TECNOLÓGICO DE MONTERREY

FUNDAMENTOS DE COMPUTACIÓN

Homework 4

Student:
Jacob RIVERA

Professor:
Dr. Hugo TERASHIMA

March 20, 2019

Tecnológico de Monterrey

1 Problems

Solve the following problems:

1. Solve problems 4.58 and 4.59 (Sara Baase)

2. Assume that the subroutine PARTITION within the quicksort algorithm produces a 9-to-1 split. First, generate the recurrence to compute the complexity of the algorithm under these conditions. Second, estimate the complexity of the recurrence and discuss if the unbalance in the partition makes the algorithm to loose the optimality.

$$\begin{aligned} T(n) &\leq cnlog(n) \\ T(n) &= T(\frac{n}{10}) + T(\frac{9}{10}n) + n \\ &\leq \frac{cn}{10}log(\frac{n}{10}) + \frac{9cn}{10}log(\frac{9n}{10}) + 10 \\ &\leq \frac{cn}{10}log(n) + \frac{cn}{10}log(\frac{1}{10}) + \frac{9cn}{10}log(n) + \frac{9cn}{10}log(\frac{9}{10}) + m \\ &\leq cnlog(n) + \frac{9cn}{10}log(9) - \frac{9cn}{10}log(10) - \frac{cn}{10}log(10) + n \\ &\leq cnlog(n) - n(clog(10) - \frac{9c}{10}log(9) - 1) \\ T(n) &\leq cnlog(n) \text{ if } clog(10) - \frac{9c}{10}log(9) - 1 > 0 \\ &= \theta(nlog(n)) \end{aligned}$$

The unbalance in the partition would certainly affect the speed of the algorithm, however, the rate of growth would stay approximately the same, and well, we know that at the worst case, which this definitely isn't, the complexity of the quicksort will have an upper bound of $\theta(n^2)$

3. Show that there is no comparison sorting algorithm whose complexity is linear for at least $n!/2$ inputs of length n, What about a fraction $1/n^3$ of inputs of length n? What about a fraction $1/2^n$?

Given that we know that an input of length n has $n!$ permutations, which means that in the worst case, a comparison algorithm has to make at worst $\Omega(nlog(n))$ comparisons, which is the height h of the tree. The quantity of inputs which are linear can be represented by m. We know that the $n! = 2^h \geq m$, which is equivalent to $h \geq log(m)$ y $log(m) = \Omega(nlog(n))$. So, for

$$\begin{aligned} log(\frac{n!}{2}) &= log(n!) - 1 \geq nlog(n) - nlog(e) - 1 \\ log(\frac{1}{n^3}) &= -log(n^3) \leq nlog(n) \\ log(\frac{1}{2^n}) &= -log(n^3) \leq nlog(n) \end{aligned}$$

So, for the first one is true that there isn-t a algorithm which is linear for the quantity of inputs stated. However, for the other two, there are algorithms that can be linear, as the number of inputs m that are required to be linear reduces as n increases.

4. Consider an array of characters R, W, and B, and you want to arrange the array in such a way that the Rs are first, then the Ws, and at the end the Bs. Describe a linear algorithm to accomplish this task.

As the input space is fairly small compared to other algorithms, a silly but fast algorithm would be to iterate the array and count how many of each characters are in the array, which takes $O(n)$ and then just write the adequate number of R, W and B in the required order, which would also only take $O(n)$.

5. Take a sequence of $2n$ numbers as input. Design an algorithm that partitions the numbers into n pairs, with the property that the partition minimizes the maximum sum of the pair. For example, say we are given the numbers (1,3,5,9). The possible partitions are ((1,3),(5,9)), ((1,5), (3,9)) and((1,9),(3,5)). The pair sums of these partitions are (4,14), (6,12) y (10,8). Thus the third partition has 10 as its maximum sum, which is the minimum over the three partitions. Establish the complexity of your algorithm.

As the maximum sum can be minimized by pairing the biggest and the smallest elements an each pair subsequently. So we first need to sort the numbers, which can be done in $nlog(n)$ and then the joining of the pairs, which can be done in $n/2$ steps, resulting in $nlog(n) + \frac{n}{2} = O(nlog(n))$