CS 202 Fundamental Structures of Computer Science II
Assignment 1 – Algorithm Efficiency and Sorting

Assigned on: 17 February 2016 (Wednesday)
Due Date: 29 February 2016 (Monday)

Question-1: Tracing (20 points)

Trace the following sorting algorithms to sort the array [9 8 2 5 7 4] into ascending order. Use the array implementation of the algorithms as described in the textbook.

a) Insertion sort.
b) Selection sort.
c) Bubble sort.
d) Merge sort; also list the calls to **`mergesort`** and **`merge`** in the order they occur.
e) Quick sort; also list the calls to **`quicksort`** and **`partition`** in the order they occur. Assume that the last item is chosen as pivot.

Question-2: Programming Assignment (60 points)

You are asked to implement the **bubble sort**, **merge sort**, and **quick sort** algorithms for *an array of integers* and then perform the measurements as detailed below.

1. For each algorithm, implement the functions that take an array of integers and the size of the array and then sort it in ***ascending order***. Add a counter to count the number of key comparisons and the number of data moves in sorting.
2. For the quick sort algorithm, you are supposed to take **the last element** of the array as pivot.
3. Write a main function to measure the time required by each sorting algorithm. To this end, use the library function `clock()`, which is defined in time.h. Invoke the `clock()` library function before and after each sorting algorithm to measure the elapsed time in milliseconds.
4. Although you will write your own main function to get the experimental results, we will also write our own main function to test whether or not your algorithms work correctly. In our main function, we call your sorting algorithms with the following prototypes.

```
void bubbleSort( int *arr, int size, int &compCount, int &moveCount);
void mergeSort( int *arr, int size, int &compCount, int &moveCount);
void quickSort( int *arr, int size, int &compCount, int &moveCount);
```

In all of these prototypes, **`arr`** is the array that the algorithm will sort, **`size`** is the array size, **`compCount`** is the number of key comparisons in sorting, and **`moveCount`** is the number of data moves in sorting. After returning from this function, **`arr`** should become sorted.

For key comparisons, you should count each comparison like "k1 < k2" as one comparison, where k1 and k2 correspond to the value of an array entry (that is, they are either an array entry like arr[i] or a local variable that temporarily keeps the value of an array entry).

For data moves, you should count each assignment as one move, where either the right-hand side of this assignment or its left-hand side or both of its sides correspond to the value of an array entry. For example, the following swap function has three such assignments (and thus three data moves)

```
void swap(DataType &x, DataType &y) {
   DataType temp = x;
   x = y;
   y = temp;
}
```

5. Put the implementations of these functions in **`sorting.cpp`**, and their interfaces in **`sorting.h`**. Do not include your main function in these files. Submit your main function inside a file, called **`main.cpp`**.
6. ***You will lose a significant amount of points if you do not comply with these naming conventions.***

After implementing the sorting algorithms,

1. Create three identical arrays with ***random 10,000 integers*** using the random number generator function `rand`. Use one of the arrays for the bubble sort, another one for the merge sort, and the last one for the quick sort algorithm. Output the number of key comparisons, the number of data moves, and the elapsed time to sort these integers using each of these algorithms. Repeat this experiment for at least 10 different input sizes that are greater than 10,000 (for instance from 10 000 to 100 000, incrementing the sizes by 10 000). With the help of a graphical plotting tool, present your experimental results graphically. Note that plot the number of key comparisons, the number of data moves, and the elapsed time in different figures.
2. Then, create three identical copies of an array ***with 10,000 integers that are sorted in descending order***. Use each array for each algorithm. Repeat all of the experiments and present your experimental results graphically. (That is, for each algorithm, create arrays with at least 10 different input sizes and output the number of key comparisons, the number of data moves, and the elapsed time to sort these arrays and present your results graphically.)
3. Lastly, create three identical copies of an array ***with 10,000 integers that are sorted in ascending order***. Use each array for each algorithm. Repeat all of the experiments (again by creating arrays with at least 10 different input sizes) and present your experimental results graphically.

Question-3 (5 points)

Interpret your experimental results that you obtained in Question-2. Compare these results with the theoretical ones for each sorting algorithm. Explain any differences between the experimental and theoretical results.

Question-4: Asymptotic Analysis and Growth-Rate Functions (15 points)

1. Prove that $\log n! = O(n \log n)$
2. Show that the solution of the following recurrence relation is $T(n) = O(n \log n)$. Use the repeated substitution technique in your solution.

   $T(n) = 2T(n/2) + O(n)$, for $n \geq 2$

   $T(1) = O(1)$
3. Show that $f(n) = 4n^5 + 3n^2 + 1$ is $O(n^5)$ by specifying appropriate *c* and n_0 values in Big-O definition.

HAND-IN

- Before 23:59 of February 29, 2016, upload your solutions using the following online submission form, http://www.cs.bilkent.edu.tr/~saksoy/courses/cs202-Spring2016/upload.html. You should upload a single zip file that contains
  - hw1.pdf, the file containing the answers to Questions 1, 3, and 4, and the graphical findings of the experiments for Question 2,
  - sorting.cpp, sorting.h, and main.cpp, the files containing the C++ source code, and
  - readme.txt, the file containing anything important on the compilation and execution of your program in Question 2.
  - **You should prepare the answers of Questions 1, 3, and 4 using a word processor (in other words, do not submit images of handwritten answers).**
  - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities (for example to measure the time).
  - Do not forget to put your name, student id, and section number, in all of these files. Well comment your implementation.
- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work in a Linux environment (specifically using the g++ compiler). We will compile your programs with the g++ compiler and test your codes in a Linux environment. Thus, you may lose a significant amount of points, if your C++ code does not compile or execute in a Linux environment.
- Keep all the files before you receive your grade.
- This homework will be graded by your TA, Cem Orhan (cem.orhan at bilkent edu tr). Thus, you may ask your homework related questions directly to him.

IMPORTANT: For this assignment, you may use the codes given in your textbook and the slides. However, you are NOT allowed to use any codes from other sources (including the codes given in other textbooks, found on the internet, and belonging to your class mates). Furthermore, you are NOT allowed to use any data structure or function from the C++ standard template library (STL).

Do not forget plagiarism and cheating will be heavily punished. Please do the homework yourself.