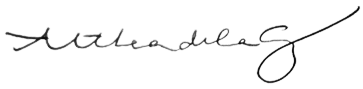
CCDSALG Term 3, AY 2024 – 2025

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	Samonte, Anne Camille T. 	✓	✓	✓	✓	✓	✓
S13	dela Cruz, Althea Leanne L. 	✓	✓	✓	✓	✓	✓
S13	Dee, Adrian Matthew L. 	✓	✓	✓	✓	✓	✓

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`	5 sample corresponding output files	✔
`12.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 -Wall main1.c graham_scan1.c graham_helper.c stack.c sort.c -o main1.exe

  C:\MCO> gcc -Wall main2.c graham_scan2.c graham_helper.c stack.c sort.c -o main2.exe

- How to run from command line

  C:\MCO>main1

  C:\MCO>main2

Next, answer the following questions:

a. Is there a compilation (syntax error) in your codes? (YES or NO). __NO__

WARNING: the project will automatically be graded with a score of **0** if there is syntax error in any of the submitted source code files. Please make sure that your submission does not have a syntax error.

b. Is there any compilation warning in your codes? (YES or NO) __NO___

WARNING: there will be a 1 point deduction for every unique compiler warning. Please make sure that your submission does not have a compiler warning.

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

The stack data structure for this program was implemented using arrays, as they provide easier access through indexing and proved to be more efficient in memory handling. We used a Point struct, consisting of x-coordinate, y-coordinate, and polar angle, in which an array-based implementation made it easier to push and access points. Given that its memory is only allocated once, it was also easier to implement the stack using an array for a maximum problem size requirement of 2^15 elements. In contrast, a linked list can lead to memory inefficiency due to multiple allocations that can potentially allocate more memory than necessary. Lastly, given that array elements are contiguous in memory, operations such as top and nextToTop became easier to implement because simple indexing could be used to access the elements.

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.

No errors found.

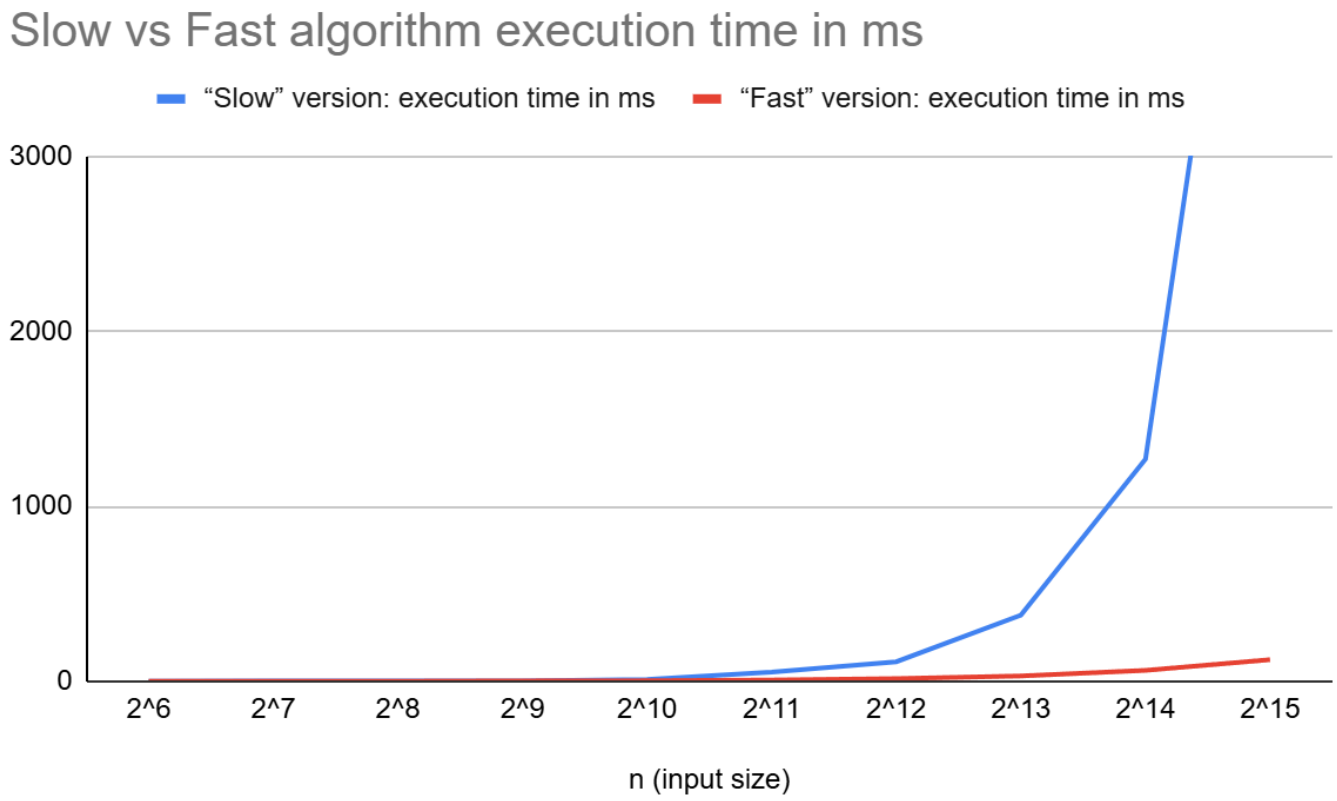
5. Based on the exhaustive testing that you did, fill-up the Comparison Table below that shows the performance between the "slow" version versus the "fast" version. Test for 10 different values of *n*, starting with *n = 2^6 = 64* points.

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	1.000000
2	*2^7*	2.000000	1.000000
3	*2^8*	3.000000	1.000000
4	*2^9*	3.000000	3.000000
5	*2^10*	11.000000	4.000000
6	*2^11*	53.000000	8.000000
7	*2^12*	112.000000	16.000000
8	*2^13*	379.000000	31.000000
9	*2^14*	1270.000000	63.000000
10	*2^15 = 32768*	6048.000000	124.000000

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.



Link to Spreadsheet

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?

The "slow" version seems to exhibit a polynomial growth as the input size n increases. The execution time appears to increase by around 2-5 times every time the input size doubles. Thus, this suggests a polynomial growth with a big Oh notation of O(n^k). If this version had an exponential growth rate, then the execution time would surge more rapidly. For instance, if it had a time complexity of O(2^n), then the input size would square the execution time.

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

The "fast" version seems to exhibit a linear growth as the execution time roughly doubles as the input size doubles. Hence, this version suggests a time complexity of O(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 main factor as to why the “fast” version computes the results faster than the “slow” version is the sorting algorithm utilized. Apart from the sorting algorithm, the implementation for both “fast” and “slow” versions are the same. In the worst and average case, the bubble sort algorithm present in the “slow” version has a big Oh notation of O(n^2). As it repeatedly swaps and compares elements, this algorithm doesn’t scale well when the input size increases. The “fast” version, on the other hand, utilizes the quicksort algorithm, which has a best case and average case complexity of O(n log n). The quicksort algorithm works better with larger input sizes as it employs the divide-and-conquer approach.

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…

サルバドル・フロランテ