Data Structures and Algorithms Lab

Lab 07 **Marks 05**

Instructions

Work on this lab individually. You can use your books, notes, handouts etc. but you are not allowed to borrow anything from your peer student.

Marking Criteria

Show your work and **tabulate your results** in the provided **Answer Sheet** and **submit** it to the instructor before leaving the lab to get some or full credit.

What you must do

Evaluate the **performance** of three **sorting algorithms** (Bubble, Selection, and Insertion) that we have discussed in class. The standard implementation of these algorithms is given in source file ***sort.cpp*** added in this lab folder.

You are going to analyze the **performance** of above algorithms on different types and sizes of data sets. There are three types of input files:

1. Random
2. Sorted (in ascending order)
3. Sorted (in descending order)

Different sizes of data sets are as follows:

1. 500 elements
2. 6000 elements
3. …
4. 20000 elements

Apart from the running time of sorting algorithms, you are also required to determine **Number of key comparisons** and **Number of data movements** for each case.

The following code snippet (*included in sort.cpp*) should give you an idea of how to determine the **running time** of some operation:

```
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
	clock_t startTime = clock();

	// Perform some operation here

	clock_t endTime = clock();
	double elapse = (double)(endTime-startTime)/CLOCKS_PER_SEC;
	cout << "The operation took " << elapse << " seconds";

	return 0;
}
```

Input files of different data sizes are given with this lab folder. There are **3 versions** of each input file. For example, the file **in500.txt** contains **500 integers** in a random order, **in500a.txt** contains those **500 integers** sorted in ascending order, and **in500d.txt** contains those **500 integers** sorted in descending order. Note that the **first number** in each file shows the **total number of elements** present in that file.

Keep the following things in mind when measuring the running time of a sorting algorithm:

1. You should display the running time, **No. of key comparisons**, and **No. of data movements** at the end of the function. Apart from this, there should be **NO input/output** done inside the function.
2. To get meaningful running times, make sure that there are no other **applications/programs** running at the time of the execution of your program.
3. If the running time of some algorithm on some input is insignificant (i.e. 0 seconds), repeat that process multiple times (in a loop). At the end, divide the running time by the number of times that process was repeated.

☺ ☺ ☺ **BEST OF LUCK** ☺ ☺ ☺