ADw3

Name : **Peter Garamvoelgyi** Student ID: **2018280070**

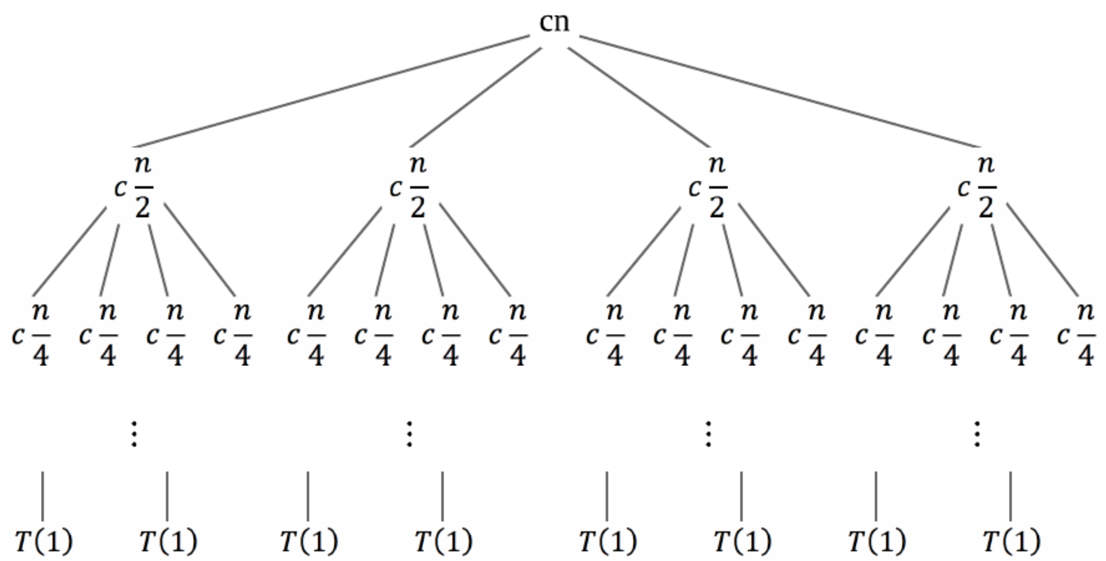
CLRS 4.4-7. Draw the recursion tree for $\boldsymbol{T(n) = 4T(\lfloor n/2 \rfloor) + cn}$, where $\boldsymbol{c}$ is a constant, and provide a tight asymptotic bound on its solution.

$$T(n) = 4 \cdot T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + cn$$

Following the method from CLRS page 88 ("*we know that floors and ceilings usually do not matter when solving recurrences*"), we create a recursion tree for the recurrence

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn \qquad T(1) = 1$$

… where we also assumed the base case to be $T(1) = 1$.



We get a recursion tree with $\log_2 n$ levels.

Relying on the fact that $T(1) = \Theta(1)$, the cost at the leaves is

$$\Theta\left(4^{\log_2 n}\right) = \Theta\left(n^{\log_2 4}\right) = \Theta(n^2)$$

The sums on the other levels are:

$$cn, 2cn, 4cn, \dots, \left(4^i \cdot c\frac{n}{2^i} = 2^i cn\right), \dots$$

Summing these up, we get

$$\sum_{i=1}^{\log_2 n-1} 2^i cn = \frac{2^{\log_2 n} - 1}{2 - 1} cn = \left(2^{\log_2 n} - 1\right) cn = cn^2 - cn$$

To get the total cost, we sum up the cost of the leaves and the cost of the other levels:

$$T(n) = cn^2 - cn + \Theta(n^2) = \Theta(n^2)$$

We can verify this using the *master method*:

$$a = 4 \quad b = 2 \quad f(n) = cn$$

In this case, $f(n)$ grows polynomially slower than $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = cn = O(n^{2-\varepsilon}) \quad \forall \varepsilon: 0 < \varepsilon \leq 1$$

… i.e. this is case 1. Thus,

$$T(n) = \Theta(n^{\log_b a}) \quad \Longrightarrow \quad T(n) = \Theta(n^2)$$

CLRS 4-1. Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

b. $T(n) = T(7n/10) + n$

Using the *master method*, our parameters are

$$a = 1 \quad b = \frac{10}{7} \quad f(n) = n$$

In this case, $f(n)$ grows polynomially faster than $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_{\frac{10}{7}} 1} = n^0 = 1$$

$$f(n) = \Omega(n^{0+\varepsilon}) \quad \forall \varepsilon: 0 \leq \varepsilon \leq 1$$

… and $f(n)$ satisfies the regularity condition

$$\forall\, c < 1: \quad af\left(\frac{n}{b}\right) \leq cf(n) \quad \rightarrow \quad \frac{7n}{10} \leq cn \quad \text{holds for } c = \frac{7}{10}$$

… i.e. this is case 3. Thus,

$$T(n) = \Theta\big(f(n)\big) \quad \Longrightarrow \quad T(n) = \Theta(n)$$

c. $T(n) = 16T(n/4) + n^2$

Using the *master method*, our parameters are

$$a = 16 \quad b = 4 \quad f(n) = n^2$$

In this case, $f(n)$ grows at a similar rate as $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_4 16} = n^2$$

$$f(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

… i.e. this is case 2. Thus,

$$T(n) = \Theta(n^{\log_b a} \lg n) \quad \Longrightarrow \quad T(n) = \Theta(n^2 \lg n)$$

d. $\boldsymbol{T(n) = 7T(n/3) + n^2}$

Using the *master method*, our parameters are

$$a = 7 \quad b = 3 \quad f(n) = n^2$$

In this case, $f(n)$ grows polynomially faster than $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_3 7} \approx n^{1.77}$$

$$f(n) = \Omega(n^{\log_3 7 + \varepsilon}) \quad \forall \varepsilon: 0 \leq \varepsilon \leq 2 - \log_3 7$$

… and $f(n)$ satisfies the regularity condition

$$\forall\, c < 1: \quad af\left(\frac{n}{b}\right) \leq cf(n) \quad \rightarrow \quad 7\left(\frac{n}{3}\right)^2 \leq cn^2 \quad \text{holds for } c = \frac{7}{9}$$

… i.e. this is case 3. Thus,

$$T(n) = \Theta\big(f(n)\big) \quad \Longrightarrow \quad T(n) = \Theta(n^2)$$

CLRS 4-2. Consider the recursive binary search algorithm for finding a number in a sorted array. Give recurrences for the worst-case running times of binary search when arrays are passed using each of the three methods above[1], and give good upper bounds on the solutions of the recurrences.

The running time of binary search is characterized by the equation

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + D(n) + C(n)$$

… i.e. we half the size of the problem and we only have one single recursive call in every iteration. $C(n) = \Theta(1)$ in every case. The difference is in the complexity of the divide step $D(n)$.

1. **<u>Pass by pointer</u>**

$$D(n) \approx \mathrm{c} \qquad \Longrightarrow \qquad T_1(n) = T_1\left(\frac{n}{2}\right) + \mathrm{c}$$

Using the *master method*, our parameters are

$$a = 1 \qquad b = 2 \qquad f(n) = c$$

In this case, $f(n)$ grows at a similar rate as $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$$

$$f(n) = \Theta(n^{\log_b a}) = \Theta(1)$$

… i.e. this is case 2. Thus,

$$T_1(n) = \Theta(n^{\log_b a} \lg n) \qquad \Longrightarrow \qquad T_1(n) = \Theta(\lg n)$$

2. **<u>Pass by copying</u>**

$$D(n) \approx c + N \qquad \Longrightarrow \qquad T_2(n) = T_2\left(\frac{n}{2}\right) + c + \mathrm{N}$$

The easiest way to prove this is to draw the recursion tree. We get a list with $\log_2 n$ levels, each of them containing $c + N$ steps.

$$S = (c + N) \cdot \log_2 n = \Theta(N \lg n)$$

[1] Pass by pointer: $\Theta(1)$. Pass by copying: $\Theta(N)$. Pass by copying $A[p..q]$ subrange: $\Theta(q - p + 1)$.

3. Pass by copying subrange

$$D(n) \approx c + \frac{n}{2} \quad \Longrightarrow \quad T_3(n) = T_3\left(\frac{n}{2}\right) + c + \frac{n}{2}$$

Using the *master method*, our parameters are

$$a = 1 \quad b = 2 \quad f(n) = c + \frac{n}{2}$$

In this case, $f(n)$ grows polynomially faster than $n^{\log_b a}$:

$$f(n) = \Omega(n^{0+\varepsilon}) \quad \forall\varepsilon: 0 \leq \varepsilon \leq 1$$

… and $f(n)$ satisfies the regularity condition

$$\forall\, c < 1: \quad af\left(\frac{n}{b}\right) \leq cf(n) \quad \rightarrow \quad c_0 + \frac{n}{4} \leq c_0 + c\frac{n}{2} \quad \text{holds for } c = \frac{1}{2}$$

… i.e. this is case 3. Thus,

$$T(n) = \Theta\big(f(n)\big) \quad \Longrightarrow \quad T(n) = \Theta(n)$$