

Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7205: Fundamentals of Computer Engineering

Spring 2022 - Homework 4

Instructions

- For programming problems:
 - Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
 - You are **not** allowed to use any advanced C++ library unless it is clearly allowed by the problem. For example, you cannot use a library function to sort a list of data if the problem is asking you to implement an algorithm to sort the list.
 - At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any).
 - Your code must compile and tested with a standard GCC compiler (available in the CoE Linux server). before submitting the source code file(s) (do not submit any compiled/executable files):
 - a. If your program does not compile with a GCC compiler due to incompatible text encoding format, then make sure the program is saved with Encoding Unicode (UTF-8). In visual studio, Save As -> Click on the arrow next to Save -> Save with Encoding -> Yes -> Unicode (UTF-8) -> Ok
 - b. Compile using `g++ -std=c++11 <filename>`
- Submit the following to the homework assignment page on Canvas:
 - Your homework report developed by a word processor and submitted as one PDF file. For answers that require drawing and if it is difficult on you to use a drawing application, which is preferred, you can neatly hand draw the answer, scan it, and include it into your report. The report includes the following (depending on the assignment contents):
 - a. Answers to the non-programming problems that show all the details of the steps you follow to reach these answers.
 - b. A summary of your approach to solve the programming problems.
 - c. The screen shots of the sample run(s) of your program(s).
 - Your well-commented programs source code files (i.e., the .cc or .cpp files).

Do NOT submit any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.

Note: You can submit multiple attempts for this homework, however, only what you submit in the last attempt will be graded. This means all required files must be included in this last submission attempt.

Problem 1 (100 Points)

In this problem you will test and analyze the Insertion, Heap, and Quick sort algorithms. To carry out this task, write a C++ program with the following requirements:

1. Analyze the algorithms by sorting, in ascending order, arrays of 1000 integers.
2. Create three arrays, BST, AVG, and WST, each has 1000 integers as follows:
 - a. BST has 1000 integers already sorted in ascending order (e.g., 10, 20, 30, ...etc.).
 - b. AVG has 1000 randomly generated integers, where each integer is between 0 and 100,000.
 - c. WST has 1000 integers sorted in descending order (e.g., 1000, 990, 980, ...etc.).
3. To test all algorithms with identical arrays, test each algorithm with the following 1000-integer arrays: tBST, tAVG, and tWST. Make sure that before testing each algorithm, these three arrays (tBST, tAVG, and tWST) are re-initialize as copies from BST, AVG, and WST respectively (you can write a function to carry out the copy task).
4. Define two global integer variables: moves and comps. Initialize these variables to zero before every call to a sorting algorithm. Add any necessary code to the algorithms' implementation to carry out the following:
 - a. Increment moves with every movement of any of the elements to be sorted from one "place" to another. Consider that the swap operation requires three movements of two elements. However, the following is not considered a move: making a "copy" of an element that will not result eventually of that element being copied to another location in the array (i.e., the element stays in its original place when it was copied). For example, in Insertion sort algorithm the copy operation "key \leftarrow A[j]" does not increment "moves" as that A[j] might ended up staying in its place (i.e., not moved). Only if we need to move that A[j] to another location in the array, then "moves" needs to be incremented,
 - b. Increment comps with each comparison operation on the elements to be sorted (not their indices).
 - c. After each call to the sorting algorithms functions (3 algorithms \times 3 arrays = 9 calls), write to a text file, sort.txt, the values of moves and comps correspond to each algorithm.
5. To ensure that the algorithms are implemented correctly, you need to verify that the arrays are sorted after calling each algorithm. One way to do that is to confirm that every element i in the array has a value less than or equal to the value of element $i+1$ (write a function for this task and re-check your code if it returns a false verification).
6. After running your program, copy the numbers in the text file, sort.txt, to an excel sheet. Use these numbers to generate two graphs. One graph compares the number of moves needed by the three algorithms for the three cases: best, average, and worst. Another graph does the same but for the number of comparisons performed by the algorithms.

In your homework report, answer the following questions:

- i. Why does the Insertion sort algorithm result in zero moves when sorting an already sorted array (the best case)?
- ii. Why does the Insertion sort algorithm result in 999 comparisons when sorting an already sorted array (the best case)?

- iii. Based on the two excel graphs, how does each algorithm perform under different scenarios (best, average, and worst)?

Programming Hints

- The following C++ code show you how to create a text file and write a text message in it:

```
include <iostream>
#include <fstream>
#include <stdexcept>

using namespace std;

int main() {
    ofstream outf;

    outf.open("sort.txt");
    if (outf.fail()) {
        cerr << "Error: Could not open output file\n";
        exit(1);
    }

    outf << "\t\t Hello World \n";

    outf.close(); //Close the file at the end of your program.
    return 0;
}
```

- The following C++ code show you how to generate 5 random integers. Each integer is between 0 and 100.

```
#include <chrono>

int main() {
    int A[5];

    srand(time(NULL)); //call this only once at the beginning to
                        // allow rand() to generate a different
                        // succession of random values.

    for (int i = 0; i < 5 ; i++)
        A[i] = rand() % 100;

    return 0;
}
```