Tecnológico de Monterrey

Fundamentos de computación

Homework 2

Student:
Jacob Rivera

Professor:
Dr. Hugo Terashima

March 21, 2019

1 Problems

Solve the following problems:

1. Use the substitution method to solve the following recurrences and determine their corresponding complexity. Establish the proper initial conditions for each problem.

   (a) $T(n) = 4T(n/2) + n^4$

$$T(n) = \begin{cases} b & \text{if } n = 1. \\ aT(\frac{n}{c}) + bn^x, & \text{otherwise.} \end{cases}$$

$$n = 2^k$$

$$\begin{aligned}
T(n) &= 2^2T(n/2) + n^4 \\
&= 2^2(2^2T(\frac{n}{2^2}) + (\frac{n}{2})^4) + n^4 \\
&= 2^4(2^2T(\frac{n}{2^3}) + (\frac{n}{2^2})^4) + \frac{n^4}{2^2} + n^4 \\
&= 2^6(2^2T(\frac{n}{2^4}) + (\frac{n}{2^3})^4) + \frac{n^4}{2^4} + \frac{n^4}{2^2} + n^4 \\
&= 2^8T(\frac{n}{2^4}) + \frac{n^4}{2^6} + \frac{n^4}{2^4} + \frac{n^4}{2^2} + n^4 \\
&= 2^{2k}T(\frac{n}{2^k}) + \sum_{i=0}^{k-1} \frac{n^4}{2^{2i}} \\
&= 2^{2k}T(\frac{n}{2^k}) + n^4 \sum_{i=0}^{k-1} \frac{1}{2^{2i}} \\
&= 2^nT(0) + n^4 \sum_{i=0}^{k-1} \frac{1}{2^{2i}} \\
&= n^4 \sum_{i=0}^{k-1} \frac{1}{2^{2i}} \\
&= n^4 \sum_{i=0}^{k-1} 2^{-2i} \\
&= n^4 * \frac{1-2^k}{1-2} \\
&= n^4 \\
&\in O(n^4)
\end{aligned}$$

   (b) $T(n) = 3T(n/3) + nlog(n)$

$$n = 3^k$$

$$\begin{aligned}
T(n) &= 3T(n/3) + nlog(n) \\
&= 3(3T(\frac{n}{3^2}) + \frac{n}{3}log(\frac{n}{3})) + nlog(n) \\
&= 3^kT(\frac{n}{3^k}) + \sum_{i=0}^{k-1} nlog(\frac{n}{3^i}) \\
&= 3^kT(0) + \sum_{i=0}^{k-1} nlog(\frac{n}{3^i}) \\
&= n\sum_{i=0}^{k-1} log(\frac{n}{3^i}) \\
&= nlog^2(n) \\
&\in nlog^2(n)
\end{aligned}$$

(c) $T(n) = 3T(n/3) + \frac{\sqrt{n}}{log(n)}$

$$n = 3^k$$

$$\begin{aligned}
T(n) &= 3T(n/3) + \frac{\sqrt{n}}{log(n)} \\
&= 3(3T(\frac{n}{3^2}) + \frac{\sqrt{n/3}}{log(n/3)}) + \frac{\sqrt{n}}{log(n)} \\
&= 3^2T(\frac{n}{3^2}) + 3\frac{\sqrt{n/3}}{log(n/3)}) + \frac{\sqrt{n}}{log(n)} \\
&= 3^2(3T(\frac{n}{3^3}) + \frac{\sqrt{n/3}}{log(n/3)}) + 3\frac{\sqrt{n/3}}{log(n/3)} + \frac{\sqrt{n}}{log(n)} \\
&= 3^3T(\frac{n}{3^3}) + 3^2\frac{\sqrt{n/3}}{log(n/3)} + 3\frac{\sqrt{n/3}}{log(n/3)} + \frac{\sqrt{n}}{log(n)} \\
&= 3^kT(\frac{n}{3^k}) + \sum_{i=0}^{k-1} 3^i\frac{\sqrt{n/3^i}}{log(n/3^i)} \\
&= 3^kT(0) + \sum_{i=0}^{k-1} 3^i\frac{\sqrt{n/3^i}}{log(n/3^i)} \\
&= \sum_{i=0}^{k-1} 3^i\frac{\sqrt{n/3^i}}{log(n/3^i)} \\
&= \sum_{i=0}^{k-1} 3^i\frac{\sqrt{n/3^i}}{log(n/3^i)} \leq 3^{log(n)}
\end{aligned}$$

$$\in O(3^{log(n)})$$

(d) $T(n) = T(n-3) + n$

$$n = 3k$$

$$\begin{aligned}
T(n) &= T(n-3)+n \\
T(n-1) &= T(n-4)+n \\
T(n-3) &= T(n-6)+n \\
&= T(n-6)+2n \\
&= T(n-3k)+kn \\
&= T(n-3\frac{n}{3})+\frac{n}{3}n \\
&= T(0)+\frac{n}{3}n \\
&= \frac{n^2}{3} \\
&\in O(n^2)
\end{aligned}$$

2. Modify the algorithm for multiplying two numbers X and Y(that with complexity $n^{1,59}$), dividing each number in three parts. Describe the algorithm and derive its complexity. Is it better than dividing by two parts? Explain.

   The algorithm would keep the same functioning as the old one, the only difference being how it's divided. As such, the recurrence would be

$$T(n) = 8T(n/3) + kn$$

   Which results in a complexity $O(n^{1.893})$ which would result in worst performance than using a two parts split

3. The run time of an algorithm A for solving a problem is given by the recurrence $T(n) = 7T(n/4)+n$. A different algorithm A' for solving the same problem has a running time given by $T'(n) = aT'(n/7) + n$. Determine the integer value a with which algorithm A is better than algorithm A'. Explain your process.

   By the use of the master theorem, we know that $T(n) \in O(n^{1.404})$ and $T'(n) \in O(n)$, so only when $n = 1$ the algorithms are going to make the same number of steps.

4. For each of the following algorithms, which each receive an integer as input, determine what they do,establish the corresponding recurrence,and solve it to find the computational complexity.

```
ALGORITHM BR(n)
    // Input a positive integer n
    if n = 1 return 1
    else return BR(floor(n/2)) + 1
```

   This algorithm returns the power of 2 closest to the input number, without surpassing it. The recurrence would be: $T(n) = T(n/2) + 1$, which complexity would be $O(log(n))$

```
ALGORITHM Question(n)
    // Input a positive integer n
    if n = 1 return 1
    else return Question(n−1) + n∗n∗n
```

   This algorithm gives us the sum of the cubes of all numbers until n. As such, the recurrence would be $T(n) = T(n-1) + n^3$, which results in complexity of $O(n^4)$

$$\begin{aligned}
n = kT(n) &= T(n-1)+n^3 \\
T(n-1) &= T(n-2)+2n^3 \\
T(n) &= T(n-k)+kn^3 \\
T(n) &= T(0)+n^4 \\
&\in O(n^4)
\end{aligned}$$

5. The Towers of Hanoi problem is often used as an example when teaching recursion. Six disks of different sizes are piled on a peg in order by size, with the largest at the bottom. There are two empty pegs. The problem is

to move all the disks to the third peg by moving only one at a time a never placing a disk on top of a smaller one. The second peg may be used for intermediate moves. The usual solution recursively moves all but the last disk from the starting peg to the spare peg, then moves the remaining disks on the start peg to the destination peg, and then recursively moves all the others from the spare peg to the destination peg. Illustrate these steps and explain the procedure in a pseudocode. Write a recurrence equation to determine the number of moves done and solve it.

The solution proposed would be fairly easy to implement following the next pseudocode:

```
Towers(disk, source, destination, auxiliar)
    //Disk is the number of the disk to be moved, source is the starting
    point, destination is the goal and auxiliar is the spare peg
    if (disk == 1)
        move(disk, destination)
    else
        Towers(n - 1, source, auxiliar, destination)
        move(disk, destination)
        Towers(n - 1, auxiliar, destination, source)
```

$$\begin{aligned}
T(1) &= 1 \\
n &= k+1 \\
T(n) &= 2T(n-1) \\
&= 2^2T(n-2) \\
&= 2^kT(n-k) \\
&= 2^{n-1}T(n-(n-1)) \\
&= 2^{n-1}T(1) \\
&= 2^{n-1} \\
&\in \theta(2^{n-1})
\end{aligned}$$