Algorithm Assignment 2

He Tianyang

March 25, 2024

1 Problem 1.

Use induction (substitution) to prove:

$$T(n) = \begin{cases} 0, & n = 1 \\ T(\frac{n}{2}) + T(\frac{n}{2}) + cn, & n > 1 \end{cases} \tag{1}$$

$$\leq cn \log_2 n \tag{2}$$

Base case: When $n = 1$,exist c greater enough that satisfy $T(1) = 0 \leq c \cdot 1 \cdot \log_2 1$. The base case holds.
Inductive step: Assume that $T(k) \leq ck \log_2 k$ for all $k < n$. We want to show that $T(n) \leq cn \log_2 n$.

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + c_2 n \\ &\leq c_1 \frac{n}{2} \log_2 \frac{n}{2} + c_1 \frac{n}{2} \log_2 \frac{n}{2} + cn \\ &= c_1 n \log_2 \frac{n}{2} + c_2 n \\ &\leq c_1 n \log_2 n + c_2 n \\ &= O(n \log_2 n) \end{aligned}$$

Therefore, by induction, $T(n) \leq cn \log_2 n$ for all $n \geq 1$.

2 Problem 2.

Use the master theorem to solve the following recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n^{\frac{1}{2}}) \tag{3}$$

Solution:
The recurrence relation is of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, where $a = 2$, $b = 2$, and $f(n) = \Theta(n^{\frac{1}{2}})$. We can compare $f(n)$ with $n^{\log_b a}$ to determine which case of the master theorem applies.

In this case, $n^{\log_b a} = n^{\log_2 2} = n$. Since $f(n) = \Theta(n^{\frac{1}{2}})$, we have $f(n) = O(n^{1-\epsilon})$ for $\epsilon = \frac{1}{2}$. Therefore, we are in case 1 of the master theorem, and the solution to the recurrence relation is:

$$\boxed{T(n) = n^{\log_b a} = n^{\log_2 2} = \Theta(n)}$$

3 Problem 3.

Expand the recurrence tree for the following recurrence relation, and apply asymptotic analysis to determine the complexity of the recurrence relation.

$$T(n) = T(2) + T(n-2) + cn \tag{4}$$

Solution:

The recurrence relation is of the form $T(n) = T(2) + T(n-2) + cn$. We can expand the recurrence tree to visualize the recursive calls.

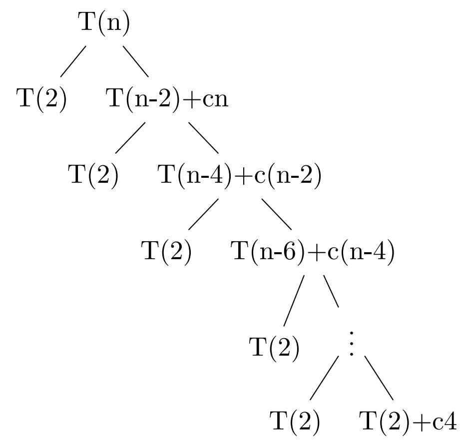

Figure 1: Recurrence tree for $T(n) = T(2) + T(n-2) + cn$

Donated h as the depth of the tree, the cost of each level is cn, where we have:

$$h = \frac{n}{2}$$

The total cost of the tree is:

$$T(n) = hT(2) + \sum_{i=0}^{h-1} 2c(i+1) = \frac{n}{2} \cdot T(2) + \frac{cn^2 + 4cn}{4} = \frac{n}{2} \cdot T(2) + \frac{c}{4} \cdot n^2 + cn$$

Compare the complexity of $\frac{n}{2} \cdot T(2)$ with $\frac{c}{4} \cdot n^2 + cn$, we have:

$$T(n) = \begin{cases} \Theta(nT(2)), & \text{if } T(2) = \Omega(n) \\ \Theta(n^2), & \text{if } T(2) = O(n) \end{cases}$$

4 Problem 4.

Expand the recurrence tree for the following recurrence relation. Calculate the depth and the asymptotic complexity of the recurrence relation.

$$T(n) = T(0.2n) + T(0.8n) + \Theta(n) \tag{5}$$

Solves:

The recurrence relation is of the form $T(n) = T(0.2n) + T(0.8n) + \Theta(n)$. We can expand the recurrence tree to visualize the recursive calls.

Each k, $T(k) = T(0.2k) + T(0.8k) + \Theta(k)$, $T(0.8)$ is the larger part, and get the slowest asymptotic speed. Therefore, the depth of the recursive tree is $\log_{0.8} n$.

For the layer in depth k, the cost is $\Theta(n)$, and the total cost of the tree is:

$$T(n) = \sum_{i=0}^{\log_{0.8} n - 1} 2^i \Theta(n) = \Theta(n \log_{0.8} n)$$

Therefore, the asymptotic complexity of the recurrence relation is $\boxed{\Theta(n \log_{0.8} n)}$.

5 Problem 5.

Calculate the time complexity of algorithm A which is defined by the following recurrence relation:

$$\begin{cases} f(n) = 2f(n-1) + 1 \\ f(0) = 1 \end{cases} \tag{6}$$

Solves: Obviously, we have

$$\begin{aligned} f(n) &= 2f(n-1) + 1 \\ &= 2(2f(n-2) + 1) + 1 \\ &= 2^2 f(n-2) + 2 + 1 \\ &= 2^3 f(n-3) + 2^2 + 2 + 1 \\ &= \dots \\ &= 2^n f(0) + 2^{n-1} + 2^{n-2} + \dots + 2 + 1 \\ &= 2^n + 2^{n-1} + 2^{n-2} + \dots + 2 + 1 \\ &= 2^{n+1} - 1 \end{aligned}$$

Therefore, the time complexity of algorithm A is $\boxed{\Theta(2^n)}$.