Ömer Faruk ABACI
150150037

BLG 335E: Analysis of Algorithms I
Project 1 Report

Compilation Instructions

Compile command: g++ Data.cpp Data.h main.cpp
Run command: ./a.out –algo A –feature F –size N
(A = 'm' for merge-sort, 'i' for insertion-sort
F = 'p' for last_price criterion, 't' for time_stamp criterion
N = size)

Part a.

Asymptotic upper bound of Merge Sort: $n \log n$

Big-O notation: $O(n \log n)$

Asymptotic upper bound of Insertion Sort: n^2

Big-O notation: $O(n^2)$

Implementation of Merge Sort: It is a divide and conquer method. So I first divided input to 2 almost equal arrays, and continue to divide these arrays to 2 until I reach inputSize/2 arrays with size 1 or 2. Starting from first half, I compared first two array's elements, sorted and merged them into the original array. I continued this process for all arrays with size 1 or 2. Then I compared merged arrays with size 2 or 3, sorted and merged them. I recursively continued this process. At the end, all elements were sorted. So this implementation fits to the asymptotic upper bound of Merge Sort.

Implementation of Insertion Sort: It's an in-place sorting algorithm. I selected a key starting from the second index, and compared the key with the items before itself, until I find the proper place where the key's value is more than the value before itself and smaller than the value after itself. Then I put the key to that place, and increased it. I iteratively continued this process until my key is the last value of the array. At the end, all elements were sorted. So this implementation fits to the asymptotic upper bound of Insertion Sort.

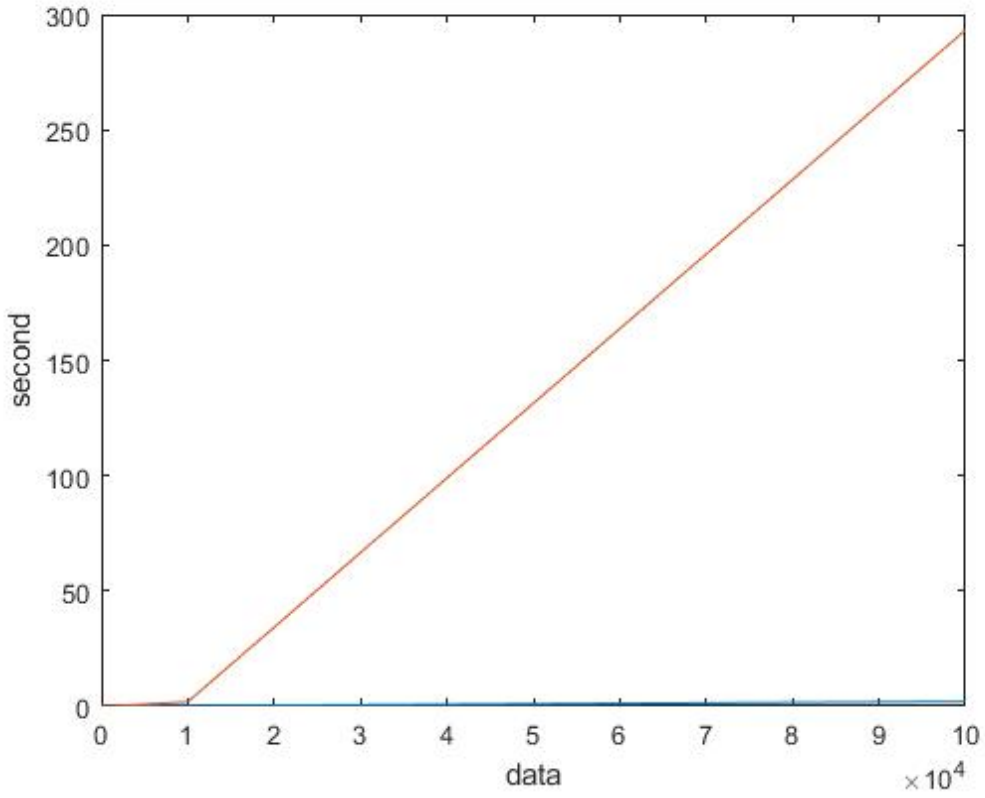
Part b.

Below you can find the table of average runtimes of 10 executions using different data sizes (N) and algorithms.

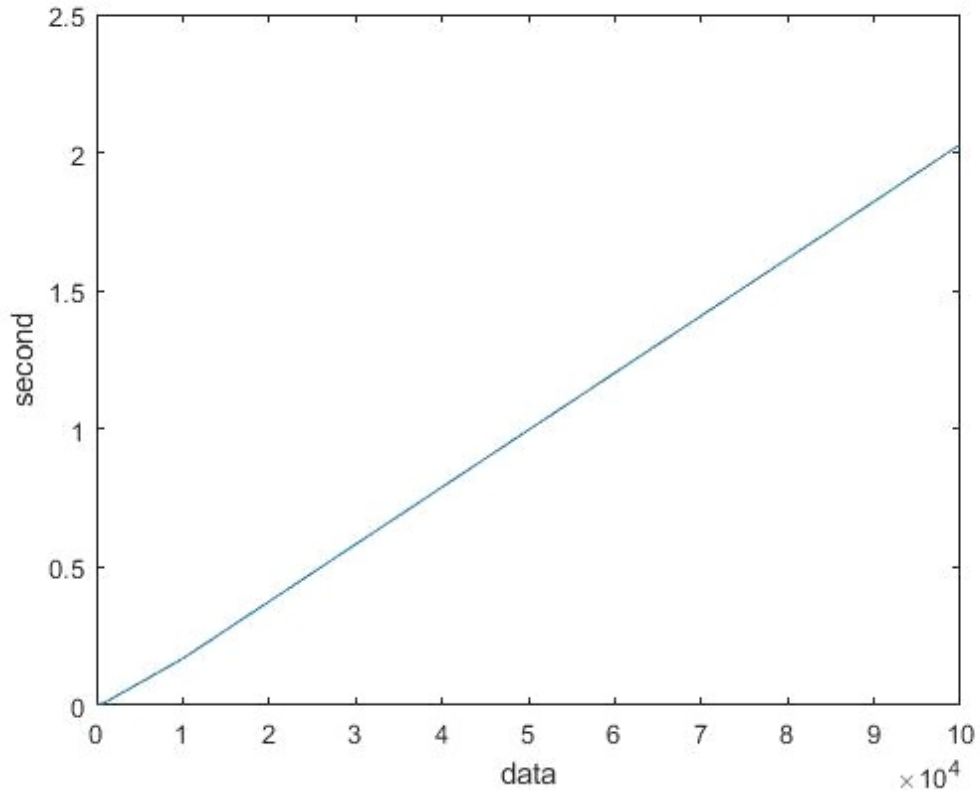
	N = 100	N = 1000	N = 10000	N = 100000
Merge Sort average time (secs)	0,001465	0,010250	0,169034	2,029931
Insertion Sort average time (secs)	0,000294	0,013937	1,907786	293,195000

Part c.

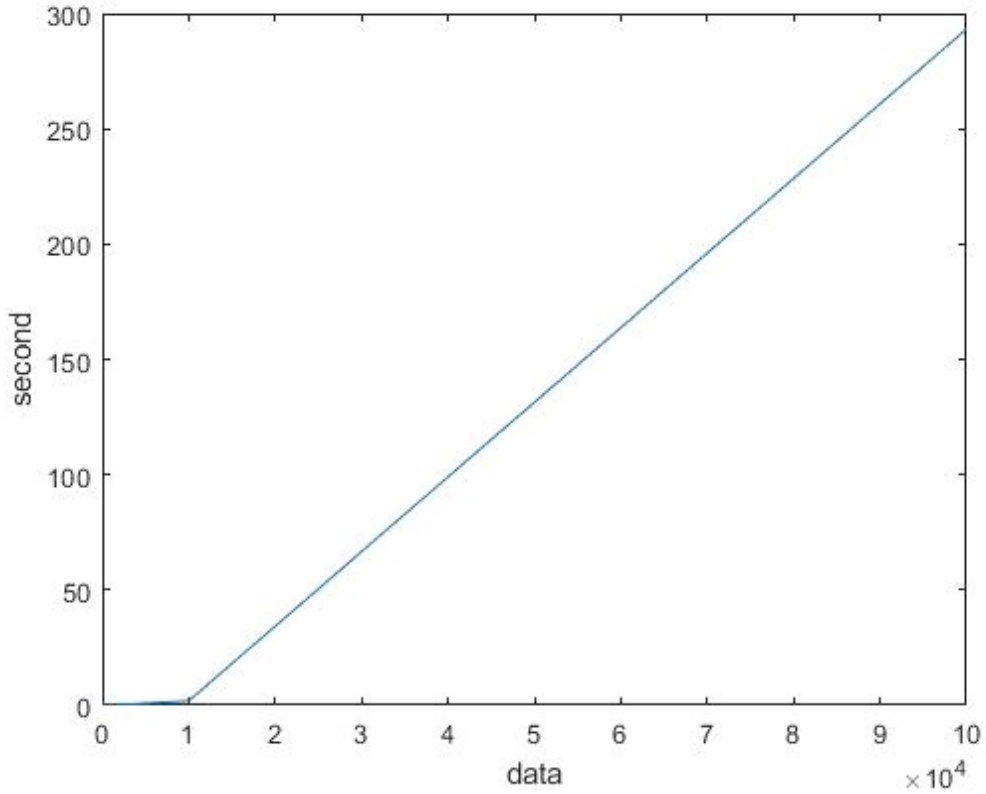
Merge Sort (blue) vs. Insertion Sort (pink)



Merge Sort



Insertion Sort



We can see that until the data size 10000, Insertion Sort performs almost equally with or better than Merge Sort but after the data size 10000, Merge Sort outperforms Insertion Sort. That's because Insertion Sort have a smaller constant than Merge Sort, but when data size is big enough constants don't change runtime much which means $n \log n < n^2$.

Part d.

When we add a new data to a sorted array, and then sort it, it is better to choose Insertion Sort because it will enter only once to the inner loop of Insertion Sort thus it's complexity will be n. However, for Merge Sort, it will divide and merge the array again and make all comparisons even though the array is almost sorted. So it's complexity is still $n \log n$.