BLG 335E – Analysis of Algorithms I, Fall 2020

Project 1 Report

Erce Can BEKTÜRE

150180009

Code Analysis

In this assignment, we are expected to implement quick sort and measure the elapsed time. All codes are written in **main.cpp** file.

Compilation Command: g++ main.cpp
Running Command: ./a.out N

a) (15 Points) Write down the asymptotic upper bound for the Quicksort for best case, worst case and average case. Prove them solving the recurrence equations.

We can write three recurrence equations for Quicksort algorithm:

Best case: (if the input distributed randomly)

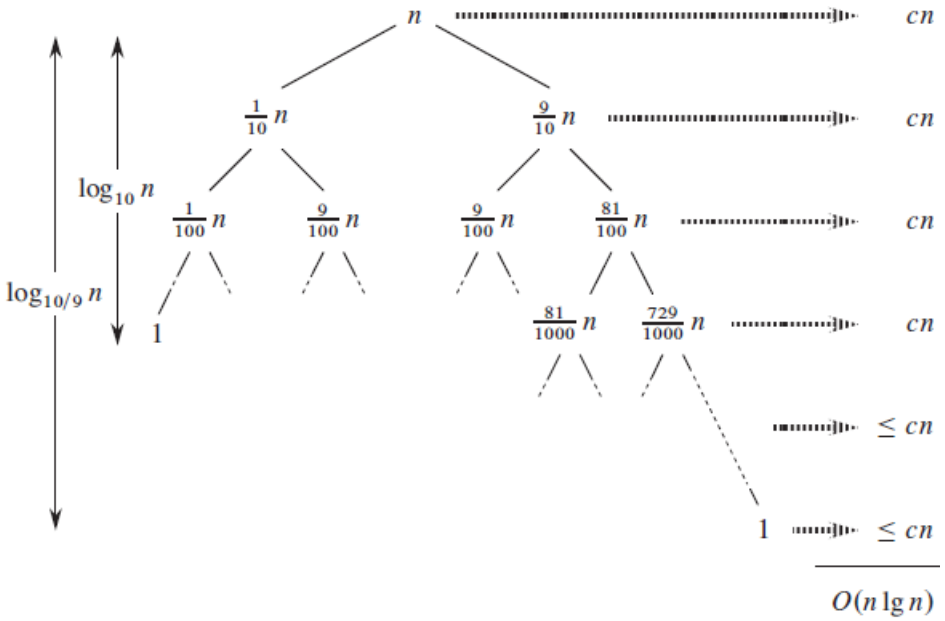
In the most even possible split, PARTITION produces two subproblems, each of size no more than n=2, since one is of size bn=2c and one of size dn=2e 1. In this case, quicksort runs much faster. The recurrence for the running time is then
We can solve this case using the Master method.

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

$$n = \theta\left(n^{\log_2 2}\right) \rightarrow T(n) = \theta\left(n^{\log_2 2} * \log_2 n\right) = \theta(n * \log_2 n) \text{ (Master thm. case 2)}$$

Average case:

Suppose, for example, that the partitioning algorithm always produces a 9-to-1 proportional split, which at first blush seems quite unbalanced. We then obtain the tree below:



We also obtain the recurrence: $T(n) = T(n/10) + T(9n/10) + cn$
on the running time of quicksort, where we have explicitly included the constant c hidden in the $\theta(n)$ term. The figure above shows the recursion tree for this recurrence. Notice that every level of the tree has cost cn, until the recursion reaches a boundary condition at depth $\log_{10} n = \theta(logn)$, and then the levels have cost at most cn. The recursion terminates at depth$\log_{\frac{10}{9}} n = \theta(logn)$. The total cost of quicksort is therefore $O(nlogn)$. Thus, with a 9-to-1 proportional split at every level of recursion, which intuitively seems quite unbalanced, quicksort runs in $O(nlogn)$ time—asymptotically the same as if the split were right down the middle. Indeed, even a 99-to-1 split yields an $O(nlogn)$ running time. In fact, any split of constant proportionality yields a recursion tree of depth $\theta(logn)$, where the cost at each level is $O(n)$. The running time is therefore $O(nlogn)$. whenever the split has constant proportionality.

Worst case: (if the input is sorted or reversed sorted)

The worst-case behavior for quicksort occurs when the partitioning routine produces one subproblem with n - 1 elements and one with 0 elements. Let us assume that this unbalanced partitioning arises in each recursive call. The partitioning costs $\theta(n)$ time. Since the recursive call on an array of size 0 just returns, $T(0) = \theta(1)$, and the recurrence for the running time is $T(n) = T(n-1) + n$

We can solve this case using the substitution method:

$$T(n) = T(n-1) + n$$
$$T(n-1) = T(n-2) + n - 1$$
$$T(n-2) = T(n-3) + n - 2$$
$$\dots$$
$$T(1) = 1$$
$$T(n) = \frac{n * (n+1)}{2} = O(n^2)$$

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 and give a simulation on a small fraction from the dataset (you may modify the results if necessary).

No because Quicksort is **not** a stable sorting algorithm by default, maybe the initial position of elements can be stored and considered in comparing step. Following example explains why quicksort is not stable: (Input => (country, profit))

(USA, 0), (Germany, 3), (Turkey, 1), (France, 2), (Turkey, 0)
When we sort this input the way we did at the implementation, we get =>
(France, 2), (Germany, 3), (Turkey, 0), (Turkey, 1), (USA, 0)

If we first sort by profit and then sort by country what we get is =>

First sorted by profit we get =>
(USA, 0), (Turkey, 0), (Turkey, 1), (France, 2), (Germany, 3)

Then equal profits are sorted by country we get=>
(Turkey, 0), (USA, 0), (Turkey, 1), (France , 2), (Germany, 3)

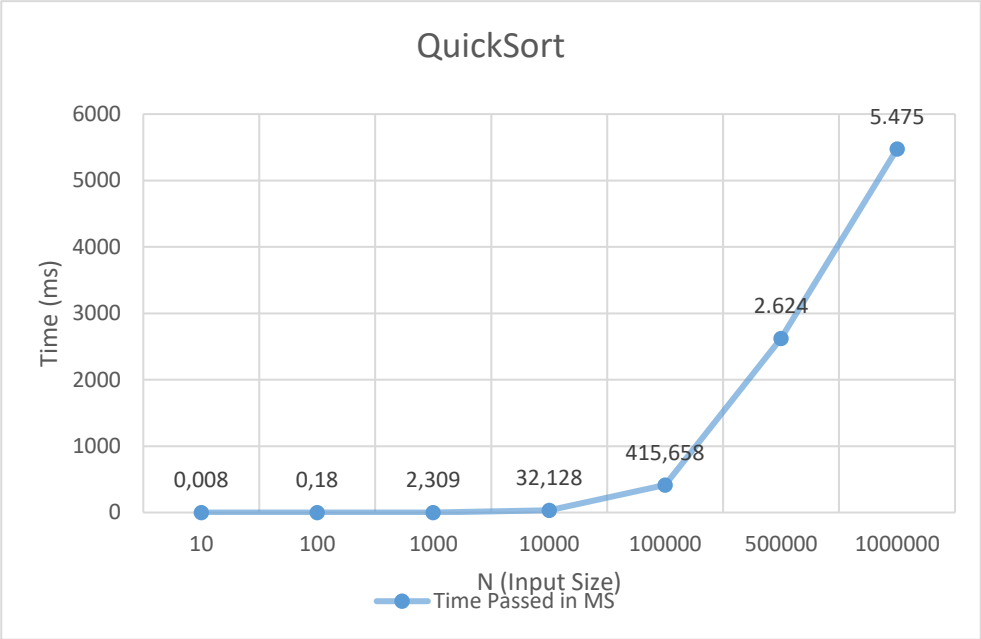
In conclusion the order is different from each other and thus we can say quicksort is unstable.

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

Three stable algorithms that could give the desired output are; Radix Sort, Merge Sort and Bucket Sort.

c) (15 Points) Run the algorithm for different **N** values {10, 100, 1000, 10K, 100K, 500K, 1M} on **sales.txt** data and calculate the average time of running. (Run the algorithm 10 times for each N value and take the average execution time 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)**

N (input size)	Elapsed Time
10	0,008 ms
100	0,18 ms
1000	2,309 ms
10000	32,128 ms
100000	415,658 ms
500000	2624 ms
1000000	5475 ms



Since the data in **"sales.txt"** was not sorted the we could say the run time of the algorithm is $T(n) = \theta(n * \log_2 n)$. Our input generally increases by the powers of 10, when we calculate $n * \log_2 n$ and it goes increasing like the table below. 33.219. t 22,5, 12.82, 13.91

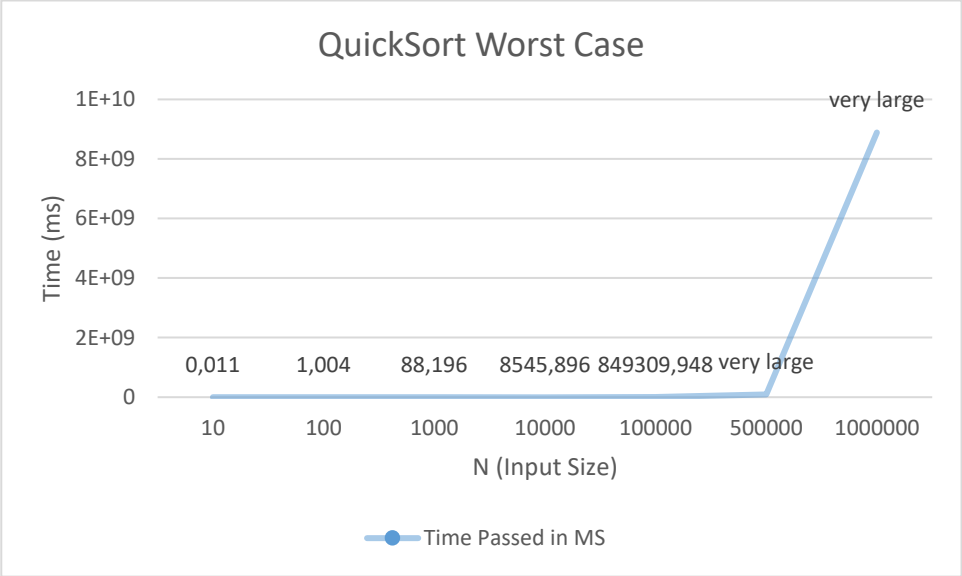
T(n)	Value	Elapsed Time
$n * \log_2 n$	33.21	0,008 ms
$n^2 * \log_2 n^2$	664.38	0,18 ms
$n^3 * \log_2 n^3$	9965.78	2,309 ms
$n^4 * \log_2 n^4$	132877.12	32,128 ms
$n^5 * \log_2 n^5$	1660964.04	415,658 ms
$5 * n^5 * \log_2 5n^5$	9465784.28	2624 ms
$n^6 * \log_2 n^6$	19931568.56	5475 ms

When we also look at the tables below we can see that the ratios of T(n)s are very similar to the ratios of elapsed time. Thus, we can conclude that our algorithms running time is $\boldsymbol{T(n) \approx \theta(n * \log_2 n)}$.

$value_i / value_{i-1}$	T_i / T_{i-1}
20.00	22.5
15.00	12.82
13.33	13.91
12.50	12.96
5.69	6.32
2.10	2.08

d) (10 Points) Run the algorithm for different N values {10, 100, 1000, 10K, 100K, 500K, 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.

N (input size)	Elapsed Time
10	0.011 ms
100	1.004 ms
1000	88.196 ms
10000	8545.896 ms
100000	849309.948 ms
500000	Very large
1000000	Very large



1. 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)**

The results are very bad compared to the results in **(c)** timewise. There happens to be a worst-case scenario for quicksort here because the data was already sorted. Since in the worst-case, the $T(n) = O(n^2)$ we cannot run N = {500000, 1000000} because it takes way too much time. Besides if we look at the ratios like we did in **(c)** we can create the table below.

$value_i / value_{i-1}$	T_i / T_{i-1}
100	91.27
100	88.60
100	96.896
100	99.382
100	Unknown
100	Unknown

By looking at the table the ratios are very similar and becoming more alike as the input grows. This proves that our runtime is $T(n) = O(n^2)$.

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

Reversely Sorted input (input sorted in descending order) would also give us similar results because it is also a worst-case scenario for quicksort

3. 3. Propose a solution to this case. **(1 sentence)**

A solution to this could be to use randomized quicksort; which inherits randomized pivot selection in the partition part of the algorithm and with that the runtime is back to $T(n) = \theta(n * \log_2 n)$ again and we can run it on N = {500000, 1000000}.