COMP0005 Group Coursework Academic Year 2023-24
Group Number: 001

1. Overview of Experimental Framework

This report presents a comparative analysis of three algorithms for finding the convex hull of a set of points: Jarvis March, Graham Scan and Chan's Algorithm.

1.1 Framework Design/Architecture

The experimental framework was designed to evaluate and compare algorithm performance on different types of inputs. The framework is accompanied by a comprehensive synthetic data generator that can produce a controlled **number of points (*n*)** and number of **convex hull points (*h*)**. The controlled synthetic data is produced using the following strategy (visualised in Figure 1):

1. If $h = 1$, it will produce a single (uniform) random point.
2. If $h = 2$, it will produce *n* random collinear points.
3. If $h = 3$, it will produce a triangle, and the rest of the points will be randomly & uniformly distributed inside the inscribed disk.
4. If $h > 3$, first it will produce a triangle, then the rest of the convex hull points will be distributed randomly & uniformly in the circle inscribing the triangle, and the rest of the points will be randomly & uniformly put inside the inscribed disk.

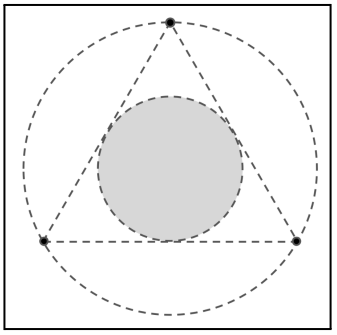

Figure 1. Illustration of the strategy at which controlled random points were generated.

The $h = 1$ case is trivial and we will not explicitly explore it, but the remaining cases are all considered later in our analysis. The data generator also has functionality to produce points that are randomly & uniformly distributed across a specified range, which we consider to measure the "average" case. Finally, all points produced are made so that their *x* and *y* are integers within the range [0, 32767].

We can have multiple instances of the experimental framework, where each instance represents an experiment designed to explore something specific. Each experiment that we run in the framework can take multiple different algorithms which we wish to compare. Then, we are able to run multiple settings on the same experiment (such as tweaking *n*) and the framework will collect and store all data (times taken and number of convex hull vertices *h*) from each run for each algorithm. We can also specify the number of trials per run the run should do so more statistics can be taken, such as mean, median, and minimum time taken for each run. Finally, the framework can plot multiple different graphs based on those statistics (such as box plot, line graph of medians, or a combination thereof as will be seen later).

1.2 Hardware/Software Setup for Experimentation

All experiments were run on an Apple M2 Macbook Pro running on MacOS Sonoma with 16 GB of RAM. Furthermore, the experiments were run on Python 3.11 within a Jupyter Notebook.

2. Performance Results

In this section, we will individually benchmark each algorithm on various scenarios as well as compare them together at the end. We define our **average case** experiment as taking multiple samples of *n* randomly & uniformly distributed points in a square plane. Next, we define the **worst case** experiment as setting $h = n$, which produces *n* points distributed randomly & uniformly on the circumference of a circle. Due to integer limitations caused by the finite range and limited computational resources, we were only able to produce worst case points up to $h{\sim}2200$.

2.1 Jarvis March

We will first explore Jarvis March, which has a theoretical time complexity of $O(nh)$. It is output sensitive as its time complexity is also relative to its output (h). Since it is $O(nh)$, it is expected to perform well when h is relatively small compared to n. Our implementation has $O(n)$ space complexity.

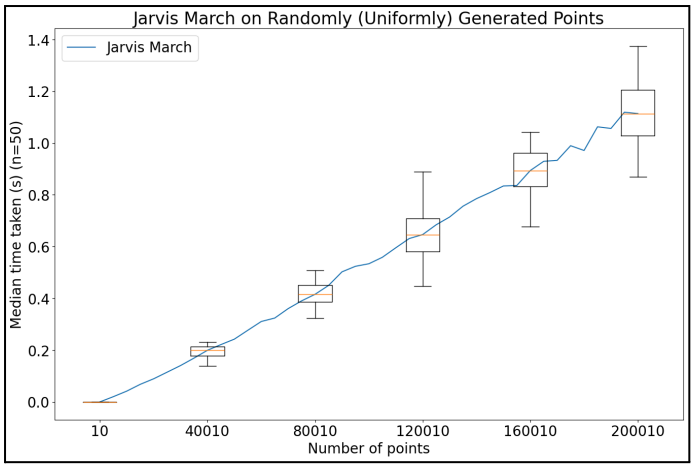

Figure 2. Boxplot and average case median times taken (s) vs n for Jarvis March as n increases.

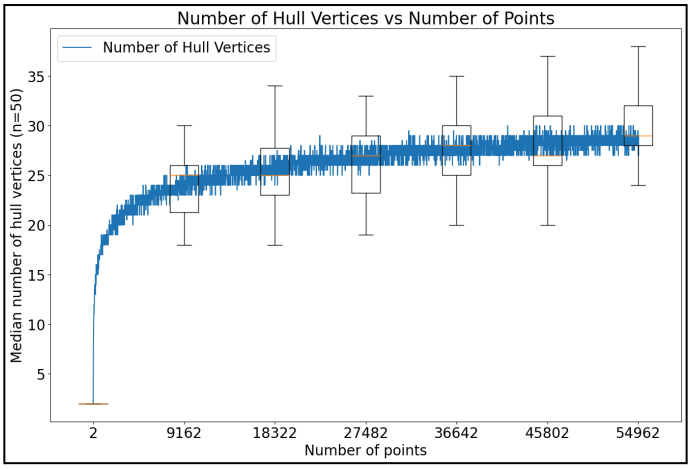

Figure 3. Plot of median h values vs n as n increases.

We initially investigate the average case (Figure 2). We repeat the average case over several n values, and for each n we run 50 trials and take their median to produce the plot on the right. If h grows within a constant factor of n, then we would expect the growth to be quadratic in nature. However, evidently the median time taken for Jarvis March resembles linear growth with respect to n. This suggests that h itself is not linear with respect to n, so we graphed the median h value as n increases (Figure 3).

Evidently, when we produce n randomly & uniformly distributed points on a square plane, h grows very slowly with respect to n. The growth is seemingly logarithmic with minimal change in variability, and by checking the existing literature, it seems that Renyi and Sulanke (1963)[1] found that the expected value of h in this case is indeed O(log n). This explains the nearly linear growth of Jarvis March in this case, as if $h = O(\log n)$, then Jarvis March's time complexity is rendered $O(n \log n)$ (linearithmic). It is worth noting that Jarvis March's variability in execution time clearly increases as the n increases, which can be attributed to system performance being increasingly inconsistent for larger ns.

Next we explore the worst case (Figure 4). This would theoretically give us a worst case time complexity of $O(n^2)$. Since there is no variability of h in the input set ($h = n$), and Jarvis will always run n times for each hull vertex, most variability in execution time will likely be due to inconsistent system performance. Thus, we choose to take the minimum time taken over 10 trials of the same input as our metric. This will help eliminate external factors such as system and cpu cycles not being free (i.e, gets close to the peak system performance).

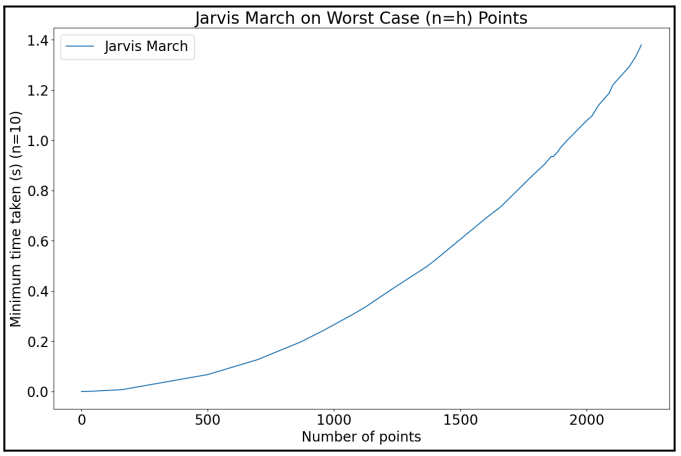

Figure 4. Line plot of the worst case minimum times taken (s) for Jarvis March vs n as n increases.

As evident from Figure 4, the results give us a graph with a quadratic order of growth, hence directly agreeing with the theoretical time initially discussed. Therefore, if h itself grows linearly with respect to n, Jarvis March is not a feasible algorithm in terms of its time complexity.

2.2 Graham Scan

We now move to Graham Scan, which has a theoretical time complexity of $O(n \log n)$. Unlike Jarvis, Graham Scan is not output sensitive, meaning its time complexity does not depend on its output. For our

[1] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gerwählten Punkten 1963.

experiments, this means that the average case should not differ from the worst case. Furthermore, since Graham Scan involves sorting the entire input, we will compare different sorting algorithms that have average time complexities of $O(n \log n)$ (Quicksort, Mergesort and Heapsort) as sorting is the bottleneck of the algorithm.

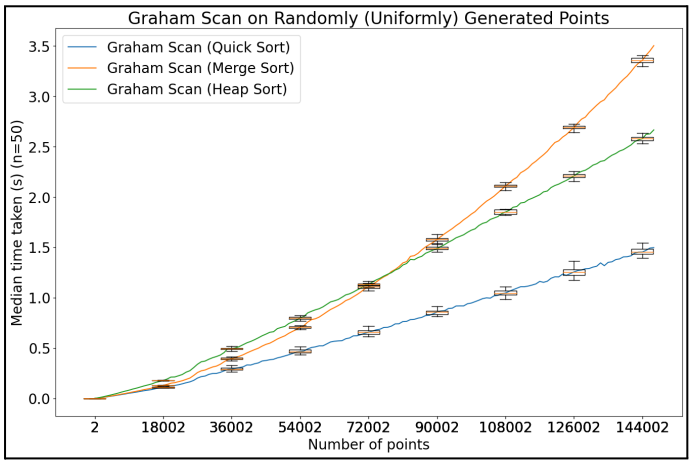

Figure 5. Boxplot and average case median times taken (s) vs n for Graham Scan as n increases with Quick Sort, Merge Sort and Heap Sort.

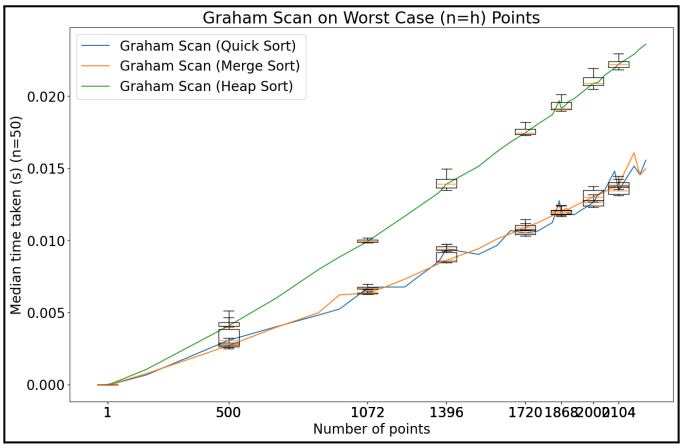

Figure 6. Line plot of the worst case median times taken (s) for Graham Scan vs n as n increases with Quick Sort, Merge Sort and Heap Sort.

We first try the average case for each sorting algorithm repeated 50 times for each n and plot the median times to get the graphs in Figure 5. All sorting algorithms resemble $O(n \log n)$ growth as expected, and all have a negligibly changing variability in times, as evident from the box plots, which is also as expected since the time complexity does not depend on h. Quicksort is clearly the best here and Mergesort becomes worse than Heapsort at around $n = 80$k. One potential reason why Mergesort performs the worst here is because it is the only one which is not in-place, requiring $O(n)$ space complexity, which becomes more significant as n gets larger. This gives Heapsort and Quicksort better cache locality (making them perform better in modern computers which make various optimizations based on how memory is laid out).

Now we explore the worst case. We repeat the worst case scenario ($h = n$) over 50 trials for each n and plot their medians up to 2.2k (Figure 6). Theoretically, the average case and worst case should be roughly the same since Graham Scan depends only on n. Up to 2.2k, the two graphs (Figure 5 and 6) look almost identical with Mergesort and Quicksort being nearly tied. If we were able to go further than 2.2k, we would likely expect Mergesort to become slower than Heapsort for similar reasons to the average case.

2.3 Chan's Algorithm

We now look at an algorithm which, similarly to Jarvis, is output sensitive. It utilises concepts from both Graham's scan and Jarvis march, with the end result giving us an algorithm with an $O(n \log(h))$ time complexity. Moreover, just like Graham Scan and Jarvis March, our Chan's Algorithm occupies $O(n)$ space complexity. We will use the Mergesort implementation of Graham Scan to implement Chan's algorithm, as Mergesort is known to always have a linearithmic worst time worst complexity regardless of how the input is shuffled, potentially giving us more stable execution times.

Firstly, we analyse the average case, where for every n we test, we repeat 25 times and take the median time to ensure consistency within our results (Figure 7). The results on Figure 7 agree with the theoretical time complexity that was discussed earlier, and by taking a brief observation of its rate of change, it seems to be very close to being linear. As mentioned previously with Jarvis March, in the average case h grows logarithmically as n increases ($h = O(\log n)$), hence, the theoretical time complexity of Chan in the average case is $O(n \log(\log n))$, which for our range of inputs makes the graph in Figure 7 seem practically linear.

Another thing worth noting is the box plots that were taken at regular intervals, displaying the spread in recorded times over the repeated trials. There is no clear trend in the variability of times taken, which can be supported by the minimal change in the value of h for large n as seen in Figure 3.

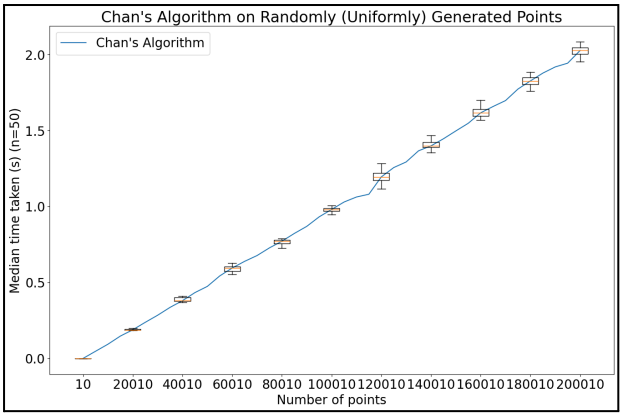

Figure 7. Boxplot and average case median times taken (s) vs n for Chan as n increases.

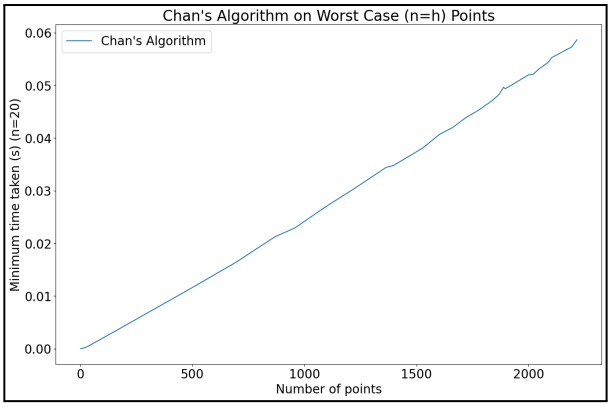

Figure 8. Line plot of the worst case minimum times taken (s) for Chan vs n as n increases.

Next, we perform the worst case identically to how we did with Jarvis March. In this case, taking the minimum value of 20 trials is justified because Mergesort (which we use in our Chan implementation) is invariable to how the input is shuffled (which can also be inferred from Figure 6), thus all variability can be attributed to inconsistent system performance, which we wish to eliminate by taking the minimum of the trials. When $n = h$ we would expect a time complexity of $O(n \log(n))$, and this graph in Figure 8 certainly validates the theory, due to its linearithmic shape. A closer inspection suggests that there is a minimally higher rate of change (gradient) in this case in comparison with the average case, which agrees with our conclusion that the average case is $O(n \log(\log n))$ while the worst case is $O(n \log n)$.

A key takeaway from this is that there is not a considerable difference in time complexity between the average and worst case scenarios for Chan's algorithm, clearly showing that despite its output sensitivity, it is a relatively stable algorithm with regards to its order of growth.

3. Comparative Assessment

We will now directly compare all the algorithms (Jarvis March, Graham Scan and Chan's algorithm) against each other. Note that the Graham Scan used within Chan's Algorithm is the same as the one we are comparing it to (specifically, the Mergesort version), which will ensure accurate results.

We start with the average case, where for each n, we repeat for 50 trials and get the medians. As evident from the graph in Figure 9, Graham and Jarvis start off with similar times taken, but for larger n values, Jarvis takes over. Furthermore, Chan starts slower than Graham but around $n = 70$k, Chan takes over. As discussed earlier, since $h = O(\log n)$, Graham here would be $O(n \log n)$, Jarvis would effectively be $O(n \log n)$ as well, but Chan would be $O(n \log(\log n))$. So although Jarvis is the clear winner up to 100k, we would expect Chan to take over for sufficiently large n (similar to how it took over Graham).

Next, we repeat the worst case scenario ($h = n$) over 20 trials for each n and plot their medians (Figure 10). It is obvious that Jarvis March is insufficient for n larger than around 1000 in the worst case, as it possesses $O(n^2)$ growth in the worst case. As for Graham Scan and Chan's Algorithm, they both have a theoretical time complexity of $O(n \log n)$ in the worst case, but Graham Scan is still the clear winner, which is likely due to the extra few iterations Chan's Algorithm needs to go through before getting to a partition size of n. From this, we conclude that Graham Scan is the clear winner when $h = n$.

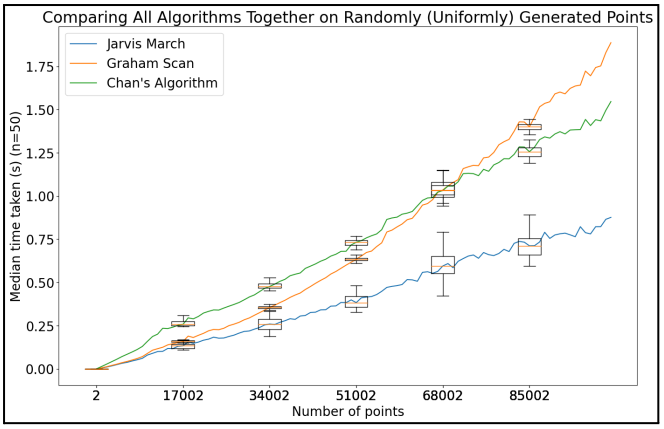

Figure 9. Boxplot and average case median times taken (s) vs n for Jarvis March, Graham Scan and Chan's Algorithm as n increases.

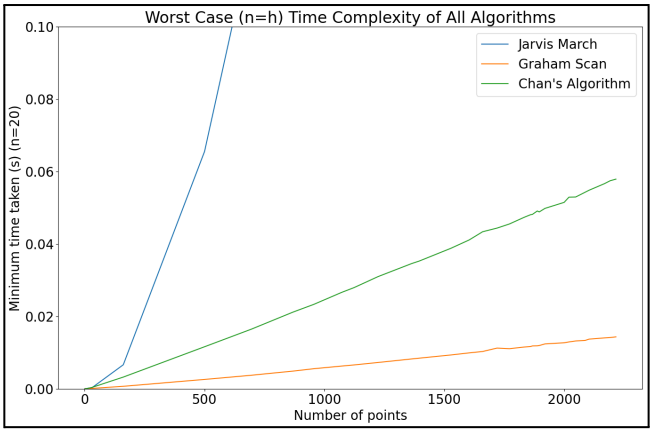

Figure 10. Line plot of the worst case minimum times taken (s) for Jarvis March, Graham Scan and Chan's Algorithm as n increases.

Another thing we explored is given an h value, at what n does Chan's algorithm overtake Graham Scan. To answer this, we plotted the ratios of the minimum times taken from 20 repeated trials (to help eliminate system performance inconsistencies), between Chan and Graham for each n. When the graph is above the $y = 1$ line, then Graham is performing better, and Chan is better when it is below $y = 1$. We repeated that over multiple h values, producing the following graph.

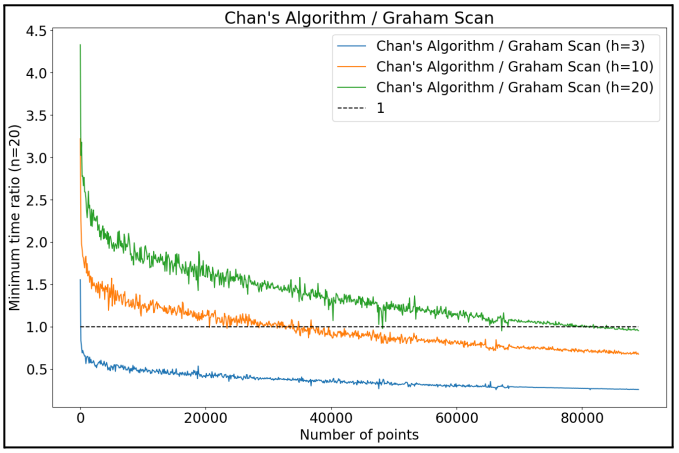

Figure 11. Line plot of the ratios of the minimum times taken (s) between Chan's Algorithm and Graham Scan for h values 3, 10 and 20.

As seen in Figure 11, for $h = 3$ Chan was almost immediately faster than Graham, but for $h = 10$ it required us to go up to n~30k, and for $h = 20$ we had to go to n~80k. The graphs in Figure 11 reveal to us that although it might take very large n values, Chan always beats Graham eventually given a fixed h.

Table 1. Algorithm Worst and Best Case Comparisons and Best Use Cases.

	Average Case ($h = O(\log n)$)	Worst Case	When Best Used?
Jarvis March	$O(nh)$ $= O(n \log n)$	$O(n^2)$	When the size of input is small and points are random (leading to smaller h).
Graham Scan	$O(n \log n)$	$O(n \log n)$	When h is known to be a large proportion of n, or we cannot risk h being large.
Chan's Algorithm	$O(n \log h)$ $= O(n \log(\log\ n))$	$O(n \log n)$	When points are random and input size is sufficiently large (leading to larger h).

** Note that average and worst case in Table 1 is used explicitly as defined at the start of Section 2.*