Analysis of Jarvis March and Graham's Scan algorithms for computing Convex Hull points

By Carlvince Tan

Introduction

In this report, the algorithms Jarvis March and Graham's Scan were implemented in the C language under the assumption that there are no existing points in the convex hull that are collinear for simplicity.

Jarvis March, utilised a binary heap to find the leftmost point of all input points and a simple brute-force algorithm which compared two other points with a known point on the convex hull starting from the leftmost point. The binary heap, used to find the leftmost point of all input points, has a time complexity of O(n). The brute-force algorithm has an average time complexity of O(nh), where "n" is the total number of the input points and "h" is the total number of points existing on the convex hull, the time complexity for the worst scenario is $O(n^2)$. So, the algorithm overall has an average time complexity of $O(n^2h)$ and a time complexity of $O(n^3)$ for the worst scenario. The points were stored in a doubly-linked list since the solution required printing of the convex hull points in both clockwise and anti-clockwise directions. The ability to traverse the doubly linked list from the head and tail enabled great efficiency for printing purposes.

Graham's Scan, is an algorithm that first sorts all input points in respect to the lowest point of all inputs by polar angle and compares two other points to determine whether the point that was part of a previous comparison should be part of the convex hull. Graham's scan also utilizes a binary heap to find the lowest point of all input points, which has a time complexity of O(n). The heapsort algorithm uses a priority queue with an average and worst time complexity of O(log(n)). Optimised through the sorting, Graham's Scan can achieve an average and worst time complexity of O(nlog(n)) through the implementation of Priority Queue (Heapsort) with a binary heap. The algorithm also utilizes a simple stack for an efficient temporary storage of the convex hull points and removal when previous stored points as temporary convex hull points can be covered by a new point. After the stack is left with only convex hull points, they are transferred to a doubly-linked list for the same reasons as discussed for the Jarvis March algorithm.

Since the time complexity of the binary heap for both algorithms are the same, the basic operation of the Jarvis March algorithm is taken to be the brute force method that compares the angles of two points with a known point in the convex hull as the pivot and the basic operation of the Graham's Scan algorithm is taken to be the heap sort algorithm. These basic operations were chosen as they are considered to be the asymptomatic parts of their respective algorithms. Thus, the focus of the analysis will be on these sections of the algorithms where "comparisons" of points occur in determining whether the points are part of the convex hull.

Method

1. Write the code for Jarvis March and Graham’s Scan in C language
2. Integrate a counter named as the variable: “OperationCount” to keep track of the operations executed
3. Create three samples for the 3 input sets by Python in sizes of n = 10, 1,000, 100,000, where n is the total number of input points using simplification that a circle to be used in “Test 2” has a centre at (0, 0) a radius of 30 units and a simple triangle convex hull to be used in “Test 3” has vertices at (0, 7), (5,0), (6,0).
4. For “Test 1”, create a convex hull from a generated input set consisting of random points as “Test 1”
5. For “Test 2”, create a convex hull from a generated input set consisting of random points that lie on the circle
6. For “Test 3”, create a convex hull from a generated input set of random points inside a set of points that make the simple convex hull

Results

	Jarvis March	
Test 1	**OperationsCount**	**Number of Convex Hull points**
n = 10	50	5
n = 1,000	25,000	25
n = 100,000	2,900,000	29
Test 2		
n = 10	100	10
n = 1,000	1,000,000	1,000
n = 100,000	TL;DR	100,000
Test 3		
n = 10	30	3
n = 1,000	3,000	3
n = 100,000	300,000	3
	Graham's Scan	
Test 1	**OperationsCount**	**Number of Convex Hull points**
n = 10	20	5
n = 1,000	8,347	25
n = 100,000	1,500,587	29
Test 2		
n = 10	19	10
n = 1,000	8,342	1,000
n = 100,000	1,501,036	100,000
Test 3		
n = 10	17	3
n = 1,000	8,321	3
n = 100,000	1,500,853	3

Table 1. Results from 3 series of tests under 3 distinct input sizes.

Note. The test code written in Python and input .txt files can be found in the PartC_test_cases folder

Discussion

The results obtained from the 3 series of tests under 3 distinct input sizes (n = 10, 1,000, 100,000) were consistent with the expected runtimes of the algorithm and observed that the runtimes to be heavily dependent on the input sizes. The testing proved the O(nh) time complexity of Jarvis March and the O(log(n)) time complexity of Graham's Scan, where n were the total number of input points and h was the number of points on the convex hull as matched with the values in *Table 1* from Results.

One of the crucial observations from the different styles of tests lies in Test 3, which differed in results compared to Test 1 and Test 2, where the runtime (operationCount) of Graham's Scan ran in fact at 1,500,853 operation counts, slower than the 300,000 operation counts for Jarvis March. From analysis of the nature of the time-complexities of both algorithms, Test 3 differs in the fact that the number of convex hull points remained small regardless of input sizes. For Jarvis March, it performed optimally as it allowed it to run at O(3n), which is around linear time-complexity. On the other hand, Graham's Scan still ran its usual O(log(n)) time complexity as its run time isn't influenced by the number of existing convex hull points. Since O(3n) < O(log(n)), we can understand that the number of convex hull points places a huge influence on the runtime of Jarvis March, and thus is the better choice out of the two algorithms for this specific scenario where the number of convex hull points is extremely small. Another observation is that the descriptor "extremely small" is determined by the proportion of convex hull against the total number of input points. For example, despite Test 1, for n = 10 only had 5 convex hull points, it isn't considered "extremely small" compared to the 10 input points. Thus it didn't perform as optimal as Graham's scan.

In Test 1 and Test 2, when the number of convex hull points is also larger when the input size is larger, Graham's Scan showed better runtimes consistently for example (20 < 50, 8,347 < 25,000 and 1,500,587 < 2,900,000) from Test 1, comparing Graham's Scan to Jarvis March respectively. In test 2, when the number of convex hull points was a large number, the OperationCount scaled extremely quickly with its time-complexity at O(n^2), to the point that a result wasn't able to be obtained within a reasonable amount of time.

Conclusion

From the results, the expected rates of growth for Jarvis March and Graham's scan were obtained and analysed, resulting in the conclusion that when the number of points on the convex hull are known to be significantly less than the number of input points, Jarvis March can run at close to a linear time-complexity, faster than Graham's Scan. Otherwise, for any other cases, the Graham's Scan algorithm will work faster than Jarvis March, especially for large datasets when the number of points on the convex hull will be expected to be greater.