Name: Başar Demir

Student Number: 150180080

BLG-335E Homework-1 Report

a) Asymptotic Upper Bounds for The Quicksort

While determining the time complexity of the cases, I use Master Theorem that is given below. If the recurrence function is

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

time complexity of our algorithm,

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

- **Best Case**

The best case of the Quicksort algorithm is partitioning the array from the middle for each iteration by choosing the element that is closest to median as a pivot. By this way, we can divide our problem into two equal sub-problems. Partitioning takes $O(n)$ time complexity because of the iteration through the array and arranging respect to pivot. Then, our recurrence function will be

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n) = 2 * T\left(\frac{n}{2}\right) + O(n)$$

We can observe that $a = 2,\ b = 2,\ d = 1$. Thanks to Master Theorem, the time complexity of the best-case Quicksort is,

$$\boldsymbol{T(n) = O(n * log(n))}$$

Proof by Substitution Method:

We have a function that gives the time complexity of our best-case algorithm. To proof the correctness of the function, we can use substitution method that can be performed by changing inner function calls with their equivalent result. We know that Quicksort's time complexity for 1 length input is 1. Therefore, we need to substitute functions until we obtain $T(1)$.

Our initial function → $T(n) = 2 * T\left(\frac{n}{2}\right) + n$

If we substitute → $T\left(\frac{n}{2}\right) = 2 * T\left(\frac{n}{4}\right) + \frac{n}{2}$

$$T(n) = 2 * \left(2 * T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 4 * T\left(\frac{n}{4}\right) + 2n$$

If we substitute → $T\left(\frac{n}{4}\right) = 2 * T\left(\frac{n}{8}\right) + \frac{n}{4}$

$$T(n) = 4 * \left(2 * T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n = 8 * T\left(\frac{n}{8}\right) + 3n$$

For the k^{th} step of the substitution, our formula will be

$$T(n) = 2^k * T\left(\frac{n}{2^k}\right) + kn$$

We only know the value of the $T(1) = 1$, therefore we need to step that satisfies $\frac{n}{2^k} = 1$ and it is,

$$\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log_2(n)$$

Therefore, our time complexity function will equal to:

$$T(n) = 2^{\log_2(n)} * T\left(\frac{n}{2^{\log_2(n)}}\right) + \log_2(n) * n = \boldsymbol{n + \log_2(n) * n}$$

$$T(n) = O(n + \log_2(n) * n) = \boldsymbol{O(n * log(n))}$$

Result of the Master Theorem is **satisfied.**

- **Worst Case**

The worst case of the Quicksort algorithm is partitioning the array from the boundaries for each iteration by choosing the element that is highest or smallest of array as a pivot. This situation provides to decrease the problem size with just 1. After partitioning operation, we get (n+1) sized sub-problem, 1 pivot. Then, our time complexity will be

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$$

$$\boldsymbol{T(n) = O(n^2)}$$

Proof by Substitution Method:

To prove the correctness of the time complexity that is given above, I will use Substitution Method again.

Our initial function → $T(n) = T(n-1) + n$

If we substitute → $T(n-1) = T(n-2) + n - 1$

$$T(n) = T(n-2) + (n-1) + (n)$$

If we substitute → $T(n-2) = T(n-3) + n - 2$

$$T(n) = T(n-3) + (n-2) + (n-1) + (n)$$

For the k^{th} step of the substitution, our formula will be

$$T(n) = T(n-k) - \sum_{i=0}^{k-1} (n-i)$$

We only know the value of the $T(1) = 1$, therefore we need to step that satisfies $n - k = 1$ and it is,

$$n - k = 1 \rightarrow k = n - 1$$

Therefore, our time complexity function will equal to:

$$T(n) = T(1) + \sum_{i=0}^{n-2}(n-i) = 1 + \sum_{i=0}^{n-2}(n-i)$$

It represents sum of all numbers from 1 to n and its formula is,

$$T(n) = \frac{(n)(n+1)}{2} = \frac{n^2+n}{2}$$

With the big-O notation, its upper bound converges to n^2

$$\boldsymbol{T(n) = O(n^2)}$$

Therefore, we have **proven** our result.

- **Average Case**

While calculating average case of the algorithms, we have to consider all possible cases and their time complexities. We can choose pivot from n possible elements. Therefore, probability of an element being a pivot is $1/n$. We are diving our problem into two sub-problems using these pivots. Average case complexity function will consist of time complexities of all pivot combinations divided by number of the pivots that is equal to length of array:

$$T_{average}(n) = \frac{\sum_{i=0}^{n-1}[T(i)+T(n-i-1)+n]}{n} = \frac{\sum_{i=0}^{n-1}[T(i)+T(n-i-1)]}{n} + n$$

$$T_{average}(n) = \frac{1}{n}\sum_{i=0}^{n-1}(T(i)+T(n-i-1)) + O(n)$$

Element that are inside of the sum operation are same, both of them sums numbers from 0 to n-1. So, we can write equation as

$$T_{average}(n) = \frac{2}{n}\sum_{i=0}^{n-1}T(i) + O(n)$$

To solve this problem, we should get rid of summation operation. To provide this we can use another n, because this equation holds for all possible n values.

$$n * T_{average}(n) - n^2 = 2\sum_{i=0}^{n-1} T(i)$$

$$(n-1) * T_{average}(n-1) - (n-1)^2 = 2\sum_{i=0}^{n-2} T(i)$$

If we subtract equations,

$$n * T(n) - n^2 - (n-1)T(n-1) + (n-1)^2 = \ 2\,T(n-1)$$

$$n * T(n) - (n-1)T(n-1) - 2n + 1 = \ 2\,T(n-1)$$

$$T(n) = \ \frac{(n\ +\ 1)T(n\ -\ 1) + \ 2n - 1}{n}$$

$$T(n) = \ \frac{(n\ +\ 1)T(n\ -\ 1)}{n} + 2 - \frac{1}{n} \to Recurrence\ function$$

To solve this recurrence function, we should expand our function until reach to $T(1)$ as I have done in previous questions and sum them. Because we are searching for the upper bound we can eliminate $\frac{1}{n}$ term and simplify our equations. If we write all equations from n to 1 and sum them all, we can obtain:

$$\frac{T(n)}{n+1} = \cancel{\frac{T(n-1)}{n}} + \frac{2}{n+1}$$

$$\cancel{\frac{T(n-1)}{n}} = \cancel{\frac{T(n-2)}{n-1}} + \frac{2}{n}$$

$$\cancel{\frac{T(n-2)}{n-1}} = \cancel{\frac{T(n-3)}{n-2}} + \frac{2}{n-1}$$

$$\vdots$$

$$\frac{T(n)}{n+1} = 2\sum_{i=2}^{n+1} \frac{1}{i} \approx 2\int_{2}^{n+1} \frac{1}{x}\, dx \approx 2\ln(n) \approx 2\log(n)$$

$$T_{average}(n) \approx 2\log(n)\,(n+1) \approx \boldsymbol{n * log(n)}$$

Finally, we obtain average case as $\boldsymbol{n * log(n)}$ which is same as Best Case approximation.

b) <u>Desired Outputs</u>

For this part, I have implemented a code that sorts sales according to total profits (decreasing order) and I have sorted it according to their county names (increasing order). I have observed that for some cases that includes same countries for several times gives wrong output.

1) I caught the wrong answer when I run simulation with N=100. I realized that Ghana items sorted differently then what we have expected. The wrong output occurs because of the pivot selection. In this homework, I chose last element of array as a pivot then, I rearrange the array according to values of the elements. If element has smaller value, it placed before the pivot but for other conditions, it placed after the pivot.
   Assume that in the first partition of the merge sort, the last element also pivot has shares same name with at least 1 element. When we perform partition, other elements that has same name will located in right hand side of the pivot because of the mechanism of Quicksort. But pivot has smallest total profit (it was last element of array) and it

placed before the elements that have higher total profit. Therefore, this case causes a wrong answer in results. You can find the simulation below.

Sorted text file according to total profits:

```
test.txt
Country Item Type    Order ID     Units Sold  Total Profit
Liberia Baby Food    146634709    1324     126918.64
Ethiopia     Cosmetics    807785928    662 115101.94
Ghana    Office Supplies 601245963    896 113120.00
United States of America     Personal Care    190777862    4264     106855.84
Ghana    Fruits   496523940    1323     3188.43
```

After sort operation according to country names:

```
test-result.txt
Ethiopia     Cosmetics    807785928    662 115101.94
Ghana    Fruits   496523940    1323     3188.43
Ghana    Office Supplies 601245963    896 113120.00
Liberia Baby Food    146634709    1324     126918.64
United States of America     Personal Care    190777862    4264     106855.84
```

The sorted profit array:

Liberia- 126918.64	Ethiopia - 115101.94	Ghana - 113120.00	USA- 106855.84	Ghana - 3188.43

Choose last element as pivot (position=0):

Liberia- 126918.64	Ethiopia - 115101.94	Ghana - 113120.00	USA- 106855.84	Ghana - 3188.43

Compares Ghana and Liberia (Liberia is bigger and position = 0):

Liberia- 126918.64	Ethiopia - 115101.94	Ghana - 113120.00	USA- 106855.84	Ghana - 3188.43

Compares Ghana and Ethiopia (Ghana is bigger swap 1-0, position =1):

Ethiopia - 115101.94	Liberia- 126918.64	Ghana - 113120.00	USA- 106855.84	Ghana - 3188.43

Compares Ghana and Ghana (position =1):

Ethiopia - 115101.94	Liberia- 126918.64	Ghana - 113120.00	USA- 106855.84	Ghana - 3188.43

Compares Ghana and USA (USA is bigger, position =1):

Ethiopia - 115101.94	Liberia- 126918.64	Ghana - 113120.00	USA- 106855.84	Ghana - 3188.43

Finishes iteration and swaps position (1) with **pivot** (4):

Ethiopia - 115101.94	Ghana - 3188.43	Ghana - 113120.00	USA- 106855.84	Liberia- 126918.64

Call Quicksort for sub-problems:

Ethiopia - 115101.94	Ghana - 3188.43	Ghana - 113120.00	USA- 106855.84	Liberia- 126918.64
Quicksort			Quicksort	

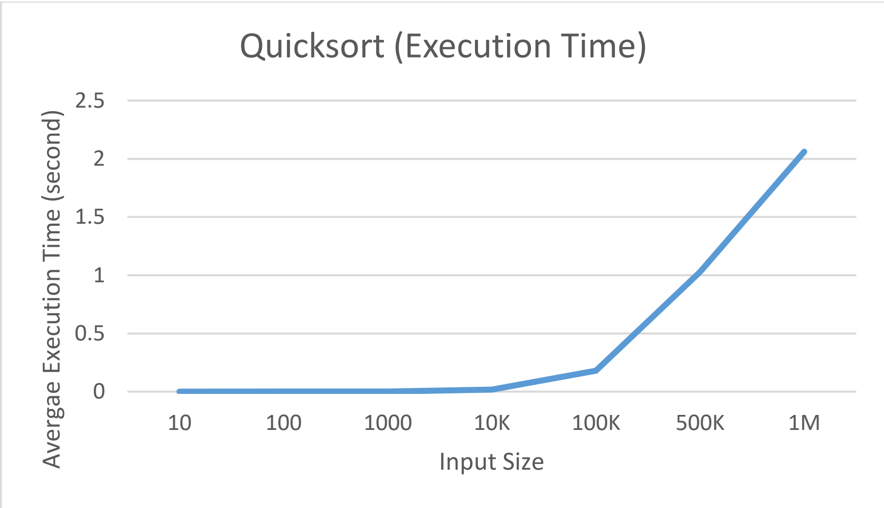
As we have observed in the example that is given above, our pivoting mechanism orders same countries wrong. After that point there **is not any possibility** that places Ghana-113120 before Ghana-3188.43.

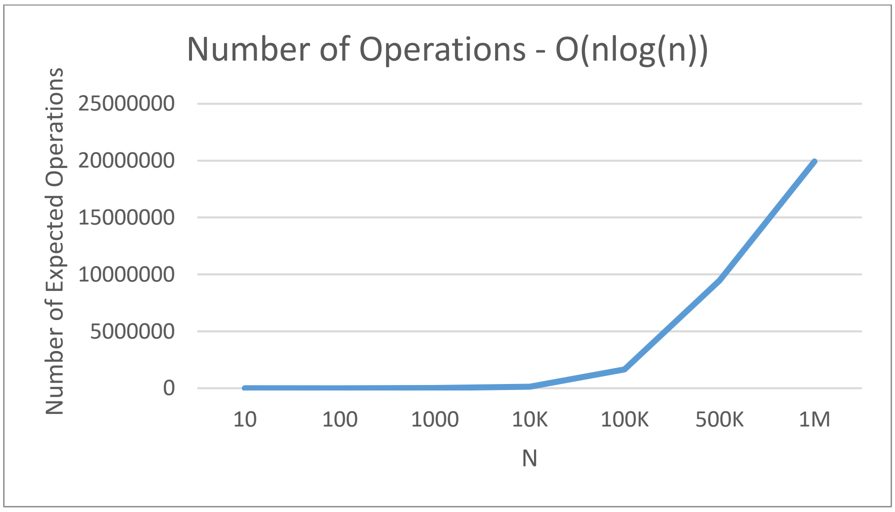
2) MergeSort, BubbleSort, InsertionSort gives desired output for this problem. They do not change orders and does not provides a possibility of changing, if two elements have same priority. Therefore, sorted elements that have same name respect to total profits does not changed orders between them. These type of sorting mechanisms are called as **Stable Sort Algorithms**.

a) <u>Execution Time of The Quicksort</u>

In part-a, I have found that asymptotic upper bound for the Quicksort for average-case as $O(n * log(n))$. As given below, our execution time table gives a plot that is similar to number of operations plot. We can say that our algorithm works at expected complexity.

Input Size	Average Execution Time (seconds)
10	0
100	0.0001
1000	0.0008
10K	0.0167
100K	0.1786
500K	1.0284
1M	2.0614





a) <u>Execution Time of The Quicksort with Sorted Array</u>

1) My deterministic Quicksort chooses last element of the array as a pivot. In sorted array, last element of array is the biggest element and it causes the worst-case of partitioning. In every partitioning, we decrease problem size with 1 and it converges to $O(n^2)$ as I mentioned in worst-case analysis.
2) Reverse sorted array also causes to worst-case approach.
3) We can choose our pivot randomly that decreases probability of getting worst-case pivot but we can never be sure without iterate over the array.

Input Size	Average Execution Time (seconds)
10	0
100	0.0007
1000	0.0421
10K	5.0347
20K	18.2472
30K	40.613