Project 1 Documentation – Convex Hull (Application of Stack Data Structure & Sorting Algorithms)
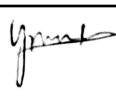
DECLARATION OF INTELLECTUAL HONESTY / ORIGINAL WORK

We declare that the project that we are submitting is the product of our own work. No part of our work was copied from any source, and that no part was shared with another person outside of our group. We also declare that each member cooperated and contributed to the project as indicated in the table below.

Section	Names and Signatures	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
<S13>	Panlilio, Yuan Miguel	X	X	X	X	X	X
<S13>	So, Nigel Henry	X	X	X	X	X	X

Fill-up the table above. For the tasks, put an 'X' or check mark if you have performed the specified task (see MCO1 specs for the detailed task descriptions). Don't forget to affix your e-signature after your first name.

1. FILE SUBMISSION CHECKLIST: put a check mark as specified in the 3rd column of the table below. Please make sure that you use the same file names and that you encoded the appropriate file contents. For the .h and .c source files: make sure to include the names of the persons who created the codes.

FILE	DESCRIPTION	Put a check mark ✔ below to indicate that you submitted a required file
`stack.h`	stack data structure header file	✔
`stack.c`	stack data structure C source file	✔
`sort.h`	"slow" and "fast" sorting algo header file	✔
`sort.c`	"slow" and "fast" sorting algo C source file	✔
`graham_scan1.c`	Graham's Scan algorithm slow version (using the "slow" sorting algorithm)	✔
`graham_scan2.c`	Graham's Scan algorithm fast version (using the "fast" sorting algorithm)	✔
`main1.c`	main module for the "slow" version	✔
`main2.c`	main module for the "fast" version	✔
`INPUT1.TXT` to `INPUT10.TXT`	10 sample input files (with increasing values of *n*)	✔
`OUTPUT1.TXT` to `OUTPUT10.TXT`	10 sample corresponding output files	✔
`GROUPNUMBER.PDF`	The PDF file of this document	✔

2. Indicate how to compile your source files, and how to RUN your exe files from the COMMAND LINE. Examples are shown below highlighted in yellow. Replace them accordingly. Make sure that all your group members test what you typed below because I will follow them verbatim (copy/paste as is). I will initially test your solution using a sample input text file that you submitted. Thereafter, I will run it again using my own test data:

- How to compile from the command line
  C:\MCO> gcc main1.c stack.c sort.c helperFunc.c -o one.exe

C:\MCO> <mark>gcc main2.c stack.c sort.c helperFunc.c -o two.exe</mark>

- How to run from command line
  C:\MCO><mark>one</mark>
  C:\MCO><mark>two</mark>

Next, answer the following questions:

a. Is there a compilation (syntax error) in your codes? (YES or NO). <mark>NO</mark>

b. Is there any compilation warning in your codes? (YES or NO) <mark>NO</mark>

---

3. How did you implement your stack data structures? Did you use an array or linked list? Why? Explain briefly (at most 5 sentences).

In our implementation, we used an array to implement our stack data structure as there is already a set maximum amount of data specified, which was 32768 coordinate inputs. Unlike a stack implementation using linked lists, which would be recommended for an unknown amount of input, an array implementation for a stack data structure may already allocate the amount of memory and space it needs. Moreover, an array implementation of stacks is beneficial in its simplicity and readability, which is better for beginners like us to work with.

---

4. Disclose **IN DETAIL** what is/are NOT working correctly in your solution. Please be honest about this. NON-DISCLOSURE will result in severe point deduction. Explain briefly the reason why your group was not able to make it work.

N/A


5. Based on the <u>exhaustive testing</u> that you did, fill-up the Comparison Table below that shows the performance between the "slow" version versus the "fast" version. Test for 10 different values of n, starting with $n = 2^6 = 64$ points.

Table: Performance Comparison Between "Slow" and "Fast" Versions

Test Case #	n (input size)	"Slow" version: execution time in *ms*	"Fast" version: execution time in *ms*
1	*2^6 = 64*	1.000000	0.000000
2	*2^7*	8.000000	0.000000
3	*2^8*	27.000000	2.000000
4	*2^9*	67.000000	5.000000
5	*2^10*	265.000000	9.000000
6	*2^11*	1257.000000	27.000000
7	*2^12*	4686.000000	81.000000
8	*2^13*	18011.000000	214.000000
9	*2^14*	71864.000000	688.000000
10	*2^15 = 32768*	303129.000000	2919.000000

NOTE: Make sure that you fill-up the table properly. It contributes 4 out of 15 points for the Documentation.

5. Create a graph (for example using Excel) based on the Comparison Table that you filled-up above. The x-axis should be the values of *n* and the y axis should be the execution time in milliseconds (ms). There should be two line graphs, one for the "slow" and the other for the "fast" data that should appear in one image. Copy/paste an image of the graph below.
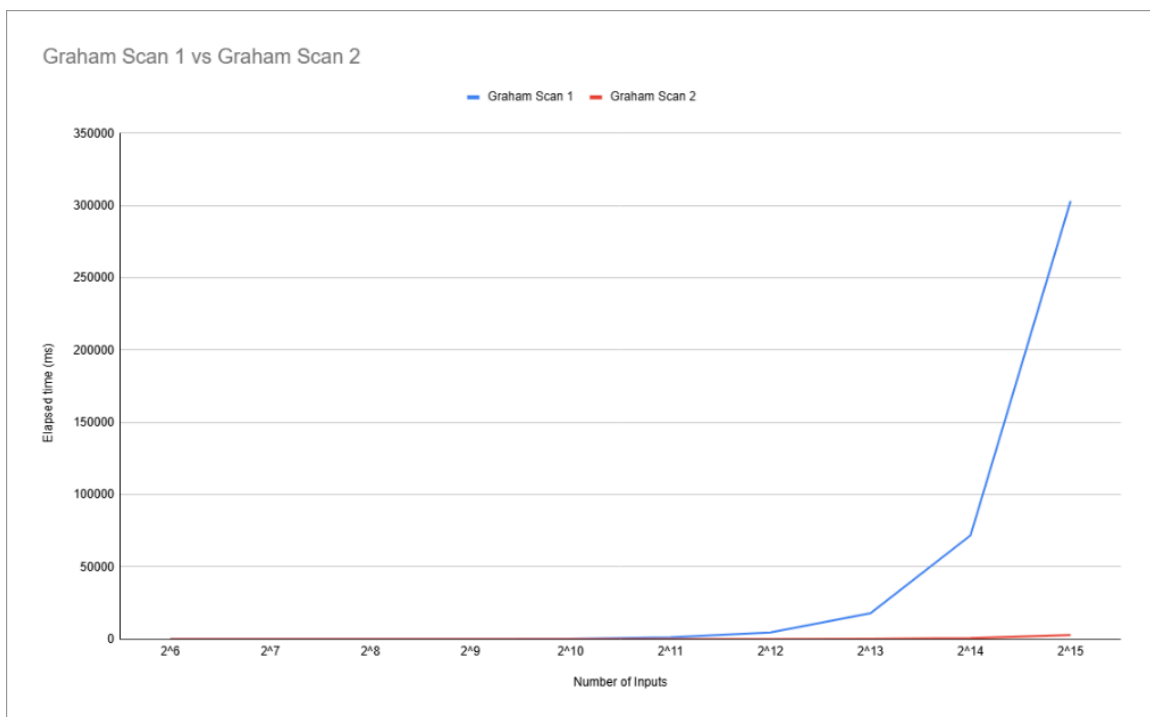
NOTE: Make sure that you provide a graph based on your comparison table data above. It contributes 4 out of 15 points for the Documentation.

6. Analysis – compare and analyze the growth rate behaviors of the "slow" and "fast" versions based on the Comparison Table and the graphs above.

Answer the following question:

a. What do you think is the growth rate behavior of the "slow" version?

Based on the graph, there is a definite correlation between the elapsed time of the slow version of the program and amount of inputs. It is apparent that there is a rapid increase in elapsed time for the slow version of the program everytime the number of inputs were to be multiplied by 2. This would mean the algorithm would most likely fall under a quadratic growth rate behavior, or $O(n^2)$, which would be appropriate as we have implemented a Selection sorting algorithm in this version of the Graham scan algorithm.

b. What do you think is the growth rate behavior of the "fast" version?

Compared to the slow version's graph, the fast version is able to find all of the points of the convex hull with less time consumed in all cases. There is a less extreme increase in elapsed time We have implemented the exact same graham scan algorithm to this as the slow version, with only its sorting algorithm replaced with a merge sorting algorithm for this version. With that said, using the same implications from the first question, unless the graham scan algorithm presents a worse growth rate behavior, then we may also present that the worst case growth behavior of the fast version would be that of the merge sorting algorithm, which would be linear-logarithmic, or $O(n \log n)$.

c. What do you think is/are the factor/s that make the "fast" version compute the results faster than the "slow" version?

The most important factor that determines why the fast version computes the results faster than the slow version is due to the sorting algorithm it uses. In the slow version of our program, we have used a Selection sorting algorithm ($O(n^2)$), while the fast version of our program uses the Merge sorting algorithm ($O(n \log n)$). This is the only prominent difference between the programs, meaning that this also most likely determines which of the two is faster. However, other factors such as the physical state of the processors or multitasking peripheral operations may contribute to the speed of running this specific program. In our case, we have used a 5-year old laptop in running the programs with other applications running in the background, which could have contributed to the results.

NOTE: Make sure that you provide cohesive answers to the three questions above. This part contributes 4 out of 15 points for the Documentation.

7. Fill-up the table below. Refer to the rubric in the project specs. It is suggested that you do an individual self-assessment first. Thereafter, compute the average evaluation for your group, and encode it below.

REQUIREMENT	AVE. OF SELF-ASSESSMENT
1. Stack	20 (max. 20 points)
2. Sorting algorithms	20 (max. 20 points)
3. Graham's Scan algorithm	40 (max. 40 points)
4. Documentation	15 (max. 15 points)
5. Compliance with Instructions	5 (max. 5 points)

TOTAL SCORE   100 over 100.

NOTE: The evaluation that the instructor will give is not necessarily going to be the same as what you indicated above. The self-assessment serves for your own reference only…