CS 202: Fundamental Structures of Computer Sciences II

Assignment 1

Due: 23:59, March 10 (Monday), 2014

Part 1

In this assignment, your aim is to implement the following sorting algorithms for integer arrays, and to empirically evaluate them. Moreover, your program has to meet some specific requirements, which are going to be briefly described hereafter. First, write the functions for the following sorting algorithms. Make sure that your functions sort a given integer array (`theArray`) with a size of `n` in descending order.

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

Next, write a function to find the time requirements of your sorting functions for different integer arrays. You have to create 9 different arrays and you have to find the time requirements of each of your functions for each of these arrays. The arrays that you have to create:

- 3 arrays with sizes of 25000, 100000 and 500000. Fill each array with integers in the ascending order (e.g., 1, 2, 3, 4, ..., 25000). Refer these arrays A25000, A100000, and A500000.
- 3 arrays with sizes of 25000, 100000 and 500000. Fill each array with integers in the descending order (e.g., 25000, 24999, 24998, ..., 1). Refer these arrays D25000, D100000, and D500000.
- 3 arrays with sizes of 25000, 100000 and 500000. Fill each array with random integers using the random number generator `rand`. Refer these arrays R25000, R100000, and R500000.

You are going to measure the time requirement of your function in two ways. First, use the `clock()` library function (include `<ctime>`). You should invoke the `clock()` function before and after the invocation of your sorting function. Then take the difference between these two returned values. The result is the time requirement for your function in clock ticks . Second, count the operations in your sorting algorithms to find the execution complexity.

After each invocation show your results. You may take each comparison as one operation. You also have to compare the results and present the user, the best and the worst algorithm in different array sizes and scenarios. To sum up, your program should display 36 time requirements (9 for each sorting algorithm) for each measurement method.

Sample Output:

When the program is run, it should ask the user what he/she wants to do. Prepare a main menu with 3 choices, two for the time measurement methods and the other for ending the program. Your main menu should look like this:

```
Please make a selection:
1. Measure with the clock() function
2. Measure by counting operations
3. Exit
```

The output of the first selection should be like as follows. Note that your program should fill "`...`" with the time requirements that it finds and "`???`" with the name of the best or worst sorting algorithm. Make sure you follow the same format in your output.

```

It takes ... clock ticks to sort A25000 by insertion sort
It takes ... clock ticks to sort A25000 by selection sort
It takes ... clock ticks to sort A25000 by quick sort
It takes ... clock ticks to sort A25000 by merge sort

It takes ... clock ticks to sort A100000 by insertion sort
It takes ... clock ticks to sort A100000 by selection sort
It takes ... clock ticks to sort A100000 by quick sort
It takes ... clock ticks to sort A100000 by merge sort

It takes ... clock ticks to sort A500000 by insertion sort
It takes ... clock ticks to sort A500000 by selection sort
It takes ... clock ticks to sort A500000 by quick sort
It takes ... clock ticks to sort A500000 by merge sort

For A25000, the best is ??? sort, and the worst is ??? sort.
For A100000, the best is ??? sort, and the worst is ??? sort.
For A500000, the best is ??? sort, and the worst is ??? sort.

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-

It takes ... clock ticks to sort D25000 by insertion sort
It takes ... clock ticks to sort D25000 by selection sort
It takes ... clock ticks to sort D25000 by quick sort
It takes ... clock ticks to sort D25000 by merge sort

It takes ... clock ticks to sort D100000 by insertion sort
It takes ... clock ticks to sort D100000 by selection sort
It takes ... clock ticks to sort D100000 by quick sort
It takes ... clock ticks to sort D100000 by merge sort

It takes ... clock ticks to sort D500000 by insertion sort
It takes ... clock ticks to sort D500000 by selection sort
It takes ... clock ticks to sort D500000 by quick sort
It takes ... clock ticks to sort D500000 by merge sort

For D25000, the best is ??? sort, and the worst is ??? sort.
For D100000, the best is ??? sort, and the worst is ??? sort.
For D500000, the best is ??? sort, and the worst is ??? sort.

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-

It takes ... clock ticks to sort R25000 by insertion sort
It takes ... clock ticks to sort R25000 by selection sort
It takes ... clock ticks to sort R25000 by quick sort
It takes ... clock ticks to sort R25000 by merge sort

It takes ... clock ticks to sort R100000 by insertion sort
It takes ... clock ticks to sort R100000 by selection sort
It takes ... clock ticks to sort R100000 by quick sort
It takes ... clock ticks to sort R100000 by merge sort

It takes ... clock ticks to sort R500000 by insertion sort
It takes ... clock ticks to sort R500000 by selection sort
It takes ... clock ticks to sort R500000 by quick sort
It takes ... clock ticks to sort R500000 by merge sort
```

```
For R25000, the best is ??? sort, and the worst is ??? sort.
For R100000, the best is ??? sort, and the worst is ??? sort.
For R500000, the best is ??? sort, and the worst is ??? sort.

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
```

The output of the second selection should be like as follows. Note that your program should fill "..." with the time requirements that it finds and "..." with the name of the best or worst sorting algorithm. Make sure you follow the same format in your output.

```
... operations are done to sort A25000 by insertion sort
... operations are done to sort A25000 by selection sort
... operations are done sort A25000 by quick sort
... operations are done sort A25000 by merge sort

... operations are done to sort A100000 by insertion sort
... operations are done to sort A100000 by selection sort
... operations are done sort A100000 by quick sort
... operations are done sort A100000 by merge sort

... operations are done to sort A500000 by insertion sort
... operations are done to sort A500000 by selection sort
... operations are done sort A500000 by quick sort
... operations are done sort A500000 by merge sort

For A25000, the best is ??? sort, and the worst is ??? sort.
For A100000, the best is ??? sort, and the worst is ??? sort.
For A500000, the best is ??? sort, and the worst is ??? sort.

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-

... operations are done to sort D25000 by insertion sort
... operations are done to sort D25000 by selection sort
... operations are done sort D25000 by quick sort
... operations are done sort D25000 by merge sort

... operations are done to sort D100000 by insertion sort
... operations are done to sort D100000 by selection sort
... operations are done sort D100000 by quick sort
... operations are done sort D100000 by merge sort

... operations are done to sort D500000 by insertion sort
... operations are done to sort D500000 by selection sort
... operations are done sort D500000 by quick sort
... operations are done sort D500000 by merge sort

For D25000, the best is ??? sort, and the worst is ??? sort.
For D100000, the best is ??? sort, and the worst is ??? sort.
For D500000, the best is ??? sort, and the worst is ??? sort.

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-

... operations are done to sort R25000 by insertion sort
... operations are done to sort R25000 by selection sort
... operations are done sort R25000 by quick sort
... operations are done sort R25000 by merge sort

... operations are done to sort R100000 by insertion sort
... operations are done to sort R100000 by selection sort
... operations are done sort R100000 by quick sort
... operations are done sort R100000 by merge sort

... operations are done to sort R500000 by insertion sort
... operations are done to sort R500000 by selection sort
... operations are done sort R500000 by quick sort
... operations are done sort R500000 by merge sort

For R25000, the best is ??? sort, and the worst is ??? sort.
For R100000, the best is ??? sort, and the worst is ??? sort.
For R500000, the best is ??? sort, and the worst is ??? sort.

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
```

Part 2

After running your programs, you are expected to prepare a 2-3 page report about your findings. Particularly, with the help of a spreadsheet program (Microsoft Excel, Matlab or other tools), present your experimental results graphically. 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:

Do not start your homework before reading these notes!!!

1. This assignment is due by 23:59 on Monday, March 10th, 2014. The standard rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

2. In this assignment, you must submit a 2-3 page report (as a pdf file) that contains all information requested above and codes in a single archive file. The name of the archive file must be in the following format.

   `Surname_Name_StudentId_Section_HW1.zip`

   You should email your homework to your TA Can Fahrettin Koyuncu (`koyuncu at cs bilkent edu tr`) before the deadline. No hardcopy submission is needed.

3. This homework will be graded by Can Fahrettin Koyuncu. Thus, you may ask your homework related questions directly to him.

4. 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 ...).

5. We will test your programs on `dijkstra.ug.bcc.bilkent.edu.tr` and we will expect your programs to compile and run on the Dijkstra machine. If we could not get your program properly work on this machine, you would lose a considerable amount of points. Thus, you are required to get your program compile and properly work on this machine.

6. 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: Can Fahrettin Koyuncu
     // ID: 2100000000
     // Section: 0
     // Assignment: 1
     // Description: This program is for comparing different
     // sorting algorithms in 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
     }
     ```

   - Indentation, indentation, indentation...
   - Pay attention to these instructions, otherwise you may lose some points even though your code has no error.

Bonus Question (15 points)

Consider the following sorting algorithm, called Monkey Sort:

```
void MonkeySort(int *theArray, int n) {
        bool sorted = false;
        while(sorted == false) {
                for( int i = 0; i < n; i++) {
                        index = (rand() % (n - 1));
                        tmp = theArray[i];
                        theArray[i] = theArray[index];
                        theArray[index] = tmp;
                }
                sorted = true;
                for( int i = 0; i < n - 1; i++) {
                        if(theArray[i] >= theArray[i + 1]) {
                                sorted = false;
                                break;
                        }
                }
        }
}
```

Derive the theoretical best, average, and worst case performance bounds of the MonkeySort algorithm. The way you prove the answer is more important than the answer here.