Recurrence Equation Analysis

◆ The conquer step of merge-sort consists of merging two sorted sequences, each with $n/2$ elements and implemented by means of a doubly linked list, takes at most bn steps, for some constant b.

◆ Likewise, the basis case ($n < 2$) will take at b most steps.

◆ Therefore, if we let $T(n)$ denote the running time of merge-sort:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

◆ We can therefore analyze the running time of merge-sort by finding a **closed form solution** to the above equation.

  ■ That is, a solution that has $T(n)$ only on the left-hand side.

Master Method
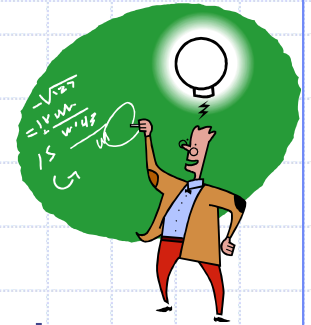
- Many divide-and-conquer recurrence equations have the form:

$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

- The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

   provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

Master Method, Example 1

◆ The form:
$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

◆ The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

   provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

◆ Example:
$$T(n) = 4T(n/2) + n$$

Solution: $\log_b a = 2$, so case 1 says $T(n)$ is $O(n^2)$.

Master Method, Example 2

◆ The form:
$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

◆ The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

◆ Example:
$$T(n) = 2T(n/2) + n \log n$$

Solution: $\log_b a = 1$, so case 2 says T(n) is O(n log^2 n).

Master Method, Example 3

◆ The form: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$

◆ The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

◆ Example:

$$T(n) = T(n/3) + n \log n$$

Solution: $\log_b a = 0$, so case 3 says T(n) is O(n log n).

Master Method, Example 4

- The form: $T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$

- The Master Theorem:

  1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

  2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

  3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

     provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

- Example:

$$T(n) = 8T(n/2) + n^2$$

Solution: $\log_b a = 3$, so case 1 says T(n) is O(n³).

Master Method, Example 5

◆ The form:
$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

◆ The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

   provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

◆ Example:

$$T(n) = 9T(n/3) + n^3$$

Solution: $\log_b a = 2$, so case 3 says $T(n)$ is $O(n^3)$.

Master Method, Example 6

◆ The form:
$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

◆ The Master Theorem:

1. if $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$

2. if $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$

3. if $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$,

   provided $af(n/b) \leq \delta f(n)$ for some $\delta < 1$.

◆ Example:

$$T(n) = T(n/2) + 1 \quad \text{(binary search)}$$

Solution: $\log_b a = 0$, so case 2 says T(n) is O(log n).

Integer Multiplication

◆ Algorithm: Multiply two n-bit integers I and J.

- Divide step: Split I and J into high-order and low-order bits

$$I = I_h 2^{n/2} + I_l$$

$$J = J_h 2^{n/2} + J_l$$

- We can then define I*J by multiplying the parts and adding:

$$I * J = (I_h 2^{n/2} + I_l) * (J_h 2^{n/2} + J_l)$$

$$= I_h J_h 2^n + I_h J_l 2^{n/2} + I_l J_h 2^{n/2} + I_l J_l$$

- So, $T(n) = 4T(n/2) + n$, which implies $T(n)$ is $O(n^2)$.
- But that is no better than the algorithm we learned in grade school.

An Improved Integer Multiplication Algorithm

◆ Algorithm: Multiply two n-bit integers I and J.

- Divide step: Split I and J into high-order and low-order bits

$$I = I_h 2^{n/2} + I_l$$

$$J = J_h 2^{n/2} + J_l$$

- Observe that there is a different way to multiply parts:

$$I * J = I_h J_h 2^n + [(I_h - I_l)(J_l - J_h) + I_h J_h + I_l J_l] 2^{n/2} + I_l J_l$$

$$= I_h J_h 2^n + [(I_h J_l - I_l J_l - I_h J_h + I_l J_h) + I_h J_h + I_l J_l] 2^{n/2} + I_l J_l$$

$$= I_h J_h 2^n + (I_h J_l + I_l J_h) 2^{n/2} + I_l J_l$$

- So, $T(n) = 3T(n/2) + n$, which implies $T(n)$ is $O(n^{\log_2 3})$, by the Master Theorem.
- Thus, $T(n)$ is $O(n^{1.585})$.