
CS202, Spring 2023

Homework 1 - Algorithm analysis and sorting

Due: 11/03/2023

Before you start your homework please read the following instructions carefully:

FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for any late submission policies and Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as studentID_name_surname_hw1.zip.
- Your ZIP archive should contain **only** the following files:
  - **studentID_name_surname_hw1.pdf**, the file containing the answers to Questions 1, and 3.
  - main.cpp(name it as: **studentID_name_surname_hw1.cpp**) file which contains the C++ source code and the **Makefile**.
  - Do not forget to put your name, student id, and section number in all of these files. Comment your implementation well. Add a header (see below) to the beginning of each file:

```
/**
 * Title: Algorithm analysis & Sorting
 * Author : Name & Surname
 * ID: 12345678
 * Section : 1
 * Homework : 1
 * Description : description of your code
 */
```

  - Do not put any unnecessary files such as the auxiliary files generated from your preferred IDE.
- Your code must compile.
- Your code must be complete.
- Your code must run on the **dijkstra.cs.bilkent.edu.tr** server.
- For any question related to the homework contact your TA: *klea.zambaku@bilkent.edu.tr*

Question 1 (15 points)

a) Show that $f(n) = 6n^4 + 9n^2 - 8$ is $O(n^4)$ by specifying the appropriate c and n_0 values in Big-O definition. (3 points)

b) Trace the below mentioned sorting algorithms to sort the array [5, 3, 2, 6, 4, 1, 3, 7] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps (after each sort pass for instance).

i) Selection Sort (3 points)
ii) Merge Sort (3 points)
iii) Quick Sort – Assume the last element is chosen as a pivot. (3 points)

c) Find the asymptotic running times in big O notation of $T(n) = 2T(n - 1) + n^2$, where $T(1) = 1$ by using the repeated substitution method. Show your steps in detail. (3 points)

Question 2-a (35 points)

You are asked to implement the following sorting algorithms for an array of integers in a `sorting.cpp` file:

- Insertion Sort (10 points)
- Bubble Sort (10 points)
- Merge Sort (10 points)
- Quick Sort (10 points)
- Hybrid Sort[1] (10 points)

Your functions must have the following parameters:

```
void insertionSort (int *arr, const int size, int &compCount, int &moveCount);

void bubleSort (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);

void hybridSort (int *arr, const int size, int &compCount, int &moveCount);
```

The parameters are described as follows:

`arr` – the array of integers

`size` – size of the array of integers

`compCount` - for **key comparisons**, you should count each comparison like `k1 < k2` as one comparison, where `k1` and `k2` correspond to the key 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).

`moveCount` - for **data moves**, you should count each key 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 . For example, the following swap function has three such assignments (and thus three data moves):

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

Question 2-b (15 points)

In this part, you will analyse the performance of the sorting algorithms that you implemented in part 2-a. In your `main.cpp`, write code which does the followings:

1. Create five identical arrays with random 1000 integers (use rand from `cstdlib`). Use one of the arrays for the insertion sort, one for the bubble sort, one for the merge sort, one for the quick sort and the last

[1] The hybrid sort algorithm starts with the quick sort, but when the partition size becomes less than or equal to 20, sorts that partition with the bubble sort.

one for the hybrid sort algorithm. Output the elapsed time in milliseconds, the number of key comparisons, the number of data moves (use clock from `ctime` for calculating elapsed time). Repeat the experiment for the following sizes: {5000, 10000, 20000} (5 points)

2. Now, instead of creating arrays of completely random integers, create arrays with elements in this partially ascending order as follows: If we have n integers, smallest $log_2 n$ elements are in the first part of the array (randomly permuted among themselves), next smallest $log_2 n$ elements are in the second part (again randomly permuted among themselves) and so on, till you have the full array. Then repeat the steps in 2-b-1. (5 points)

3. Lastly, create arrays with elements in this partially descending order: If we have n integers, largest $log_2 n$ elements are in the first part of the array (randomly permuted among themselves), next largest $log_2 n$ elements are in the second part (again randomly permuted among themselves) and so on, till you have the full array. Then repeat the steps in 2-b-1 (5 points)

When the code in your `main.cpp` is compiled, it needs to produce an output similar to the following one:

Part 2-b-1: Performance analysis of random integers array

	Elapsed time	Comp. count	Move count
Array Size: 1000			
Insertion Sort			
Bubble Sort			
Merge Sort			
Quick Sort			
Hybrid Sort			

	Elapsed time	Comp. count	Move count
Array Size: 5000			
Insertion Sort			
Bubble Sort			
Merge Sort			
Quick Sort			
Hybrid Sort			

……..

Part 2-b-2: Performance analysis of partially ascending integers array

----

	Elapsed time	Comp. count	Move count
Array Size: 1000			
Insertion Sort			
Bubble Sort			
Merge Sort			
Quick Sort			
Hybrid Sort			

	Elapsed time	Comp. count	Move count
Array Size: 5000			

Insertion Sort
Bubble Sort
Merge Sort
Quick Sort
Hybrid Sort
…….

Part 2-b-3: Performance analysis of partially descending integers array

----

	Elapsed time	Comp. count	Move count
Array Size: 1000			
Insertion Sort			
Bubble Sort			
Merge Sort			
Quick Sort			
Hybrid Sort			

	Elapsed time	Comp. count	Move count
Array Size: 5000			
Insertion Sort			
Bubble Sort			
Merge Sort			
Quick Sort			
Hybrid Sort			

…….

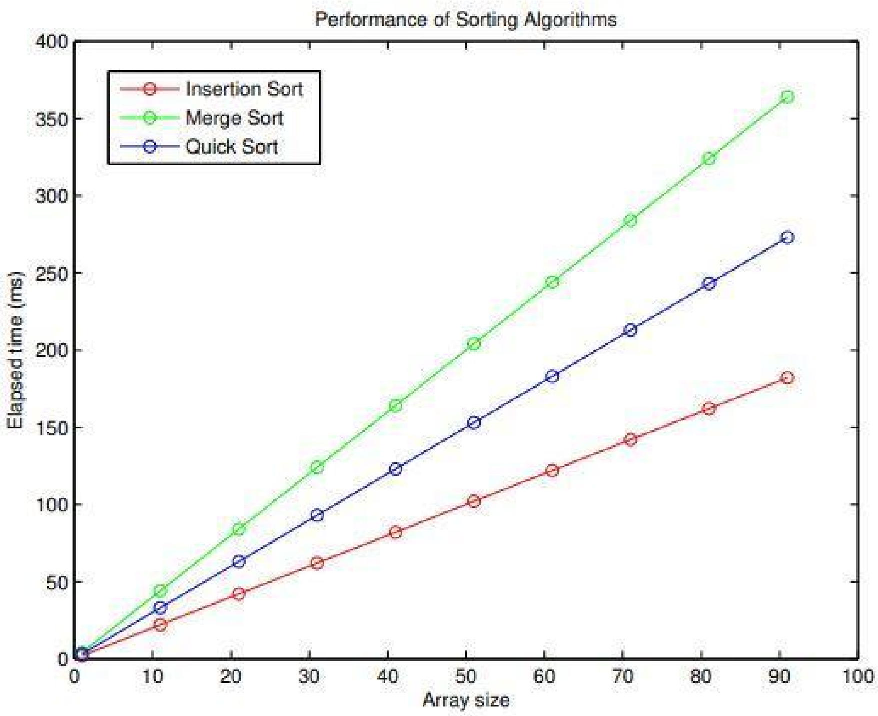
Question 3 (35 points)

a) (10 points) After running your programs, you are expected to prepare a 2-page report about the experimental results that you obtained in Question 2-b. First, create a table similar to the one below and fill it with the results you obtained. In this table, for instance: R1K - array with 1000 random integers, A1K - array of 1000 partially ascending integers, D1K - array of 1000 partially descending integers.

Array	Elapsed Time (ms)					Number of comparisons					Number of Data Moves				
	Insertion Sort	Bubble sort	Merge Sort	Quick Sort	Hybrid Sort	Insertion Sort	Bubble sort	Merge Sort	Quick Sort	Hybrid Sort	Insertion Sort	Bubble sort	Merge Sort	Quick Sort	Hybrid Sort
R1K															
R5K															
R10K															
R20K															
A1K															
A5K															
A10K															
A20K															
D1K															
D5K															

D10K														
D20K														

b) (10 points) Then, with the help of a spreadsheet program (Microsoft Excel, Matlab or other tools), plot elapsed time versus the size of array. Note that you will need to plot 3 figures, one for each array type (random, partially ascending and partially descending). A sample figure is given below (these values do not reflect real values, although not shown in this figure, there should be 5 lines, one for each algorithm):



c) (15 points) Comment on the results, contrasting the results for varying sorting algorithm and varying input array type.

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:

[A Simple Makefile Tutorial]

[Another Makefile Tutorial]

[Another Makefile Tutorial]