Design and Analysis of Algorithms

Mohammad GANJTABESH

mgtabesh@ut.ac.ir

School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

Analysing the Divide-and-Conquer Algorithms

In general we have the following recurrence equation:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{Otherwise.} \end{cases}$$

where:

- $T(n)$**:** is the time requred for an input of size n
- n**:** is the size of problem
- c**:** is a constant number
- a**:** is the number of subproblems
- n/b**:** is the size of each subproblem
- $D(n)$**:** is the time needed for Divide
- $C(n)$**:** is the time needed for Combine

Example: Analysing Merge Sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1, \\ 2T(n/2) + O(1) + O(n) & \text{Otherwise.} \end{cases}$$

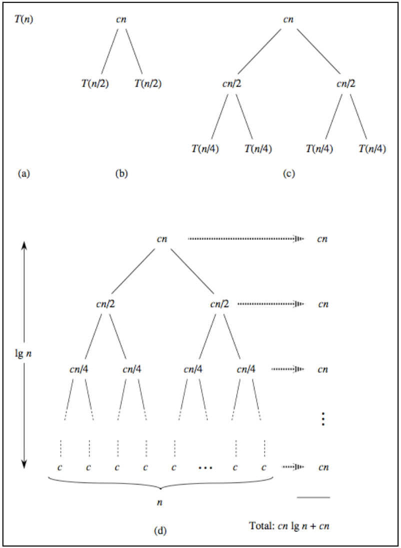
- What is the implicit formula of $T(n)$?
- How we can find it?

Solving the recurrence equations

There are different approaches to do this:

- Constructing Recursion Tree
- Performing Substitution
- Using Induction
- Master Theorem
- Generating Functions

Example: Analysing Merge Sort by constructing recursion tree



Example: Analysing Merge Sort by Performing Substitution

Suppose that $n = 2^k$, for some $k \in \mathbb{Z}$. We can write $T(n)$ as follows:

$$\begin{aligned}
T(n) &= 2T(n/2) + cn \\
&= 2T(2^{k-1}) + c2^k \\
&= 2\left(2T(2^{k-2}) + c2^{k-1}\right) + c2^k \\
&= 2^2 T(2^{k-2}) + 2c2^k \\
&= 2^2\left(2T(2^{k-3}) + c2^{k-2}\right) + 2c2^k \\
&= 2^3 T(2^{k-3}) + 3c2^k \\
&\vdots \\
&= 2^k T(1) + kc2^k \\
&= c'n + cnlog_2(n) \\
&= O(nlog(n)).
\end{aligned}$$

Master Theorem

Theorem (Master Theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n).$$

Then $T(n)$ can be bounded asymptotically as follows:

I. *If $f(n) = O(n^{log_b(a)-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{log_b(a)})$.*

II. *If $f(n) = \Theta(n^{log_b(a)})$, then $T(n) = \Theta(n^{log_b(a)} log(n))$.*

III. *If $f(n) = O(n^{log_b(a)+\varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

Master Theorem

Lemma

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n).$$

Then $T(n)$ can be written as follows:

$$T(n) = \Theta(n^{log_b(a)}) + \sum_{j=0}^{log_b(n)-1} a^j f(n/b^j).$$

Proof.

Let $n = b^i$ for some $i \in \mathbb{Z}$ and perform the substitution... $\square$

Master Theorem: Proof

In order to proof the Master Theorem, let

$$g(n) = \sum_{j=0}^{log_b(n)-1} a^j f(n/b^j)$$

So $T(n)$ becomes as follows:

$$T(n) = \Theta(n^{log_b(a)}) + g(n).$$

No we discuss about $g(n)$ ...

Master Theorem: Proof

Proof.

Part I.

$$f(n) = O(n^{log_b(a)-\varepsilon}) \Longrightarrow f(n) \leq cn^{log_b(a)-\varepsilon}.$$

$$\begin{aligned}
g(n) &= \sum_{j=0}^{log_b(n)-1} a^j f(n/b^j) \\
&\leq c \sum_{j=0}^{log_b(n)-1} a^j (n/b^j)^{log_b(a)-\varepsilon} \\
&= cn^{log_b(a)-\varepsilon} \sum_{j=0}^{log_b(n)-1} (b^\varepsilon)^j \\
&= cn^{log_b(a)-\varepsilon} \frac{n^\varepsilon - 1}{b^\varepsilon - 1} \\
&\leq c' n^{log_b(a)} \\
&= O(n^{log_b(a)}).
\end{aligned}$$

So $T(n) = \Theta(n^{log_b(a)})$. □

Master Theorem: Proof

Proof.

Part II.

$$f(n) = O(n^{log_b(a)}) \Longrightarrow f(n) \leq cn^{log_b(a)}.$$

$$\begin{aligned} g(n) &= \sum_{j=0}^{log_b(n)-1} a^j f(n/b^j) \\ &\leq c \sum_{j=0}^{log_b(n)-1} a^j (n/b^j)^{log_b(a)} \\ &= cn^{log_b(a)} \sum_{j=0}^{log_b(n)-1} 1 \\ &= cn^{log_b(a)} log_b(n) \\ &= O(n^{log_b(a)} log_b(n)). \end{aligned}$$

Similarly $g(n) = \Omega(n^{log_b(a)} log_b(n))$. So $T(n) = \Theta(n^{log_b(a)} log_b(n))$. $\square$

Master Theorem: Proof

Proof.

Part III. We should proof that $g(n) = \Theta(f(n))$.

- $g(n) = \Omega(f(n))$ since:

$$g(n) = \sum_{j=0}^{log_b(n)-1} a^j f(n/b^j) = f(n) + af(n/b) + \cdots \Longrightarrow g(n) = \Omega(f(n)).$$

- $g(n) = O(f(n))$ since:

$$af(n/b) = cf(n) \Longrightarrow f(n/b^j) \leq c^j/a^j f(n)$$

  So we have:

$$g(n) \leq f(n) \sum_{j=0}^{log_b(n)-1} c^j \leq f(n) \sum_{j=0}^{\infty} c^j \leq \frac{f(n)}{1-c}$$

  which implies $g(n) = O(f(n))$.

$\square$

Exercises

1. Solve the following recurrence equations:
   a. $T(n) = T(n/2) + 1.$
   b. $T(n) = 4T(n/2) + n^3.$
   c. $T(n) = 2T(n/4) + \sqrt{n}.$
   d. $T(n) = 3T(n/2) + nlog(n).$
   e. $T(n) = 2T(n/2) + n/log(n).$
   f. $T(n) = 4T(n/2) + n^3.$
   g. $T(n) = T(\sqrt{n}) + 1.$
   h. $T(n) = T(n-1) + 1/n.$