Department of Computer Engineering

BLG 335E
Analysis of Algorithms I
Project 1 Report

Fall 2018

Muhammed Raşit EROL

150150023

1 INTRODUCTION

In this project, it required that analysing real time stock market data with implementing Merge Sort and Insertion Sort algorithms. Data for this project is the file which is named as "log_inf.csv". It is used two feature for sorting market data which are "timestamp" and "last_price" columns. N which is provided by user, is the number of read line from the file. Also, it corresponds the number of line in sorting market data. Moreover, user can provide criterion of sorting using "–feature", value of the N using "-size" and algorithm using "-algo" keywords. Sorted data is stored in the file named as "sorted_csv".

2 IMPLEMENTATION

In this project, it is used two class for storing timestamp and last price. In order to get better performance timestamp stored as several integer variables instead of string. Comparison between integers is much faster than byte-byte string comparison.

Read lines are stored in string array for faster writing on the file after sorting. Also, instead of swapping each line, indexes of the each line stored in the classes. Meanly, data sorted with sorting criterion like price, but the line which corresponds to that price stored as index on the string array. In the writing to the file part, this indexes are used. Thus, whole lines are stored memory when reading and used after that.

Furthermore, for conditional statements during the sorting, ">" and "<=" operators are overloaded. Merge and insertion sort algorithms are implemented using pseudo codes on the slides. Using void pointers functions are implemented such that they can work with both class.

My code compiled at ITU SSH server successfully like that:

```
g++ *.cpp –o algo -std=c++11
```

3 ANALYSIS

3.1 PART A

Time complexity of the Merge Sort is O(nlogn). In the merge sort, first we split our array very small pieces. Each element of the array are divided to the individual elements. After divide process, array elements are merged back from small to big pieces. In this stage, comparison are done. There is logn level as merged like tree notation. In the each level, n comparison are done. Hence, time complexity of the merge sort become O(nlogn). In terms of the N, time complexity of merge sort is NlogN.

In my project, the array is split into small pieces which have size of 1 and each pieces are compared the other siblings. The high of the tree is logn and there are n comparison each level.

In the figure 1, summed up over the sub problems on each level, we see that the total merging time for that level is c*n(c is some coefficient). There are logn level and each level merging time time is cn. Hence; complexity of my code is also cnlogn which is equal to O(nlogn).

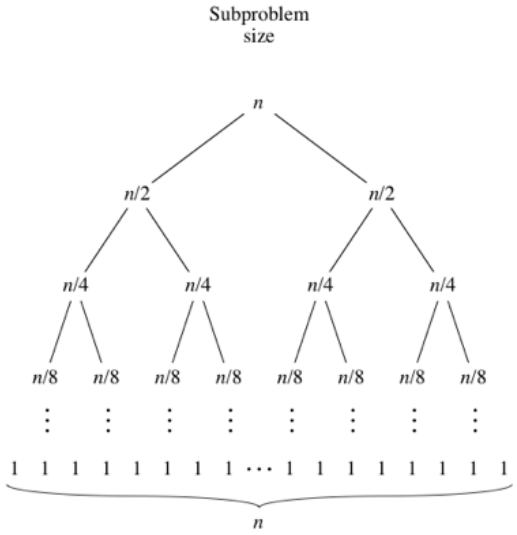


Figure 1 Merge Sort

Time complexity of the Insertion Sort is O(n^2). In the insertion sort, there are two nested loop. First loop iterates the input elements by growing the sorted array at each iteration. In the first iteration array can be accepted as sorted. Each iteration on the first loop array size increases and it compares the last added element with the whole array in the second loop in order to find correct position for that value. If position is found then first loop continue to iteration until the end of the array. There are N iteration on the first loop and N comparison on the second loop; hence time complexity of the insertion sort is O(n^2). In terms of the N, time complexity of insertion sort N^2.

In my project, there are two nested loop. First loop iterates N-1 times. Maximum number of the second iteration is N-1. Hence complexity of my code is (N-1)^2 which is equal to O(n^2).

3.2 PART B

Average times for both algorithm are shown below (see Figure 2).

Time Stamp							
Average Time(milisecond)	N	1000	10000	50000	100000	500000	~1000000
	Merge Sort	2,9	27,4	157,7	326,3	1431,9	2698,5
	Insertion Sort	0,3	0,8	2,6	5,1	26,4	41,2
Last Price							
Average Time(milisecond)	N	1000	10000	50000	100000	500000	~1000000
	Merge Sort	2,2	28,2	162	323,4	1465,5	2558,9
	Insertion Sort	0,6	83,2	2273,1	7935,3	198194,7	675120,4

Figure 2 Average time for algorithms

All execution times for both algorithm and both for sorting criterion are shown below (see Figure 3 and Figure 4).

Run Time Scores for Time Stamp											
Run Times:			1	2	3	4	5	6	7	8	9	10
Average(ms)/Algorithm		N	(milisecond)									
2,9	Merge	1000	3	2	3	2	3	2	2	4	4	4
0,3	Insertion		1	0	0	1	0	0	0	0	0	1
Average(ms)/Algorithm		N	(milisecond)									
27,4	Merge	10000	30	24	23	30	30	23	23	30	31	30
0,8	Insertion		1	1	1	1	0	1	1	1	0	1
Average(ms)/Algorithm		N	(milisecond)									
157,7	Merge	50000	123	171	170	164	152	162	168	154	162	151
2,6	Insertion		2	3	2	3	3	3	2	2	3	3
Average(ms)/Algorithm		N	(milisecond)									
326,3	Merge	100000	356	251	339	338	252	345	326	355	347	354
5,1	Insertion		4	5	6	4	5	5	6	6	5	5
Average(ms)/Algorithm		N	(milisecond)									
1431,9	Merge	500000	1371	1384	1913	1374	1385	1380	1371	1379	1388	1374
26,4	Insertion		26	25	26	24	29	28	26	27	27	26
Average(ms)/Algorithm		N	(milisecond)									
2698,5	Merge	916723	2585	3543	2656	2590	2546	2632	2814	2487	2596	2536
41,2	Insertion		48	45	42	46	44	39	36	37	38	37

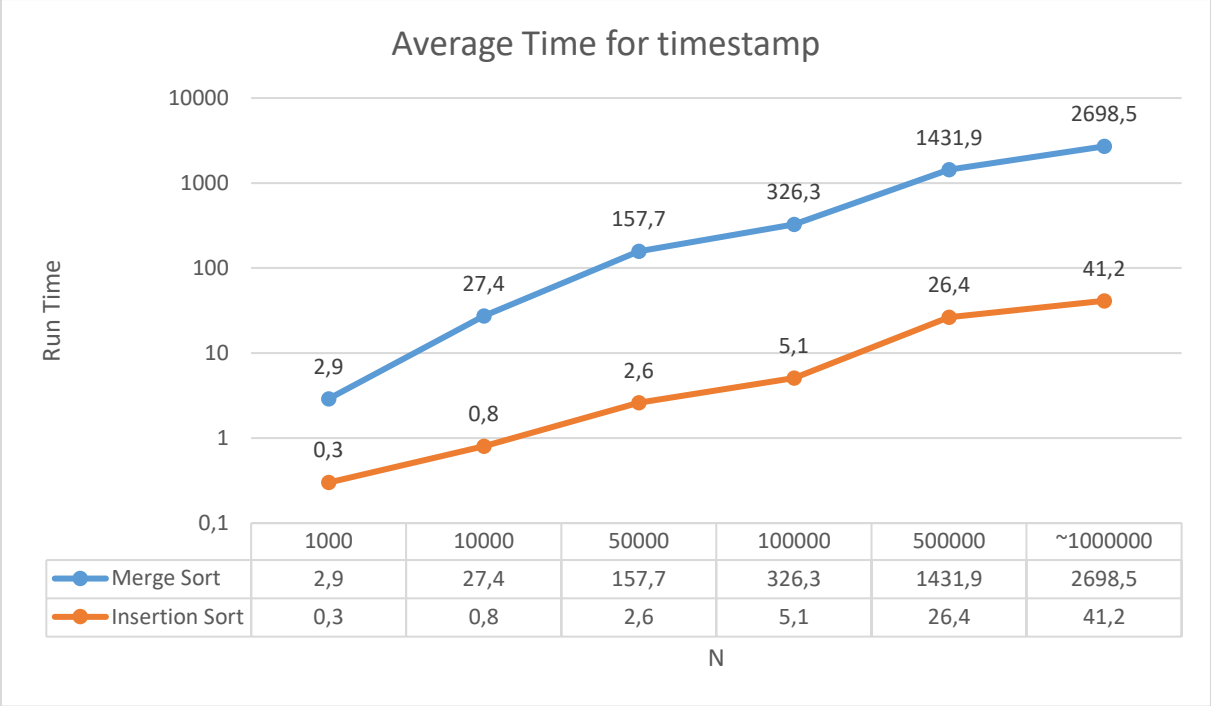
Figure 3 Run time scores for timestamp

Run Time Scores for Last Price											
Run Times:			1	2	3	4	5	6	7	8	9	10
Average(ms)/Algorithm		N	(milisecond)									
2,2	Merge	1000	2	2	2	2	2	2	3	3	2	2
0,6	Insertion		1	0	0	1	1	0	1	1	0	1
Average(ms)/Algorithm		N	(milisecond)									
28,2	Merge	10000	22	29	27	23	30	31	30	29	30	31
83,2	Insertion		89	88	87	88	86	88	88	87	65	66
Average(ms)/Algorithm		N	(milisecond)									
162	Merge	50000	160	160	158	159	161	166	164	172	161	159
2273,1	Insertion		2239	2282	2197	2368	2451	2279	2131	2232	2281	2271
Average(ms)/Algorithm		N	(milisecond)									
323,4	Merge	100000	253	347	336	329	331	342	298	338	324	336
7935,3	Insertion		6686	6725	6621	6865	8961	7958	9047	8964	8734	8792
Average(ms)/Algorithm		N	(milisecond)									
1465,5	Merge	500000	1773	1345	1398	1679	1337	1467	1541	1340	1312	1463
198194,7	Insertion		196278	203985	197532	198541	196912	201667	198521	197854	195328	195329
Average(ms)/Algorithm		N	(milisecond)									
2558,9	Merge	916723	2575	2527	2561	2569	2536	2565	2574	2570	2566	2546
675120,4	Insertion		696867	665659	662189	675247	689878	692341	673698	662178	663296	669851

Figure 4 Run time scores for last price

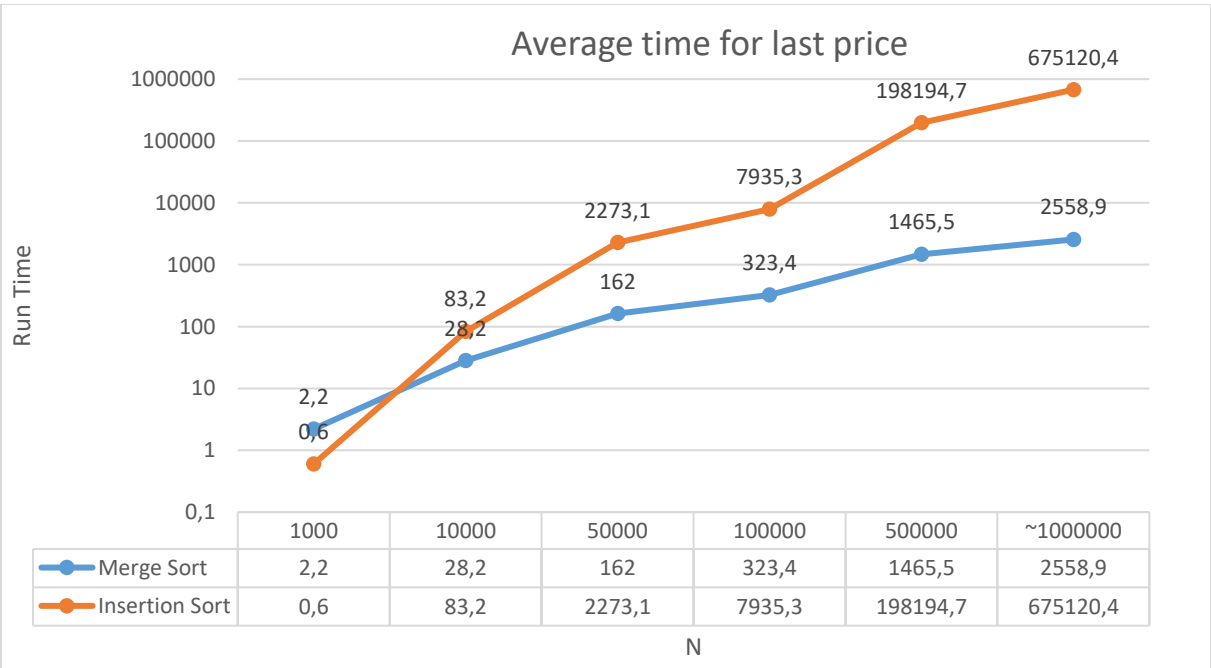
3.3 PART C

Average time graphs are shown below (see Figure 5 and Figure 6). These graphs are scaled with logarithm.



	1000	10000	50000	100000	500000	~1000000
Merge Sort	2,9	27,4	157,7	326,3	1431,9	2698,5
Insertion Sort	0,3	0,8	2,6	5,1	26,4	41,2

Figure 5 Average time for timestamp



	1000	10000	50000	100000	500000	~1000000
Merge Sort	2,2	28,2	162	323,4	1465,5	2558,9
Insertion Sort	0,6	83,2	2273,1	7935,3	198194,7	675120,4

Figure 6 Average time for last price

Given data is sorted by timestamp. Because of that, insertion sort works faster than merge sort if we sort the data with timestamp (see Figure 5). However, data is not sorted by last price. Hence, merge sort works faster than insertion sort when sorting criterion is last price (see Figure 6).

For merge sort, it can be seen that complexity is nlogn for both sorting criterion (see Figure 5 and Figure 6). In the both figures there is no big difference between timing because data is random.

However, for insertion, due to sorted timestamp, complexity is n in the Figure 5. In the figure 6, due to last price is not sorted, complexity of insertion sort is n^2.

Shortly, for N is 1.000.000 when sorting criterion is last price, complexity of insertion sort is O(n^2) with value of 675120 and complexity of merge is O(nlogn) with value of 2558. This proves the asymptotic upper bounds found in part a.

3.4 PART D

In this part it is used sorted data with last price criterion. I have inserted new entry into the end of the file. My entry has biggest last price. In the Figure 7, there are values before the adding new entry and after the adding new entry for both algorithm. For sorting last price, due to sorted array, insertion sort worked faster than merge sort. Because for sorted array insertion sort complexity is O(n).

Moreover, I have inserted another entry into the end of the file. This time my entry has smallest last price. In the Figure 7, there are values before the adding new entry and after the adding new entry for both algorithm. For sorting last price, due to sorted array, insertion sort worked faster than merge sort. Because for sorted array insertion sort complexity is O(n). Furthermore, after adding new entry, insertion sort worked little slower when last element of array is smallest. However, this difference is small because just one element is shifted. Moreover, merge sort worked as same in the other situation.

Shorty, in the sorted arrays, insertion sort has better performance than the merge sort. Because in the sorted arrays, insertion sort complexity is O(n) but merge sort complexity is O(nlogn). In the other situations, merge sort has better performance. This result can be seen in the Figure 7.

Last Price			
Add New Entry	Before	After Biggest Entry	After Smallest Entry
Merge	2475ms	2495ms	3282ms
Insertion	4ms	4ms	7ms

Figure 7 Adding new entry