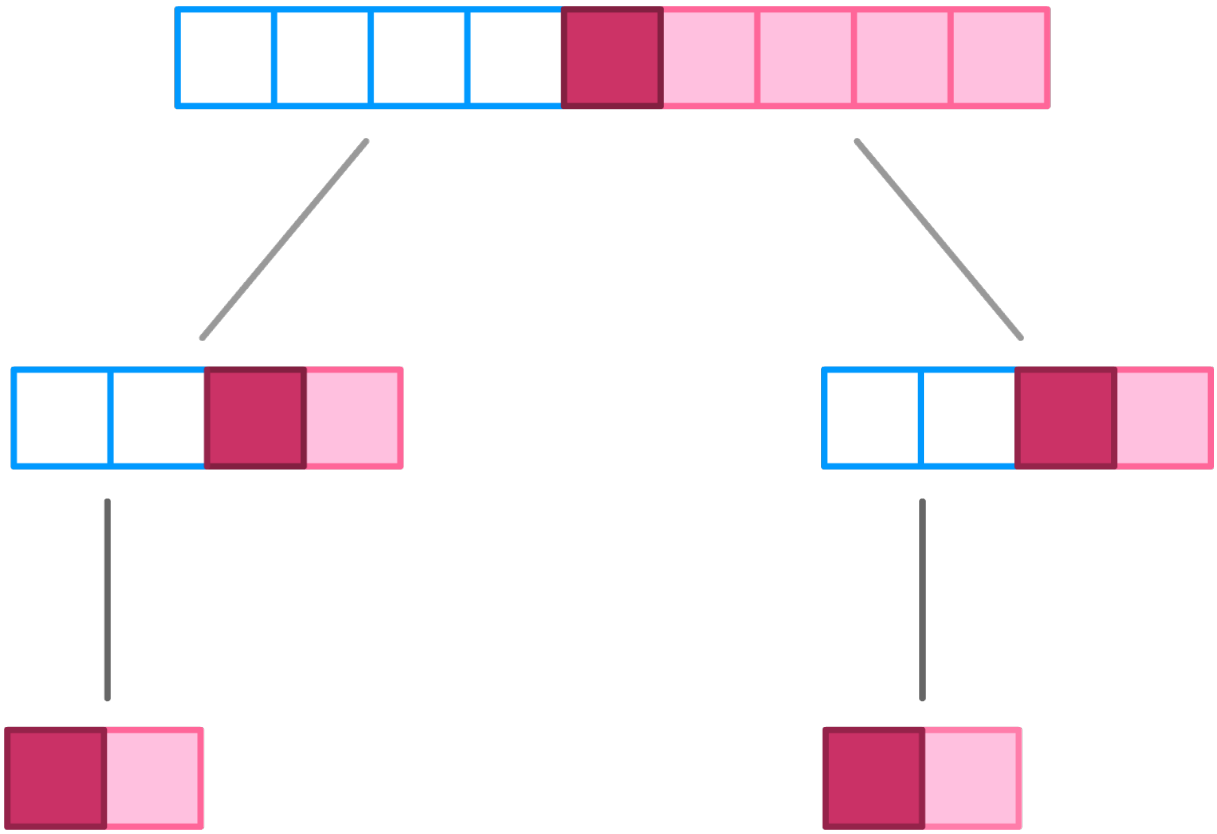
Image taken from Kirupa



BLG 335E

Homework 1 Report
Quicksort

Berke Dönmez

150170045

December 10, 2020

a)

The runtime function $T(n)$ for Quicksort is

$$T(n) = T(|L|) + T(|R|) + cn$$

Where $|L|$ is the number of elements less than the pivot, $|R|$ is the number of elements greater than or equal to the pivot and c is a constant real number not less than 1. For this homework, c is much greater than 1 because of string comparisons.

Upper bound for best case

For the best case, after every partition, $|L|$ and $|R|$ would be very close values so that "height of the recursion tree" is minimal. This is best provided by $|L| = |R| = \frac{n}{2}$. We can ignore rounding downs as they are not that effective. Let's conduct this into equation:

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + cn \\
&= 2T\left(\frac{n}{2}\right) + cn \\
&= 2(2T\left(\frac{n}{4}\right) + c\frac{n}{2}) + cn \\
&= 4T\left(\frac{n}{4}\right) + 2cn \\
&= 8T\left(\frac{n}{8}\right) + 3cn \\
&. \\
&. \\
&. \\
&= 2^k T(1) + k * cn
\end{aligned}$$

If we observe carefully, k is such an integer that $\frac{n}{2^k} \geq 1$ (since n may not be a power of 2) and k is maximal. If we modify this inequality, we get $n \geq 2^k \rightarrow log_2 n \geq k$. It is clear that $k = \lfloor log_2 n \rfloor$. Therefore, we get

$$T(n) = n + cnlog_2 n$$

Finally, the upper bound for the best case is $\mathcal{O}(n + cnlog_2 n) = \mathcal{O}(nlogn)$.

Upper bound for worst case

Contrary to the best case, for worst case, we want the height of the tree to be maximal, i.e. $|L|$ and $|R|$ should be as further as possible. Well, this can be achieved by having one of them equal to zero. It does not matter which is zero, it can be considered as a symmetrical situation.

$$\begin{aligned} T(n) &= T(0) + T(n-1) + cn \ (T(0) \text{ is negligible}) \\ &= T(n-2) + c(n-1) + cn \\ &= T(n-3) + c(n-2) + c(n-1) + cn \\ &. \\ &. \\ &. \\ &= c(1+2+3+...+n) = c\frac{n(n+1)}{2} \end{aligned}$$

We directly get the upper bound of $\mathcal{O}(\frac{c}{2}(n^2+n)) = \mathcal{O}(n^2)$ for the worst case.

Upper bound for average case

Calculating the upper bound for average case is the trickiest part because of its more advanced math requirements. First of all, the following assumption is made: **In the beginning of every partitioning, choosing every index of the interval as the pivot is equiprobable (pivot is randomly selected).** If the length of the interval is n, then the probability of selecting a random index is $\frac{1}{n}$.
Thus, every possible $(|L|, |R|)$ pair will appear with the probability of $\frac{1}{n}$. The possible $(|L|, |R|)$ pairs are $(0, n-1), (1, n-2), ...(n-1, 0)$. Therefore, the runtime equation can be written as

$$T(n) = \frac{1}{n}\left(\sum_{i=0}^{n-1} T(i) + T(n-1-i)\right) + cn$$

Since for every i $(0 \leq i \leq n-1)$, $T(i)$ is included in the sum $(\sum_{i=0}^{n-1} T(i) + T(n-1-i))$ exactly twice, we can rewrite the equation as

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

Multiplying both sides by n, we get

$$nT(n) = 2\left(\sum_{i=0}^{n-1} T(i)\right) + cn^2 \tag{1}$$

If we write $n-1$ instead of n

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

Subtracting (2) from (1), we get

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

Because $\sum_{i=0}^{n-1} T(i) = T(n-1) + \sum_{i=0}^{n-2} T(i)$ and $n^2 - (n-1)^2 = (n-(n-1))(n+(n-1)) = 2n-1$. The rearranged form of the equation is

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

We can assume $2n-1 \approx 2n$,

$$\begin{aligned}
T(n) &= \frac{n+1}{n}T(n-1) + 2c \\
&= \frac{n+1}{n} * \frac{n}{n-1}T(n-2) + 2c\left(1 + \frac{n+1}{n}\right) = \frac{n+1}{n-1}T(n-2) + 2c\left(1 + \frac{n+1}{n}\right) \\
&= \frac{n+1}{n-1} * \frac{n-1}{n-2}T(n-3) + 2c\left(1 + \frac{n+1}{n} + \frac{n+1}{n-1}\right) = \frac{n+1}{n-2}T(n-3) + 2c\left(1 + \frac{n+1}{n} + \frac{n+1}{n-1}\right) \\
&= \frac{n+1}{n-3}T(n-4) + 2c\left(1 + \frac{n+1}{n} + \frac{n+1}{n-1} + \frac{n+1}{n-2}\right) \\
&\;\;\vdots \\
&= 2c\left(1 + (n+1)\left(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + ... + 1\right)\right)
\end{aligned}$$

The sum $1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$ is approximately $ln(n)$. Thus,

$$T(n) \approx 2c(1 + (n+1)ln(n))$$

Finally, upper bound for average case is $\mathcal{O}(nlogn)$.

b)

Given modified sorting algorithm:

- Sort the sales.txt data by the total profits and write it into sorted_by_profits.txt
- Sort the sorted_by_profits.txt data according to country names using QuickSort

It is asked to decide whether this algorithm works as same as the first algorithm (given in the implementation part) in whose result the sales are sorted by country (ascending) and if countries are same, sorted by total profit (descending). Well, it can be thought that the result of the modified algorithm is the same as the original but it **may** give a different result (depending on the input, I think) due to minor reasons. There might be more than one reason but I found only one of them: **Think of the last swapping, at the end of the partition. When the "right bar" (right pointer) reaches to the end, we swap the pivot (currently at the end) with the element "left bar" (left pointer) is behind of without checking whether the pivot and this element is equal (in country) or not. The unwanted swaps appear here: Relative order of the sales with the same country changes due to these last swaps.** Again, it is important to highlight that these unwanted swaps might not appear at all, depending on input. Let's take 3 sales in order to show that the result might be different. As the attributes except country and total profit are unnecessary (they do not affect anything), it is enough to just write countries and total profits.

1. Tanzania — 23114.16
2. Seychelles — 9349.02
3. Tanzania — 1350622.16

After sorting by sales (descending order),

1. Tanzania — 1350622.16
2. Tanzania — 23114.16
3. Seychelles — 9349.02

After sorting by country names, we expect the result to be,

1. Seychelles — 9349.02
2. Tanzania — 1350622.16
3. Tanzania — 23114.16

But the actual result is,

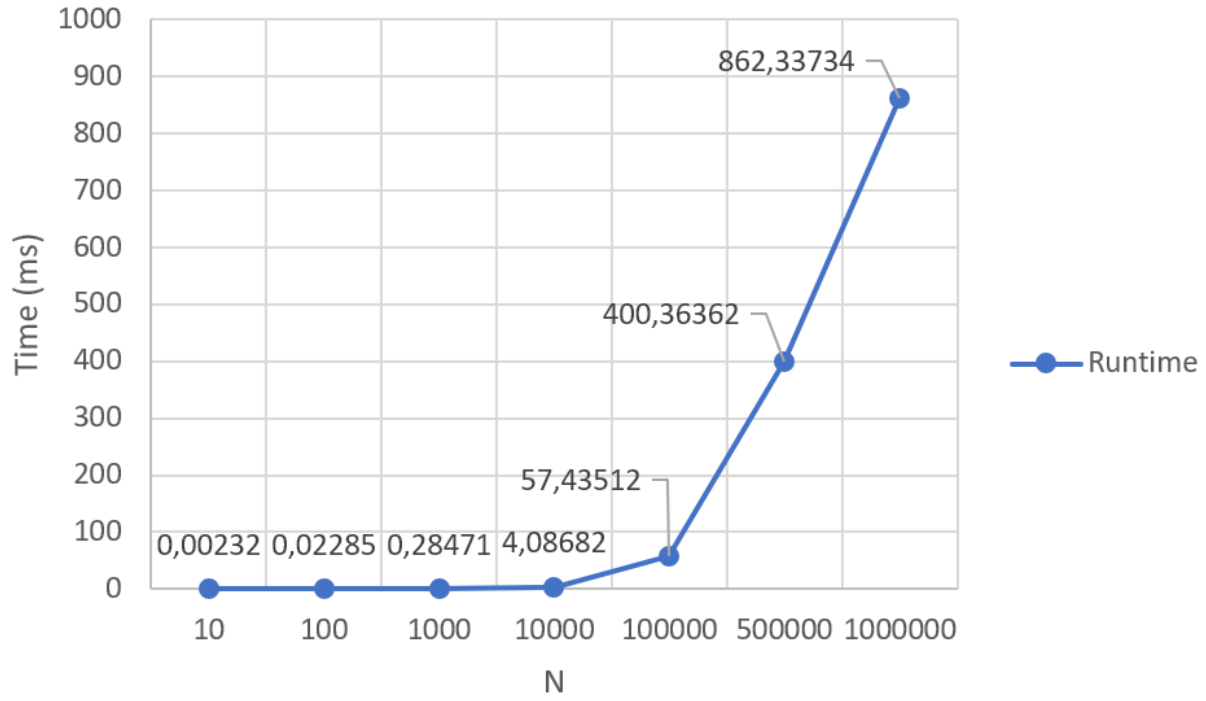
1. Seychelles — 9349.02
2. Tanzania — 23114.16
3. Tanzania — 1350622.16

That is because what I have just explained above. The pivot was “Seychelles” and left pointer did not move at all because right pointer did not encounter any element whose country name is lexicographically smaller than “Seychelles”. Then at the end, “Tanzania” with the total profit of 1350622.16 and pivot were swapped. Thus, “Tanzania” with bigger profit got below the one with smaller profit. This is called “unstability” since relative order of elements with the same country names might change.

What we want is a “stable” sorting algorithm: Relative order of elements with the same country names must not change (profits of them must be in ascending order). Three examples are as follows:

- Bubble Sort
- Insertion Sort
- Radix Sort

c)

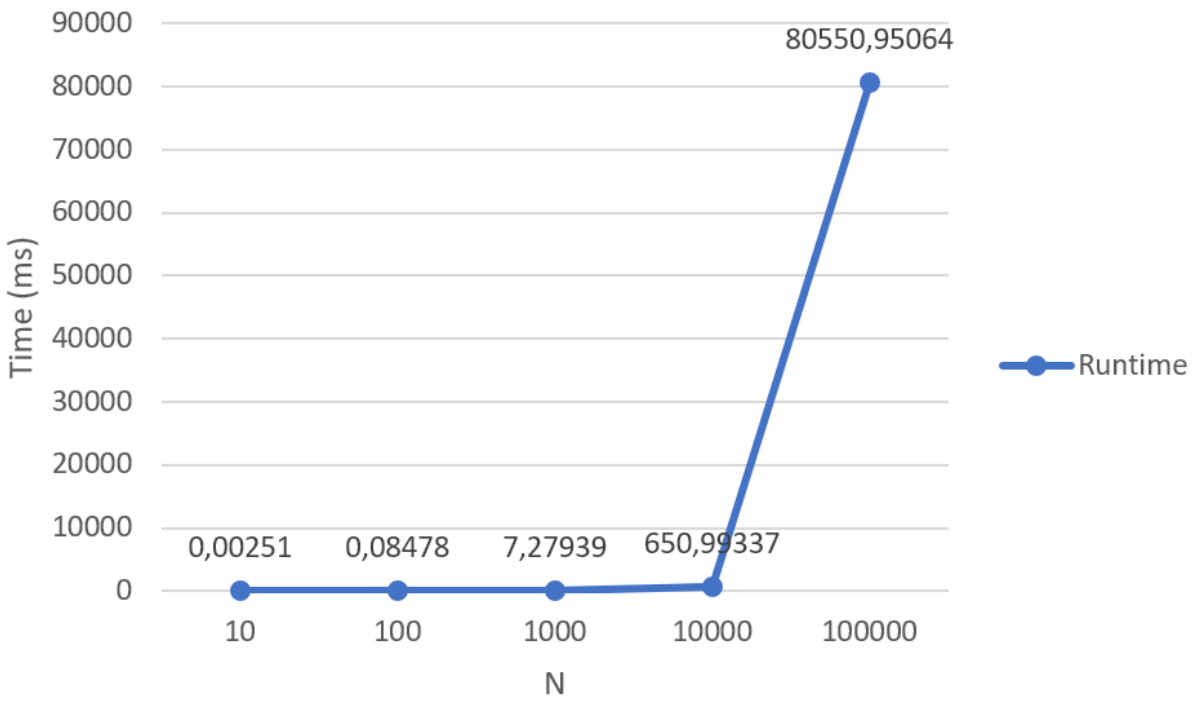


Runtimes (ms) for different N when the input file is "sales.txt"

It is clearly seen that runtime is not directly proportional to N; **the ratio between the runtimes of for consecutive N values is higher than the ratio between the N values**. The most clear example is the relation between runtimes of N = 100K and N = 500K: The ratio between runtimes is approximately 7 while ratio between Ns is 5. Well, it is obviously because of the extra $logN$ factor (average time complexity was $\mathcal{O}(NlogN)$): When N increases, $logN$ also increases. Thus, if $N_1 < N_2$,

$$\frac{N_2 logN_2}{N_1 logN_1} = \frac{N_2}{N_1} * \frac{logN_2}{logN_1} > \frac{N_2}{N_1}$$

d)



Runtimes (ms) for different N when the input file is "sorted.txt"

Compared to the previous graph in c), the runtime ratio between consecutive N values is huge. This situation is highlighting an important fact: Time complexity is higher than $\mathcal{O}(NlogN)$. It can be said that having a sorted input while always selecting pivot as the first or the last element is the worst case of all because partitioning is pretty bad. Since I am always selecting the last element, there can never be any element greater than the pivot ($|R| = 0$). Thus, I can tell that the current time complexity is $\mathcal{O}(N^2)$. Just because of that, I could not calculate the runtime for N = 500K and N = 1M. Even N = 100K takes 80 seconds!

As I said above, having a sorted input is the worst case. Reversing the sorted input is also a worst case because then $|L|$ would be 0 instead of $|R|$ (there would not be any element less than pivot).

Selecting the middle element as pivot instead of first or last element directly makes the sorted input be the best case. If you ever wonder, the runtime for N = 1M in this scenario is approximately 290 ms.