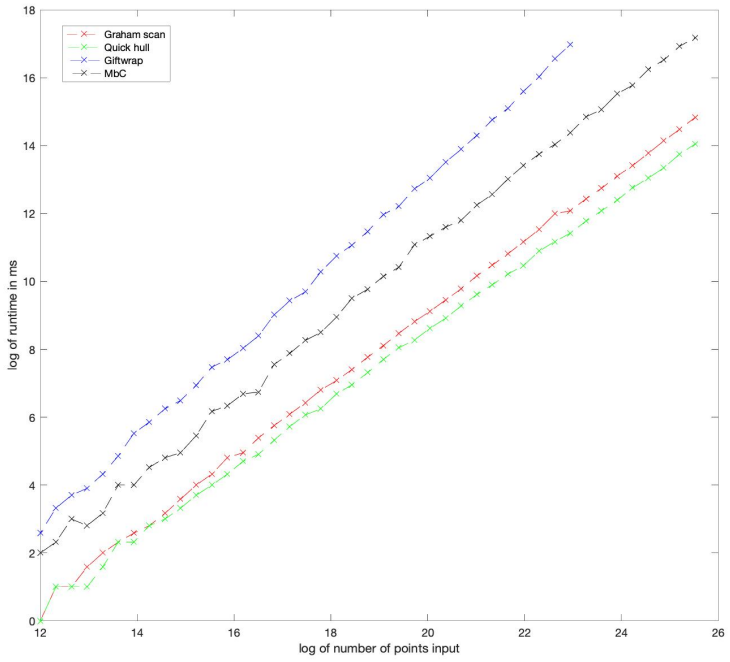


Figure 3: run time on circle input case

Quick hull is still the most fast, but Gift wrap and MsB get slower, this is because the number of points on the convex hull is quite bigger than square input case as shown in section 6.1.

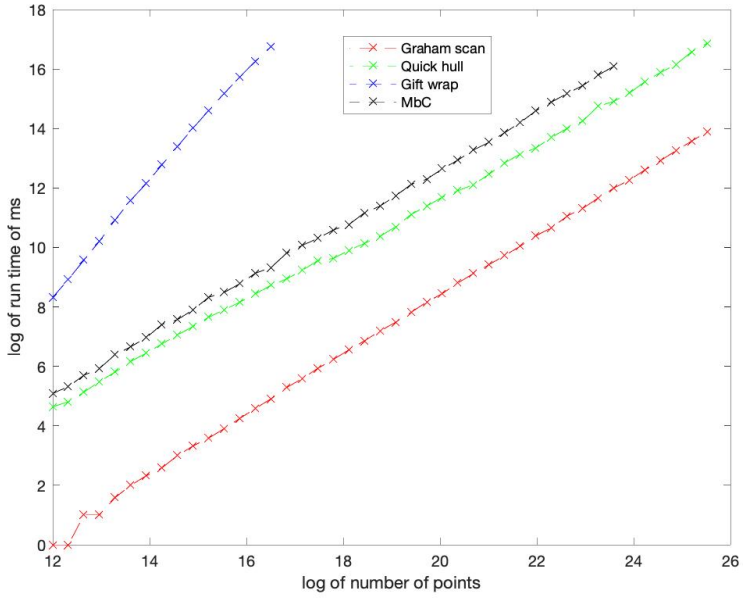


Figure 4: run time on quadratic curve input case

In this input case, even the Quick hull is far behind, because on the curve, quick hull can only remove 2

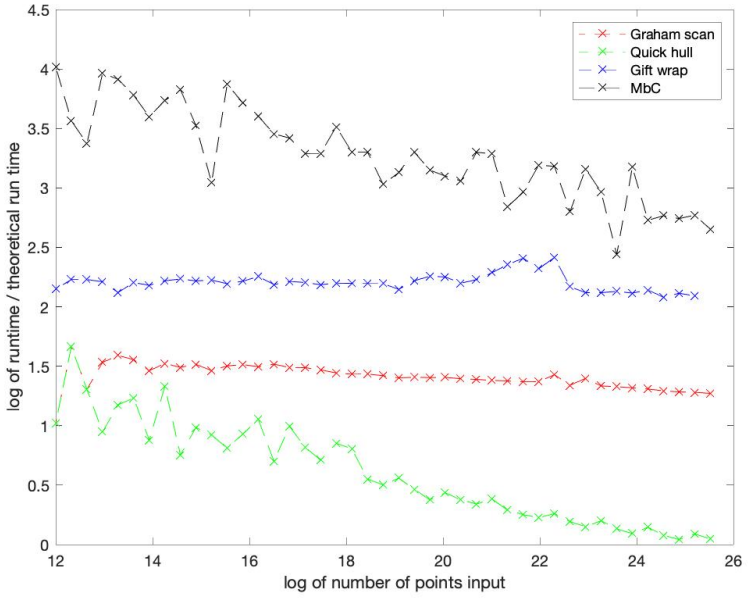
points at one recursion and leave all the points to compute.

6.3 Constant Analysis

At here, we define the constant of an algorithm with theoretical time complexity $O(f(n))$ to be Nice!

$$\epsilon \cdot runtime/f(n)$$

in which ϵ is some constant float to make the result not too small or big.
The $f(n)$ is $n \log n$, $n \log n$, $n * h$ and $n \log h$ for graham scan, quickhull, gift wrap and MbC in following figures.
Following that definition, we measured the constant of the 4 convex hull algorithm in different input case.



In this plot, we use n log n as f(n). But we should use f(n)=n to match the theoretical analysis.

What f(n) did you use for QuickHull?

Figure 5: constant on the square input case

We can see that the constant of Quick hull get smaller, this is because when the number of points get larger, the shape of convex hull get close to square, and the quick hull can always first to get the 4 corner of the square.

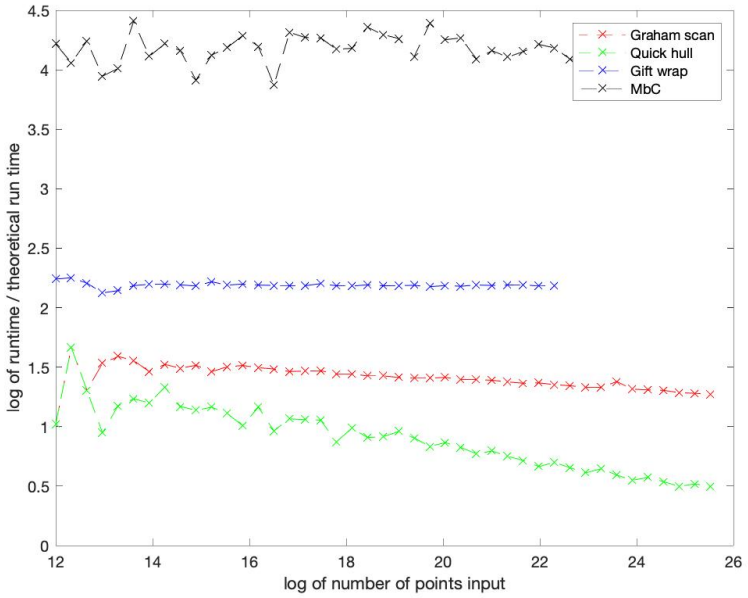
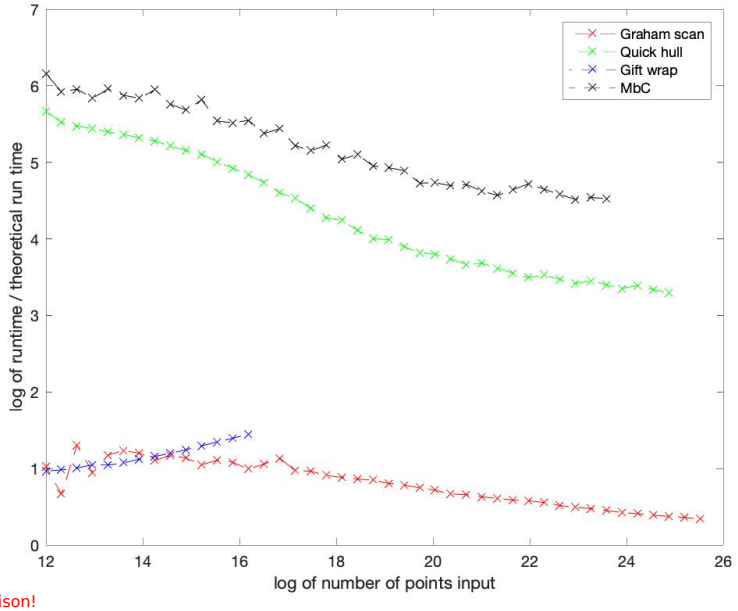


Figure 6: constant on the circle input case



These plots are more interesting than runtime comparison!

Figure 7: constant on the quadratic curve input case

7 Conclusion

Combine the theoretical analysis and the experimental performance, we can see that even the MbC only have $O(nlog(h))$ theoretically, it has a very large constant, which make it slow than Graham scan and Quick hull, at the same time, Gift wrapping is always slowest.
The quick hull has a very good performance on general square and circle, but get very slow on quadratic

curve, maybe we can do some improvement to Quick hull to avoid this?
In general, we can see that Graham scan is always fast and steady, which makes it a very widely used convex hull algorithm.