Lab 3: Sorting

Each group chooses either of the following sets to complete the project.

- **Set 1** (7 algorithms): Selection Sort, Insertion Sort, Bubble Sort, Heap Sort, Merge Sort, Quick Sort, and Radix Sort.

- **Set 2** (11 algorithms): Selection Sort, Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Flash Sort.

Please consult the following content to get the requirements.

1 Programming

1.1 Algorithms

You are asked to implement ALL algorithms (for **ascending order** only) present in the set you have selected, using *C/C++* programming language.

1.2 Experiments

The experiments necessary for this project should be conducted following the below scenario.

```
for each Data Order S₁
    for each Data Size S₂
        for each Sorting Algorithm S₃
            1. Create an array with Data Order S₁ and Data Size S₂
            2. Sort the created array using the Sorting Algorithm S₃, while:
                + Measuring the running time (in millisecs), and
                + Counting the number of comparisons in the algorithm
            3. Take note of S₁, S₂, S₃, running time and number of comparisons
        end for
    end for
end for
```

1.2.1 Data Order Examine the selected sorting algorithms on different data arrangements, including: `Sorted data (in ascending order)`, `Nearly sorted data`, `Reverse sorted data`, and `Randomized data`. See `DataGenerator.cpp` for more information.

1.2.2 Data Size Examine the selected sorting algorithms on data of the following sizes: `10,000`, `30,000`, `50,000`, `100,000`, `300,000`, and `500,000` elements.

2   Submission

2.1   Submission regulations

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.

- Report.pdf: the report file of extension **.pdf**.

- Checklist.xlsx: the Excel template file filled with your own information.

Compress the above folder into a file of extension **zip** or **rar** and name it following your Group ID (Ex: *"1.rar"*). Submission that violates any regulation will get no credit (zero).
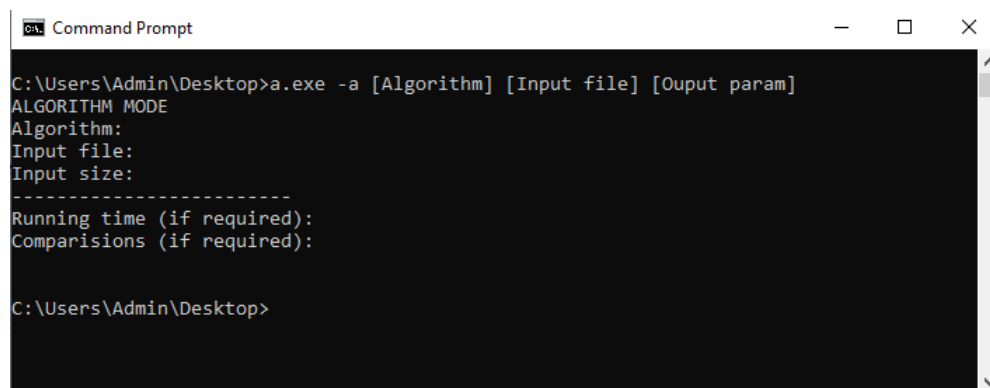
Plagiarism and Cheating will result in a "0" (zero) for the entire course and will be subject to appropriate referral to the Management Board for further action.

2.2   Output specifications

You must compile your source codes into an executable file (.exe) that can be triggered by using command line.

1. **Algorithm mode**: In this mode, you are asked to run a specified sorting algorithm on the input data, which is either given or generated automatically, and present the resulting running time and/or number of comparisons.

   - Command 1: Run a sorting algorithm on the given input data.
     - **Prototype:** `[Execution file] -a [Algorithm] [Given input] [Output parameter(s)]`
     - **Ex**: `a.exe -a radix-sort input.txt -both`
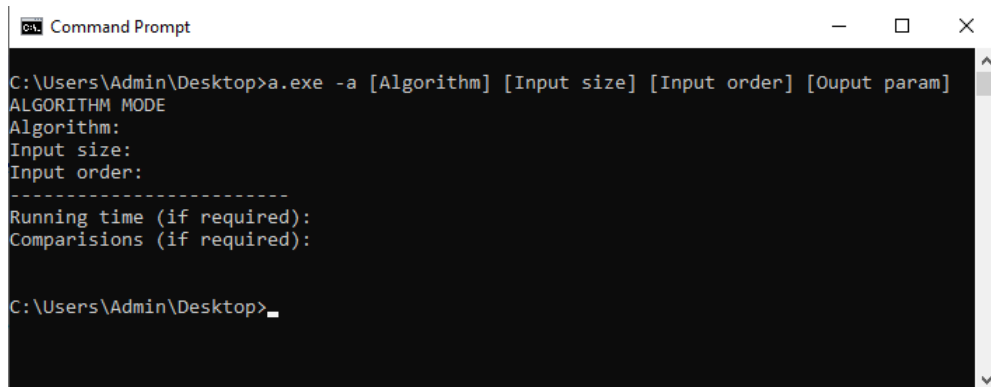   - **Console output**:

- **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)]
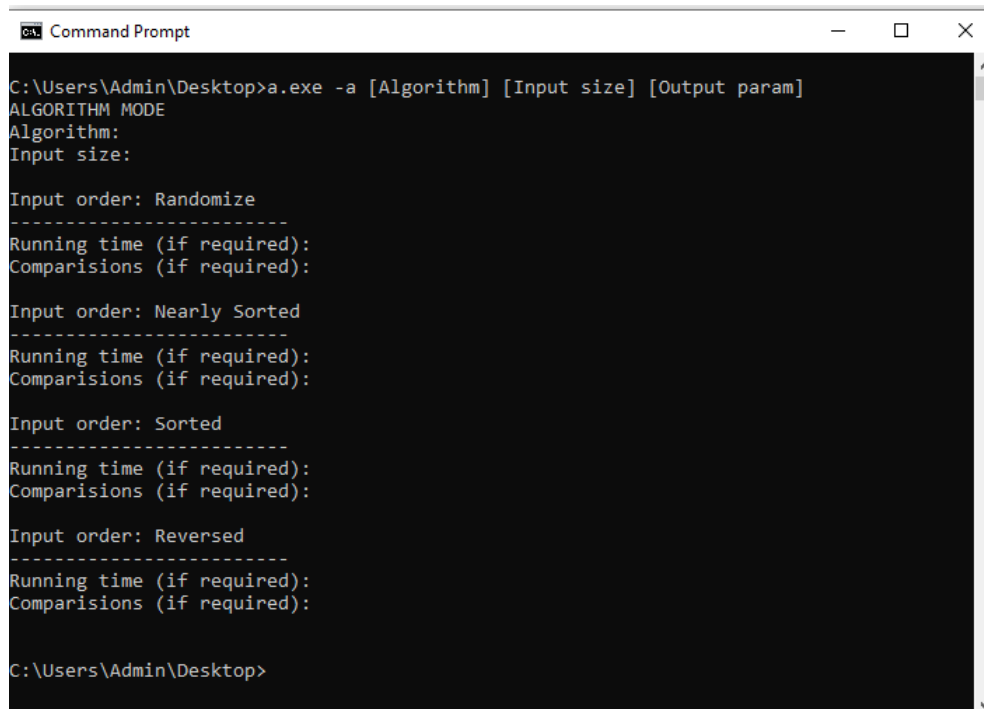 - **Ex**: `a.exe -a selection-sort 50 -rand -time`
  - **Console output**:

```
Command Prompt                                    —    □    ×

C:\Users\Admin\Desktop>a.exe -a [Algorithm] [Input size] [Input order] [Ouput param]
ALGORITHM MODE
Algorithm:
Input size:
Input order:
------------------------
Running time (if required):
Comparisions (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)]
  - **Ex**: `a.exe -a binary-insertion-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>
```
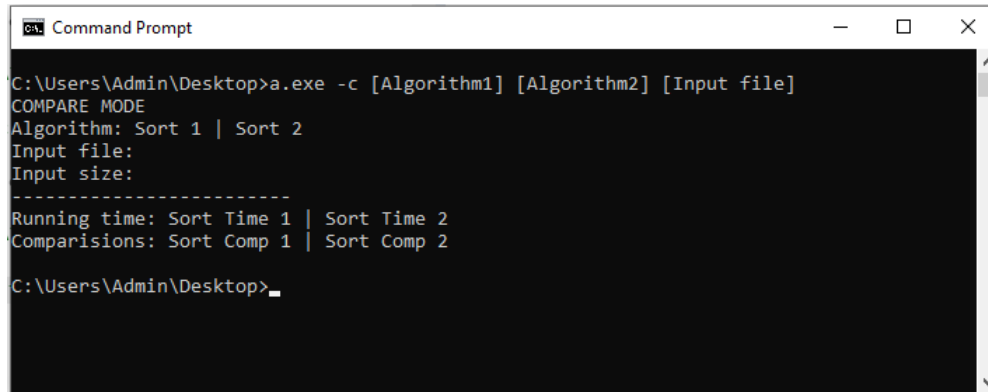
2. **Comparison mode:** In this mode, you have to run TWO specified sorting algorithms on the input data, which is either given or generated automatically, and present the resulting running times and/or numbers of comparisons.

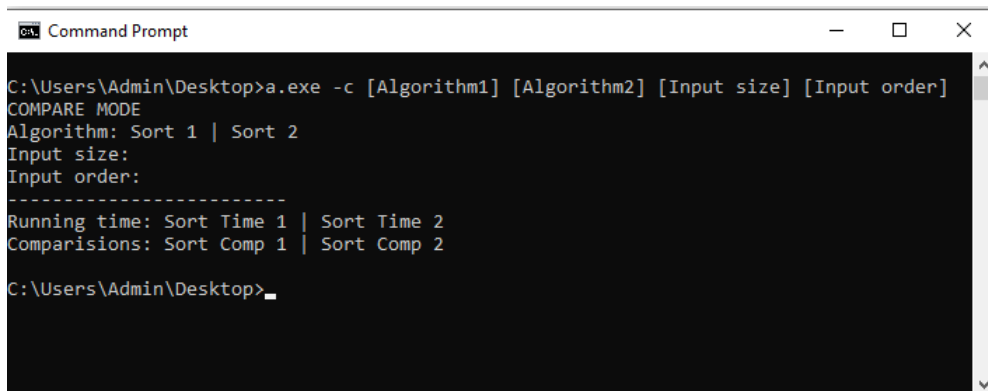- Command 4: Run two sorting algorithms on the given input.
    - **Prototype**: [Execution file] -c [Algorithm 1] [Algorithm 2] [Given input]
    - **Ex**: a.exe -c heap-sort merge-sort input.txt
    - **Console output**:

```
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.
    - **Prototype**: [Execution file] -c [Algorithm 1] [Algorithm 2] [Input size]
                                                                      [Input order]
    - **Ex**: a.exe -c quick-sort merge-sort 100000 -nsorted
    - **Console output**:

```
Command Prompt                                        —   □   ×

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
Comparisions: 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.

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

    f. Output parameters

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

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

- For <u>Command 1</u> and <u>Command 2</u>: Write down the sorted array to the *"output.txt"* file.
- For <u>Command 2</u> and <u>Command 5</u>: Write down the generated input to the *"input.txt"* file.
- For <u>Command 3</u>: 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.

2.3   Report

Your report file should include the following sections:

1. **Information page**

2. **Introduction page**

3. **Algorithm presentation**: In this section, you present the algorithms implemented in the project: ideas, step-by-step descriptions, and complexity evaluations (in terms of time complexity and space complexity, if possible). Variants/improvements of an algorithm, if there is any, should be also mentioned.

4. **Experimental results and comments**:

   - You are required to organize the experimental results into FOUR tables, each represent 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 arrangment. The table template is shown as 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 corrresponds 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.
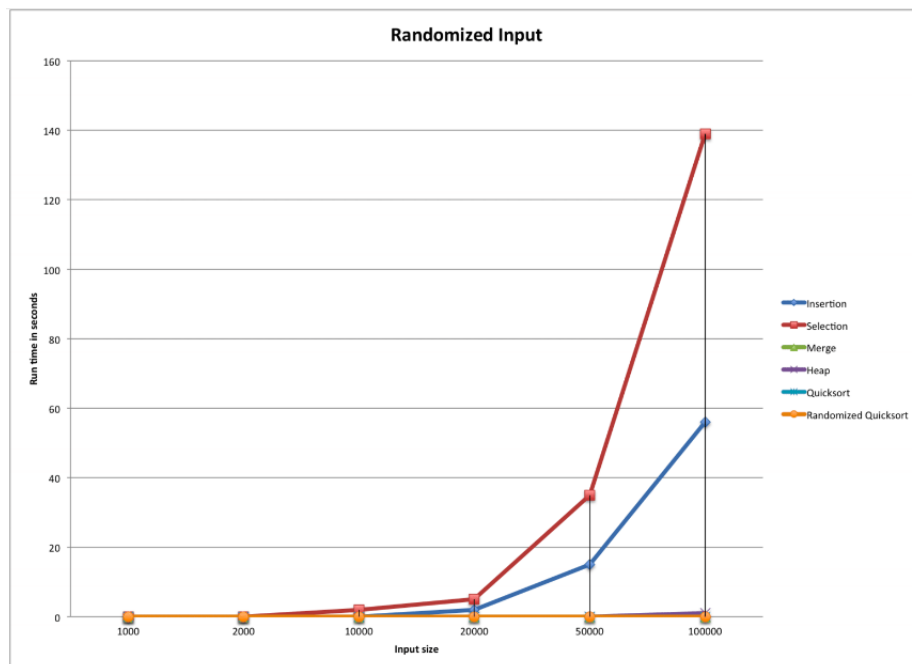

Figure 1: An example of a line graph for visualizing the algorithms' running times on randomized input data.

– 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 numbers of comparisons, as shown in Figure 2.
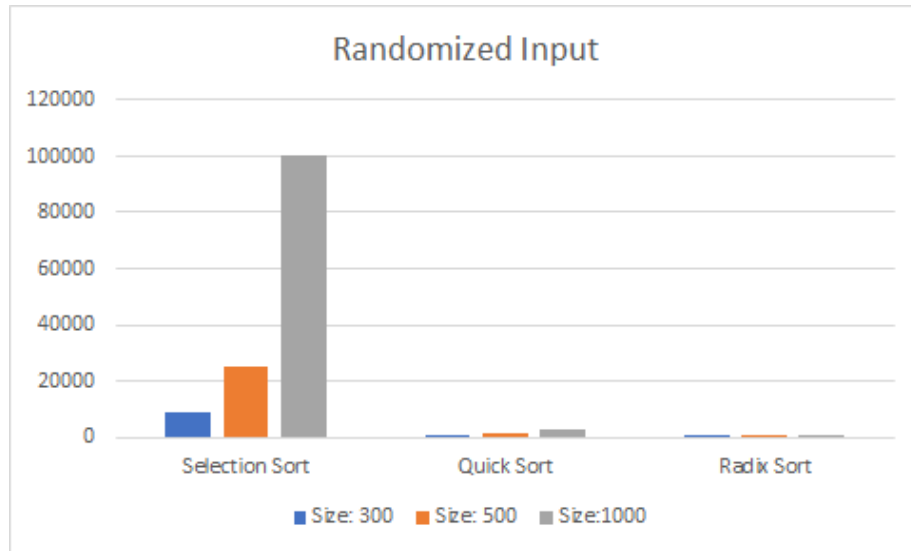

Figure 2: An example of a bar chart for visualizing the algorithms' numbers of comparisons on randomized input data.

- Make comments based on your own observations on each graph (e.g., the fastest / slowest or the most / least comparisions 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 soure codes, and notes of any special libraries/data structures used.

6. **List of references**.