COMP0005 Group Coursework
Academic Year 2023-24

Group Number: 11

1. Overview of Experimental Framework

1.1. Framework Design/Architecture

The class TestDataGenerator is designed to generate a valid set of test data. It has two public attributes n and h, which are to be provided for instantiation. It contains one public method, which is generate().

First of all, input checks are applied.

If h < n or h = 1, then calling the generate() function would result in popping an error message. Note that case h = 1 is not being discussed.

If h = 2, then n points on the same line are generated. Several checks are applied to ensure the points have integer coordinates between [0, 32767].

If h >= 3, this is the general case that the method deals with. First, we generate a random convex polygon with h vertices based on *Valtr's algorithm*[1]. Next, we put n-h points into the generated convex polygon. This is done by dividing the h-polygon into h-2 triangles, and the n-h points are distributed to these triangles randomly[2], ensuring an even distribution of points in the polygon.

Each time the method generate() works, it returns an array of object Point(x, y) representing the generated data.

The class ExperimentalFramework is implemented to test the performance of an algorithm given n and h. It has three public attributes alg, n and h. The alg attribute is of type function, and n and h integer. It contains one public method time() which generates a random set of data with n points and h convex hull vertices, and returns the application time of alg on that data set.

Several pieces of code are written to plot the relation between n/h and time later on.

1.2. Hardware/Software Setup for Experimentation

All the test results are obtained on a computer with Apple M2 chip and 16 GB of RAM.

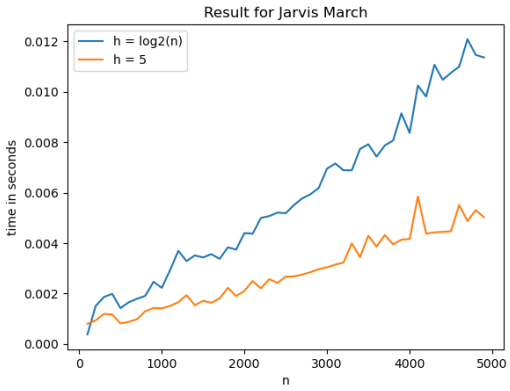
All the tests are carried out on the local Jupyter Notebook host, with Python kernel version 3.11.5.

2. Performance Results

The h=n case for Jarvis March and Chan are excluded from the graph due to too much discrepancy between the h= $log_2 n$ and h=5 cases. The figures for Jarvis March and Chan climb up to 1.6 and 0.7 seconds at n=5000.

2.1. Jarvis march algorithm

Average Case: The average case time complexity depends on various factors such as the distribution of points, the dimensionality of the problem, and the specific implementation details. However, on average, the Jarvis march algorithm tends to perform better than its worst-case complexity but worse than its best-case complexity. In many practical scenarios, the average case complexity is closer to O(nh), where h is the number of points on the convex hull and can vary based on the problem instance.

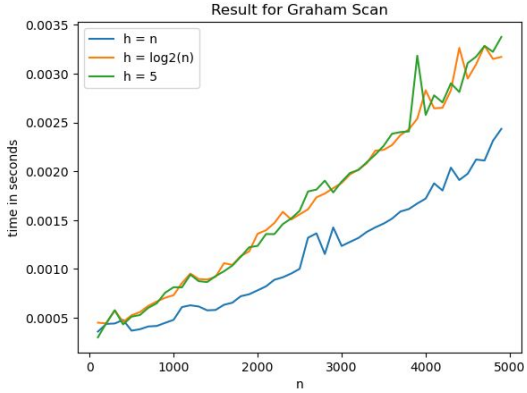


Case: h = $log_2 n$:
Here, the size of the convex hull grows logarithmically with the number of points in the set. The algorithm's time complexity is O($n \log h$), but if h= $log_2 n$:, then the time complexity is O($n \log\log h$). As n increases, the algorithm's performance improves logarithmically, but it's not as efficient as when h is a constant.

Case: h = 5:
In this scenario, the size of the convex hull is constant, regardless of the number of points in the set. The time complexity is linear with respect to the number of points, as the algorithm only needs to iterate through the points once. As n increases from 0 to 5000, the time complexity remains linear, making it more efficient than the other scenarios. This scenario represents an ideal situation for Jarvis March algorithm.

Worst Case: The worst-case scenario happens when all points lie on the convex hull, forming a complete loop. The algorithm needs to iterate over all the points in the convex hull for each point. For example, consider a scenario where all points lie on the convex u, such as in a case where the points are sorted in increasing order of x-coordinate. Therefore, the time complexity becomes O(n^2).

2.2. Graham scan algorithm

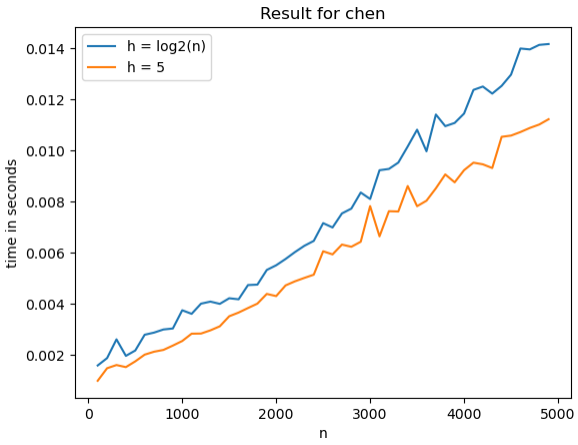


For all three scenarios, the O(n log n) complexity of the sorting phase seems to be the dominating factor. This explains why all lines increase and have a similar shape.

The blue line is the least steep. This seems

unexpected, but looking at the y-axis, there isn't much difference between the three lines. This further demonstrates the fact that h is not a crucial factor of the algorithm's efficiency.

2.3. Chan's algorithm



The Graham scan function is called for each subset of size m in the Hull2D function. The time complexity of Graham's scan is O (m log m) for each search, executing n/m times, which simplifies to O(n log m).

The linear search, as described by the author [3], has the same overall time complexity as the binary search, and the procedure is performed as a linear search of the whole candidate list for possible points. The search will be performed to take O(n); this gives a total of O (n log m + nh) per call to hull2D. For m=H, the overall thing has the time complexity of O (n log h+nh), which can be simplified to O(n log h), which is also for the general case. The binary search will perform a time complexity of O (n log h + n log h), which will perform better at high h while linear would perform better with low h values.

The iterations in the Chan's loop are log (log(h)). As the t increments by one for each iteration, giving O(n*2**t) time for each iteration. A formula described below shows the overall complexity being O(n log h) in best cases, and average cases. Situation may be different with worst case n=h with nh potentially more significant.

$$O\left(\sum_{t=1}^{\log\log h} n\,2^{t}\right) \Rightarrow O\left(n\,2^{\log\log h}\right) \Rightarrow O(n \log h)$$

(the most significant. is pick)

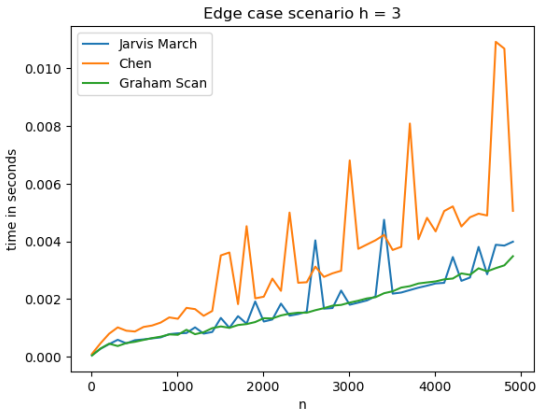
However, the algorithm does not show an advantage compared to the Graham scan with n log n time complexity. This is because a trial-and-error procedure is done for the vertex of the final hull.

3. Comparative Assessment

1. **Specific circumstances where the Jarvis march algorithm might be preferable over Graham scan and Chan's algorithm:**

Small Datasets with Simple Convex Hulls: When dealing with small datasets where the number of points is relatively low (e.g., less than a few hundred points) and the convex hull is

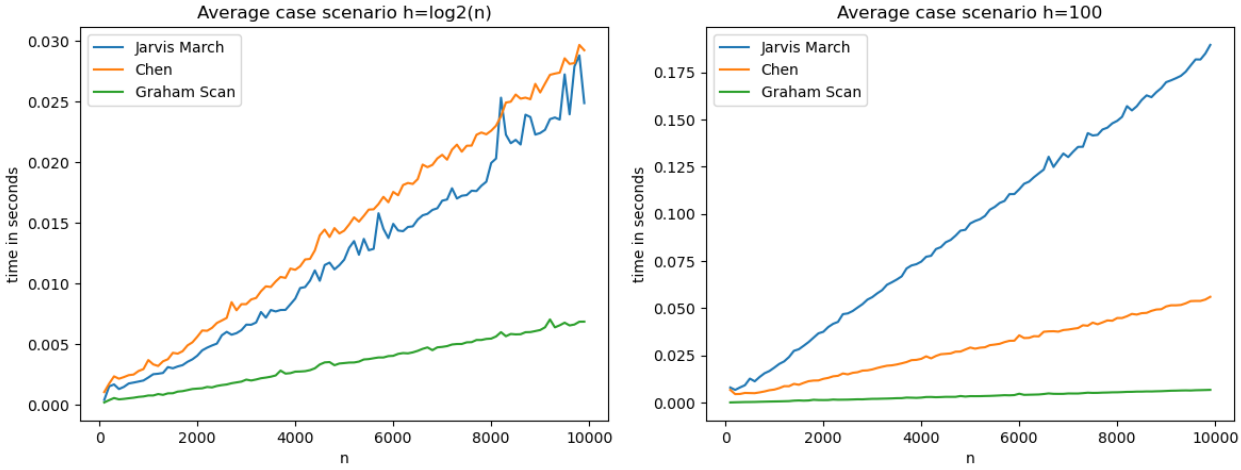
relatively simple, the simplicity of the Jarvis march algorithm can be advantageous. In such cases, the overhead of implementing more complex algorithms like Graham scan or Chan's algorithm might not be justified.



Special Cases or Degenerate Point Sets: In scenarios where the point set exhibits special characteristics or degeneracies, such as a large number of collinear points or a highly regular distribution, the simplicity of Jarvis march might suffice. While its worst-case time complexity is high, in specific cases, it can still perform reasonably well.

Overall, while the worst-case scenario of $O(n^2)$ can be concerning for large datasets with specific configurations, in practice, the algorithm often performs reasonably well, especially for well-distributed point sets. In practical scenarios, the Jarvis March algorithm is efficient for moderate-sized point sets and when the convex hull is expected to have relatively fewer points. It can be less efficient for dense point sets or when the convex hull contains a significant portion of the points.

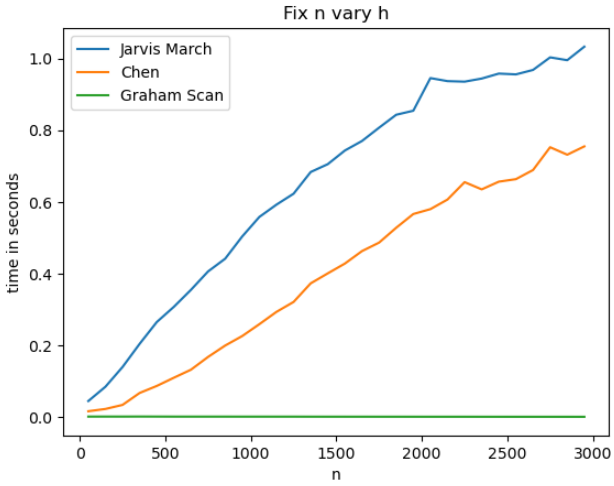
2. Graham scan being the fastest algorithm



Although we'd expect Chan's algorithm to be faster than the graham scan algorithm, performance results indicate that the graham scan algorithm is the fastest of the three. This can be attributed to the use of python's 'sorted()' function, which provides a significant performance advantage due to the highly optimised nature of the 'Timsort' algorithm that the 'sorted()' function uses. 'Timsort' is a hybrid algorithm which combines the best of merge sort and insertion sort, using natural runs in the data to optimise sorting. It uses a stack to manage runs and uses a merging strategy that considers the size of the top runs, merging

them under certain conditions to maintain efficiency.[4] From the graphs above, we can see that Graham scan is almost a straight horizontal line, while chan's show advantage to Jarvis with fixed h since less trial and error is performed.

3. Circumstances for Chan algorithm being efficient



From the graph, we can see that the Chan algorithm is very efficient, while the h values are known for each complex hull: the time of execution is almost a flat line. This is caused by a low amount of O(n log h) hull2d operation executed. However, while an h=n scenario is performed, is where the worst case is prepared, where the loop is executed from log(log(h)) times all the way up to the result, which the linear search complexity being more noticeable, similar to Jarvis March and with the additional "idle runtime", making it the one with extremely low efficiency.

Compared to the other two algorithms, Chan's algorithm has a middle efficiency at most times, and it can perform similar efficiency when comparing to Graham scan if H is known. If a trial and error is used, it can still perform faster than the Jarvis March except for some extreme cases.

4. Team Contributions

Student Name	Student Portico ID	Key Contributions	Share of work
Mike Wu	23031567	Coding and analysing of Jarvis march algorithm (part 2 and 3)	25 %
Haocheng XU	22076533	Coding and analysing of Chan algorithm (part 2 and 3)	25 %
Sai Tenneti	23005737	Coding and analysing of Graham scan algorithm (part 2 and 3)	25 %
Mark Wu	23008102	Experimental framework and comparative assessment (part 1 and 3)	25 %

References:

[1] https://refubium.fu-berlin.de/handle/fub188/17874

[2] https://blogs.sas.com/content/iml/2020/10/19/random-points-in-triangle.html

[3] T. M. Chan, "Optimal output-sensitive convex hull algorithms in two and three dimensions," *Discrete and Computational Geometry*, vol. 16, no. 4, pp. 361–368, Apr. 1996, doi: 10.1007/bf02712873.

[4] *(PDF) Merge Strategies: From merge sort to Timsort*. Available at: https://www.researchgate.net/publication/282679394_Merge_Strategies_from_Merge_Sort_to_TimSort (Accessed: 03 March 2024)