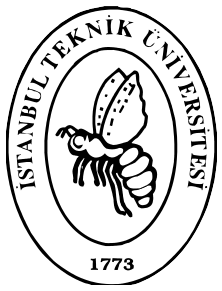
ISTANBUL TECHNICAL UNIVERSITY

Computer Engineering Department



BLG 335E
ANALYSIS OF ALGORITHMS

HOMEWORK 1-REPORT

EMRE UYSAL

PART A

Insertion Sort

In my implementation of Insertion Sort, I used template classes for giving capability to compare values according to its type. Also, Insertion Sort includes one "for" and one conditional "while" loop in it. After calling Insertion Sort from main function, "for" loop starts from second element and iterates until reaching the n^{th} element. So, for loop creates n-1 operations. Initializations in Insertion Sort occurs as constants for our cost. I made 6 initializations in "for" loop. So, it takes 6 steps and 6*(n-1) operations occur in the "for" loop. If the input is sorted inner while loop does not work because of the condition. So, this creates Best-Case for Insertion Sort. In the Best-Case my function makes 6n-6 operations. So, the Best-Case becomes O(n). If the input is in reverse order, this creates the Worst-Case. In the Worst-Case inner loop makes comparisons n times. This creates n operations and in the loop, there exists 5 operations in the inner loop. So according to inner loop our Worst-Case becomes $5(n-1)+(n-1).n/2 = (n^2-9n-10)/2$. So, in asymptotic upper bound notation Worst-Case becomes $O(n^2)$ and my algorithm fit these values.

Merge Sort

In my implementation of Merge Sort there are two functions for divide and conquer operations. In the divide step, I divided my input half by half until it is not available to division operation. In the division operation, subroutines are called. So, because of dividing input into 2 parts $\log_2 n$ operations occurs in this step (Usually computer science use 2-part division but it is not wrong to divide 3 or 4 in Merge Sort.). After the division operation Merge function comes and costs as c*n operation. After that we can say that all Merge Sort cost us $c*n*(\log_2 n)$. After ignoring the law order term Merge Sort's asymptotic upper bound exists as O(nlogn). So, my algorithm fit these values. Also in Merge Sort, Best-Case and Worst-Case have the same complexity because of for every input types Merge Sort makes division and combining parts.

PART B

In this part I tested the algorithms according to 2 cases. In the first case, unsorted csv file given to algorithms for calculating the execution time. In the second case, sorted data given to algorithms and execution times calculated. According to unsorted input I can say that Merge Sort is faster than Insertion Sort in the worst case. But according to sorted input, Insertion is faster than Merge. Because of division and combination operation in Merge, it becomes slower but insertion checks it and pass it.

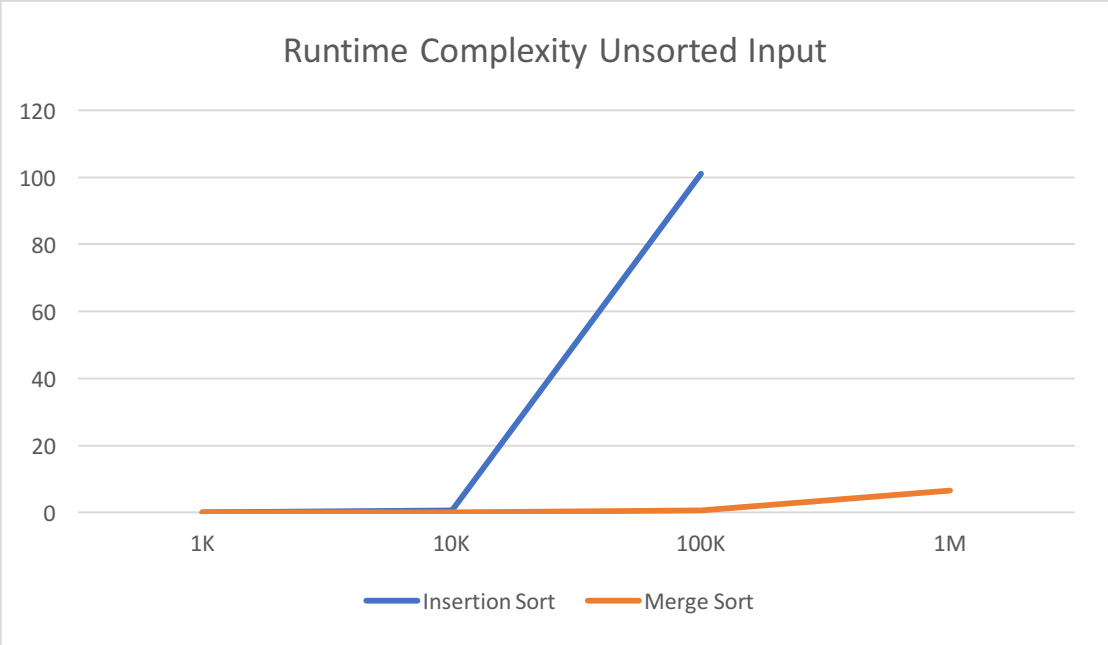
UNSORTED INPUT				
Algorithm/Size	**1K**	**10K**	**100K**	**1M**
Insertion Sort	0,007016	0,687398	101,17	-
Merge Sort	0,0035	0,049	0,601	6,55

Unsorted input times(seconds) according to input size

SORTED INPUT				
Algorithm/Size	**1K**	**10K**	**100K**	**1M**
Insertion Sort	0,00013	0,0012	0,014	0,102
Merge Sort	0,0035	0,043	0,504	5,21

Sorted input times(seconds) according to input size

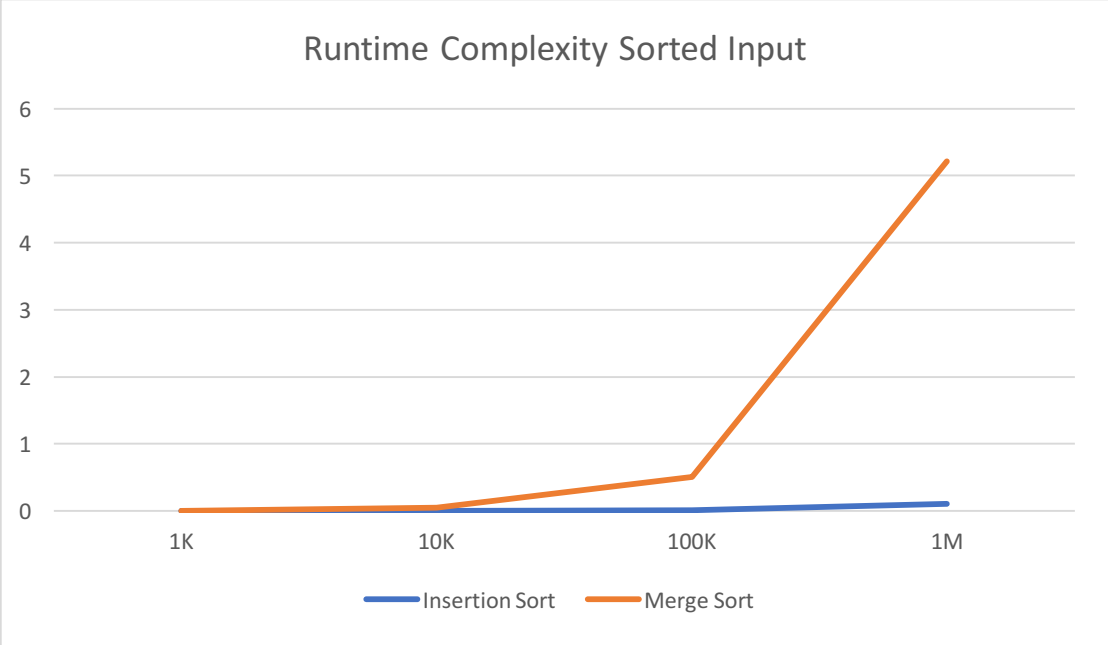
PART C



Graphic 1- Seconds / Input Size Comparison

According to Insertion Sort results, we can say that Insertion Sort's elapsed time highly increases when unsorted input data size increases. When we look at the asymptotic upper bound that we explained in part A, we can see $O(n^2)$ notation because of when unsorted data size increases (take reversed data as base) a quadratic function occurs.

Merge Sort is a divide and conquer algorithm. So, it makes merge operation in small sizes and sorts the data. When we look at the asymptotic upper bound in part A we can say that Merge Sort creates a logarithmic function and it decreases the elapsed time in big sized unsorted data according to Insertion Sort.



Graphic 2-Seconds / Input Size Comparison

According to runtime complexity sorted input graphic, we can say that Insertion Sort is faster than the Merge Sort because of check condition in the inner loop. The best case of Insertion Sort is sorted data O(n). But best case does not change for Merge Sort. It always divides and conquers. So, we can say that in best case Insertion Sort is better.

PART D

Assume that after sorted input of 1M according to price feature I inserted new data into sorted data. The results according to elapsed time are;

Sorted Input:

-Merge Sort = 5,21 seconds

-Insertion Sort = 0,102 seconds

After Adding A New Element:

-Merge Sort = 5,24 seconds

-Insertion Sort = 0,160 seconds

Merge Sort always realizes divide and conquer method if the input sorted or not. Because of that merge sort give the same result of time. Complexity does not change for Merge Sort.

Insertion Sort checks the array. Because of this check operation Insertion pass the loop that changes indexes of the array. So, in sorted input Insertion gives a faster result. But when a data added to input, complexity changes and increases for Insertion Sort.

All in all, if I have a sorted input and if I need to add a new item into it I always use Insertion Sort. Because it passes sorted ones and go through to unsorted one and puts it to required place. Also, if I have small sized data I choose Insertion Sort. But if I have big sized unsorted data I choose Merge Sort. Because of divide and conquer methodology Merge Sort decreases elapsed time.