

Lab 3: Sorting (Group Project)

This project will utilize the following sorting algorithms:

- Selection Sort, Insertion Sort, Shell Sort, Bubble Sort, Heap Sort, Merge Sort, Quick Sort, Radix Sort, and Counting Sort.
- Bonus exploration: Binary Insertion Sort, Shaker Sort, Flash Sort.

Details about the requirements can be found in the information below.

1 Programming

1.1 Algorithms

You are tasked with implementing all the above sorting algorithms in **C/C++** for **ascending order**. The implementation of bonus algorithms for additional credit is encouraged.

1.2 Experiments

For this project, the following scenario outlines the necessary steps for conducting the experiments:

```
for each Data Order S1
  for each Data Size S2
    - Create an original array A1 with Data Order S1 and Data Size S2
    for each Sorting Algorithm S3
      - Make a duplicate array A2 of the original array A1
      - Sort array A2 using the Sorting Algorithm S3, while:
        + Measuring the running time (in millisecs), and
        + Counting the number of comparisons in the algorithm
      - Take note of S1, S2, S3, running time and number of comparisons
    end for
  end for
end for
```

1.2.1 Data Order Evaluate the sorting algorithms on various data arrangements including **sorted** (in ascending order), **nearly sorted**, **reverse sorted**, and **randomized** data.

Refer to `DataGenerator.cpp` for more information.

1.2.2 Data Size Evaluate the algorithms on datasets of varying sizes: 10,000, 30,000, 50,000, 100,000, 300,000, and 500,000 elements.

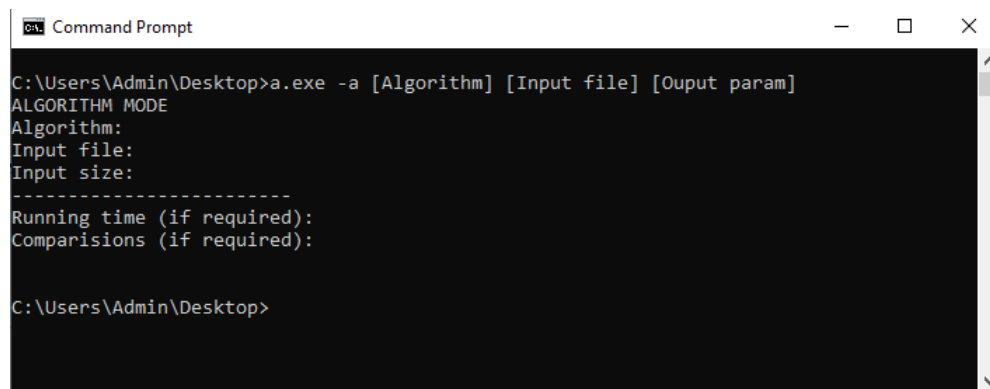
1.3 Output specifications

Your source code must be compiled into an executable file (.exe) that can be executed using commands in the command prompt.

1. **Algorithm mode:** This mode allows you to execute a specific sorting algorithm on input data. This data can be either user-provided or automatically generated. The mode then outputs the measured execution time and/or the number of comparisons performed by the algorithm.

- Command 1: Run a sorting algorithm on user-provided data.

- **Prototype:** [Execution file] -a [Algorithm] [Input filename] | [Output parameter(s)]
- **Example:** a.exe -a radix-sort input.txt -both
- **Console output:**

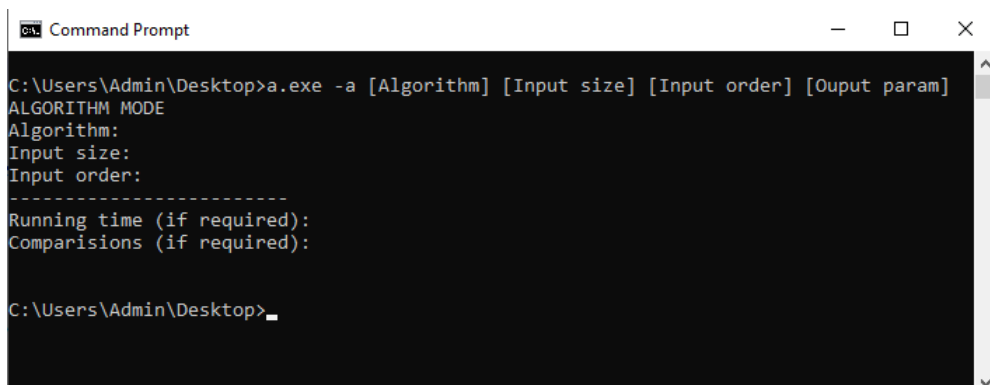


```
Command Prompt
C:\Users\Admin\Desktop>a.exe -a [Algorithm] [Input file] [Output param]
ALGORITHM MODE
Algorithm:
Input file:
Input size:
-----
Running time (if required):
Comparisons (if required):

C:\Users\Admin\Desktop>
```

- Command 2: Run a sorting algorithm on the data generated automatically with specified size and order.

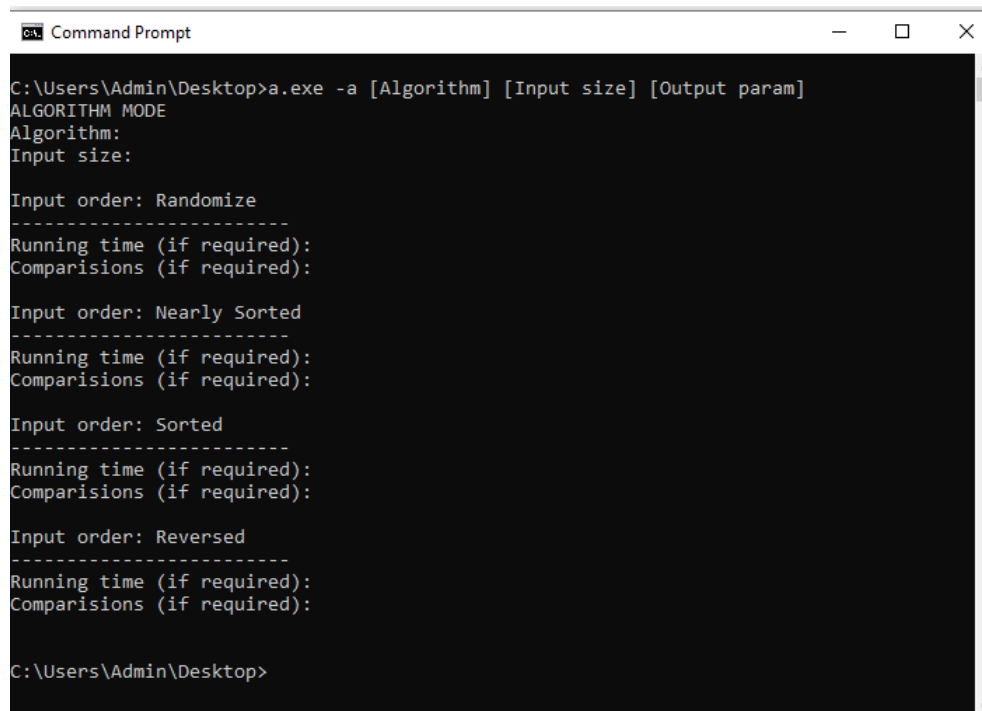
- **Prototype:** [Execution file] -a [Algorithm] [Input size] [Input order] | [Output parameter(s)]
- **Example:** a.exe -a selection-sort 50 -rand -time
- **Console output:**



```
Command Prompt
C:\Users\Admin\Desktop>a.exe -a [Algorithm] [Input size] [Input order] [Output param]
ALGORITHM MODE
Algorithm:
Input size:
Input order:
-----
Running time (if required):
Comparisons (if required):

C:\Users\Admin\Desktop>
```

- **Command 3:** Run a sorting algorithm on ALL data arrangements of a specified size.
 - **Prototype:** [Execution file] -a [Algorithm] [Input size] [Output parameter(s)]
 - **Example:** a.exe -a quick-sort 70000 -comp
 - **Console output:**



```
Command Prompt
C:\Users\Admin\Desktop>a.exe -a [Algorithm] [Input size] [Output param]
ALGORITHM MODE
Algorithm:
Input size:

Input order: Randomize
-----
Running time (if required):
Comparisions (if required):

Input order: Nearly Sorted
-----
Running time (if required):
Comparisions (if required):

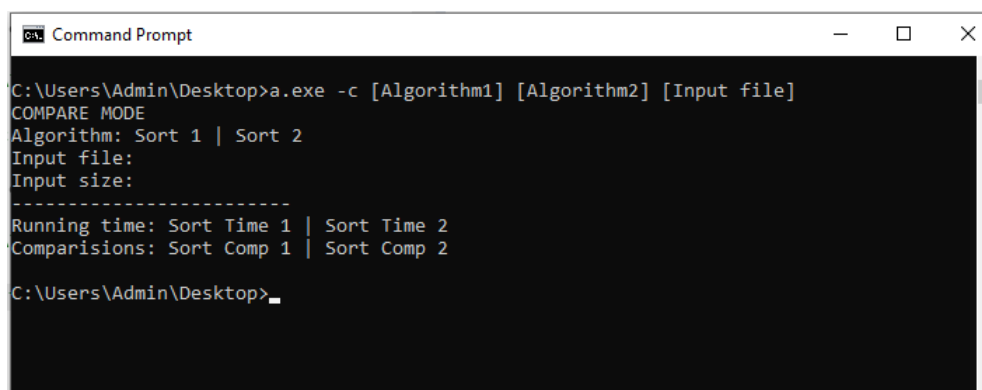
Input order: Sorted
-----
Running time (if required):
Comparisions (if required):

Input order: Reversed
-----
Running time (if required):
Comparisions (if required):

C:\Users\Admin\Desktop>
```

2. Comparison mode: This mode allows you to compare two sorting algorithms on input data. This data can be either user-provided or automatically generated. The mode then outputs the measured execution times and/or the number of comparisons performed by each algorithm.

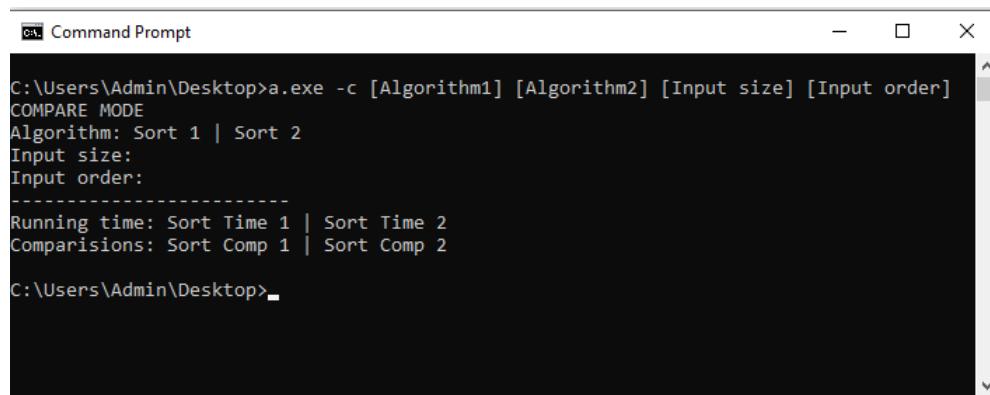
- **Command 4:** Run two sorting algorithms on user-provided data.
 - **Prototype:** [Execution file] -c [Algorithm 1] [Algorithm 2] [Input filename]
 - **Example:** a.exe -c heap-sort merge-sort input.txt
 - **Console output:**



```
Command Prompt
C:\Users\Admin\Desktop>a.exe -c [Algorithm1] [Algorithm2] [Input file]
COMPARE MODE
Algorithm: Sort 1 | Sort 2
Input file:
Input size:
-----
Running time: Sort Time 1 | Sort Time 2
Comparisions: Sort Comp 1 | Sort Comp 2

C:\Users\Admin\Desktop>
```

- **Command 5:** Run two sorting algorithms on the data generated automatically with specified size and order.
 - **Prototype:** [Execution file] -c [Algorithm 1] [Algorithm 2] [Input size] [Input order]
 - **Ex:** a.exe -c quick-sort merge-sort 100000 -nsorted
 - **Console output:**



```

C:\Users\Admin\Desktop>a.exe -c [Algorithm1] [Algorithm2] [Input size] [Input order]
COMPARE MODE
Algorithm: Sort 1 | Sort 2
Input size:
Input order:
-----
Running time: Sort Time 1 | Sort Time 2
Comparisons: Sort Comp 1 | Sort Comp 2

C:\Users\Admin\Desktop>

```

3. Input arguments: *The following arguments are applied for both modes:*

a. Mode:

- **-a:** Algorithm mode
- **-c:** Comparison mode

b. Algorithm name: Lowercase, words are connected by "-" (Ex: selection-sort, binary-insertion-sort, ...)

c. Input size: Integer ($\leq 1,000,000$)

d. Input order:

- **-rand:** randomized data
- **-nsorted:** nearly sorted data
- **-sorted:** sorted data
- **-rev:** reverse sorted data

e. Given input (file): Path to the input file. The file format is as follows:

- 1^{st} line: an integer n , indicating the number of elements in the input data
- 2^{nd} line: n integers, separated by a single space

f. Output parameters

- **-time:** algorithms's running time
- **-comp:** number of comparisons
- **-both:** both above options

4. Writing files: In addition to the console output described above, you are required to write down the corresponding input(s) or output(s).

- For Command 1 and Command 2: Write down the sorted array to the *"output.txt"* file.
- For Command 2 and Command 5: Write down the generated input to the *"input.txt"* file.
- For Command 3: Write down all four generated input:
 - *"input_1.txt"*: random order data
 - *"input_2.txt"*: nearly sorted data
 - *"input_3.txt"*: sorted data
 - *"input_4.txt"*: reversed data

The file format (for both input and output files) is as follows:

- 1st line: an integer n , indicating the number of elements in the input data
- 2nd line: n integers, separated by a single space

2 Report

Structure your report file with the following sections:

1. **Information page**
2. **Introduction page**
3. **Algorithm presentation:** This section delves into the implemented sorting algorithms. Here, you'll provide a comprehensive breakdown for each algorithm, including:
 - Core Concepts: Briefly discuss the core ideas behind each algorithm.
 - Step-by-step Explanations: Provide a clear and step-by-step explanation for each algorithm, including examples for more details.
 - Complexity Analysis: Analyze the time complexity of each algorithm. If applicable, also discuss the space complexity.
 - Variants and Optimizations: If any of the algorithms have known variants or optimizations, discuss them here.
4. **Experimental results and comments:**
 - You are required to organize the experimental results into FOUR tables, each representing one **Data order**. In each table, present the resulting statistics (i.e., running times or numbers of comparisons) of all sorting algorithms following a specific data arrangement. The table template is shown below:

Data order: ...						
Data size	10,000		50,000		...	
Resulting statics	Running time	Comparision	Running time	Comparision	Running time	Comparision
Sorting algorithm 1						
Sorting algorithm 2						
...						

- You are also required to make visualization by graphs.
 - There will be four LINE GRAPHS, each of which corresponds to a table of running times. In every graph, the horizontal axis is for **Data Size** and the vertical axis is for running time, as shown in Figure 1.

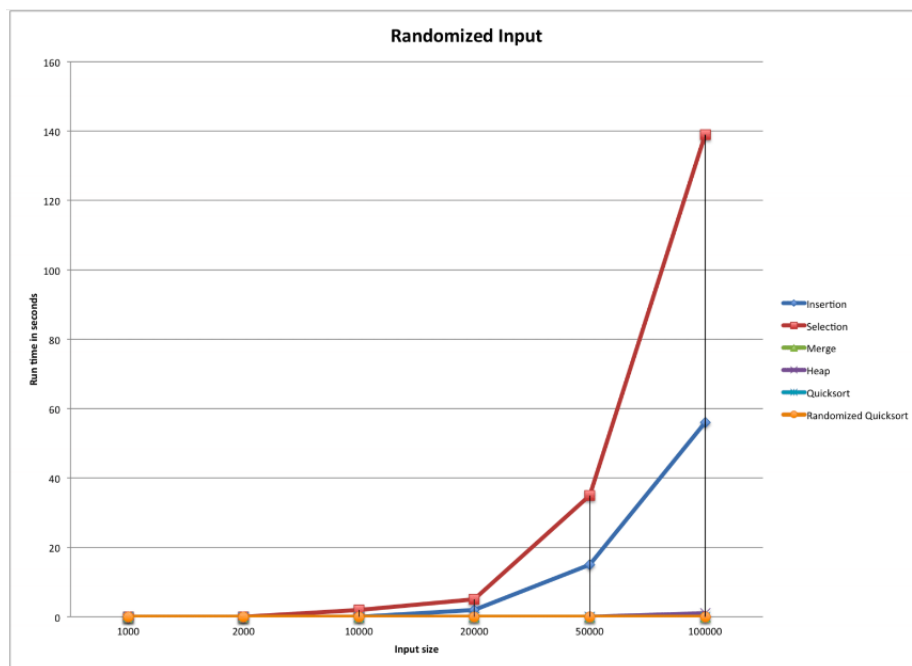


Figure 1: An example of a line graph for visualizing the algorithms' running times on randomized input data.

- There will be four BAR CHARTs, each of which corresponds to a table of numbers of comparisons. In every graph, the horizontal axis is for **Data Size**, and the vertical axis is for the number of comparisons, as shown in Figure 2.
 - Make comments based on your own observations on each graph (e.g., the fastest/slowest or the most/least comparisons algorithm(s) in each case, time or comparisons acceleration of algorithms, etc.). Explain your comments.
 - Make an overall comment of algorithms on all **Data Order** and all **Data Size** (the fastest/slowest algorithms overall, grouping the stable/unstable algorithms, etc.)
5. **Project organization and Programming notes:** A brief explanation of how you organized your source codes, and notes of any special libraries/data structures used.

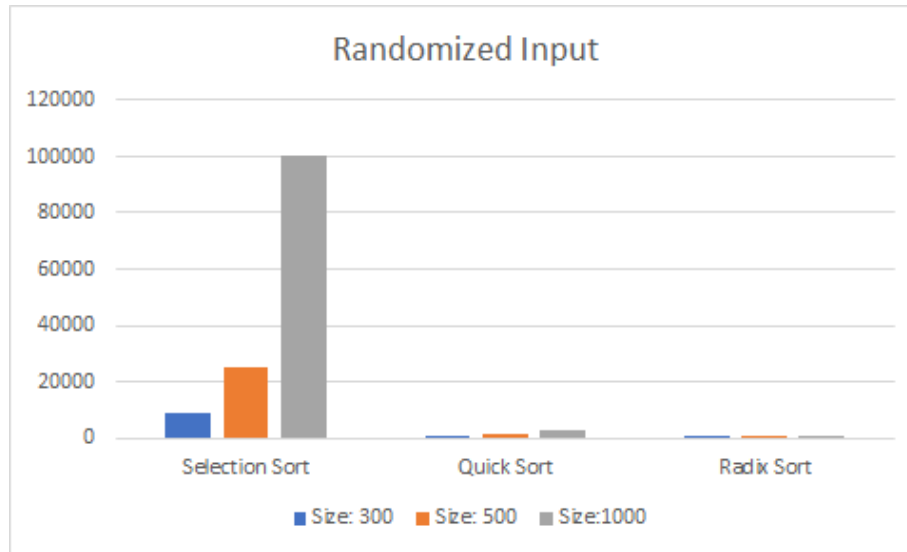


Figure 2: An example of a bar chart for visualizing the algorithms' numbers of comparisons on randomized input data.

6. List of references.

3 Submission

This is a 4-person group project. Formation of groups with fewer than 4 members requires permission from the instructors. Individual assignments will not be accepted.

- Create the folder <Group ID> to include the following materials:
 - SOURCE folder: the project's source codes, only files of extensions **.cpp** and **.h** are required.
 - Executable file: <Group ID>.exe (e.g. 01.exe)
 - Report.pdf: the report file of extension **.pdf**.
 - Checklist.xlsx: the Excel template file filled with your information.
- Compress the above folder into a file of extension **zip** and name it following your Group ID (e.g. 01.zip).
- Only one member representing the group submits the assignment.

Submissions that violate any regulations will get a score of ZERO.

Plagiarism and Cheating will result in a score of ZERO for the entire course and will be subject to appropriate referral to the Management Board for further action.