CPTS 223 Advanced Data Structure C/C++ Fall 2024

PA4: Sorting

1 Learning Objectives

At the conclusion of this programming assignment, participants should be able to:

- Develop a C++ program that applies hash table data structure;
- Design and implement three sorting algorithms: insertion sort, quick sort and merge sort;
- Comparing the actual running time of three sorting algorithms.

2 Prerequisite

Before starting this programming assignment, participants should be able to:

- Apply and implement class templates;
- Design, implement, and test medium programs in C++;
- Cross-platform compilation with CMake;
- Maintain coding projects with Git on github.
- Finish reading §7.1, 7.2, 7.3, 7.6, 7.7 and 7.8 of Textbook.

3 Overview and Requirements

3.1 Overview

In this assignment, you will implement three important sorting algorithms: *Insertion Sort*, *Quick Sort*, and *Merge Sort*. Each of them represents a different approach to organizing data efficiently.

- *Insertion Sort* is a simple and intuitive algorithm that works by gradually building a sorted portion of the array. For each element, it is inserted into its correct position in the sorted part by shifting larger elements to the right. Although it has a worst-case time complexity of $O(n^2)$, it performs well on small or nearly sorted datasets, making it valuable in scenarios where data is partially ordered or input size is small.
- *Quick Sort* is a highly efficient, divide-and-conquer algorithm that works by partitioning the array around a pivot element, sorting elements around it so that those smaller are on the left and those larger are on the right. This process is repeated recursively on each partition until the entire array is sorted. With an average-case complexity of $O(n \log(n))$, it is one of the fastest algorithms for large datasets. However, its worst-case performance can degrade to $O(n^2)$, if the pivot selection is poor, which is often mitigated by using randomized or median-based pivot strategies.

- *Merge Sort* is another divide-and-conquer algorithm, which recursively splits the array into halves until each subarray has a single element. It then merges these sorted sub-arrays back together to form a fully sorted array. With a consistent time complexity of $O(n \log(n))$ for both average and worst cases, it is highly reliable for large datasets and particularly useful for applications requiring stable sorting. However, it requires additional memory space for merging, making it slightly less space-efficient than Quick Sort.

By implementing and comparing these algorithms, you will learn not only the mechanics of each approach but also gain insights into their performance characteristics across different data sizes. This practical exercise provides a hands-on exploration of algorithmic efficiency, helping students appreciate the trade-offs between time complexity, memory usage, and real-world applicability. The assignment culminates in a performance analysis section, where students will observe how these algorithms handle small, medium, and large datasets, allowing them to make informed decisions about algorithm choice in various scenarios.

3.2 Requirements

In this project, you will implement functions for three sorting algorithms (Insertion Sort, Quick Sort and Merge Sort). Our goal is to compare their running time on datasets in different scales (we will sort $1000, 10000, 100000$ random integer numbers, respectively).

The starter code provides an implementation of the class `SortingComparison` that includes the implementations for comparing the running time of the three sorting algorithms on different scales of random datasets. The implementations for the three sorting algorithms are missing and you will finish them in this assignment:

```
// insrtion sort
void insertionSort(std::vector<int>& arr) {
    // TODO
}

// quick sort
void quickSort(std::vector<int>& arr, int left, int right) {
    // TODO
}

// merge sort
void mergeSort(std::vector<int>& arr, int left, int right) {
    // TODO
}
```

You are recommended to refer to Textbook to finish the implementations of the three sorting algorithms:

- Insertion Sort (20 pts): Figure 7.2, a simple insertion sort implementation.
- Quick Sort (40 pts): Figure 7.13, 7.15, 7.16 and 7.17. You may choose to use `median3()` optionally to determine the pivot.
- Merge Sort (40 pts): Figure 7.11 and 7.12.

The starter code also includes three functions to help analyze the running time by the three sorting algorithms:

- a function to generate random integers `generateRandomArray()`.
- a function to test if the array has been sorted `isSorted()`.
- a function to perform this anlaysis for running time comparison `compareSortingAlgorithms()`.

Figure 1 gives the results of running the starter code on left and the result of running the final code on right.

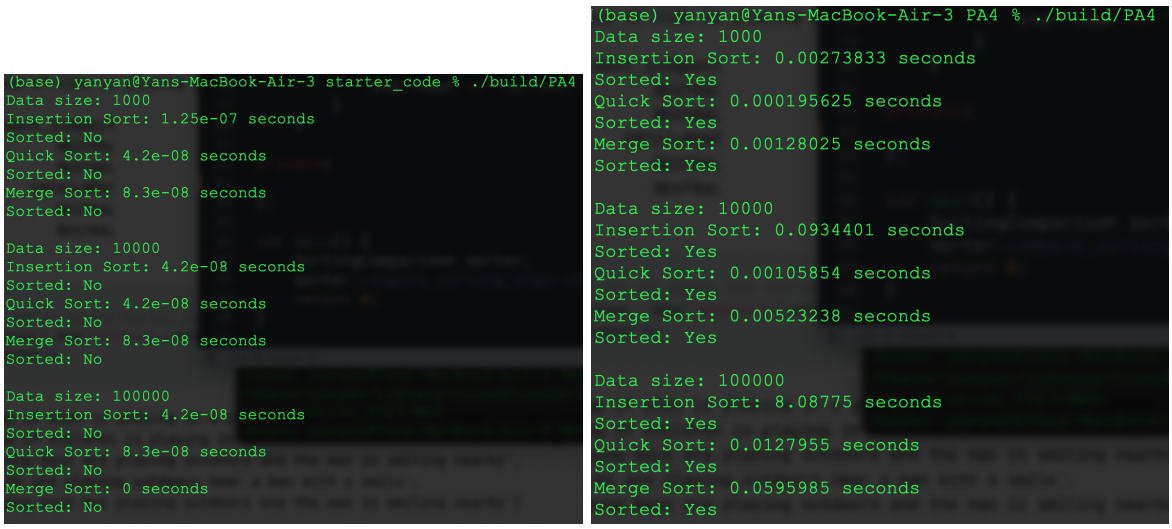

Figure 1: Example of running the starter code and final code.

4 How to Submit: Github (and Share with TA)

1. In your Git repository for this class's coding assignments, **create a new branch called "PA4"** (Refer to this YouTube video about how to create a branch from Github web interface or the terminal). In the current working directory, also create a new directory called "PA4" and place all PA4 files in the "PA4" directory. All files for PA4 should be added, committed and pushed to the remote origin which is your private GitHub repository created when during PA1 (NO NEED TO CREATE A NEW REPO).

2. You should submit at least the following files:

   - the header files if any;
   - the main C++ source file ("main.cpp"), where you will have all running and tests passed;
   - a "CMakeLists.txt" file containing your CMake building commands on Linux/WSL/MacOS which can compile your code.

3. You can refer the example GitHub template project for how to use CMake at `https://github.com/DataOceanLab/CPTS-223-Examples`.

4. Please invite the GitHub accounts of TAs (see Syllabus page and check TA's names as well as their Github usernames before submitting) as the collaborators of your repository. You should submit a URL link to the branch of your private GitHub repository on Canvas. Otherwise, we will not be able to see your repository and grade your submission.

5. Please push all commits of this branch before the due date for submitting your Github link to Canvas portal. Otherwise it might be considered as late submission.

Here is a checklist for submitting your code:

- Invite all TAs and instructor as collaborators of your created repository "CPTS223_assignments". Their github username can be found in Syllabus on Canvas.
- Commit and push your local code to your repository.
- Make sure that your repository reflects and contains all your latest files.
- Copy the link to your repository in the Canvas submission portal.

5 Grading Guidelines

This assignment is worth 100 points. We will grade according to the following criteria: See Section 3.2 for individual points.