COMP20007 Assignment 1
Task 1, Part C: Report

Name: Quoc Khang Do

Student ID: 1375531

NOTE: All names of files, folders, functions, etc. in this report are italicized, and can be found the assignment's "Task 1" directory.

Choice of data structures and algorithms

The Jarvis' March algorithm in part A was implemented using a doubly linked list to store the coordinates of the points in the convex hull. This was convenient because the cost of inserting an element at the head or tail of the list takes $O(1)$ time. Moreover, a doubly linked list is preferred over a singly linked list since the list can be traversed bidirectionally, e.g., the function *traverseRingBackwards*() in the *linkedList* module requires for the list to be printed backwards.

The Graham's Scan algorithm in part B was implemented using a stack (whose underlying data structure is a one-dimensional array) to store the indexes of the points – which is cheaper then storing the coordinates themselves. Using a stack helps to quickly pop points form the stack that cause a non-counterclockwise turn – in $O(1)$ time. Also, by using a one-dimensional array as the underlying data structure, this made it more efficient to implement the sorting algorithm (explained more below). After the scan is complete, the array of indexes gets converted into a doubly linked list, which stores the coordinates of the convex hull points (like in part A). This was done to maintain consistency with the *solution* structure in *convexHull.h*.

The sorting algorithm used in Graham's Scan was Mergesort, whose time complexity is $O(nlogn)$ – as required by the assignment specifications. Mergesort relies heavily on accessing elements, so by using a one-dimensional array (as stated above), the elements can be directly accessed in $O(1)$ time just through the index. However, using a linked list would take worst case $O(n)$ time due to having to traverse the list.

Experimental evaluation

Generating input sets

The two convex hull algorithms were each tested with 9 different types of inputs, which were stated in the assignment specifications. The text files for these input sets were generated using a Python script (called *inputGenerator.py*), and they can be found in the *my_input_sets* folder.

	Condition1: random points	**Condition 2: points on a circle**	**Condition 3: random points contained within a set of points making a simple hull**
Small (n=10)	*input1.txt*	*inpu4.txt*	*input7.txt*
Medium (n=100)	*input2.txt*	*input5.txt*	*input8.txt*
Large (n=10000)	*input3.txt*	*input6.txt*	*input9.txt*

Table 1: file names for each input type – can be found in the *my_input_sets* folder.

The three input conditions were chosen to be:

- **Condition 1: random points** were generated in in the range: $0 \leq x \leq 1000$ and $0 \leq y \leq 1000$
- **Condition 2: points on a circle** were generated on a circle of radius = 500 units, and center at (500, 500)
- **Condition 3: random points contained within a set of points making a simple hull** were generated by first making a square convex hull of 4 points (0, 0), (1000, 0), (1000, 1000), and (0, 1000). Then generating the remaining points within, but not on, the square hull.

For each of these conditions, they are to be tested for three distinct input sizes n = 10 (small), n = 100 (medium), and n = 10000 (large).

Comparing algorithms

Given an input, the two programs *problem1c_a.c* and *problem1c_b.c* were made to count the total number of basic operations of the Jarvis' March algorithm and the Graham's scan algorithm respectively. These were the results after running both programs for the 9 input sets.

	Number of points on convex hull (h)	**For condition 1, the number of basic operations in…**	
		Jarvis' March algorithm	**Graham's Scan algorithm**
input1 (n=10)	6	60	21
input2 (n=100)	12	1,200	547
input3 (n=10000)	22	220,000	120,498

Table 2: number of basic operations in each algorithm for condition 1

	Number of points on convex hull (h)	**For condition 2, the number of basic operations in…**	
		Jarvis' March algorithm	**Graham's Scan algorithm**
input4 (n=10)	10	100	20
input5 (n=100)	100	10,000	381
input6 (n=10000)	10000	100,000,000	71,508

Table 3: number of basic operations in each algorithm for condition 2

	Number of points on convex hull (h)	**For condition 3, the number of basic operations in…**	
		Jarvis' March algorithm	**Graham's Scan algorithm**
input4 (n=10)	4	40	25
input5 (n=100)	4	400	531
input6 (n=10000)	4	40,000	120,417

Table 4: number of basic operations in each algorithm for condition 3

Let $J(n)$ represent the order of growth for Jarvis' March algorithm, and $G(n)$ for Graham's Scan algorithm. The time complexity of each algorithm can be defined based on their total number of basic operations. For Jarvis' March, this is the comparison between the angles of points, so $J(n) \in O(nh)$, where n is the input size and h is the number of points on the convex hull. For Graham's Scan, a basic operation is the comparison between angles of points during the Mergesort, so $G(n) \in O(nlogn)$.

For condition 1 (*input1, input2, input3*), where points were selected at random, Graham's Scan performed better than Jarvis' March, with a lower number of total basic operations across all three input sizes (as seen in Table 2). This means, at random, $J(n)$ dominates $G(n)$.

For condition 2 (*input4, input5, input6*), where points lay on a circle, Jarvis' March performed significantly worse than Graham's scan, with a substantially higher number of total basic operations across all the input sizes (as seen in Table 3). This was expected because when all points lie on a circle, this becomes the worst-case scenario for the Jarvis' March algorithm – where the number of points on the hull is now equal to the total number of input points ($h = n$), so in this case, $J(n) \in O(n \cdot n) = O(n^2)$. And since $O(nlogn) \prec O(n^2)$, therefore $J(n)$ dominates $G(n)$.

For condition 3 (*input7, input8, input9*), where points were selected randomly within, but not on, a set of points making up a simple hull, Graham's Scan initially outperformed Jarvis' March for a small input size of n = 10 (25 basic operations vs. 40 basic operations in Table 4). However, as n became larger, the total number of basic operations in Jarvis' March became lower than that of Graham's Scan (as seen in Table 4). This was also expected. Since we were generating random points within, but not on, the simple hull, h stays constant regardless of the input size, so we can say $J(n) \in O(cn)$, where c is a constant, so essentially, $J(n) \in O(n)$. And since $O(n) \prec O(nlogn)$, therefore $G(n)$ dominates $J(n)$ asymptotically.

Conclusion

In general, it is preferable to use Graham's Scan over Jarvis' March algorithm because it yields, on average, a lower cost/number of basic operations. However, for input that has points contained within a set of a simple hull, we should consider using Jarvis' March, as it outperforms Graham's scan in the long run.

Assumptions/simplifications

1. The code does not handle the case where three points are collinear.
2. The code assumes the points are unique, and do not overlap.
3. It is assumed that the time complexity of the algorithm is based on the total number of basic operations. Although it may be simpler to implement, measuring it this way does not consider other factors that might affect the performance and real run time of these algorithms.