Two Methods for Solving Recurrences

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

We have seen how to analyze the running time of recursive algorithms by recurrences. It is thus important to strengthen our skills for solving recurrences. Today we will learn two techniques for this purpose: the master theorem and the substitution method.

Master Theorem

The Master Theorem

Let $f(n)$ be a function that returns a positive value for every integer $n > 0$. We know:

$$\begin{aligned} f(1) &= O(1) \\ f(n) &\le \alpha \cdot f(\lceil n/\beta \rceil) + O(n^{\gamma}) \qquad \text{(for } n \ge 2\text{)} \end{aligned}$$

where $\alpha \ge 1$, $\beta > 1$, and $\gamma \ge 0$ are constants. Then:

- If $\log_\beta \alpha < \gamma$, then $f(n) = O(n^{\gamma})$.
- If $\log_\beta \alpha = \gamma$, then $f(n) = O(n^{\gamma} \log n)$.
- If $\log_\beta \alpha > \gamma$, then $f(n) = O(n^{\log_\beta \alpha})$.

The theorem can be proved by carefully applying the "expansion method" we saw earlier (recall that the method writes $f(n)$ into terms with increasingly smaller values of n). The details are tedious and omitted from this course.

Example 1

Consider the recurrence of binary search:

$$\begin{aligned} f(1) &\leq c_1 \\ f(n) &\leq f(\lceil n/2 \rceil) + c_2 \qquad \text{(for } n \geq 2) \end{aligned}$$

Hence, $\alpha = 1$, $\beta = 2$, and $\gamma = 0$. Since $\log_\beta \alpha = \gamma$, we know that $f(n) = O(n^0 \cdot \log n) = O(\log n)$.

Example 2

Consider the recurrence of merge sort:

$$\begin{aligned} f(1) &\le c_1 \\ f(n) &\le 2 \cdot f(\lceil n/2 \rceil) + c_2 n \qquad \text{(for } n \ge 2) \end{aligned}$$

Hence, $\alpha = 2$, $\beta = 2$, and $\gamma = 1$. Since $\log_\beta \alpha = \gamma$, we know that $f(n) = O(n^\gamma \cdot \log n) = O(n \log n)$.

Example 3

Consider the recurrence:

$$\begin{aligned} f(1) &\leq c_1 \\ f(n) &\leq 2 \cdot f(\lceil n/4 \rceil) + c_2\sqrt{n} \qquad \text{(for } n \geq 2) \end{aligned}$$

Hence, $\alpha = 2$, $\beta = 4$, and $\gamma = 1/2$. Since $\log_\beta \alpha = \gamma$, we know that $f(n) = O(n^\gamma \cdot \log n) = O(\sqrt{n} \log n)$.

Example 4

Consider the recurrence:

$$\begin{aligned} f(1) &\leq c_1 \\ f(n) &\leq 2 \cdot f(\lceil n/2 \rceil) + c_2\sqrt{n} \qquad \text{(for } n \geq 2) \end{aligned}$$

Hence, $\alpha = 2$, $\beta = 2$, and $\gamma = 1/2$. Since $\log_\beta \alpha > \gamma$, we know that $f(n) = O(n^{\log_\beta \alpha}) = O(n)$.

Example 5

Consider the recurrence:

$$\begin{aligned} f(1) &\leq c_1 \\ f(n) &\leq 13 \cdot f(\lceil n/7 \rceil) + c_2 n^2 \qquad \text{(for } n \geq 2) \end{aligned}$$

Hence, $\alpha = 13$, $\beta = 7$, and $\gamma = 2$. Since $\log_\beta \alpha < \gamma$, we know that $f(n) = O(n^\gamma) = O(n^2)$.

The Substitution Method

Example 6

Consider the recurrence:

$$\begin{aligned} f(1) &= 1 \\ f(n) &\leq f(n-1) + 3n \qquad \text{(for } n \geq 2) \end{aligned}$$

We suspect that $f(n) = O(n^2)$; the problem is how to prove it. The substitution method provides a way of proving it by mathematical induction.

Let us reason as follows. Assume $f(n) \leq c \cdot n^2$ for some constant c. For the base case of $n = 1$, this holds as long as $c \geq 1$.

Suppose that this is correct for all $n \leq k - 1$. Now let us see what conditions c needs to satisfy to ensure correctness for $n = k$. We have:

$$\begin{aligned} f(k) &\leq f(k-1) + 3k \\ &\leq c(k-1)^2 + 3k \\ &= ck^2 - 2ck + c + 3k \end{aligned}$$

To make the above at most ck^2, it suffices to guarantee

$$c + 3k \leq 2ck$$

As $k \geq 2$, the above holds for all $c \geq 2$.

Combining all the above, we conclude that $f(n) \leq 2n^2 = O(n^2)$.

Remark: It is important to have a good guess about $f(n)$. If the guess is wrong, you will not be able to make the argument work. To see this, try "proving" $f(n) \leq cn$.

Example 7

Consider the recurrence:

$$\begin{aligned} f(1) &= 10 \\ f(n) &\leq 5f(\lfloor n/5 \rfloor) + 3n \qquad \text{(for } n \geq 2) \end{aligned}$$

We will prove that $f(n) = O(n \log n)$ with the substitution method.

Assume $f(n) \le 1 + c \cdot n \log_5 n$ for some constant c. For the base case of $n = 1$, this holds as long as $c \ge 1$.

Suppose that this is correct for all integers $n \le k - 1$. Now let us see what conditions c needs to satisfy to ensure correctness for $n = k$. We have:

$$\begin{aligned} f(k) &\le 5f(\lfloor k/5 \rfloor) + 3k \\ &\le 5(1 + c\lfloor k/5 \rfloor \log_5 \lfloor k/5 \rfloor) + 3k \\ &\le 5c(k/5)\log_5(k/5) + 3k + 5 \\ &= ck(\log_5 k - 1) + 3k + 5 \\ &= ck \log_5 k - ck + 3k + 5 \end{aligned}$$

To make the above at most $1 + ck \log_5 k$, it suffices to have

$$3k + 4 \le ck$$

for all $k \ge 2$. It suffices to set $c \ge 4$. This concludes the proof that $f(n) \le 1 + 4n \log_5 n = O(n \log n)$.

Example 8

Consider the recurrence:

$$f(n) = O(1) \quad \text{(for } n \le 40\text{)}$$
$$f(n) \le f(\lceil n/5 \rceil) + f\left(\left\lceil \frac{7n}{10} \right\rceil\right) + n \quad \text{(for } n > 40\text{)}$$

We will prove that $f(n) = O(n)$ using the substitution method. This demonstrates the true power of the method because this is really a non-trivial recurrence (the master theorem is not applicable here).

Assume $f(n) \leq cn$ for some c. The base case $n \leq 40$ holds as long as c is larger than a certain constant.

Suppose that this is correct for all real-valued $n \leq k - 1$. Now let us see what conditions c needs to satisfy to ensure correctness for $n = k > 40$. We have:

$$\begin{aligned} f(k) &\leq c(\lceil k/5 \rceil) + c(\lceil (7/10)k \rceil) + k \\ &\leq c(k/5 + 1) + c((7/10)k + 1) + k \\ &= c(9/10)k + 2c + k \end{aligned}$$

To make the above at most ck, it suffices to have

$$\begin{aligned} c(9/10)k + 2c + k &\leq ck \\ \Leftrightarrow k &\leq c(k/10 - 2) \\ \Leftarrow k &\leq c(k/20) \quad \text{(by } k > 40\text{)} \\ \Leftrightarrow 20 &\leq c. \end{aligned}$$

We now conclude that $f(n) = O(n)$.