Beyza Ozan
ID: 150180056

BLG 335E – Analysis of Algorithms I

Homework 1

Part 2. Report

a)

The asymptotic upper bound for **best case** -> Each time partitioning is done, each subarray contains n/2 of elements from previous call. So we choose pivot as n/2.

Recurrence equation -> $\mathbf{T(n) \leq 2T(n/2) + \Theta(n)}$

Solving this recurrence by Master Method. According to this method a = 2 , b = 2 , d = 1.

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

In order to $a = b^d (2 = 2^1)$, it satisfies case 1 and the result $O(n^d log(n))$.

–> $T(n) = O(nlogn)$

The asymptotic upper bound for **worst case** -> Each time partitioning is done, one subarray contains n -1 of n elements from previous call and the other is empty. So we choose pivot as n-1.

Recurrence equation -> $\mathbf{T(n) = T(n\text{-}1) + T(0) + \Theta(n) = T(n\text{-}1) + \Theta(n)}$

Solving recurrence by iteration,

T(n) = Θ(n) + T(n-1)

= Θ(n) + Θ(n-1) + Θ(n-2) + ... + Θ(1)

$$= \sum_{k=1}^{n} \Theta(k)$$

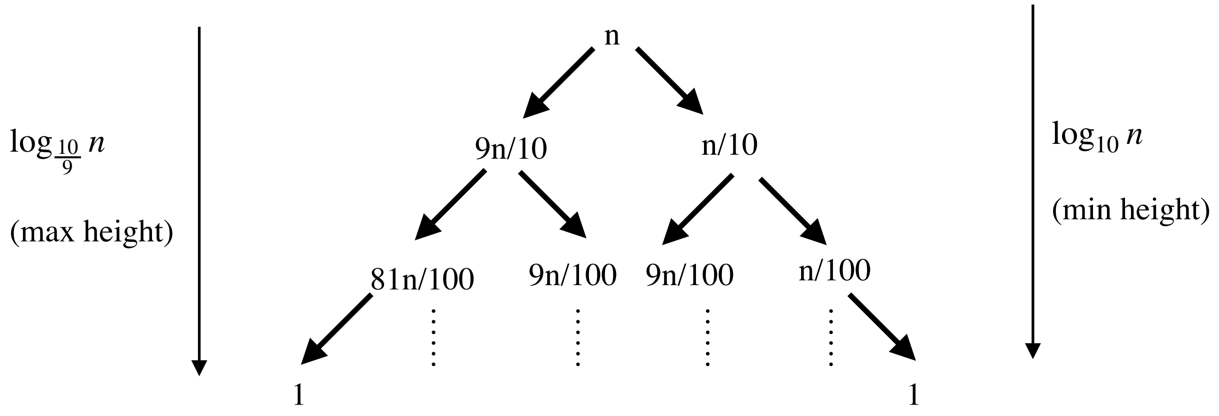
$$= \Theta(\sum_{k=1}^{n} k) = \Theta\left(\frac{n\,.\,(n+1)}{2}\right)$$

$$= \Theta(n^2)$$

The asymptotic upper bound for **average case** ->Suppose split is always 9-to-1.

Recurrence equation -> **T(n) ≤ T(9n/10) + T(n/10) + Θ(n) = T(9n/10) + T(n/10) + cn**

Solving recurrence by recursion tree,



Due to we get a 9-to-1 split, one side gets 9n/10 elements and the other side n/10 elements. The right child of each node represents a subproblem size 1/10 as large, and the left child represents a subproblem size 9/10 as large. Since the smaller subproblems are on the right, by following a path of right children, we get from the root down to a subproblem size of 1 faster than along any other path. As the figure shows, after $\log_{10} n$ levels, we get down to a subproblem size of 1 .
Since the larger subproblems are on the left, by following a path of left children, we get from the root down to a subproblem size of 1 slower than any other path. The figure shows that it take $\log_{\frac{10}{9}} n$ levels to get down to a subproblem of size 1. each left child is 9/10 of the size of the node above it (its parent node). Therefore T(n) is greater than and equal to $n x \log_{10} n$ **.** Also it is smaller than and equal to $n x \log_{\frac{10}{9}} n$.

$$n \log_{10} n \leq \quad \text{T(n)} \quad \leq n \log_{\frac{10}{9}} n$$

$$T(n) \leq n \frac{\log n}{\log \frac{10}{9}} \quad \text{Constant}$$

$$T(n) = O(nlogn)$$

b) Stable sort is the relative order of equal sort items is not preserved. QuickSort are not a stable sort. Our problem is to ask us to sort by two different variables one by one. Therefore this assuming method not give us a desired solution since first we sort the data by total profits, after then sort by country names (you will see the example below). To sum up, implementation and assuming result differ from each other because of QuickSort is unstable sorting algorithm.

In implementation we sorted the sales by **alphabetical order of country names** and then by their **total profits**. Just below is a piece of code how the data are sorted in this way. Next to it is a part of the sorted version that is written on the "*sorted.txt*" file:

```
bool larger(myset first, myset second){
    if(first.country > second.country){
        return true;
    }
    else if(first.country < second.country){
        return false;
    }
    else{
        if(first.float_total <= second.float_total)
            return true;
        return false;
    }
}
```

The larger function which provides us compare data in order to sort by country names and then by their total profits.

```
sorted.txt
Country	Item Type	Order ID	Units Sold	Total Profit
Algeria	Cosmetics	761723172	9669	1681149.03
Djibouti	Clothes	880811536	562	41273.28
Dominican Republic	Baby Food	824714744	274	26265.64
Ethiopia	Cosmetics	807785928	662	115101.94
France	Cosmetics	324669444	5758	1001143.46
Ghana	Office Supplies	601245963	896	113120
Haiti	Office Supplies	485070693	2052	259065
Morocco	Clothes	667593514	4611	338631.84
Nicaragua	Household	573998582	7791	1291202.43
Papua New Guinea	Clothes	647164094	9092	667716.48
Papua New Guinea	Meat	940995585	360	20592
Seychelles	Beverages	425793445	597	9349.02
Singapore	Snacks	176461303	7676	423254.64
Slovakia	Beverages	174590194	3973	62217.18
South Africa	Fruits	443368995	1593	3839.13
Sri Lanka	Fruits	830192887	1379	3323.39
Taiwan	Fruits	732588374	8034	19361.94
Tanzania	Cosmetics	739008080	7768	1350622.16
Tanzania	Beverages	659878194	1476	23114.16
The Bahamas	Personal Care	246248090	9137	228973.22
Turkmenistan	Vegetables	116205585	6670	421077.1
Uganda	Personal Care	539471471	451	11302.06
United Kingdom	Cosmetics	135178029	1038	180477.06
Vietnam	Personal Care	314505374	7984	200079.04
Zimbabwe	Office Supplies	953361213	9623	1214903.75
```

Figure 1

1) Now let's sort the *sales.txt* data by the **total profits** by adding the new function in the code and write it into *sorted_by_profits.txt* :

```
bool larger1(myset first, myset second){

    if(first.float_total <= second.float_total)
        return true;
    else if(first.float_total > second.float_total)
        return false;
    return false;
}
```

The larger1 function which provides us compare data in order to sort by their total profits.

```
sorted_by_profits.txt
Country	Item Type	Order ID	Units Sold	Total Profit
Algeria	Cosmetics	761723172	9669	1681149.03
Tanzania	Cosmetics	739008080	7768	1350622.16
Nicaragua	Household	573998582	7791	1291202.43
Zimbabwe	Office Supplies	953361213	9623	1214903.75
France	Cosmetics	324669444	5758	1001143.46
Papua New Guinea	Clothes	647164094	9092	667716.48
Singapore	Snacks	176461303	7676	423254.64
Turkmenistan	Vegetables	116205585	6670	421077.1
Morocco	Clothes	667593514	4611	338631.84
Haiti	Office Supplies	485070693	2052	259065
The Bahamas	Personal Care	246248090	9137	228973.22
Vietnam	Personal Care	314505374	7984	200079.04
United Kingdom	Cosmetics	135178029	1038	180477.06
Ethiopia	Cosmetics	807785928	662	115101.94
Ghana	Office Supplies	601245963	896	113120
Slovakia	Beverages	174590194	3973	62217.18
Djibouti	Clothes	880811536	562	41273.28
Dominican Republic	Baby Food	824714744	274	26265.64
Tanzania	Beverages	659878194	1476	23114.16
Papua New Guinea	Meat	940995585	360	20592
Taiwan	Fruits	732588374	8034	19361.94
Uganda	Personal Care	539471471	451	11302.06
Seychelles	Beverages	425793445	597	9349.02
South Africa	Fruits	443368995	1593	3839.13
Sri Lanka	Fruits	830192887	1379	3323.39
```

Figure 2

2) Let’s sort the *sorted_by_profits.txt* data according to country names by adding the new function in the code and write it into the *sorted_by_names.txt* :

```cpp
bool larger2(myset first, myset second){
    if(first.country > second.country)
        return true;
    return false;
}
```

The larger2 function which provides us compare data in order to sort by their country names.

sorted_by_names.txt

Country	Item Type	Order ID	Units Sold	Total Profit
Algeria	Cosmetics	761723172	9669	1681149.03
Djibouti	Clothes	880811536	562	41273.28
Dominican Republic	Baby Food	824714744	274	26265.64
Ethiopia	Cosmetics	807785928	662	115101.94
France	Cosmetics	324669444	5758	1001143.46
Ghana	Office Supplies	601245963	896	113120
Haiti	Office Supplies	485070693	2052	259065
Morocco	Clothes	667593514	4611	338631.84
Nicaragua	Household	573998582	7791	1291202.43
Papua New Guinea	Meat	940995585	360	20592
Papua New Guinea	Clothes	647164094	9092	667716.48
Seychelles	Beverages	425793445	597	9349.02
Singapore	Snacks	176461303	7676	423254.64
Slovakia	Beverages	174590194	3973	62217.18
South Africa	Fruits	443368995	1593	3839.13
Sri Lanka	Fruits	830192887	1379	3323.39
Taiwan	Fruits	732588374	8034	19361.94
Tanzania	Beverages	659878194	1476	23114.16
Tanzania	Cosmetics	739008080	7768	1350622.16
The Bahamas	Personal Care	246248090	9137	228973.22
Turkmenistan	Vegetables	116205585	6670	421077.1
Uganda	Personal Care	539471471	451	11302.06
United Kingdom	Cosmetics	135178029	1038	180477.06
Vietnam	Personal Care	314505374	7984	200079.04
Zimbabwe	Office Supplies	953361213	9623	1214903.75

Figure 3

Now if we look at orders of “Papua New Guinea” both in figure1 and figure3, the difference in the sorting is clearly seen.

In the assuming method, we sorted first by total profits , then by country name. Since QuickSort is an unstable algorithm, it makes no sense to sort by profits first. While sorting by country names in step 2, countries with the same name can be sorted in random order.

b - 2) Examples for sorting algorithms:

1- Insertion Sort

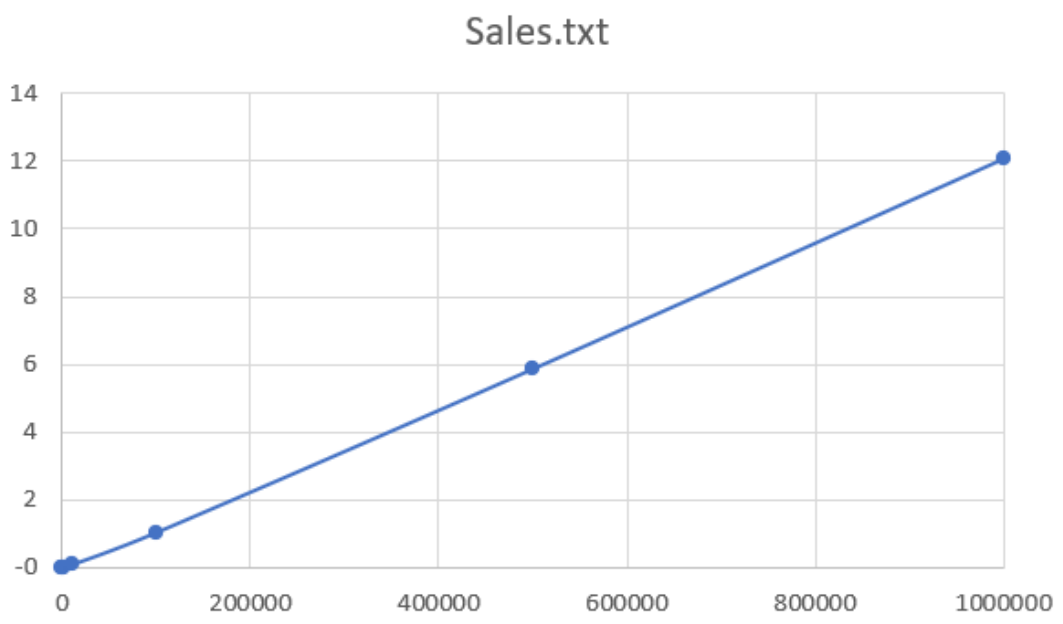
2 - Merge Sort

3 - Radix Sort

These sort algorithms give us a desired input since they are stable sort algorithms.

c)

sales.txt											
N	**1.time**	**2.time**	**3.time**	**4.time**	**5.time**	**6.time**	**7.time**	**8.time**	**9.time**	**10.time**	**Average time of running**
10	0,000039	0,000046	0,000040	0,000029	0,000039	0,000023	0,000040	0,000047	0,000049	0,000046	0,0000398
100	0,000364	0,000634	0,000745	0,000770	0,000708	0,000633	0,000527	0,000422	0,000643	0,000493	0,0005939
1000	0,006659	0,008440	0,007727	0,007664	0,007093	0,007573	0,005961	0,008239	0,007728	0,007090	0,007417
10K	0,078566	0,079142	0,077482	0,078048	0,077791	0,078136	0,086477	0,078155	0,077847	0,079863	0,079151
100K	1,031865	1,033589	1,029941	1,039618	1,031138	1,033667	1,035203	1,029902	1,030030	1,043578	1,033853
500K	5,826004	5,836720	5,833702	5,922258	5,839569	5,981344	5,904358	5,833580	5,936765	5,840868	5,875516
1M	11,734048	11,924064	11,919458	12,298997	12,461699	11,973245	12,110600	12,026571	11,968876	12,163543	12,094117

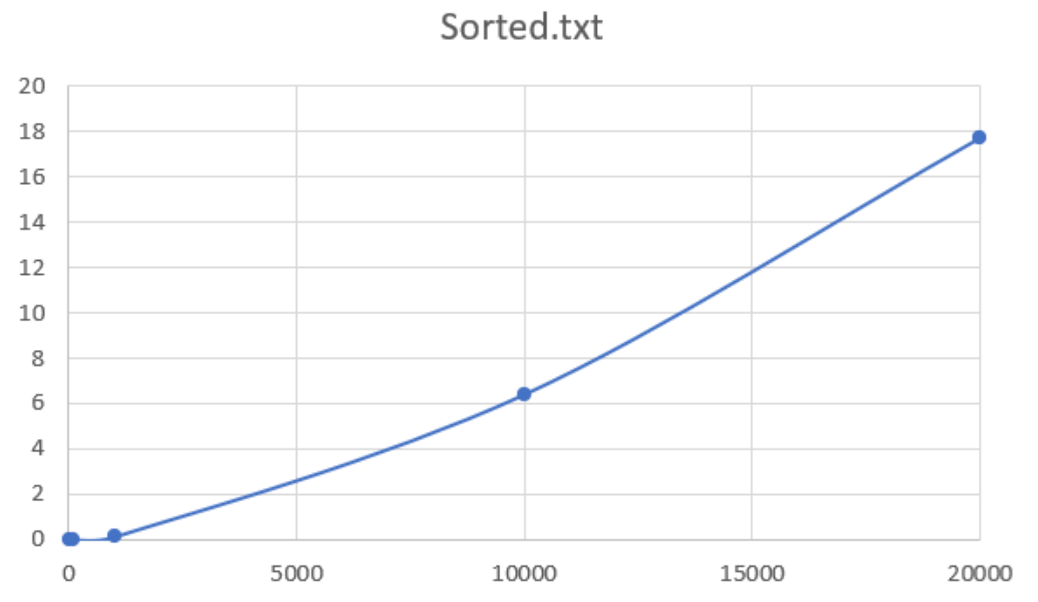


Since the inputs are in random order in the sales.txt file, this case can be the average case. We know from (a) that the asymptotic upper bound for the average case is $O(nlogn)$. Let's try to prove that this function is close to $nlogn$, by looking at the table and graph. When **N = 10K** , running time is approximately **1 seconds**. Then looking at **N = 1M**, running time is approximately **12 seconds**. If we rate these numbers, we see that when N goes up **10** times, running time goes up **12** times. Therefore, it can be said that our function grows by a factor of $nlogn$ and the time complexity of our algorithms is $O(nlogn)$ in this case.

d)

sorted.txt											
N	**1.time**	**2.time**	**3.time**	**4.time**	**5.time**	**6.time**	**7.time**	**8.time**	**9.time**	**10.time**	**Average time of running**
10	0,000026	0,000084	0,000053	0,000046	0,000053	0,000053	0,000054	0,000052	0,000084	0,000052	0,000056
100	0,001096	0,001147	0,001809	0,001841	0,001973	0,001288	0,001744	0,001340	0,001955	0,001891	0,0016084
1000	0,110996	0,113623	0,109847	0,116155	0,113731	0,120004	0,114867	0,112754	0,123643	0,110414	0,1146034
10K	6,376890	6,434521	6,450395	6,472135	6,296743	6,505078	6,397697	6,391833	6,348560	6,281254	6,3955106
20K	16,851109	17,909389	17,847342	17,689426	17,566772	17,753107	18,068720	17,718321	17,936104	17,590027	17,6930317
100K	–	–	–	–	–	–	–	–	–	–	–
500K	–	–	–	–	–	–	–	–	–	–	–
1M	–	–	–	–	–	–	–	–	–	–	–

(Since recursion is highly branched , 100K, 500K and 1M was not written.)



1) In QuickSort Algorithm, if the pivot is chosen the biggest or the smallest item, the worst case occurs. We know from (a), the asymptotic upper bound for the worst case is $O(n^2)$.
When we try to sort the data that is already sorted, the largest item would be chosen.
Therefore, sorting sorted.txt is the worst case ($O(n^2)$).
It is obviously seen on the tables and graphs, if we compare the running times in (c) and (d) at the same values of N, (d) is more slower than (c). For instance, When **N = 10K**, average running time is **0.079151 seconds** in the *sales.txt* table (c) and **6.281254 seconds** in the *sorted.txt* table (d).
To summarize, sorting *sales.txt* has an average case and its time complexity is $O(nlogn)$, on the other hand, sorting *sorted.txt* has a worst case and its complexity is $O(n^2)$.

2) If all inputs in the file are the same, it is the worst case again.

3) Choosing the pivot **randomly** is the solution for this case.