Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7205: Fundamentals of Computer Engineering

Fall 2019 - Homework 4

Instructions

- For programming problems:
  - Your code must compile and run on the COE Linux server before submitting it on Blackboard.
  - 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.
  - At the beginning of your source code files write your full name, students ID , and any special compiling/running instruction (if any).

- Submit the following to the homework assignment page on Blackboard:
  - Your homework report submitted as one PDF file. The report includes the answers to the non-programming problems and the screen shots of your program's sample runs for the programming problems. Your report must be developed by a word processor (no hand written or drawn contents are acceptable).
  - Your well-commented source code file(s) for the programming problems.
  - Do NOT submit your files (the PDF and source code) as a compressed (zipped) package. Rather, upload each file individually.

Note: Yon can submit multiple attempts for this homework, however, only your last submitted attempt will be graded.

Problem 1 (100 Points)

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

1. Analyze the algorithms by sorting, in ascending order, the BST, AVG, and WST arrays. Each one of these three arrays has 1000 integers where:
   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.).

   Note: Make sure that all algorithms are tested with identical arrays. Therefore, at the beginning of your program, create three identical versions of the BST, AVG, and WST arrays where each algorithm sorts its own version.

2. 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. `moves` is incremented with each movement operation of the elements to be sorted (in this case they are the integer values in the arrays). Consider that the swap operation requires three movements of two elements.
   b. `comps` is incremented with each comparison operation on the elements to be sorted.
   c. After each call to the sorting algorithms functions (2 algorithms × 3 arrays = 6 calls), write to a text file, `sort.txt`, the values of `moves` and `comps` correspond to each algorithm.
3. After calling an algorithm function to sort an array, verify that the array is sorted. 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 in element *i*+1 (write a function for this task and re-check your code if it returns a false verification).
4. 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:

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. From your two excel graphs, comment on the performance of the sort algorithms under different scenarios (best, average, and worst).

Programming Hints

- The following C++ code creates a text file and writes 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 generates 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;
}
```