BLG 335E

Analysis of Algorithms I

Assignment 1 – Quicksort Analysis

CRN: 15175

Lecturer's Name : Hazım Kemal EKENEL

Student's Name : Yusuf Utku GÜL

Number : 150150006

Date of Delivery : 10.12.2020

1. **Introduction**

In this assignment we were asked to use deterministic quick-sort function to sort a list of sales data of countries around the world. The sorting will be first by alphabetically and later according to profits.

2. **Development and Environment**

The code is written on Dev-C++ with compiler run option set to “-std=c++11”. The application also compiled and run succesfully on ITU SSH server with line of “g++ main.cpp -std=c++11”.

3. **Analysis**
   Part a)

   Best case for quicksort happens when the partition pivot selected as the middle value and the list divided into 2 equal parts.

   T(n) = T(n/2) + T(n/2) + O(n) = 2*T(n/2) +O(n) and the master theorem states that for any a*T(n/b) + f(n) if f(n) = O(n^logb^a) then T(n) = O(n^logb^a * lg(n)) so the best case for quicksort is T(n) = O(n*lg(n))

   Worst case happens when input list is sorted and pivot selected as the minimum or maximum value, since 1 partition always contain 0 element. If one side always has 0 element and other side has (n-1) element;

```
T(n) = T(0) + T(n–1) + cn
                 /    \
               T(0)    c(n-1)
                        /    \
                     T(0)     c(n-2)
                               …
```

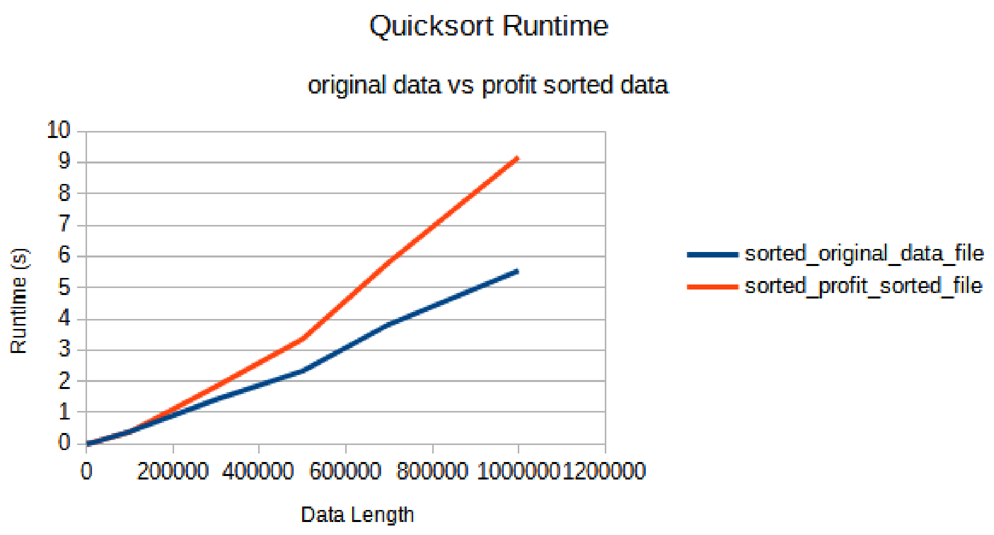
   = $cn+c(n-1)+c(n-2)+\cdots+1c$ = c((n+1)*n)/2 and in big O notation we take the biggest exponial item so; T(n) = O(n^2)

   We can also try to calculate a average case for quicksort by selecting a possible split rate. For example if we select a pivot split rate 1/10 – 9/10, then;

```
T(n) =T((1/10)n) + T((9/10)n) +      cn
                                           /    \
                          T((1/10)n)     T((9/10)n)
                          /        \          /          \
            T((1/100)n)    T((9/100)n)   T((9/100)n)   T((81/10)n)
            ….                  …………………………..
```

   shortest time would be the left side which has a length of log10n and longest time would be right side length of log10/9n, so n*log10n <=T(n)<=n*log10/9n + O(n) .

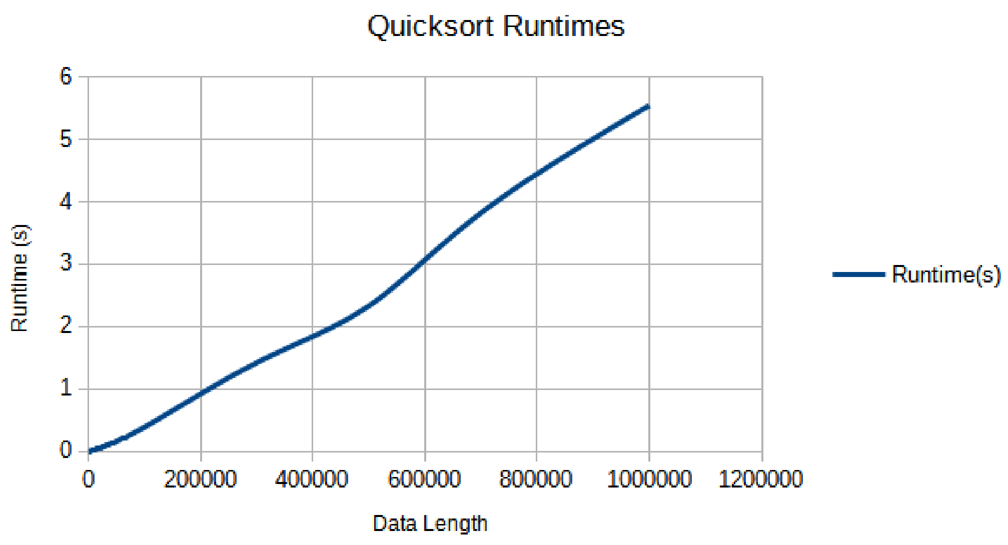
Part b)



1) When tested sorting the original data gave a better result because in sorted_by_profit file, the list is partially sorted which gives quicksort a hard time.

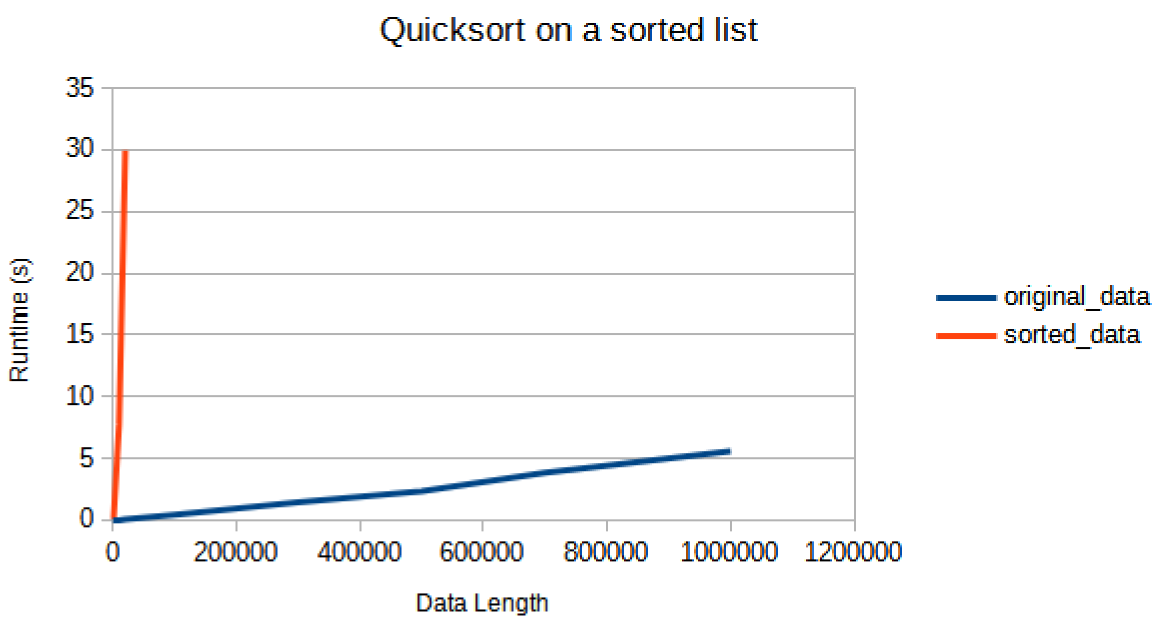
2) Insertion sort, Modified bubble sort, Merge sort could be used since they work well on sorted data.

Part c)



When looked at the data length and runtime plot, it seems that it grew quite linearly. It indicates that the worst case result didn't occur so we can say that the data file was not sorted. It can be said that the result is showing an average case.

Part d)



1) As mentioned in part a, the worst result for quicksort is when it is applied on an already sorted data. The one part of the partition is always empty so there is so much wasted time. And as the data length gets bigger, the runtime grows exponentially like shown in the plot.

2) Trying to sort a reversely sorted list would also give us a similar result.

3) The best solution to this problem probably would be using randomized quicksort. Since it determines the pivot randomly the pivot would not be minimum or maximum value of the list.