ISE315 – Analysis of Algorithms
Fall 2021
Homework-1 Report

Aral ŞEN
150170217

a) (10 points) Write down the asymptotic upper bound for deterministic Quicksort for best case and worst case. Prove them solving the recurrence equations.

Best case -> occurs generally when the input is distributed random manner

The best-case performance is found when at each recursive level of the quicksort, the pivot is chosen such that it divides the problem to be sorted into two subproblems of exactly the same size. The reccurence for the running time is :

$$T(n) = 2 * T(n/2) + n$$

We can solve this reccurence using Master Method

a = 2, b = 2, f(n) = Θ(n)

$\Theta(n^{\log_2 2}) = \Theta(n)$ **(2nd condition of Master thm.)**

So the complexity will be: $\Theta(n * \log_2 n)$

Worst case -> occurs generally when the input is sorted or reversed sorted

The worst-case occurs when the partitioning step produces one subproblem with (n – 1) elements and one with zero elements since there will be (n – 1) recursive calls to the algorithm that creates subproblems of size n - 1 and 0, and then at the next step produces subproblems of size $n - 2$ and 0, if this unbalanced partitioning continues in each recursive call, the partitioning takes $\theta(n)$ time. The recurrence for the running time is $T(n) = T(n-1) + n$

Solving the reccurence relation using the substitution method:

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

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

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

…

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

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

b) (10 points) 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. Give an example on a small fraction from the dataset proving your answer. (You may choose the samples as you want to simulate your answer).

No, because **QuickSort is an unstable algorithm**. Without considering their original positions, we do swapping of elements according to pivot's position. Example below demonstrates why quicksort is not stable:

Sample data:

Country	Item Type	Order ID	Units Sold	Total Profit
Egypt	Office Supplies	683694493	9249	1167686.25
Turkey	Fruits	435268723	4543	10948.63
Turkey	Vegetables	761274707	6958	439258.54
China	Cosmetics	640492475	4811	836488.57
Azerbaijan	Snacks	955892789	8773	483743.22

If we sort our sample data the way we did at the implementation,

Country	Item Type	Order ID	Units Sold	Total Profit
Azerbaijan	Snacks	955892789	8773	483743.22
China	Cosmetics	640492475	4811	836488.57
Egypt	Office Supplies	683694493	9249	1167686.25
Turkey	Vegetables	761274707	6958	439258.54
Turkey	Fruits	435268723	4543	10948.63

First, sort by profit we get

Country	Item Type	Order ID	Units Sold	Total Profit
Egypt	Office Supplies	683694493	9249	1167686.25
China	Cosmetics	640492475	4811	836488.57
Azerbaijan	Snacks	955892789	8773	483743.22
Turkey	Vegetables	761274707	6958	439258.54
Turkey	Fruits	435268723	4543	10948.63

Then sort the data above according to country names we get

Country	Item Type	Order ID	Units Sold	Total Profit
Azerbaijan	Snacks	955892789	8773	483743.22
China	Cosmetics	640492475	4811	836488.57
Egypt	Office Supplies	683694493	9249	1167686.25
Turkey	Fruits	435268723	4543	10948.63
Turkey	Vegetables	761274707	6958	439258.54

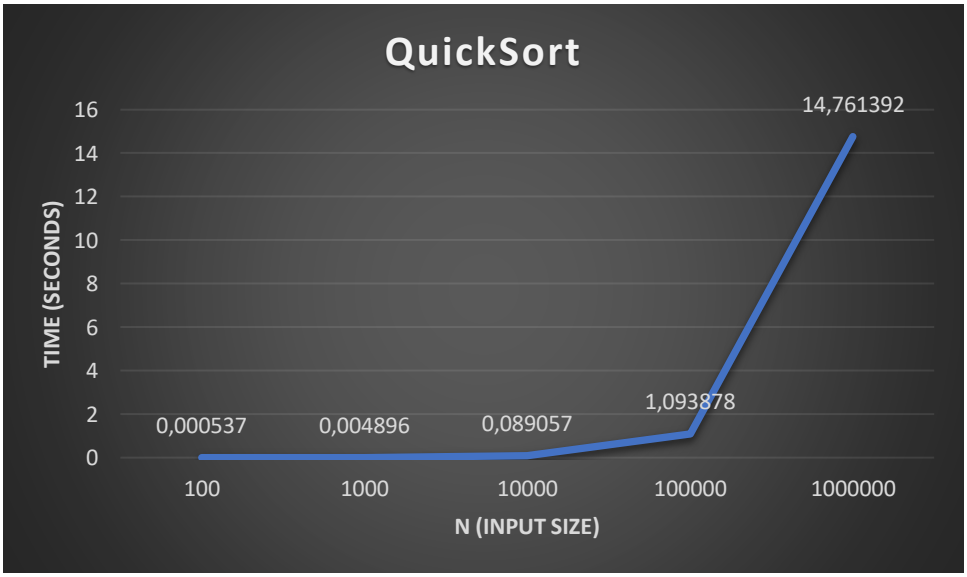
The sorted datas are different thus we can say that quicksort is unstable.

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

Marge Sort, **Counting Sort**, and **Insertion Sort** are the three stable sorting algortihms that give the desired output.

c) (15 points) Run the algorithm for different N values {100, 1000, 10K, 100K, 1M} on sales.txt data and calculate the average time of running for multiple executions (at least 5 times) for each N value. Report the average execution times in a table and prepare an Excel plot which shows the N – runtime relation. Comment on the results considering the asymptotic bound that you have found **in (a). (3-4 sentences)**

Input Size (N)	Elapsed Time
100	**0.000537 seconds**
1000	**0.004896 seconds**
10K	**0.089057 seconds**
100K	**1.093878 seconds**
1M	**14.761392 seconds**



Our input data in sales.txt file is not sorted order, as we stated **in (a),** if we run quicksort on a set of inputs that give good splits, the average run time will be close to best-case. For this case, we could say the run time of the algorithm is $T(n)$ = $\Theta(n * \log_2 n)$

d) (25 points) You get sorted files at the output of Part 1. Run the same Quicksort algorithm from Part 1 for different N values {100, 1000, 10K, 100K, 1M} on sorted.txt data and calculate the average time of running as you have done in (c). Report the results in a table and plot them similarly.

Input Size (N)	Elapsed Time
100	0.002007 seconds
1000	0.195144 seconds
10K	Very large (maximum recursion depth exceed)
100K	Very large (maximum recursion depth exceed)
1M	Very large (maximum recursion depth exceed)

1. Compare the results with the results you have obtained at (c) and explain the difference in detail referring the equations you have given **in (a). (4-5 sentences)**

There is a quite a lot of time difference compared to the results in **(c)** timewise. This happens because my pivot-selection strategy is simply selecting the last element as a pivot, and this causes a worst-case performance when the input is already sorted, in this case, the algorithm runs in T(n) = $\mathbf{O}(n^2)$ time that is why we cannot run our program for N = {10K, 100K, and 1M} inputs since their recursions are really deep, and using up more stack than available (stack overflow).

2. Which other input cases would give us the similar results? **(1 sentence)**

Since it is one of the worst-case scenarios for the quicksort, reversely-sorted input (descending order) would also give us similar results.

3. Which part of the deterministic Quicksort should be changed to prevent this problem? Just explain briefly referring the code written in Part 1, you do not have to implement it. Next, write down the analysis of this changed algorithm. **(Hint: You need to use probabilistic analysis.)**

One of the solutions could be to use randomized Quicksort. We can modify Partition and set pivot the random element in Arr rather than the last element in Arr before partitioning. For every input of length n, the average running time of Quicksort with random pivots is O(n log n)

```
def partition(arr, p, r):
    pivot = random(p, r)   # pivot
```

Assume T(n) is the runtime of our new algorithm:

- Running time of the algorithm is the total running time spent in all Partition calls.
- We call partition n times.
- Every call of Partition takes O(1) time + (time/number of iterations of for-loop)

In each iteration of for-loop we compare an element with the pivot. (A[j] ≤ pivot)

If X is the number of comparisons in Partition over the entire execution of algortihm then we can say that the runtime is O(n + X).

We get,

E[T(n)] = E[O(n + X)] = n + E[X]

To analyze the expected running time we need to compute E[X]. Let X denote the total number of comparisons made by the algorithm. Define random variable X_{ij} to be 1 if the algorithm does compare the ith smallest and jth smallest elements in term of sorting, and 0 if it does not (Here we gonna use Z_{ij} to denote {zi , zi+1, . . . , zj}, Z_i is the ith smallest) since we never compare the same pair of elements twice.

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \text{ where,}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr[Z_i \text{ compared to } Z_j]$$

To compute $\Pr[Z_i \text{ compared to } Z_j]$ it is useful to consider when two elements are not compared. If the pivot we choose is between Z_i and Z_j then these two will never compared. On the other hand, if zi is chosen as pivot before any other element in Z_{ij} then it is compared to each element in Z_{ij} . Similar for zj. Each element of $\{Z_i , Z_{i+1}, \ldots , Z_j\}$ is same chance to going to be first to get selected as pivot. The probability of Z_i or $Z_j$to be the first to get selected is Z_{ij} is $\frac{1}{j-i+1}$, since $\Pr[Z_i \text{ compared to } Z_j]$ contains two elements as $\{Z_i, Z_j\}$, $\Pr[Z_i \text{ compared to } Z_j] = \frac{2}{j-i+1}$

We have,

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr[Z_i \text{ compared to } Z_j]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k}$$

$$= \sum_{i=1}^{n-1} O(\log n)$$

= O(nlogn) , E[X] = Θ(nlogn)

E[T(n)] = Θ(nlogn)