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

Assigned on: 10 October 2016 (Monday)
Due Date: 24 October 2016 (Monday) – 23:55

Question-1 (30 points)

Trace the following sorting algorithms to sort the array `[7, 5, 1, 2, 8, 3, 9, 4]` into ***ascending order***. Use the array implementation 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 (60 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 ***non-ascending (descending) order*** Add counter to count the number of operations during sorting. You may take each comparison as one operation.
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 `clock()` library function. (include `<ctime>`) Invoke the `clock()` library function before and after each sorting algorithm to measure the elapsed time in milliseconds. Then take the difference between these two returned values. The result is the time requirement for your function in clock ticks.

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 SelectionSort ( int * theArray, int n ) ;
void QuickSort ( int * theArray, int n ) ;
void MergeSort ( int * theArray, int n ) ;
```

In all of these prototypes, **`theArray`** is the array that the algorithm will sort, **`n`** is the array size. After returning this function, **`theArray`** should become sorted.

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 25,000 integers*** using the random number generator function `rand`. (Refer this array R25K) 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 operations and the elapsed time to sort these integers using each of these algorithms. Repeat this experiment for input sizes 100,000 and 500,000. (Refer these arrays R100K and R500K) With the help of a graphical plotting tool, present your experimental results graphically. Note that plot the number of operations, and the elapsed time in different figures.
2. Then, create three identical copies of an array ***with 25,000 integers that are sorted in descending order***. (e.g., 25000, 24999, 24998, . . . , 1). Refer this array D25K. 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 input sizes input sizes 100,000 and 500,000, refer these arrays D100K and D500K, and output the number of operations and the elapsed time to sort these arrays and present your results graphically.)
3. Lastly, create three identical copies of an array ***with 25,000 integers that are sorted in ascending order***. (e.g., 1, 2, 3, 4, . . . , 25000) Refer this array A25K. 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 input sizes input sizes 100,000 and 500,000, refer these arrays A100K and A500K, and output the number of operations and the elapsed time to sort these arrays and present your results graphically.)

Question-3 (10 points)

After running your programs, you are expected to prepare a 2-3 page report about your experimental results that you obtained in Question-2. Particularly, with the help of a spreadsheet program (Microsoft Excel, Matlab or other tools), present your experimental results graphically. Interpret and compare your empirical results with the theoretical ones for each sorting algorithm. Explain any differences between the empirical and theoretical results, if any.

IMPORTANT NOTES

Please do not start the assignment before reading these notes.

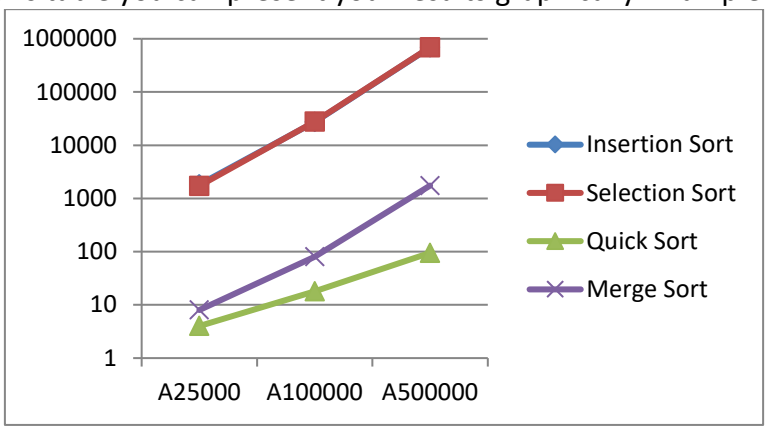
- Before 23:55 on due date as specified above upload your solutions to the moodle.
- The name of the archive file must be in the following format.

  `Surname_Name_StudentId_Section_HW1.zip`

- You should upload a single zip file that contains
  - hw1.pdf, the file containing the answers to Questions 1 and 3, 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, and
  - 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.
- You are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).
- To show and compare the results for the number of operations and the elapsed time to sort first you are expected to make a table as following:

Input	Selection Sort	Merge Sort	Quick Sort
A25K			
A100K			
A500K			
D25K			
D100K			
D500K			
R25K			
R100K			
R500K			

Then by using this table you can present your results graphically. Example:



- **<u>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.</u>**
- Keep all the files before you receive your grade.
- To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.
  - Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your main file. Example:

    ```
    //-------------------------------------------------------
    // Title: Algorithm Analysis and Sorting Program
    // Author: Miray Ayşen
    // ID: 21000000
    // Section: 0
    // Assignment: 1
    // Description: This program is for comparing different sorting algorithms for
    different scenarios.
    //-------------------------------------------------------
    ```

  - Since your codes will be checked without your observation, you should report everything about your implementation. Well comment your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

    ```
    void SelectionSort(int *theArray, int n)
    //-------------------------------------------------------
    // Summary: Sorts array items into descending order.
    // Precondition: theArray is an array of n integers.
    // Postcondition: theArray is sorted into descending
    // order, n is unchanged.
    ```

```
//--------------------------------------------------
{
// body of the function
}
```

- Be careful about indentation.
- Pay attention to these instructions, otherwise you may lose some points even though your code has no error.
- This homework will be graded by your TA, Miray Ayşen (miray.aysen at bilkent edu tr). Thus, you may ask your homework related questions directly to her.

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