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

Assigned on: 2 October 2015 (Friday)
Due Date: 16 October 2015 (Friday)

Question-1: Tracing(20 points)

Trace the following sorting algorithms to sort the array `[25,9,3,7,11,0,2,20,28,5,16]` into **ascending** order. Use the array implementation **exactly** 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 (45 points)

Programming Assignment -- You are asked to implement the **selection 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 counters to count the number of key comparisons and the number of data moves during sorting.
2. For the quick sort algorithm, you are supposed to take the first element of the array as the 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 will call your sorting algorithms with the following prototypes.

```
void insertionSort( 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 this function, **`arr`** should become sorted.

For counting key comparisons, you should count each comparison like "a<b" as 1 comparison. For counting number of moves, you should count each assignment as 1 move, for example, swap method:

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

has 3 moves. If you shift left the following array: [1,2,3]->[2,3,1], this operation has 4 moves.

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 separate 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 20,000 integers*** using the random number generator function `rand`. Use one of the arrays for the selection 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 5 different input sizes that are greater than or equal to 20,000 (for instance, 20 000, 30 000, 40 000, 50 000, 60 000). 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 20,000 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 5 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 20,000 integers that are sorted in ascending order***. Use each array for each algorithm. Repeat all of the experiments and present your experimental results graphically.

Question-3: Interpretation (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: Algorithm Analysis (15 points)

1. Write a recursive algorithm called **`findMin`** for finding the minimum element of a given arbitrary array of integers. You may assume a **`min(a,b)`** method that returns the minimum of input integers a and b.
2. Prove that your algorithm works correct.
3. Compute the complexity (in terms of comparisons) of your algorithm with recurrence relations.

Question-5: Asymptotic Analysis and Growth-Rate Functions (15 points)

1. Prove that $\log(n!) = O(n \log n)$
2. Prove that the solution for the recurrence relation

$$T(1) = 0$$
$$T(n) = 2T\left(\frac{n}{2}\right) + n\,, n \geq 2$$

   is

$$T(n) = n \log n$$

   Hint: Use induction.
3. Show that $f(n) = 4n^5 + 3n^2 + 1$ is order of $O(n^5)$ by giving appropriate c and n_0 values.

HAND-IN

- Before 23:59 of October 16, 2015, upload your solutions to Moodle. You should upload a single **zip** file that contains
  - **hw1.pdf**, the file containing the answers to questions 1, 3, 4 and 5, the sample output of the program, and the graphical findings of the experiments for Question 2.
  - **sorting.cpp**, **sorting.h**, and **main.cpp**, the files containing the C++ source code
  - **readme.txt**, the file containing anything important on the compilation and execution of your program in question 2.
  - Do not forget to put your name, student id, and section number, in all of these files. Well comment your implementation.
  - For questions 1, 3, 4 and 5, your solutions should be **typed** (you may use a word processor like MS Word and convert it to **pdf** file or you may use Latex). **Do not submit scans or photographs** of your solutions, you will get an immediate 0 (zero). For sample output of question 2, you should put your screen outputs as text, **no** screenshots.
  - Do not put any unnecessary files such as the files generated from your favorite IDE, and name your zip file as follows: "**SECTION_ID_NAME_SURNAME.zip**"(without quotations); any violation of these will cause a significant point deduction from your grade.
  - 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: Although you may use any platform and any operating system in implementing your algorithms and obtaining your experimental results, your code should work in a Linux environment with the g++ compiler. We will 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.**

DO THE HOMEWORK YOURSELF. PLAGIARISM AND CHEATING ARE HEAVILY PUNISHED!!!