# Title: Sorting and Algorithm Efficiency

# Author: Tolga Han Arslan

# ID: 22003061

# Section: 1

# Assignment: 1

# Description: HW1 Report

# Question 1:

(a) [*5 points*] Show that 𝑓(𝑛) = 4𝑛5 + 2𝑛3 + 3𝑛 is 𝑂(𝑛5) by specifying appropriate c and n0 values in Big-O definition.

(b) [*10 points*] Trace the following sorting algorithms to sort the array [ 40, 25, 65, 45, 50, 35, 55, 38, 30, 42 ] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.

# Selection Sort:

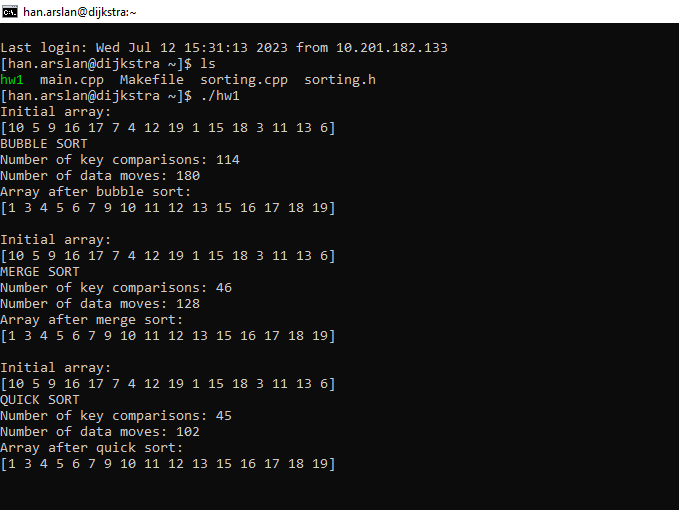


# Insertion Sort:



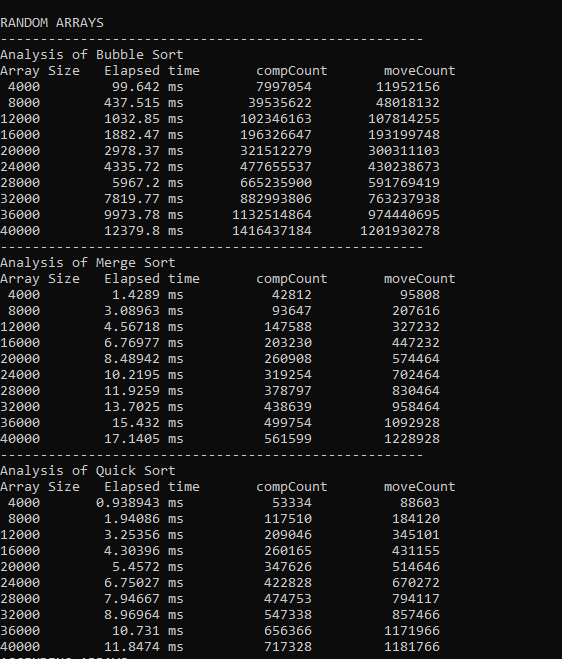
# Question 2:

## Screenshot of the output in part c:



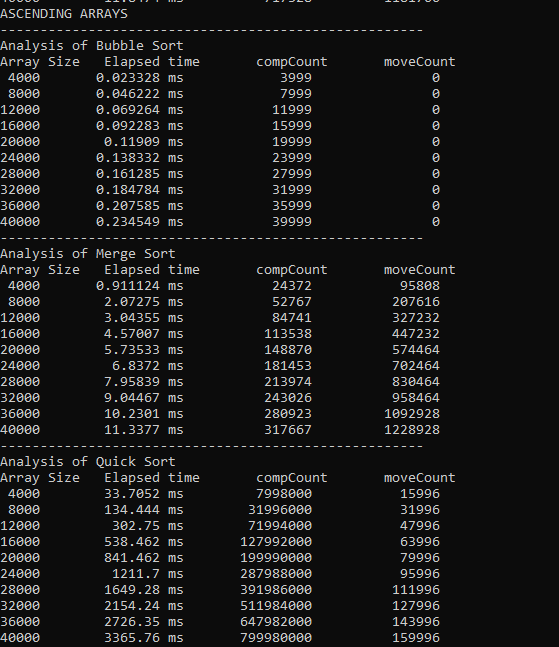
# Screenshots of the output in part d:

# For Random Arrays:



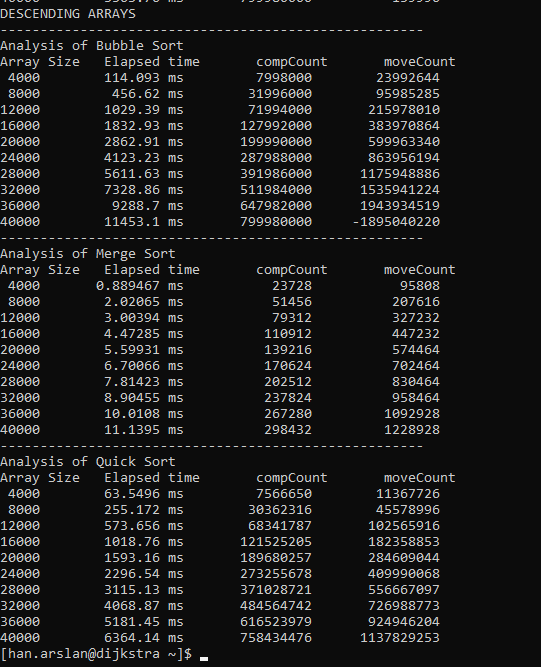
*Output 1: Algorithm comparison with randomly generated arrays*

# For Ascending Arrays:



*Output 2: Algorithm comparison with ascending arrays*

# For Descending Arrays:



*Output 3: Algorithm comparison with descending arrays*

# Question 3:

*Graph 1: Sorting algorithm comparison with randomly generated arrays*

*Graph 2: Sorting algorithm comparison with ascending arrays*

*Graph 3: Sorting algorithm comparison with descending arrays*

# Comments:

* Bubble sort worst case behaviour is O(n2) and it happens when the array is in reverse order. By looking at the graph 3, experimental results with descending arrays are consistent with theoretical results.
* Bubble sort average case behaviour is O(n2), by looking at the graph 1 experimental results are consistent with theoretical results since the arrays are randomly generated.
* Bubble sort makes O(n2) key comparisons and moves in worst case and average case, as seen in output 3 and output 1.
* Bubble sort best case behaviour is O(n) and it happens when the array is already sorted. This case happens in graph 2, when the array is in ascending order. The moveCount variable is 0 in all array samples, but bubble sort algorithm stil do (n-1) key comparisons as seen in output 2.
* There is integer overflow in descending array analysis with bubble sort of array size 40000, hence moveCount variable is negative.
* Merge sort is O(n\*logn) in all cases, independent of the array configuration. Graph 1, 2 and 3 are consistent with this theoretical behaviour.
* Merge sort makes O(n\*logn) key comparisons in average case and worst case, hence all three outputs are consistent with this theoretical behaviour.
* Quick sort is O(n\*logn) algorithm in best case and average case, results in graph 1 is consistent with this behaviour.
* In worst case, when the array is already sorted or in reverse order, quick sort is O(n2). Graph 2 and 3 shows this behaviour of quick sort algorithm’s worst case.
* Although merge sort is stable and O(n\*logn) in every case, quick sort is faster in terms of elapsed time in average case, as seen in output 1. This behaviour is most likely caused by merge sort’s requirement of an extra array in its execution.
* Bubble sort’s best case behavior O(n) is actually faster than merge sort and quick sort in all cases, as seen in the outputs, however this case rarely occurs in real-life implementations.