ISE 315
HW2
Selin Dinç
150150229

REPORT

a)
Quick Sort Pseudo-code

```
QuickSort(vector,int,int,char)

if low<high
        pivotIndex← partition(vector,low,high,feature)
        QuickSort(vector,low,pivotIndex-1,feature)
        QuickSort(vector,pivotIndex+1,high,feature)

Partition(vector,int,int,char)

pivot←high
i← low-1

for s← low to high
        if vector[s]<=pivot
                i++
                swap(vector[i],vector[s])
swap(data[i+1],data[high])
return i+1
```

QuickSort has asymptotic bound of O(nlgn) for best cases and almost best cases. If our partitions are as evenly balanced as possible, we can obtain the best case. However, if our data is sorted or reverse sorted, pivot may be the highest or the lowest element in our data array. That lead us to encounter the worst case which is O(n^2). That is because our partitions will not be balanced. In my QuickSort function, I'm calling the partition function one time which is O(n), since there is one for loop in it. Then, I call QuickSort recursively two times. For the two parts of my partitions. If my partitions are evenly balanced, then that makes lgn. That makes the asymptotic bound O(nlgn). However, if my data is sorted or reverse sorted, because of the reason that I choose my pivot from the end of the data set, that makes the pivot the highest or the lowest element. That makes one of my partition doesn't contain any elements and the other contains n-1 elements. That makes O(n). That is why my asymptotic bound would be O(n^2).

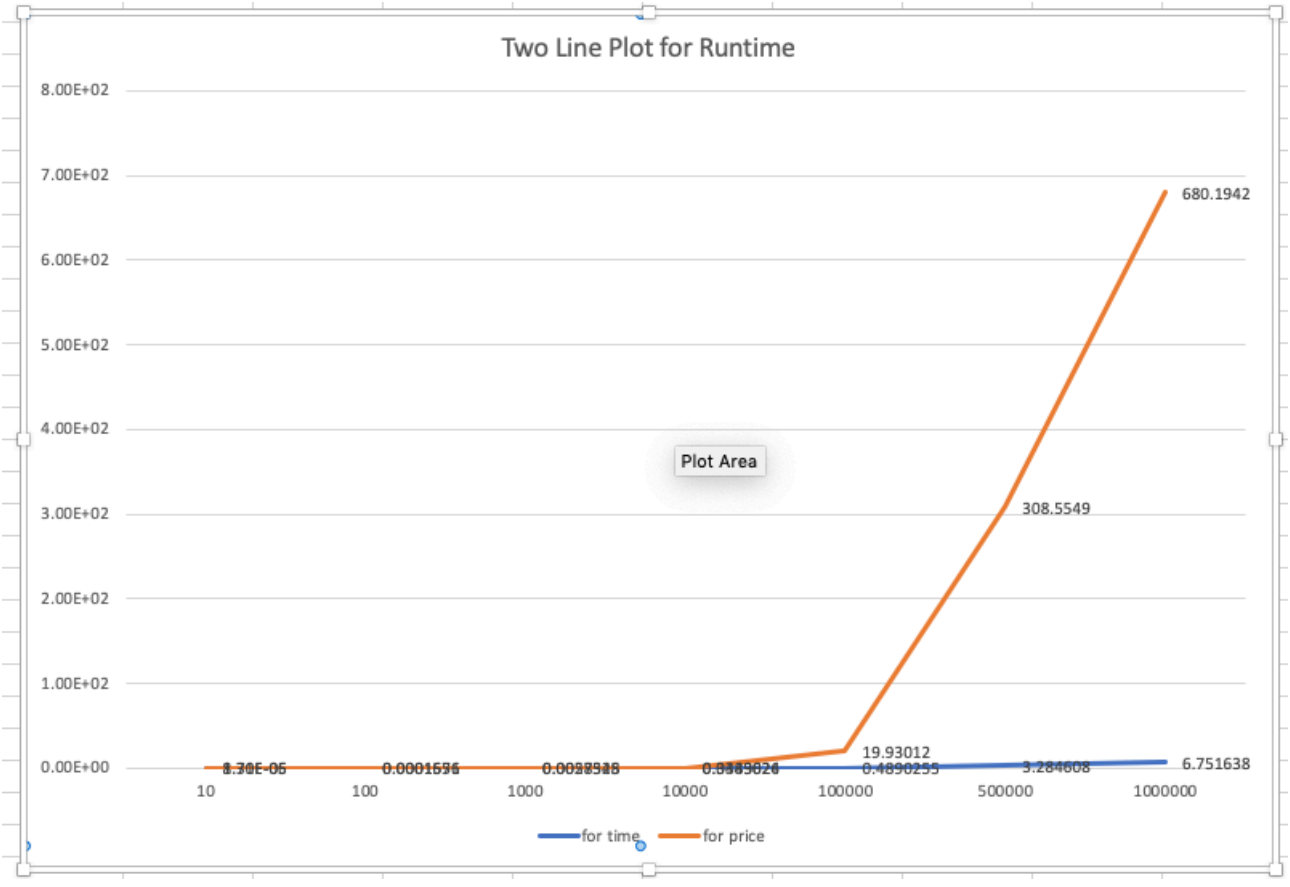
b)

for timestamp:

	10	100	1000	10000	100000	500000	1000000
QuickSort	1.30E-05	0.00021	0.002987	0.045563	0.516483	3.27386	6.372
	1.10E-05	0.000233	0.002946	0.040225	0.485219	3.1459	6.34967
	1.20E-05	0.000123	0.002807	0.042988	0.482604	3.17494	6.35388
	1.20E-05	0.000329	0.002683	0.045153	0.483578	3.20749	7.61891
	1.10E-05	0.000123	0.003126	0.05501	0.483292	3.40167	6.33013
	1.60E-05	0.000123	0.00272	0.046219	0.478511	3.26277	6.97908
	1.20E-05	0.000123	0.002626	0.057458	0.485	3.17008	6.89118
	1.70E-05	0.000143	0.003007	0.045496	0.498775	3.33101	6.71271
	1.60E-05	0.000132	0.002771	0.047596	0.491356	3.2083	6.88755
	1.10E-05	0.000132	0.002652	0.057318	0.485437	3.67006	7.02127
Average:	1.31E-05	0.0001671	0.0028325	0.0483026	0.4890255	3.284608	6.751638

for last_price:

	10	100	1000	10000	100000	500000	1000000
QuickSort:	8.00E-06	0.000293	0.004969	0.336518	20.4381	308.779	680.701
for price	8.00E-06	0.000108	0.005486	0.34936	21.2101	307.455	679.577
	8.00E-06	0.000156	0.00599	0.333891	18.043	308.356	680.467
	8.00E-06	0.000156	0.005938	0.333891	18.06	309.676	681.145
	8.00E-06	0.000117	0.005767	0.338131	21.6549	308.366	680.701
	7.00E-06	0.000162	0.005867	0.333422	20.8314	307.455	679.467
	8.00E-06	0.000116	0.005552	0.336669	19.1651	308.775	680.845
	8.00E-06	0.000159	0.005796	0.329416	22.3144	309.788	679.284
	1.10E-05	0.000172	0.005553	0.3313	18.5791	308.779	680.372
	1.30E-05	0.000117	0.00663	0.327026	19.0051	308.12	679.383
Average:	8.70E-06	0.0001556	0.0057548	0.3349624	19.93012	308.5549	680.1942



Y-axis: runtime
X-axis: data size

In this plot, I run my Quick sort algorithm to sort timestamp and last_price values in my data. The plot shows that, sorting last_price values has higher rate of increase than sorting timestamp values. This shows that, for timestamp values my program run with the asymptotic bound O(nlgn) which is best or almost best case for Quick sort. However, for last_price values, my program has asymptotic bound O(n^2) which is worst case for Quick sort.

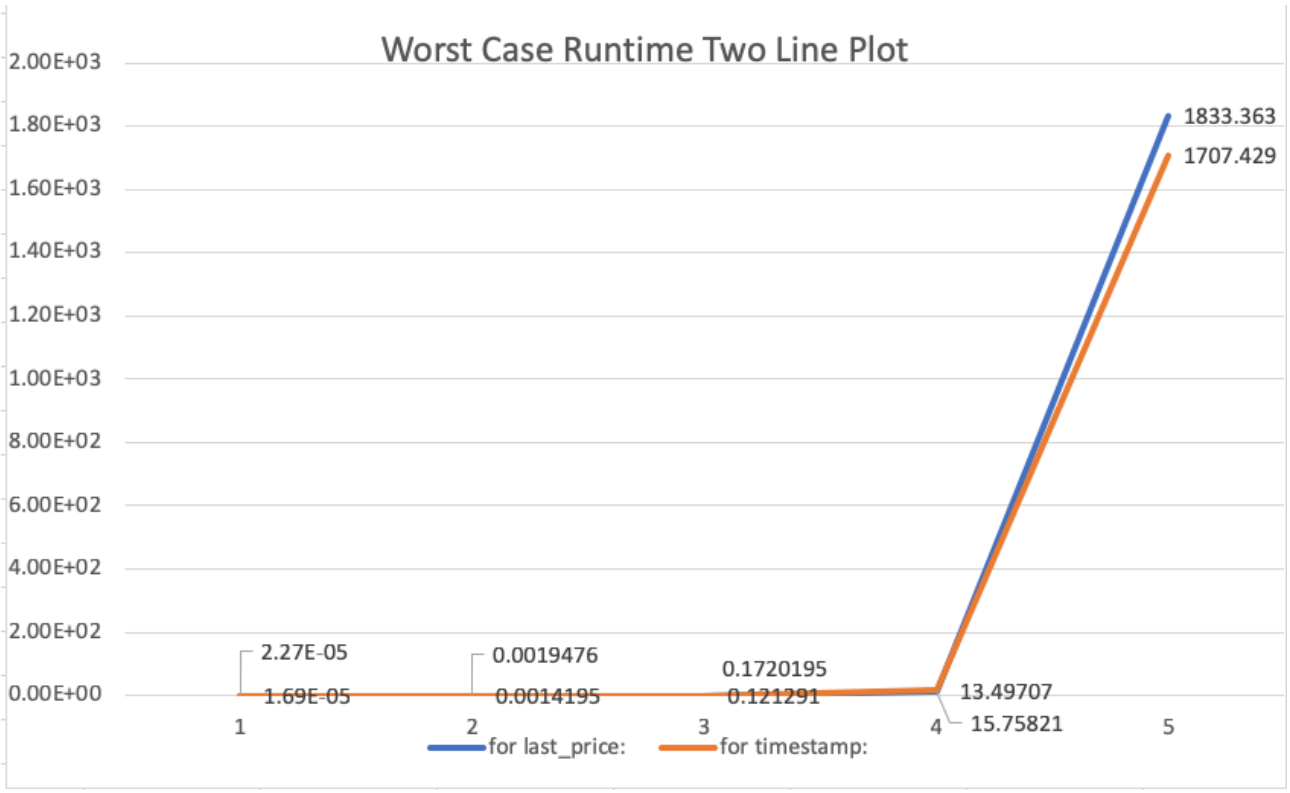
c)
Worst case for QuickSort is O(n^2). We encounter the worst case when our data set is ordered, or reverse ordered. In this case if we choose our pivot from the end or start, this leads that one of our partition will be empty and the other will contain n-1 elements. Thus, I will make a sorted data set and execute my program.

For timestamp:

	10	100	1000	10000	100000
WorstCase:	2.30E-05	0.001723	0.172678	15.0637	1707.68
for timestamp:	1.90E-05	0.00197	0.171181	15.5154	1706.98
	2.00E-05	0.001645	0.165041	15.971	1707.74
	2.10E-05	0.001427	0.161079	15.8909	1706.98
	2.10E-05	0.001775	0.16318	16.0603	1708
	2.10E-05	0.001579	0.169811	15.5894	1707.23
	3.40E-05	0.004392	0.170162	15.8406	1706.99
	2.00E-05	0.001834	0.174471	15.8445	1708.01
	2.10E-05	0.001577	0.204043	15.9218	1707.68
	2.70E-05	0.001554	0.168549	15.8845	1707
Average:	2.27E-05	0.0019476	0.1720195	15.75821	1707.429

For last_price:

	10	100	1000	10000	100000
WorstCase:	1.50E-05	0.001157	0.001361	11.7758	1833.09
for last_price:	1.60E-05	0.001329	0.135034	12.954	1833.24
	1.60E-05	0.00131	0.13492	12.6051	1832.98
	1.50E-05	0.001204	0.135511	13.3637	1832.72
	2.10E-05	0.001212	0.130443	13.9336	1834.04
	1.50E-05	0.002772	0.139872	13.3849	1833.09
	1.70E-05	0.001137	0.132014	13.5779	1832.78
	1.50E-05	0.00139	0.139327	13.8997	1833.04
	2.30E-05	0.001323	0.137792	16.3688	1833.67
	1.60E-05	0.001361	0.126636	13.1072	1834.98
Average:	1.69E-05	0.0014195	0.121291	13.49707	1833.363



In this plot, we can see that Quick sort works at its worst case for both timestamp and last_price values because of the sorted data input. Worst case of Quick sort is O(n^2).

My solution to overcome the worst case of Quick Sort would be shuffle my input data before calling the Quick sort. By shuffling my data randomly, I would reduce the possibility of getting the higher or lower element as a pivot. Another solution would be choosing the pivot randomly from data set.

d)

No Quick sort is not a stable algorithm. Stable algorithm means; if two elements with equal keys appear in the same order in sorted output as they appear in the input data. Since, quick sort puts all elements greater than pivot to the right and smaller ones to the left of the pivot, that makes the quick sort unstable.

For example,

data: 3,1,2,2,**2**,1,2,3,1

2 is the pivot (bold)

sorted: 1,1,1,2,2,2,**2**,3

Here we can see that Quick sort is unstable, since the 2's are in a different order in the output than our original data.