COMP0005 Group Coursework **Academic Year 2023-24**

Group Number: 32

1 Overview of Experimental Framework

1.1 Framework Design/Architecture

Our experimental framework leverages a TestDataGenerator class to ensure fair testing of algorithms by generating random test cases within specified x and y ranges [0, 32767]. The ExperimentalFramework class utilises the matplotlib library's pyplot module and the timeit module to plot graphs and measure algorithms' average execution times. To ensure fairness, tests are run nine times, with the median value selected to mitigate outliers.

Testing Scenarios

Average Case Scenario: To mimic realistic conditions, we assessed algorithms' performance across a wide spectrum of random points, varying the number of points (n) from 3 to 1,000,000 in 50,000-point increments.
Worst Case Scenario: Focusing on Jarvis March and Chan's Algorithm, crucial for applications like surveillance, we examined their performance under extreme conditions. We created test cases with points exclusively on a polygon's edge, with n equal to the convex hull's point count (h), incrementing n by 10 from 3 to 600.
Constant n, Variable h: This scenario explores the impact of varying h while keeping n constant at 10,000, focusing on the range where algorithm behaviour diverges, specifically for h values between 3 and 12.
Varying n/h Ratio: We examined each algorithm's performance trend by altering the ratio of n to h to understand different operational efficiencies.

Graphical Analysis

Our framework plots the average and worst-case runtimes of algorithms against the number of points (n) using specific plot functions for comprehensive comparison. For average-case runtime analysis, random points are generated within a 0 to 32766 range for both x and y values. In contrast, the worst-case analysis for Jarvis March and Chan's Algorithm involves generating points forming a circle, ensuring all points are on the convex hull, reflecting a scenario where n = h. The Graham Scan algorithm is tested under conditions aimed to maximise sorting time, as its runtime is independent of h. We can do this by maximising n.

By incrementally increasing the input set size, our tests, starting from a single point up to one million points, are designed to encapsulate larger runtimes for industry-scale comparison. This approach allows us to evaluate algorithmic performance thoroughly and develop strategies to mitigate vulnerabilities and optimise efficiency.

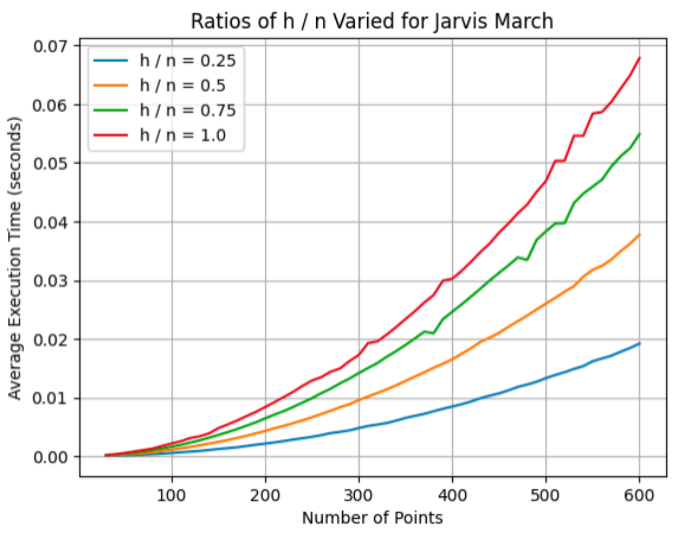
1.2 Hardware/Software Setup for Experimentation

We used a Windows laptop with AMD Ryzen 5 4600H with Radeon Graphics 3.00Ghz. With 8GB of RAM. The device runs on a 64-bit operating system. We edited the code using the VS Code IDE and ran it using Jupyter Notebook, running on Python 3.11.

2 Performance Results

2.1 Jarvis march algorithm

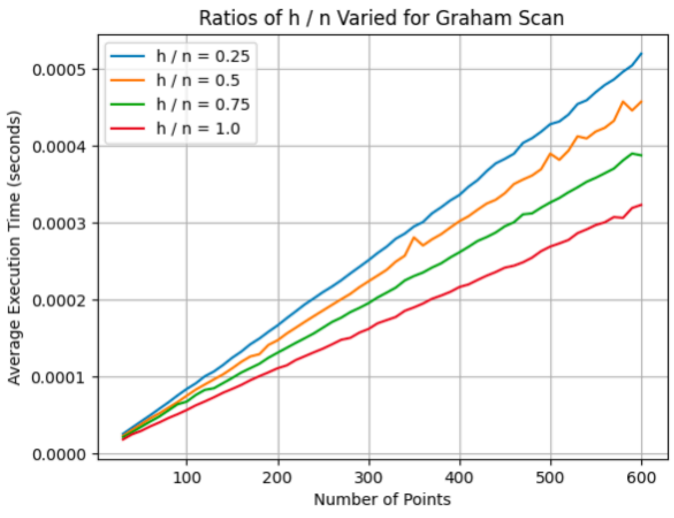
The algorithm commences by locating the leftmost point, ensuring inclusion in the hull due to its minimal x-value. Subsequent points are then iteratively identified by selecting the most counterclockwise point relative to the current endpoint, employing a cross-product to ascertain the orientation of the involved points. Points triggering a clockwise turn are candidates for the next point in the hull, while collinear points are considered based on their distance, opting for the furthest point. This process circulates through the entire set until it reconvenes at the start point, thereby completing the hull.



The runtime complexity of Jarvis March is O(nh), where n is the total number of points and h is the number on the hull, with optimal performance when either is small. Our analysis indicates a runtime increase as the h/n ratio escalates, decelerating the algorithm proportionally. The trend intensifies towards O(n^2) as h approaches n, characterising the worst-case scenario that occurs when all points form a convex polygon, necessitating that each point be evaluated.

2.2 Graham scan algorithm

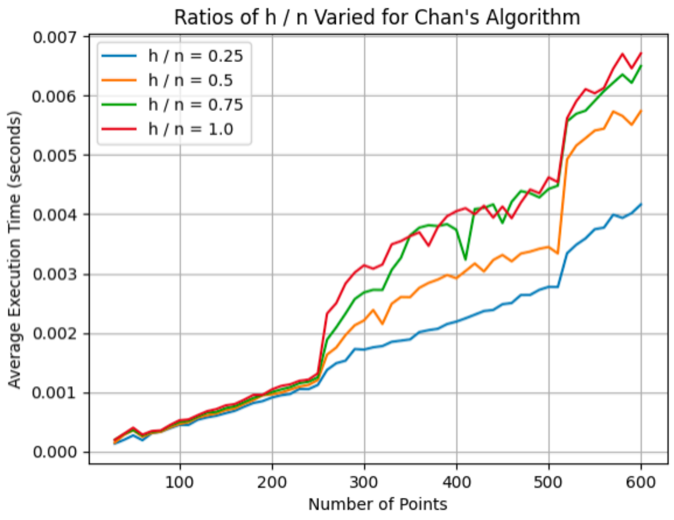
The Graham Scan algorithm identifies the convex hull by first locating the point with the lowest y-coordinate. After anchoring this starting point, the remaining points are sorted by polar angle. As it processes each point in this sorted order, the algorithm ensures a counter-clockwise turn is maintained, discarding points that would cause a clockwise deviation. This stepwise validation continues until a complete convex hull is traced out. The algorithm's efficiency largely hinges on the sorting method employed. Utilising Python's Timsort, the average-case complexity is O(nlogn), as it effectively handles varied input distributions.



Our empirical analysis of the Graham Scan algorithm reveals a correlation between the h/n ratio and computational speed: a higher ratio indicates fewer points within the polygon relative to those on the hull, thus reducing the iterations needed to sort and validate the turn criteria. Consequently, the runtime decreases as the algorithm benefits from a more streamlined set of points to process.

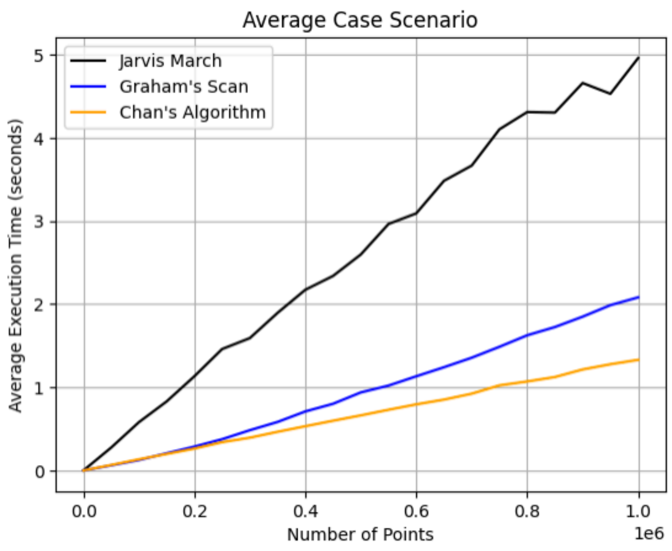
2.3 Chan's algorithm

The algorithm begins by partitioning the input point set. The input set should be divided into subsets of size m forming ⌈n/m⌉ subsets. Where m is an integer chosen where 3<=m<=n. Where n is the total number of points in the input set. Each subsets' convex hull can be computed in O(mlogm) using Graham Scan. Therefore, the process of computing the mini convex hulls is O(nlogm). Subsequently, a modified Jarvis March is done to find the all-encompassing convex hull of the point set. We take the minimum point from the convex hulls processed and find the rightmost tangent from this point to all the convex hulls. Select the steepest tangent and continue this "gift-wrapping" until we arrive at the initial point. The tangent calculation takes O(logm) time via binary search. Hence finding all tangents will take O((n/m)logm). By taking the m value that is equal to h, where h is the number of points on the convex hull, we will be able to attain an average time complexity of O(nlog(h)). Where the worst case occurs when h == n with a O(nlog(n)) runtime.

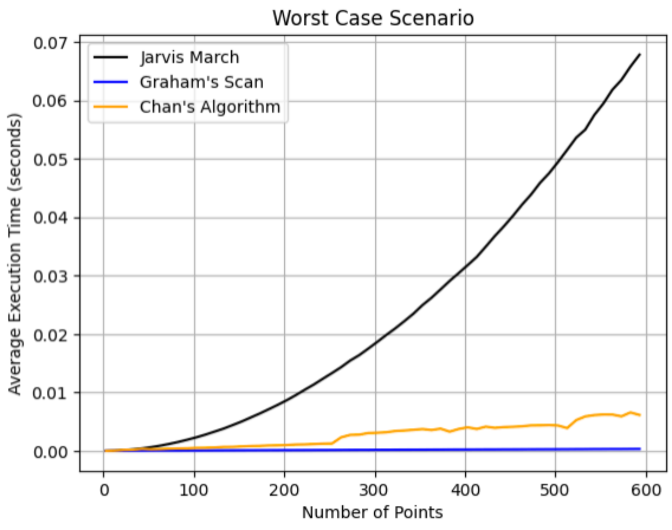


Our experimental results with Chan's algorithm indicate that a larger h/n ratio correlates with increased runtimes, approaching the worst-case time complexity O(nlogn). Notably, the runtime spikes align with certain n values, potentially due to increased subset counts affecting the main loop's iteration frequency. This can be attributed to how we calculate m by using a logarithm of the number of points.

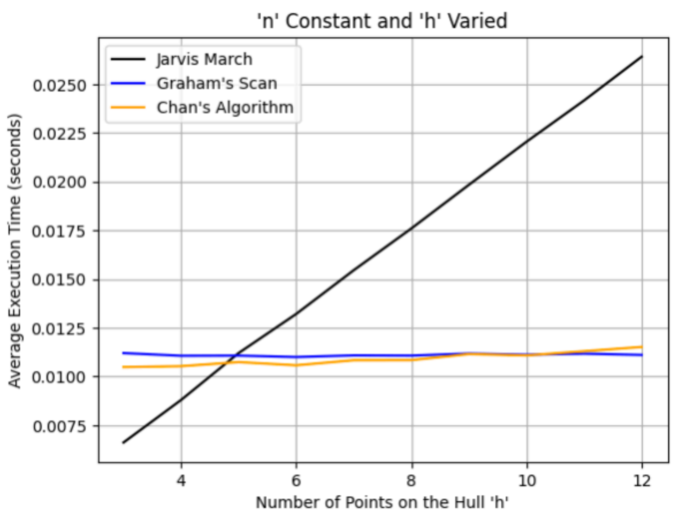
3 Comparative Assessment



In the average case scenario analysis, the performance trends for Jarvis March, Graham's Scan, and Chan's Algorithm are indicative of their computational complexities. The Jarvis March algorithm's execution time rises sharply with the number of points, reflecting its O(nh) complexity and suggesting a decline in efficiency for datasets with many points on the hull. Graham's Scan, on the other hand, shows a steadier, more gradual increase in execution time, which is symptomatic of its O(nlogn) time complexity due to the initial sorting step. As dataset size grows, its growth rate remains logarithmic, indicative of a more scalable algorithm for larger datasets. Chan's Algorithm, demonstrating the most stable and gentle slope among the three, supports its O(nlogh) theoretical complexity, and its consistent performance underscores its scalability for larger datasets. For small to medium datasets, the analysis suggests that either Graham's Scan or Chan's Algorithm remains efficient and practical choices.



The worst-case scenario graph reveals distinctive performance trajectories for Jarvis March, Graham’s Scan, and Chan's Algorithm when confronted with a convex hull comprising all input points. As expected, Jarvis March's runtime escalates sharply, underscoring its O(nh) complexity, which manifests most prominently when h approaches n. In stark contrast, Graham’s Scan demonstrates a stable and flat response; since every point influences the hull, the pre-sorting phase, generally the complexity determinant, does not adversely impact the runtime. In fact, this scenario represents an optimal case for Graham’s Scan. Chan's Algorithm presents an intermediary curve, suggesting that while it experiences performance degradation in the presence of many hull points—similar to Jarvis March—it remains more efficient due to strategic partitioning and merging steps that effectively navigate the computational challenges of a fully convex dataset.



In scenarios with a consistent number of points (n) and varying hull size (h), the graph illustrates distinct trends for Jarvis March, Graham's Scan, and Chan's Algorithm. When the hull size is small (h ranging from 3 to 5), Jarvis March outperforms the others, affirming its suitability for cases with fewer points on the hull. As h increases, particularly past the threshold of 10, the performance dynamics shift: Graham's Scan supersedes Chan's Algorithm, leveraging its O(nlogn) complexity which remains uninfluenced by the number of hull points. The transition observed in the graph for Chan's Algorithm—initially trailing behind Jarvis March but eventually overtaking Graham's Scan around the 120,000 points mark—highlights its adaptive nature. Chan's hybrid approach, initially slower, becomes more effective as the dataset grows, benefiting from its O(nlogh) complexity in larger contexts.

Summarising these insights, the graph suggests that for small datasets or those with minimal hull points, Jarvis March is advantageous. Graham's Scan emerges as a preferable choice for medium-sized datasets. In contrast, Chan's Algorithm is geared towards larger datasets, where it ultimately achieves the best performance.

4 Team Contributions

Student Name	Student Portico ID	Key Contributions	Share of work
Neethesh Neethesh	23170961	Implementation of Chan's Algorithm Report Writing	25 %
Amir Shamsuddin	23012535	Implementation of Graham Scan Algorithm Report Writing	25 %
Jash Shah	23163657	Implementation of Experimental Framework and Testing Report Writing	25 %
Meteor Pu	23028587	Implementation of Jarvis March Algorithm Report Writing	25 %