Weekly Homework 3

April 2, 2025

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 Comparison of Sorting Algorithms

1.1 Bubble Sort

- **Advantages:** Simple and easy to implement. Stable sort.
- **Disadvantages:** Very slow ($O(n^2)$ in all but the best case). Inefficient for large datasets.
- **Best Use-Case:** Only useful for small datasets or as an introductory sorting algorithm in education.

1.2 Selection Sort

- **Advantages:** Performs a minimal number of swaps, making it suitable for scenarios with costly swapping operations.
- **Disadvantages:** Always $O(n^2)$ time complexity, even if the array is already sorted.
- **Best Use-Case:** Useful when swapping is expensive, such as sorting files with large records but small keys.

1.3 Insertion Sort

- **Advantages:** Efficient for nearly sorted data (O(n) in the best case). Works well for small datasets. Stable.
- **Disadvantages:** Still O(n^2) in the worst case. Not suitable for large datasets.
- **Best Use-Case:** Great for online sorting (data arriving in real-time) and small datasets.

1.4 Merge Sort

- **Advantages:** O(n log n) complexity in all cases. Stable. Works well with linked lists.
- **Disadvantages:** Requires extra memory (O(n) space complexity). Slower for small arrays.
- **Best Use-Case:** Ideal for sorting linked lists and external sorting (e.g., sorting large files stored on disk).

1.5 Heap Sort

- **Advantages:** Always O(n log n) time complexity. Works efficiently even with large datasets.
- **Disadvantages:** Not stable. More comparisons than Quick Sort.
- **Best Use-Case:** Suitable for priority queues and scenarios where worst-case performance guarantees are needed.

1.6 Quick Sort

- **Advantages:** One of the fastest sorting algorithms in practice. O(n log n) on average. In-place sorting (low memory usage).
- **Disadvantages:** Worst case is O(n^2) (when the pivot is poorly chosen). Not stable.
- **Best Use-Case:** General-purpose sorting when speed is a priority, such as database sorting and competitive programming.

1.7 Radix Sort

- **Advantages:** O(nk) time complexity (faster than comparison-based sorts for large numbers). Stable.
- **Disadvantages:** Only works for numbers and requires extra memory. Less flexible than other sorts.
- **Best Use-Case:** Used for sorting integers, phone numbers, IP addresses, and other digit-based data efficiently.

2 Programming

2.1 Algorithms

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

2.2 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
```

2.2.1 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.

2.2.2 Data Size

Examine the selected sorting algorithms on data of the following sizes: 10,000, 30,000, 50,000, 100,000 elements.

3 Submission

3.1 Submission regulations

Create the folder <MSSV1_MSSV2> 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** and name it following your MSSV (Ex: *"MSSV1_MSSV2.zip"*). Submission that violates any regulation will get no credit (zero).

3.2 Output specification

3.2.1 Input arguments

Algorithm name

- Algorithm name: Lowercase, words are connected by "-" (Ex: selection-sort, insertion-sort, ...)
- Option -a (Ex: ./main.exe -a selection-sort)

Input file

- Option -i (Ex: ./main.exe -i input.txt)

```
// input.txt
10
3 4 5 6 7 8 9 2 1 10
```

Output file

- Option -o (Ex: ./main.exe -o output.txt)

```
// output.txt
10
1 2 3 4 5 6 7 8 9 10
```

Example command: ./main.exe -a merge-sort -i ./input.txt -o ./output.txt

3.2.2 Report

- There will be four **LINE GRAPHS** (Sorted data , Nearly sorted data, Reverse sorted data, and Randomized data), 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.
- 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.

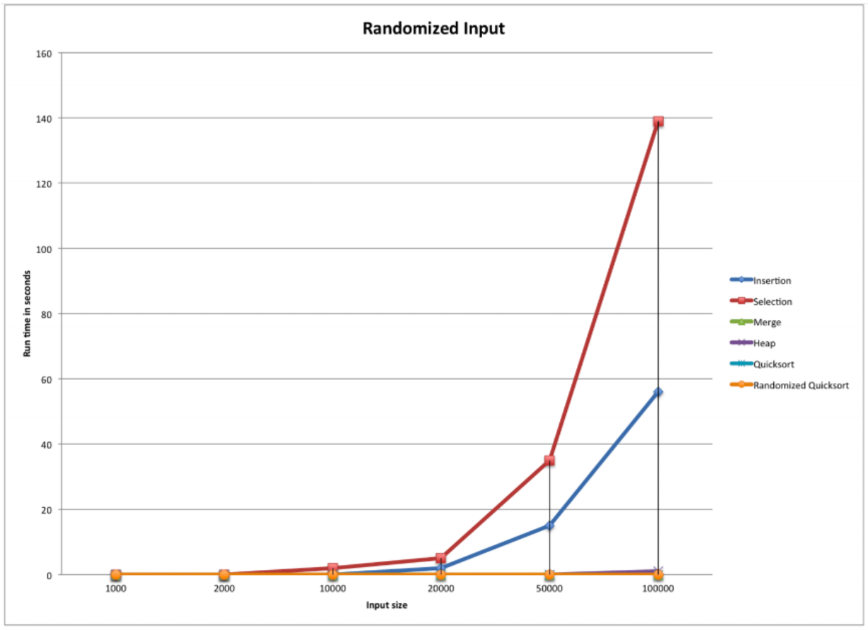

Figure 1:

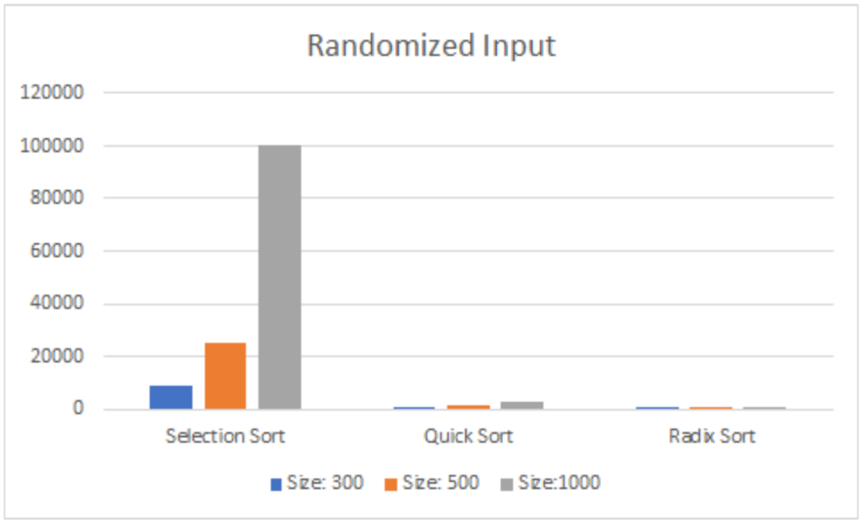

Figure 2: