Lab 4: Sorting (Group Project)

In this project, your group will implement the following 12 sorting algorithms and analyze their performance:

- **List of algorithms:** Selection Sort, Insertion Sort, Binary Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Flash Sort.

Please refer to the following content for the requirements.

1 Programming

1.1 Algorithms

You are asked to implement all the algorithms mentioned above (for **ascending order** only) using the *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 Evaluate the sorting algorithms on different data arrangements, including `Sorted` (in ascending order), `Nearly sorted`, `Reverse sorted`, and `Randomized`. Check the file `DataGenerator.cpp` for more information.

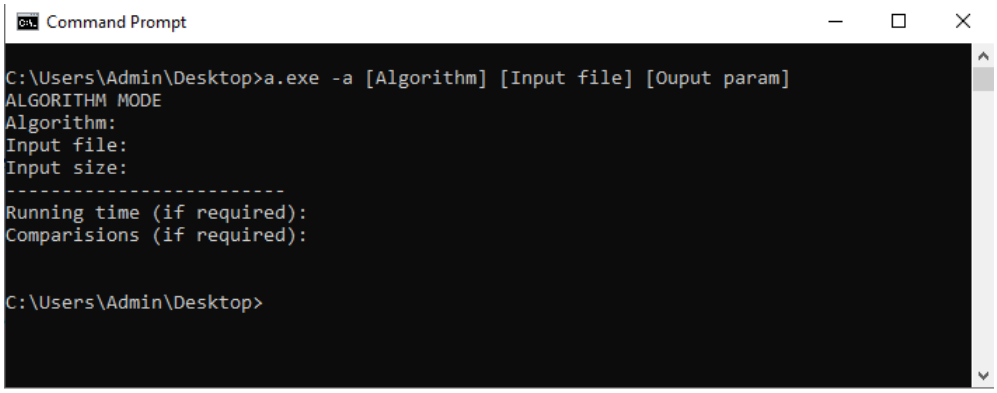
1.2.2 Data Size Evaluate the sorting algorithms on data of the following sizes: `10,000`, `30,000`, `50,000`, `100,000`, `300,000`, and `500,000` elements.

1.3 Output specifications

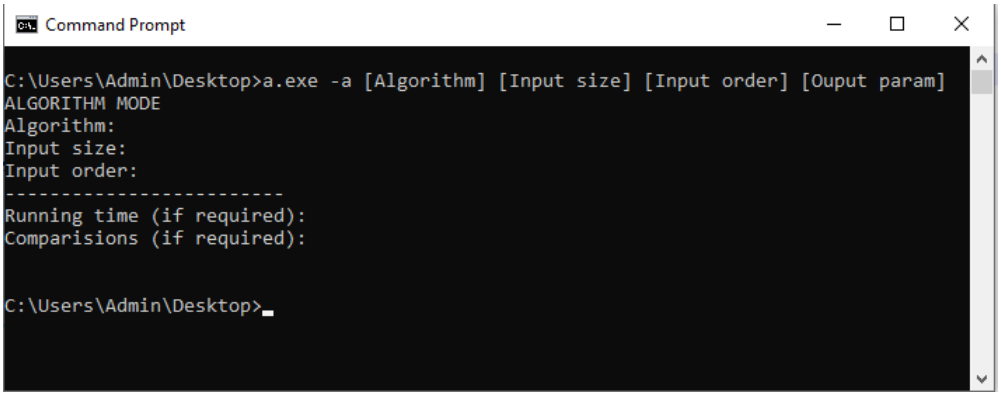
You must compile your source codes into an executable file (.exe) that can be triggered by using a command line.

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 presents the resulting running time and/or number of comparisons.

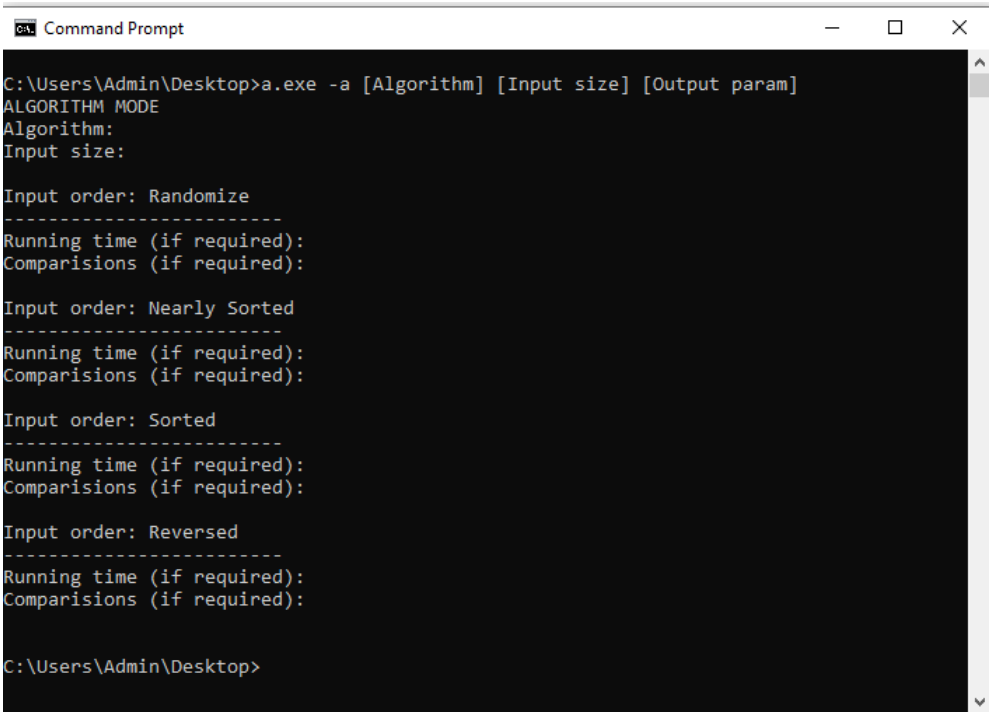
   - Command 1: Run a sorting algorithm on the given input data.
     - **Prototype:** `[Execution file] -a [Algorithm] [Given input] [Output parameter(s)]`
     - **Ex**: `a.exe -a radix-sort input.txt -both`
     - **Console output**:



   - 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**:

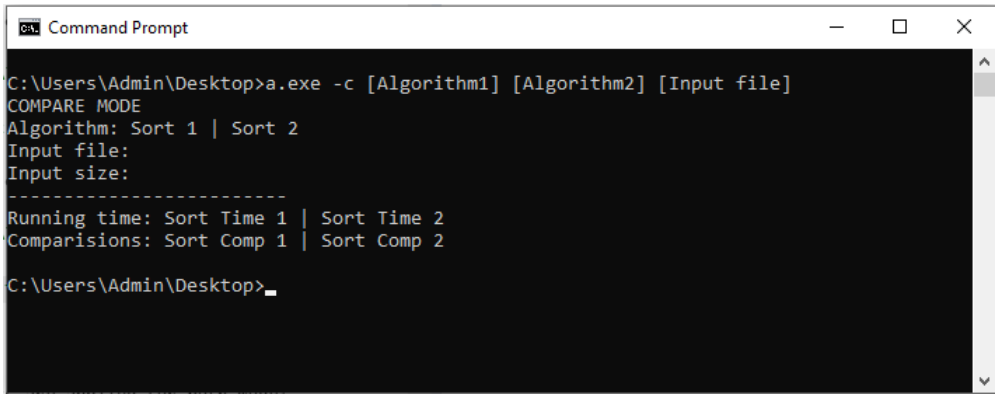


- Command 3: Run a sorting algorithm on ALL data arrangements of a specified size.
  - **Prototype**: `[Execution file] -a [Algorithm] [Input size] [Output parameter(s)]`
  - **Ex**: `a.exe -a binary-insertion-sort 70000 -comp`
  - **Console output**:

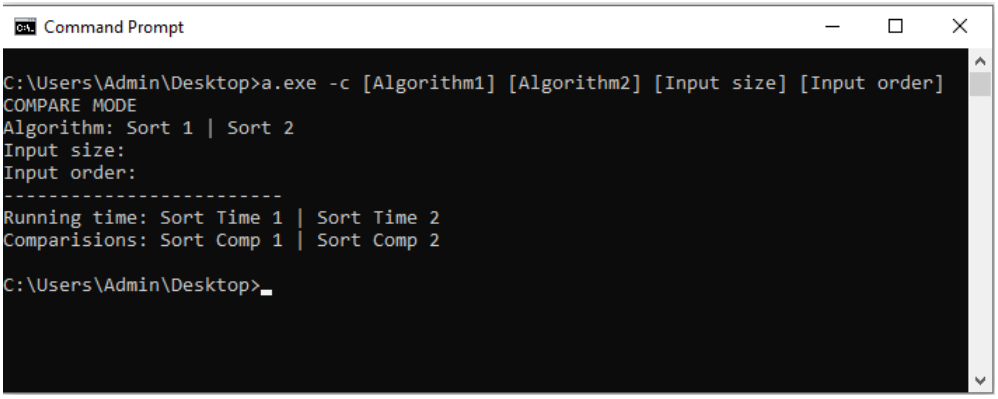


2. **Comparison mode:** In this mode, you have to run TWO specified sorting algorithms on the input data, which is either given or generated automatically and presents the resulting running times and/or numbers of comparisons.

- 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 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**:



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 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 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.

   - 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.

2 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 with examples, and complexity evaluations (in terms of time complexity and space complexity, if possible). Variants/improvements of an algorithm, if there are any, should be also mentioned.
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.

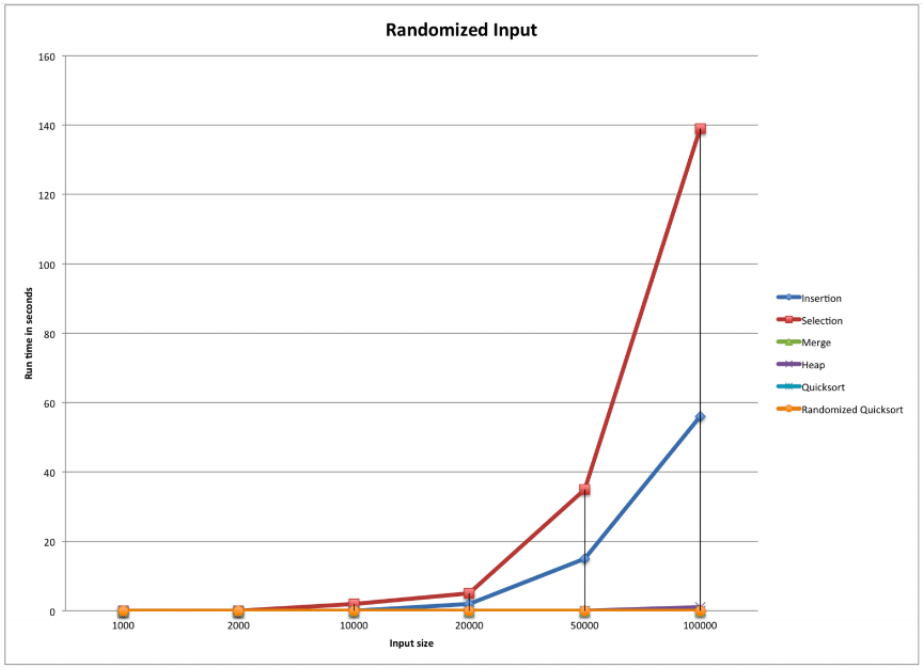


Figure 1: An example of a line graph for visualizing the algorithms' running times on randomized input data.

  - 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.
6. **List of references**.

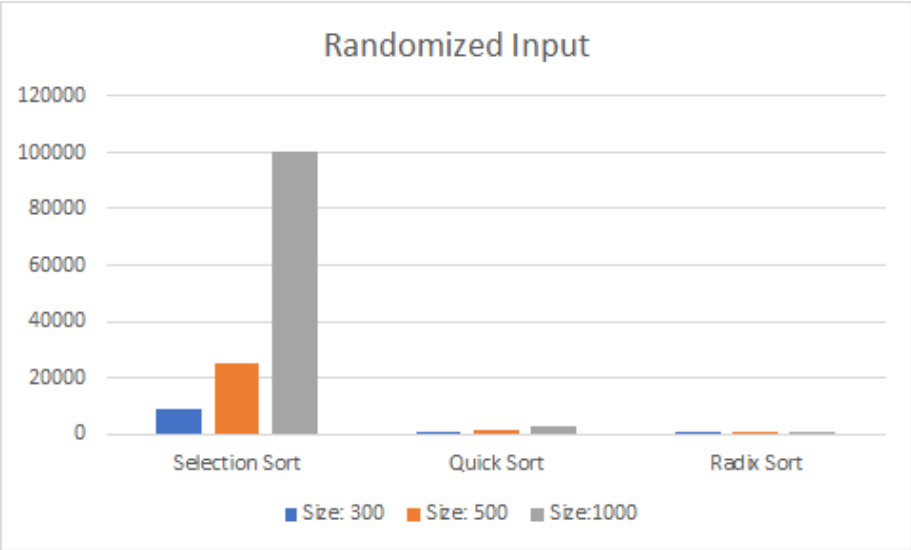


Figure 2: An example of a bar chart for visualizing the algorithms' numbers of comparisons on randomized input data.

3 Submission

This is a 4-person group project. If you want to form a group with fewer than 4 people, you need to get 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.
    - Please remove any files or folders that are not needed (`.vs`, `Debug`, `x64`, etc.) to avoid making your submission too large for Moodle.
    - **Do not submit .exe files** to prevent your submission from being deleted by antivirus software.
  - `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** and name it following your Group ID (Ex: *"1.zip"*). You can find your Group ID on the registration sheet on Moodle.
- Only **one** member representing the group to submits the assignment via Moodle.

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.