Introduction to Algorithms

Chapter 4
Recurrences

Solving Recurrences

- A **recurrence** is an equation or inequality that describes a function in terms of itself by using smaller inputs
- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

- is a *recurrence*.

Solving Recurrences

- Examples:
  - T(n) = 2T(n/2) + Θ(n) ➔ T(n) = Θ(n lg n)
  - T(n) = 2T(n/2) + n ➔ T(n) = Θ(n lg n)
  - T(n) = 2T(n/2) + 17 + n ➔ T(n) = Θ(n lg n)

- Three methods for solving recurrences
  - **Substitution method**
  - **Iteration method**
  - **Master method**

Recurrence Examples

$$T(n) = \begin{cases} 0 & n = 0 \\ c + T(n-1) & n > 0 \end{cases}$$

$$T(n) = \begin{cases} 0 & n = 0 \\ n + T(n-1) & n > 0 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Substitution Method

- The substitution method
  - **"making a good guess method"**
  - **Guess the form of the answer, then**
  - **use induction to find the constants and show that solution works**

- Our goal: show that
  $\boldsymbol{T(n) = 2T(n/2) + n = O(n \lg n)}$

Substitution Method
$T(n) = 2T(n/2) + n = O(n \lg n)$

- Thus, we need to show that $T(n) \leq c\, n \lg n$ with an appropriate choice of c
  - **Inductive hypothesis: assume** $\boldsymbol{T(n/2) \leq c\,(n/2) \lg (n/2)}$
  - **Substitute back into recurrence to show that** $\boldsymbol{T(n) \leq c\, n \lg n}$ **follows, when** $\boldsymbol{c \geq 1}$
    - $$\begin{aligned} T(n) &= 2\,\underline{T(n/2)} + n \\ &\leq 2\,\underline{(c\,(n/2) \lg (n/2))} + n \\ &= cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n && \text{for } c \geq 1 \\ &= O(n \lg n) && \text{for } c \geq 1 \end{aligned}$$

Substitution Method

- Consider $$T(n) = 2T(\sqrt{n}) + \lg n$$
  - Simplify it by letting **m = lg n** ➔ ***n* = 2*m***

$$T(2^m) = 2T(2^{m/2}) + m$$

  - Rename $S(m) = T(2^m)$
  - $S(m) = 2S(m/2) + m = O(m \lg m)$
  - Changing back from $S(m)$ to $T(n)$, we obtain
  - $T(n) = T(2^m)$
    $= S(m)$
    $= O(m \lg m)$
    $= O(\lg n \lg \lg n)$

Iteration Method

- Iteration method:
  - **Expand the recurrence *k* times**
  - **Work some algebra to express as a summation**
  - **Evaluate the summation**

$$T(n) = \begin{cases} 0 & n = 0 \\ c + T(n-1) & n > 0 \end{cases}$$

- $T(n) \;=\; c + T(n\text{-}1) \;=\; c + c + T(n\text{-}2)$
  $\;=\; 2c + T(n\text{-}2) \;=\; 2c + c + T(n\text{-}3)$
  $\;=\; 3c + T(n\text{-}3) \;\ldots\; kc + T(n\text{-}k)$
  $\;=\; ck + T(n\text{-}k)$
- So far for $n \geq k$ we have
  - ***T(n) = ck + T(n-k)***
- To stop the recursion, we should have
  - $n - k = 0 \;➔\; k = n$
  - ***T(n) = cn + T(0) = cn***
- Thus in general ***T(n) = O(n)***

$$T(n) = \begin{cases} 0 & n = 0 \\ n + T(n-1) & n > 0 \end{cases}$$

- $T(n)$
  $= n + T(n\text{-}1)$
  $= n + n\text{-}1 + T(n\text{-}2)$
  $= n + n\text{-}1 + n\text{-}2 + T(n\text{-}3)$
  $= n + n\text{-}1 + n\text{-}2 + n\text{-}3 + T(n\text{-}4)$
  $= \ldots$
  $= n + n\text{-}1 + n\text{-}2 + n\text{-}3 + \ldots + (n\text{-}k\text{+}1) + T(n\text{-}k)$
  $$= \sum_{i=n-k+1}^{n} i \;+\; T(n-k) \qquad \text{for } n \geq k$$
- To stop the recursion, we should have $n - k = 0$ ➔ $k = n$

$$\sum_{i=1}^{n} i \;+\; T(0) \;=\; \sum_{i=1}^{n} i \;+\; 0 \;=\; n\frac{n+1}{2}$$

$$T(n) \;=\; n\frac{n+1}{2} = O(n^2)$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

- $T(n)$

$T(n)$	$=$	$2\ T(n/2) + c$	1
	$=$	$2(2\ T(n/2/2) + c) + c$	2
	$=$	$2^2\ T(n/2^2) + 2c + c$	
	$=$	$2^2(2\ T(n/2^2/2) + c) + (2^2\text{-}1)c$	3
	$=$	$2^3\ T(n/2^3) + 4c + 3c$	
	$=$	$2^3\ T(n/2^3) + (2^3\text{-}1)c$	
	$=$	$2^3(2\ T(n/2^3/2) + c) + 7c$	4
	$=$	$2^4\ T(n/2^4) + (2^4\text{-}1)c$	
	$=$	…	
	$=$	$2^k\ T(n/2^k) + (2^k - 1)c$	k

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

- So far for $n \geq k$ we have
  - $T(n) = 2^k\ T(n/2^k) + (2^k - 1)c$
- To stop the recursion, we should have
  - $n/2^k = 1 \Rightarrow k = \lg n$
  - $T(n) = 2^{\lg n}\ T(n/2^{\lg n}) + (2^{\lg n} - 1)c$

    $= n\ T(n/n) + (n - 1)c$

    $= n\ T(1) + (n\text{-}1)c$

    $= nc + (n\text{-}1)c$

    $= nc + nc - c = 2cn - c = cn - c/2$

    $\leq cn = O(n)$ for all $n \geq ½$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

- $T(n) =$ $aT(n/b) + cn$ 1
  - $=$ $a(aT(n/b/b) + cn/b) + cn$ 2
  - $=$ $a^2T(n/b^2) + cna/b + cn$
  - $=$ $a^2T(n/b^2) + cn(a/b + 1)$
  - $=$ $a^2(aT(n/b^2/b) + cn/b^2) + cn(a/b + 1)$ 3
  - $=$ $a^3T(n/b^3) + cn(a^2/b^2) + cn(a/b + 1)$
  - $=$ $a^3T(n/b^3) + cn(a^2/b^2 + a/b + 1)$
  - $=$ …
  - $=$ $a^kT(n/b^k) + cn(a^{k-1}/b^{k-1} + a^{k-2}/b^{k-2} + \ldots + a^2/b^2 + a/b + 1)$ k

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

- So we have
  - $T(n) = a^kT(n/b^k) + cn(a^{k-1}/b^{k-1} + \ldots + a^2/b^2 + a/b + 1)$
- To stop the recursion, we should have
  - $n/b^k = 1$ ➔ $\boldsymbol{n = b^k}$ ➔ $k = \log_b n$
  - $T(n)$ $= a^kT(1) + cn(a^{k-1}/b^{k-1} + \ldots + a^2/b^2 + a/b + 1)$
    - $= a^kc + cn(a^{k-1}/b^{k-1} + \ldots + a^2/b^2 + a/b + 1)$
    - $= ca^k + cn(a^{k-1}/b^{k-1} + \ldots + a^2/b^2 + a/b + 1)$
    - $= cna^k/b^k + cn(a^{k-1}/b^{k-1} + \ldots + a^2/b^2 + a/b + 1)$
    - $= cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1)$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

- So with $k = \log_b n$
  - **$T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1)$**
- What if $a = b$?
  - **$T(n) = cn(1+\ldots+1+1+1)$ //k+1 times**

  $= cn(k + 1)$

  $= cn(\log_b n + 1)$

  $= \Theta(n \log_b n)$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

- So with $k = \log_b n$
  - **$T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1)$**
- What if $a < b$?
  - **Recall that**
    - **$\Sigma(x^k + x^{k-1} + \ldots + x + 1) = (x^{k+1} - 1)/(x-1)$**
  - **So:**

$$\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \frac{1 - (a/b)^{k+1}}{1 - (a/b)} < \frac{1}{1 - a/b}$$

  - **$T(n) = cn \cdot \Theta(1) = \Theta(n)$**

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

- So with $k = \log_b n$
  - $T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1)$

- What if $a > b$?

$$\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 \quad = \quad \frac{(a/b)^{k+1} - 1}{(a/b) - 1} \quad = \quad \Theta\left((a/b)^k\right)$$

  - $T(n) = cn \cdot \Theta(a^k / b^k)$

$= cn \cdot \Theta(a^{\log_b n} / b^{\log_b n}) = cn \cdot \Theta(a^{\log_b n} / n)$

recall logarithm fact: $a^{\log_b n} = n^{\log_b a}$

$= cn \cdot \Theta(n^{\log_b a} / n) = \Theta(cn \cdot n^{\log_b a} / n)$

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

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

- So…

$$T(n) = \begin{cases} \Theta(n) & a < b \\ \Theta(n \log_b n) & a = b \\ \Theta\left(n^{\log_b a}\right) & a > b \end{cases}$$

The Master Theorem

- Given: a *divide and conquer* algorithm
  - **An algorithm that divides the problem of size *n* into *a* subproblems, each of size *n/b***
  - **Let the cost of each stage (i.e., the work to divide the problem + combine solved subproblems) be described by the function *f*(n)**
- Then, the Master Theorem gives us a cookbook for the algorithm's running time:

The Master Theorem

- if T(n) = aT(n/b) + f(n) where a ≥ 1 & b > 1
- then

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta\left(n^{\log_b a} \lg n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \& \\ & af(n/b) < cf(n) \text{ for large } n \end{cases} \left.\begin{matrix} \\ \varepsilon > 0 \\ c < 1 \\ \\ \end{matrix}\right\}$$

Understanding Master Theorem

- In each of the three cases, we are comparing f(n) with $\mathbf{n^{\log_b a}}$, the solution to the recurrence is determined by the larger of the two functions.
- In case 1, if the function $\mathbf{n^{\log_b a}}$ is the larger, then the solution T(n) = $\Theta(\mathbf{n^{\log_b a}})$.
- In case 3, if the function f(n) is the larger, then the solution is T(n) = Θ(f(n)).
- In case 2, if the two functions are the same size, then the solution is T(n) = $\Theta(\mathbf{n^{\log_b a} \lg n})$ = Θ(f(n) lg n).

Understanding Master Theorem

- In case 1, not only must f(n) be smaller than $\mathbf{n^{\log_b a}}$, it must be polynomially smaller. That is f(n) must be asymptotically smaller than $\mathbf{n^{\log_b a}}$ by a factor of n^{ε} for some constant $\varepsilon > 0$.
- In case 3, not only must f(n) be larger than $\mathbf{n^{\log_b a}}$, it must be polynomially larger and in addition satisfy the "regularity" condition that:

$$a\,f(n/b) \le c\,f(n).$$

Understanding Master Theorem

- It is important to realize that the three cases do not cover all the possibilities for f(n).
- There is a gap between cases 1 and 2 when f(n) is smaller than $\mathbf{n^{\log_b a}}$ but not polynomially smaller.
- There is a gap between cases 2 and 3 when f(n) is larger than $\mathbf{n^{\log_b a}}$ but not polynomially larger.
- If f(n) falls into one of these gaps, or if the regularity condition in case 3 fails to hold, the master method cannot be used to solve the recurrence.

Using The Master Method
Case 1

- T(n) = 9T(n/3) + n
  - **a=9, b=3, f(n) = n**
  - $\mathbf{n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)}$
  - **Since $\mathbf{f(n) = O(n^{\log_3 9 - \varepsilon}) = O(n^{2-0.5}) = O(n^{1.5})}$**
  - **where ε=0.5**
  - **case 1 applies:**

$$T(n) = \Theta\left(n^{\log_b a}\right) \text{when } f(n) = O\left(n^{\log_b a - \varepsilon}\right)$$

  - **Thus the solution is**
    - **$\mathbf{T(n) = \Theta(n^2)}$**

Using The Master Method
Case 2

- $T(n) = T(2n/3) + 1$
  - a=1, b=3/2, f(n) = 1
  - $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
  - Since $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$
  - case 2 applies:

$$T(n) = \Theta\left(n^{\log_b a} \lg n\right) \text{when } f(n) = \Theta\left(n^{\log_b a}\right)$$

  - Thus the solution is
    - $T(n) = \Theta(\lg n)$

Using The Master Method
Case 3

- $T(n) = 3T(n/4) + n \lg n$
  - a=3, b=4, f(n) = n lg n
  - $n^{\log_b a} = n^{\log_4 3} = n^{0.793} = n^{0.8}$
  - Since $f(n) = \Omega(n^{\log_4 3+\varepsilon}) = \Omega(n^{0.8+0.2}) = \Omega(n)$
  - where $\varepsilon \approx 0.2$, and for sufficiently large n,
    - $a \,.\, f(n/b) = 3(n/4) \lg(n/4) < (3/4)\, n \lg n$ for $c = 3/4$
  - case 3 applies:

$$T(n) = \Theta\left(f(n)\right) \text{when } f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$$

  - Thus the solution is
    - $T(n) = \Theta(n \lg n)$

When the Master Method does not apply to recurrence

- $T(n) = 2T(n/2) + n \lg n$
  - $a = 2$, $b = 2$, $f(n) = n \lg n$, and $n^{\log_b a} = n$
  - Note that $f(n) = n \lg n$ is asymptotically larger than $n^{\log_b a} = n$.
  - The problem is that it is not polynomially larger.
  - The ratio $f(n) / n^{\log_b a} = (n \lg n) / n = \lg n$ is asymptotically less than n^{ε} for any positive constant ε.
  - The recurrence falls into the gap between case 2 and case 3.