CS 202, Summer 2022

Homework 1 – Algorithm Efficiency and Sorting

Due: 23:55, June 22, 2022

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, June 22, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_secNo_hw1.zip.
2. Your ZIP archive should contain the following files:
   - **`hw1.pdf`**, the file containing the answers to Questions 1, 2 and 3.
   - **`sorting.cpp`**, **`sorting.h`**, **`main.cpp`** files which contain the C++ source codes, and the **`Makefile`**.
   - Do not forget to put your name, student id, and section number in all of these files. Well comment your implementation. Add a header as given below to the beginning of each file:

```
/*
 * Title: Sorting and Algorithm Efficiency
 * Author: Name Surname
 * ID: 21000000
 * Section: 1
 * Assignment: 1
 * Description: description of your code
 */
```

   - 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).
   - You should prepare and upload **handwritten** answers for Question 1 (in other words, do not submit answers prepared using a word processor).
   - Use the exact algorithms shown in lectures.
3. Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the `dijkstra` server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. Thus, you will lose significant amount of points if your C++ code does not compile or execute on the `dijkstra` server.
4. This homework will be graded by your TA, Saeed Karimi. Thus, please contact him directly (saeed.karimi at bilkent edu tr) for any homework related questions.

Attention: For this assignment, 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 other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

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

Question 1 – 15 points

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

(b) [*10 points*] Trace the following sorting algorithms to sort the array [31, 17, 58, 35, 45, 27, 36, 28, 24, 34] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.

- Selection sort
- Insertion sort

Question 2 – 70 points

Implement the following functions in the `sorting.cpp` file:

(a) [*40 points*] Implement the bubble sort, merge sort, and quick sort algorithms. Your functions should take an array of integers and the size of that array and then sort it in ascending order. Add two counters to count and return the number of key comparisons and the number of data moves for all sorting algorithms. For the quick sort algorithm, you are supposed to take the first element of the array as pivot. Your functions should have the following prototypes (all prototypes should be in `sorting.h`):

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

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 (e.g., a swap function has three data moves).

(b) [*15 points*] To test your implementation and conduct the experiments described below, you must write additional auxiliary global functions as follows. The prototypes of these functions are given below. The first function displays the array items on the screen. The other ones are to create four identical arrays that will be used for testing the sorting algorithms with random numbers (generated using the random number generator function `rand`), numbers in ascending order, and numbers in descending order, respectively.

```
void displayArray(const int *arr, const int size);
void createRandomArrays(int *&arr1, int *&arr2, int *&arr3, int *&arr4, const int size);
void createAscendingArrays(int *&arr1, int *&arr2, int *&arr3, int *&arr4, const int size);
void createDescendingArrays(int *&arr1, int *&arr2, int *&arr3, int *&arr4, const int size);
```

(c) [*5 points,* ***mandatory***] Create a `main.cpp` file which does the following in order:

- creates an array from the following: {20, 15, 19, 26, 27, 17, 14, 22, 29, 11, 25, 28, 13, 21, 23, 16}.
- calls the `bubbleSort` function, displays the number of key comparisons and the number of data moves to sort this array, and calls the `displayArray` function to show the contents of the array after bubble sorting
- calls the `mergeSort` function, displays the number of key comparisons and the number of data moves to sort this array, and calls the `displayArray` function to show the contents of the array after merge sorting
- calls the `quickSort` function, displays the number of key comparisons and the number of data moves to sort this array, and calls the `displayArray` function to show the contents of the array after quick sorting

At the end, write a basic `Makefile` which compiles all of your code and creates an executable file named `hw1`. Check out this tutorial for writing a simple makefile:

http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Please make sure that your `Makefile` works properly, otherwise you will not get any points from Question 2.

Important: Then run your executable and add the screenshot of the console to the solution of Question 2 in the pdf submission.

(d) [*10 points*] In this part, you will analyze the performance of the sorting algorithms that you will have implemented. Write a `performanceAnalysis` function to systematically call these algorithms. This function should call the auxiliary global functions to create the arrays with different contents. For each scenario (random, ascending, descending), use the following sizes for the arrays: 4000, 8000, 12000, 16000, 20000, 24000, 28000, 32000, 36000, 40000. Run each of the sorting algorithms with each scenario and output the elapsed time in milliseconds, the number of key comparisons and the number of data moves.

The `performanceAnalysis` function needs to produce an output similar to the one given on the next page. Include this output to the answer of Question 2 in the pdf submission.

```
-----------------------------------------------------
Analysis of Bubble Sort
Array SizeElapsed timecompCount         moveCount
4000      x ms              x                 x
8000      x ms              x                 x
...

-----------------------------------------------------
Analysis of Merge Sort
Array SizeElapsed timecompCount         moveCount
4000      x ms              x                 x
8000      x ms              x                 x
...

-----------------------------------------------------
Analysis of Quick Sort
Array SizeElapsed timecompCount         moveCount
4000      x ms              x                 x
8000      x ms              x                 x
...
```

Question 3 – 15 points

After running your programs, prepare a single page report about the experimental results that you will have obtained in Question 2. With the help of a spreadsheet program (Microsoft Excel, Matlab or other tools), plot elapsed time versus the array size for each sorting algorithm implemented in Question 2. Combine the outputs of each sorting algorithm in a single graph.

In your report, interpret and compare your empirical results with the theoretical ones. Explain any differences between the empirical and theoretical results, if any.

Do not forget to discuss how the time complexity of your program changed when you applied the sorting algorithms to already sorted arrays (ascending and descending) instead of an array containing randomly generated numbers. Also briefly explain the rationality behind this change.