




PROJECT-2 REPORT

Analysis of Algorithms

Emre KÖSE – 150130037
09.11.2017



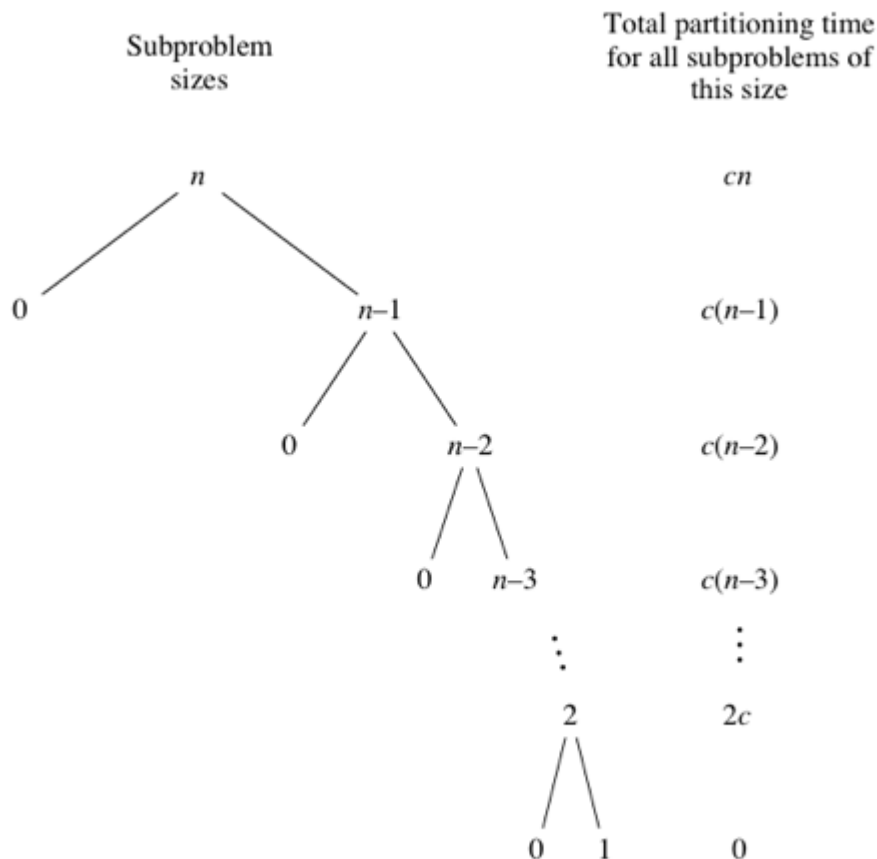
BLG 335E - Analysis of Algorithms I, Fall 2017

Project 2 - Report

- a) Give the asymptotic upper bound on the running time for Quicksort. If there is a recurrence for the algorithm, give and solve it.

The upper bound is determined with the worst case scenario of the sorting algorithm. So we should use big-O notation for that and we know Quick sort's worst case running time is $O(n^2)$. If we want to calculate this running time with recurrence (because quick sort works recursively) for worst case:

Firstly, we should declare the situation that when the worst case occur for quick sort, the answer is that when we are so unlucky. *In particular, suppose that the pivot chosen by the partition function is always either the smallest or the largest element in the n -element subarray [1].* In this case, in every recursive step of the quick sort, the cost of the partitioning time will be $(n-k)$ which n is the size of array.



To solve the recurrence of the worst case of quick sort, we can apply to equations, which are given below:

$$T(N) = N + T(N-1) \text{ (in worst case) and } T(0) = T(1) = 0 \text{ (base case)}$$

If we try to iterate recurrence:

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

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

$$T(N-2) = (N-2) + T(N-3)$$

...

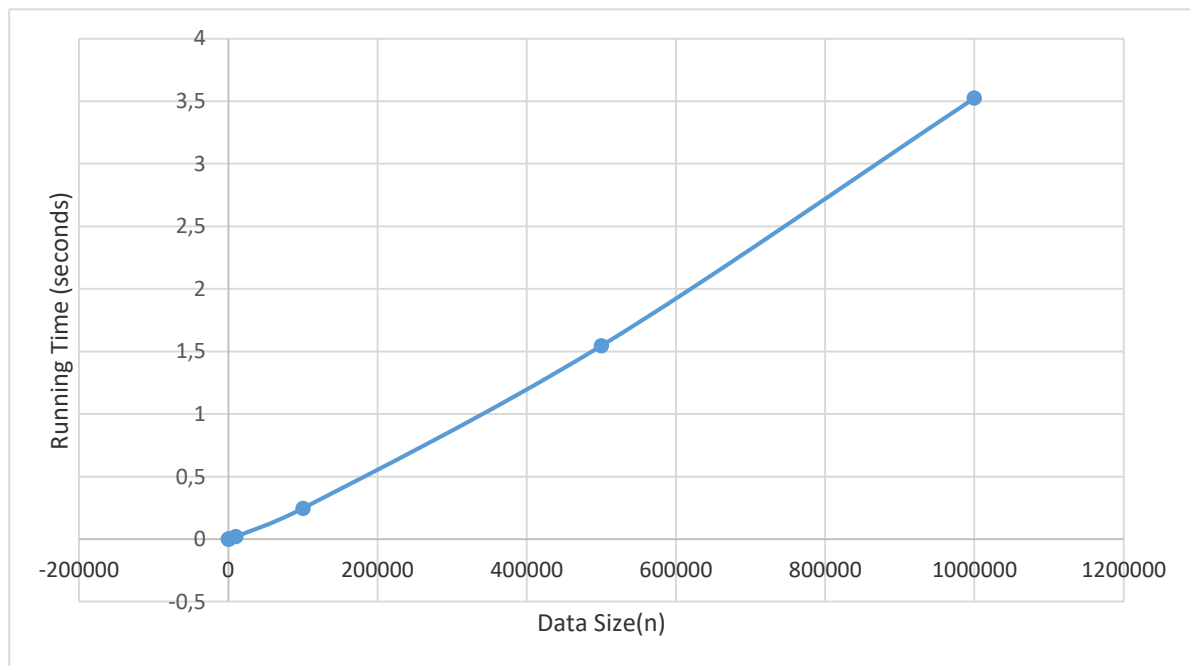
$$T(3) = 3 + T(2)$$

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

$$T(1) = 0$$

Hence, $T(N) = N + (N-1) + (N-2) \dots + 3 + 2 \approx N^2 / 2$ and this is again in $O(N^2)$

- b)** Run your program for N as $\{10, 100, 1000, 10000, 100000, 500000, 1000000\}$, and calculate the average time of execution (for example run 10 times for $N = 10$ and take the average, 10 times for $N = 100$ and take the average, and so on...) for each value of N . After calculating average execution times, you are required to prepare a two-lined plot in order to visualize the run time complexity of Quicksort for different values of N . Comment on the results by considering the asymptotic bound that you have found in (a).



Average running time of Quick Sort for different input sizes

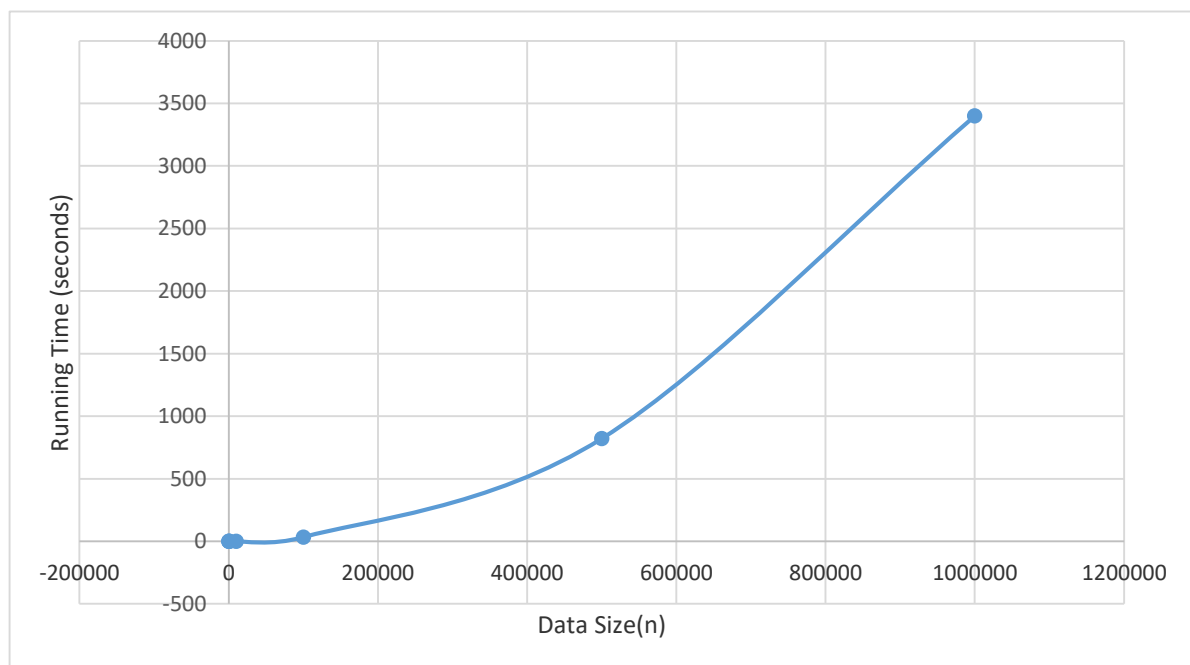
As seen in the graph, the line is increasing like linear form even if it is not entirely linear (Actually the graph form is form of the function $f(n) = n \cdot \log(n)$). We found the quicksort's upper bound which is $O(n^2)$ in part-a and in this situation this is not illogical because as we can see, the running time in average case is increasing slower than the running

time in worst case that grows quadratically. Hence, we can say $\theta(n \log(n))$ is subset of $O(n^2)$ according to asymptotic notation.

- c) *What is the worst case for Quicksort? Construct a data set in order to simulate this case, and execute your program for different values of N as $\{10, 100, 1000, 10000, 100000, 500000, 1000000\}$. Report the average execution times, and introduce a plot like you have done in part (b). Introduce a solution in order to overcome the worst case of Quicksort (You do not need to implement that solution).*

The worst case of quick sort is $O(n^2)$ which we found in part-a as an upper-bound. As mentioned in part a, if pivot chosen by the partition function is always either the smallest or the largest element in the n -element subarray, this happens when we give sorted or reverse sorted input data.

Note: The running time could not be measured because duration is too much in this case



Quick sort worst-case time complexity $O(n^2)$

We can overcome this problem with choosing a random pivot element. Then the running time will be independent of the input order and there is no variation of input will occur a worst-case time complexity. In this case, the worst case time complexity will be $O(n \log(n))$ which is like the average case of original quick sort.

d) *Is Quicksort stable? Explain your answer by illustrating your answer with a small fraction from the data set. You may modify the values if necessary.*

Quick sort is not stable algorithm. If we want to give an example from our project:


Firstly, I will give an original order of two rows in input data and then, I will give situations of the same two rows from sorted data.

First, original possessions: As seen from the pictures, our two rows those we will check, are in lines 247053 and 408744.



247051	221,18,19,male,90732,8600000US90732						
247052	1,22,24,female,24411,8600000US24411						
247053	0,10,14,male,00906,8600000US00906						
247054	83,50,54,female,46745,8600000US46745						
247055	6,25,29,male,68759,8600000US68759						

Original list





408742	16,30,34,male,24270,8600000US24270						
408743	11,25,29,female,18843,8600000US18843						
408744	0,10,14,male,00906,8600000US00906						
408745	53,0,4,female,61914,8600000US61914						
408746	26,35,39,male,64874,8600000US64874						

Original list

Secondly, sorted possessions: We know our sorting algorithm sorts the data according to the population and geo-id and we see our sample rows have equal values in terms of these two parameters (population and geo-id). As seen from picture below (sorted) and pictures from above (originals), even if the data values of our samples (we look only population and geo-id) are same, the orders are different after the sorting process. Therefore, this shows us quick sort is an unstable algorithm.

Note: I take the picture in below from sorted file that has 500k elements

	A	B	C	D	E	F	G	H
1	0,10,14,male,906,8600000US00906							
2	0,21,21,female,906,8600000US00906							
3	0,75,79,female,906,8600000US00906							
4	0,67,69,female,906,8600000US00906							
5	0,22,24,female,906,8600000US00906							
6	0,20,20,male,906,8600000US00906							
7	0,85,0,male,906,8600000US00906							
8	0,70,74,female,906,8600000US00906							
9	0,18,19,male,906,8600000US00906							
10	0,10,14,female,906,8600000US00906							
11	0,62,64,male,906,8600000US00906							
12	0,70,74,male,906,8600000US00906							
13	0,21,21,male,906,8600000US00906							
14	0,5,9,female,906,8600000US00906							

REFERENCES

[1] <https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort>