CS 202, Fall 2020
Homework 1 – Algorithm Efficiency and Sorting

Due: 23:55, March 9, 2020

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, March 9, 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: Algorithm Efficiency and Sorting
 * 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 the answers using a word processor (in other words, do not submit images of handwritten answers).
   - 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. Please make sure that you are aware of the homework grading policy that is explained in the rubric (https://docs.google.com/document/d/1jyGik6lsghu7KdSk75wwAcjTwo__4nbtIXrWK4_L75w/edit) for homework assignments.
4. This homework will be graded by your TA, Can Taylan Sarı. Thus, please contact him directly (can.sari 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 $f(n) = 20n^4 + 20n^2 + 5$ 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 [18, 4, 47, 24, 15, 24, 17, 11, 31, 23] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.

- Selection sort
- Bubble sort

Question 2 – 60 points

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

(a) [*30 points*] Implement the insertion sort, quick sort, and merge 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:

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

To test your implementation and conduct the experiments required below, use the auxiliary global functions which you are provided with (see `auxArrayFunctions.h` and `auxArrayFunctions.cpp` files). The headers of these functions are given below. The first function displays the array items on the screen. The other ones are to create three identical arrays that will be used for testing the sorting algorithms. Their use will be detailed later.

```
void displayArray(int *arr, int len);
void createRandomArrays(int *&arr1, int *&arr2, int *&arr3, int N);
void createAlreadySortedArrays(int *&arr1, int *&arr2, int *&arr3, int N);
void createNearlySortedArrays(int *&arr1, int *&arr2, int *&arr3, int N, int K);
```

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

- creates an array from the following: {9, 5, 8, 15, 16, 6, 3, 11, 18, 0, 14, 17, 2, 9, 11, 7}.

- calls the `insertionSort` 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 insertion 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 these tutorials for writing a simple makefile:

http://mrbook.org/blog/tutorials/make/

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.

(c) [*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 do the following.

- First generate three identical arrays of 5000 random integers by using the `createRandomArrays` function, which you are provided with. This function creates three identical arrays with a size of N. Use one of the arrays for the insertion sort, second for the merge sort, and last for the quick sort. Output the elapsed time (in milliseconds), the number of key comparisons and the number of data moves in the format given in the next page. Do not include the time required for creating these arrays. Repeat the experiment for the following sizes: 10000, 15000, 20000, 25000, 30000.

```
void createRandomArrays(int *&arr1, int *&arr2, int *&arr3, int N);
```

- Repeat the experiment, this time using already sorted arrays. For that, use the `createAlreadySortedArrays` function, which you are also provided with. This function creates three identical arrays with a size of N. Likewise, use one of the arrays for the insertion sort, second for the merge sort, and last for the quick sort. Output the elapsed time (in milliseconds), the number of key comparisons and the number of data moves in the format given in the next page. Do not include the time required for creating these arrays. Repeat the experiment for the following sizes: 5000, 10000, 15000, 20000, 25000, 30000.

```
void createAlreadySortedArrays(int *&arr1,int *&arr2,int *&arr3,int N);
```

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

```
-----------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size      Time Elapsed      compCount         moveCount
5000            x ms              x                 x
10000           x ms              x                 x
...

-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size      Time Elapsed      compCount         moveCount
5000            x ms              x                 x
10000           x ms              x                 x
...

-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size      Time Elapsed      compCount         moveCount
5000            x ms              x                 x
10000           x ms              x                 x
...
```

(d) [*15 points*] After running your programs, prepare a single page report about the experimental results that you will have obtained in Question 2c. 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 an already sorted array instead of an array containing randomly generated numbers. Also briefly explain the rationality behind this change.

Question 3 – 25 points

Now consider sorting a nearly sorted array, in which each item is at most K away from its target location. You can generate such kind of arrays for different values of N and K using the `createNearlySortedArrays`, which you are provided with.

```
void createNearlySortedArrays(int *&arr1, int *&arr2, int *&arr3, int N, int K);
```

Prepare a single page report that discusses which algorithm among the three (i.e., the insertion sort or the merge sort or the quick sort) you should select for the most efficient solution of this problem. Discuss how the value of K (with respect to N) will affect your selection. You have to support your discussion with the experimental results. The quality of your experiments will greatly affect your grade.