CCDSALG/GDDASGO Term 1, 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
<S11>	Barlaan, Bahir Benjamin		X			X	
<S11>	Jocson, Vince Miguel	X			X	X	
<S11>	Vergara, Royce Aaron Adam			X	X		X

Fill-up the table above. For the tasks, put an 'X' or check mark if you have performed the specified task. Please refer to the project specifications for the description of each task. 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.

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 `INPUT5.TXT`	5 sample input files (with increasing values of *n*)	✔
`OUTPUT1.TXT` to `OUTPUT5.TXT`	5 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. 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 -std=c99 main1.c -o main1.exe
C:\MCO> gcc -Wall -std=c99 main2.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).

Our stack data structure was implemented using an array. Since most of us have been more used to arrays we felt more confident and comfortable in using arrays for implementation. Aside from simplicity, we also used an array because there was a fixed size indicated and arrays are suitable for this circumstance.

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.

For example:

The following are NOT working (buggy):

a.

b.

We were not able to make them work because:

a.

b.

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 <u>at least</u> 5 different values of *n*, starting with *n = 2^6 = 64*

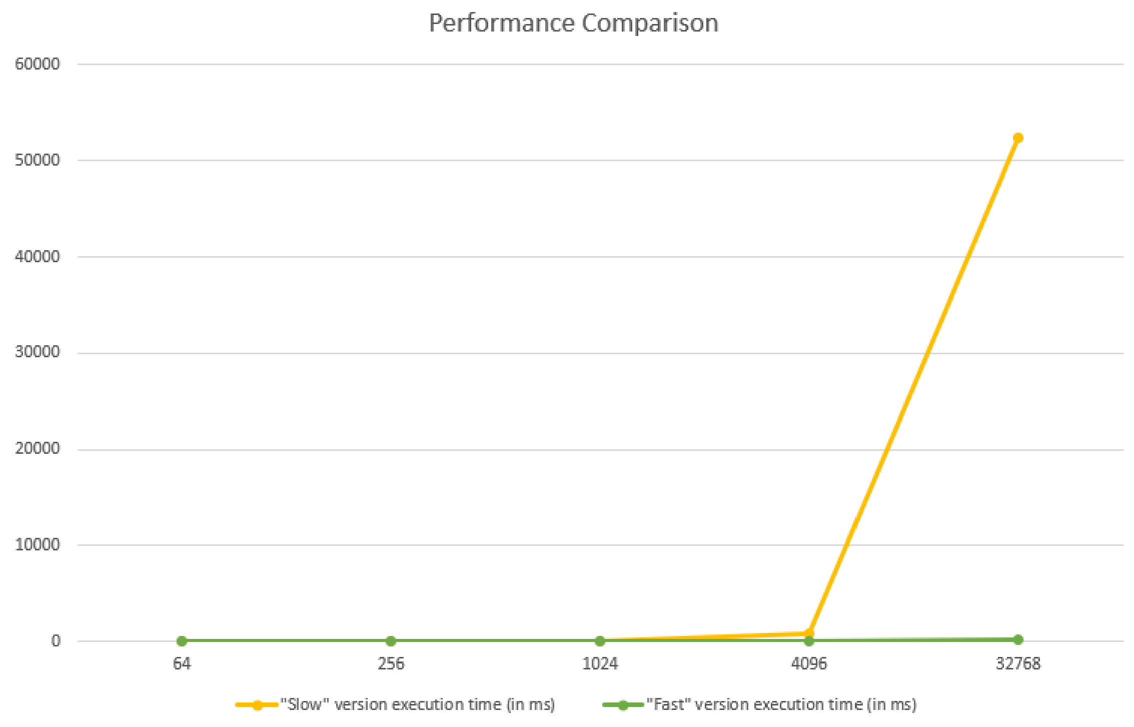
points. Set the values of *n* such that they are expressed using 2 as exponent (for example *n = 2^8 = 256*). Your last test case should have *n = 2^15 = 32768* points.

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	0.000000	0.000000
2	2^8 = 256	3.000000	0.000000
3	2^10 = 1024	51.000000	2.000000
4	2^12 = 4096	805.000000	10.000000
5	2^15 = 32768	52375.000000	107.000000

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?

O(n^2)

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

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?

One factor that makes the "fast" version compute results faster than the "slow" version is the number of comparisons and swaps in each algorithm. In our group's case, our slow version was sorted using bubble sort and our fast version was sorted using heap sort. Bubble sort compares adjacent values and swaps them if needed, which makes it go through the entire array multiple times to make sure each element is properly sorted. Heap sort on the other hand performs fewer swaps and comparisons because of the Heapify function, which works because of the heap structure. It compares a child node to its parent node and swaps if the child node is bigger.

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	25 (max. 25 points)
2. Sorting algorithms	15 (max. 15 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 primarily as a guide.

サルバドール・フロランテ