Report

NAME: BURAK ŞEN
ID: 150170063

a) Write down the asymptotic upper bound for the Quicksort for best case, worst case and average case. Prove them solving the recurrence equations.

In the asymptotic upper bound we have to use Big-O notation.

BEST CASE:

In the best case scenario, when we doing partition if our list divide by nearly two half, our best case scenario occurs. We can express this with this. $T(n) = 2T(n/2) + O(n)$, $O(n)$ has to be in every iteration for looking up the items. $2T(n/2)$ comes from two equal half pieces. We can solve this with master theorem.

According to master theorem.

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d).$$

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

$T(n) = 2T(n/2) + O(n)$'s parameters are like this, a = 2, b = 2 and d = 1, and if we evaluate according to these parameters, we can see our equation is $T(n) = O(nlogn)$ as we can see quick sort algorithm in the best case scenario takes O(nlogn) time to sort our list.

WORTS CASE:

In the worst case scenario, if our list line up in a way that every iteration we have to look up previous iterated items minus 1. So we have to recurse our function with $n - 1$. And we now can say this $T(n) = T(n - 1) + O(n)$, $O(n)$ has to be in every iteration for looking up the items. Then we can solve $n - 1$ with series.

$\sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n}(n) - \sum_{i=1}^{n}(i) = n^2 - \frac{n*(n+1)}{2} = \frac{n^2-n}{2}$ which is,

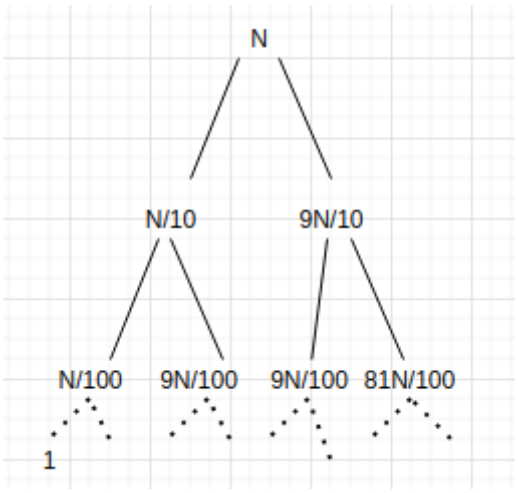
$T(n) = O(\frac{n^2-n}{2}) + O(n)$ and we can arrange like this, $T(n) = O(\frac{n^2}{2} - \frac{n}{2}) + O(n)$ and we can ignore $n / 2$ and O(n) from $n^2 / 2$ we can ignore ½ and finally we have this;

$T(n) = O(n^2)$ as we can see quicksort algorithm in the worst case scenario takes $O(n^2)$ time to sort our list.

AVERAGE CASE:

In the avarge case scenario, if our pivot ends up between the middle (best case scenario) and one of the edges (the worst case scenario) we can calculate our average case.

If we assume our pivot always ends up at least 10% from either edge, we can create this tree.



As we can see that for each level our subproblem number is N then we can calculate from this how many levels this tree has. We could calculate this if we came from the subproblem with size of 1 from 1 if we multiply each level with 10 we can reach to N, which gives us;

$10^x = N$ and if we calculate x from this, we have $x = \log_{10}N$ for the left side of the problem. If we want to calulate for the right side of the problem, we have to calculate this;

$(10/9)^x = N$ and if we calculate x from this, we would have $x = \log_{(10/9)}N$;

If we look at the number nodes in the first $\log_{10}N$ levels there would be N nodes in the levels, for the remaining levels the number of nodes would be fewer than n nodes and its partition time would be fewer than cn therfore we there are $\log_{(10/9)}N$ levels.

We can ommit some part of the $\log_{(10/9)}N$;

If we look at this,

$\log_{(10/9)}N = \frac{\log_2(N)}{\log_2(10/9)}$ we can see in here there is a constant value that we can ommit. If we ommit this constant value we would have $\log_2(N)$.

For each level there is N subproblem which we can calculate our Big O with this information. If we calculate it our Big O is $O(N\log N)$.

b) In implementation, we wanted to sort the sales by alphabetical order of country names and then by their total profits. Let's assume that we are having this kind of method:

1) Sort the sales.txt data by the total profits and write it into sorted_by_profits.txt

2) Sort the sorted_by_profits.txt data according to country names using QuickSort

Does this solution give us the desired output for all cases?

1. Explain why or why not and give a simulation on a small fraction from the dataset (you may modify the results if necessary).

If we start sorting from the total profits and after this we sort the output according to country names, this would give us desired output. If we sorted first according to country names and after this if we sort according to total profit we couldn't have the desired output because if we sort the total profit lastly it would be the primary comparison and country name would be second comparison. And our list would be sorted by total profits and if two total profits are equal they would be sorted by country name this time.

I will explain this with an example;

If we have a data set like this,

Not Sorted		First Sorted by country name		First Sorted by profit	
Country name	Profit	Country Name	Profit	Country Name	Profit
a	1	a	3	a	3
a	2	c	3	a	2
b	1	d	3	a	1
d	2	a	2	b	1
c	1	c	2	c	3
a	3	d	2	c	2
c	3	a	1	c	1
d	3	b	1	d	3
c	2	c	1	d	2

As we can see that if we sort our list firstly by country name it won't sort as we desired. But if we sort like in the question we can have the desired output.

2. Give 3 examples for the sorting algorithms that give the desired output.

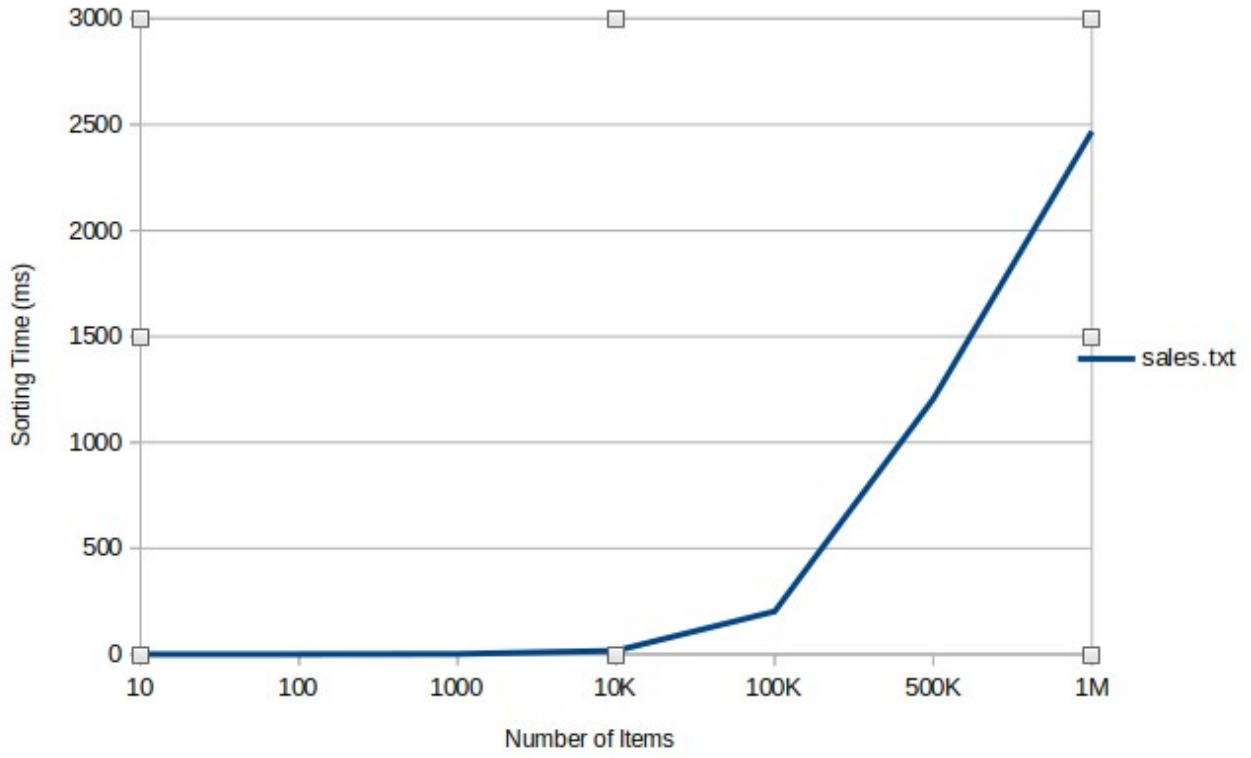
Mergesort, Insertion Sort, Counting Sort

c)

Average Execution Time

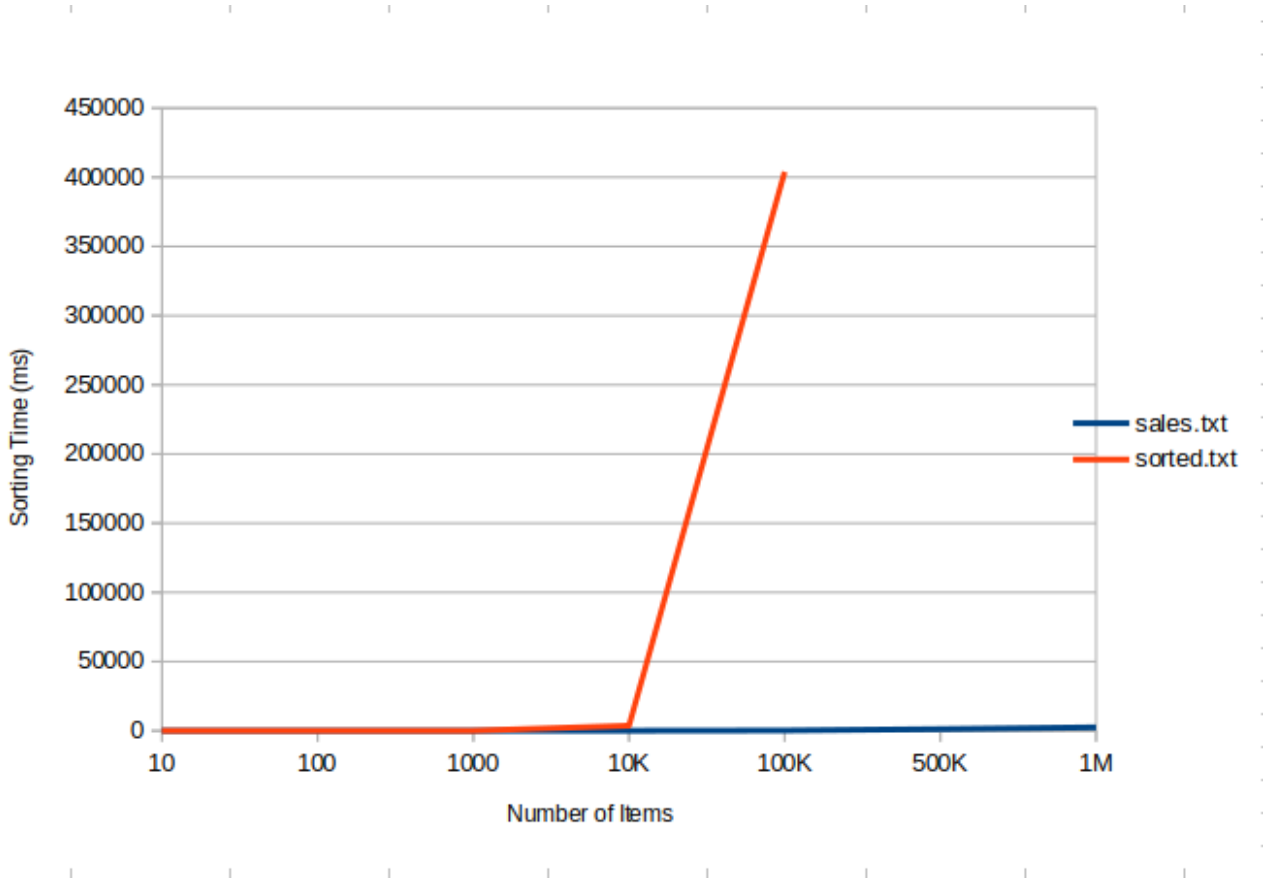
10 items	100 items	1000 items	10K items	100K items	500K items	1M items
0,0139669 ms	0,2339742 ms	1,154901 ms	16,56555 ms	202,3193 ms	1204,807 ms	2467,075 ms

Execution time plot



When we look at the results we can see that our average times nearly linear to the number of items. We can clearly see that it is not linear but it closer to linear from an O(n^2) algorithm.

d)



1)

As we can see that our quicksort algorithm did not work well in the sorted list, if we can look more deeply to this problem we can easly see that when we try to quicksort algorithm on the sorted list our time will be $O(n^2)$ which we know this from the worst case scenario which is if we recurse our function n – 1 times it would happen why it is recurse n – 1 times, for each recursion we take the pivot the last item, if we take the last item as a pivot and if all the list is sorted it has to be looking for the all comparisons for unsorted items. However there is no unsorted items for sorted list but the algorithm does not know that thus it has to finish all the comparisons, but in the sales list we can see that it takes the same time with average case O(nlogn). It is an ordinary execution it has to be an average time to sort.

2)

If we sort the sorted list in the reverse we can see a similar result for the sorted list, for the sales list we can improve with pivot's location if pivot is first or the last item we can see similar results but if pivot is in the middle we could see maybe better results.

3)

If we can choose pivots other then the last or first items we can improve and solve this problem, our sorting time would be an average case.