Methods for Solving Recurrences

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(\mathrm{n})$$

T(n) = ?

- Substitution method
- Recursion-tree
- Master method

Substitution Method

- (1) Guess the form of the solution
- (2) Use mathematical induction to find the constants and show the solution works
    - Works well when it is easy to guess
    - Can be used for upper or lower bounds
- *Example*: merge sort
    - $T(n) = 2T(n/2) + cn$
    - We guess that the answer is $O(n\log n)$
    - Prove it by induction

Substitution Method

$$T(n) = 2T(n/2) + n = ?$$

- Guess T(n) = O($n \log n$)
- Prove T(n) $\leq cn \log n$ for some c

Inductive base: prove the inequality holds for some small n

T(2)=2T(1) + 2 = 4
$cn \log n$ = c * 2 * log 2 = 2c   choose any c $\geq$ 2

Assume true for n/2

$$T(n/2) \leq c\,(n/2)\log(n/2)$$

Substitution Method

Prove that it then must hold for n:

$$
\begin{aligned}
T(n) &= 2\,\underline{T(n/2)} + n \\
&\le 2\,\underline{(c\,(n/2)\log(n/2))} + n \\
&= cn\log(n/2) + n \\
&= cn\log n - cn\log 2 + n \\
&= cn\log n - cn + n \\
&\le cn\log n \qquad \text{for } c \ge 2 \\
&= O(n\log n) \qquad \text{for } c \ge 2
\end{aligned}
$$

Substitution Method

- Experience helps… you know it when you see it…
  - If a recurrence looks familiar… guess a similar solution

$$\boldsymbol{T(n) = 2T(n/2+17) + n}$$

Guess: $T(n) = O(n\log n)$

Why?

- The 17 cannot substantially affect the solution to the recurrence (just a constant)

Iterative Substitution

- Look at the recurrence relation:

$$T(n) = 0 \quad \text{if } n=0$$
$$= T(n - 1) + n \quad \text{if } n > 0$$

- Substituting n – 1 for n in the relation above we get:
  $T(n - 1) = T(n - 2) + (n - 1)$
- Substitute for n – 1 in the original relation:
  $T(n) = (T(n - 2) + (n - 1)) + n$
- We know that $T(n - 2) = T(n - 3) + (n - 2)$
- So substitute this for $T(n - 2)$ above:
  $T(n) = (T(n - 3) + (n - 2)) + (n - 1) + n$

Iterative Substitution

- We see the following pattern:
  $T(n) = T(n - 1) + n$
  $T(n) = (T(n – 2) + (n – 1)) + n$
  $T(n) = (T(n – 3) + (n – 2)) + (n – 1) + n$
  . . .
  $T(n) = T(n – (n – 2)) + 2 + 3 + … + (n – 2) + (n – 1) + n$
  $T(n) = T(n – (n – 1)) + 2 + 3 + … + (n – 2) + (n – 1) + n$
  $T(n) = T(n – (n - 0)) + 2 + 3 + … + (n – 2) + (n – 1) + n$
- We can rewrite $(n – (n – 0))$ as $(n – n)$ or as (0), thus:
  $T(n) = T(0) + 1 + 2 + 3 + … + (n – 2) + (n – 1) + n$

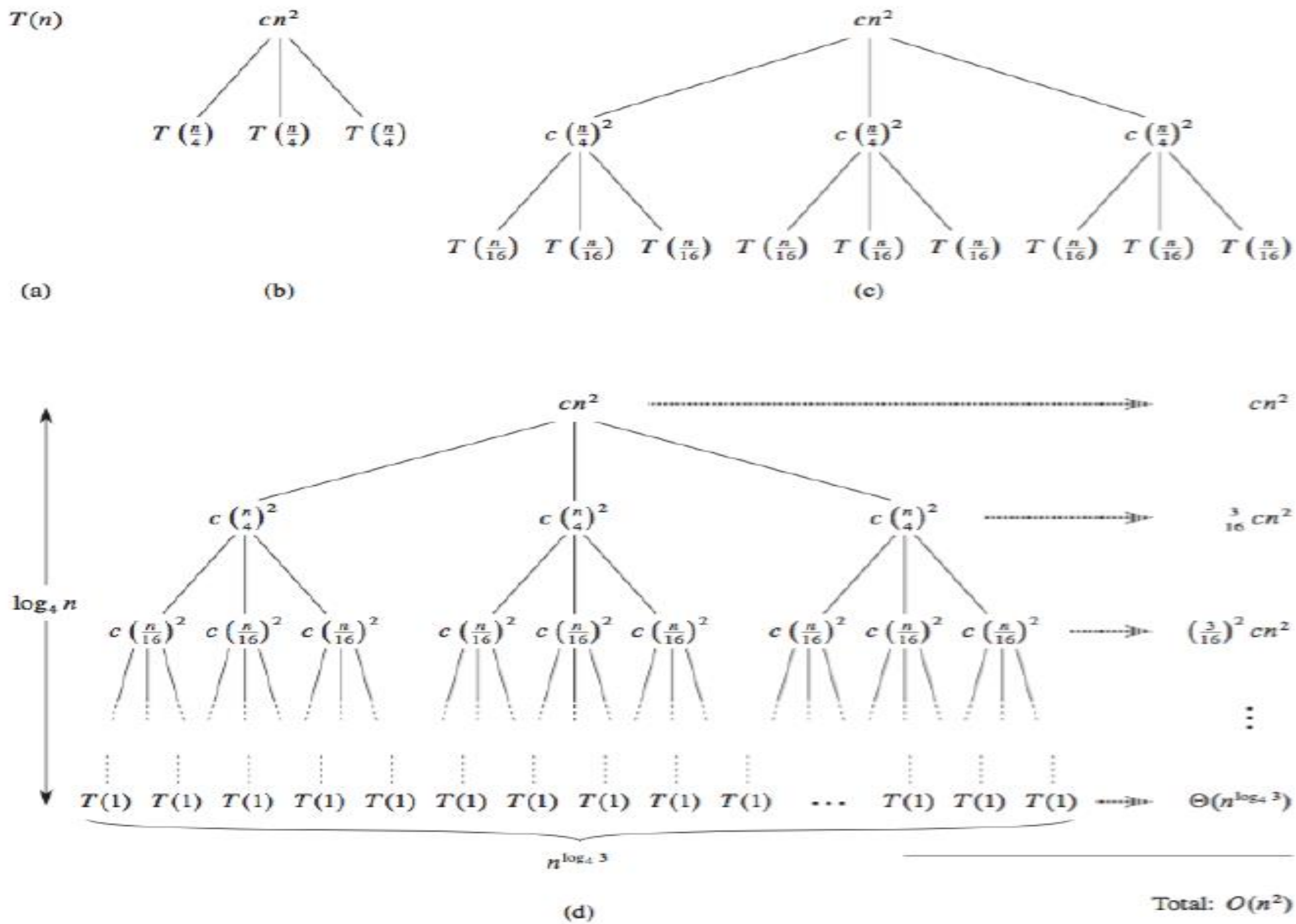
Iterative Substitution

- But we know that $T(0) = 0$ is the base case, so:
  $T(n) = 0 + 1 + 2 + 3 + \ldots + (n - 2) + (n - 1) + n$
- The summation of
  $T(n) = 0 + 1 + 2 + 3 + \ldots + (n - 2) + (n - 1) + n$
  is $T(n) = (n\ (n + 1)\ /2) = ½\ n^2 + ½\ n$
  which we recognize as $O(n^2)$.

Recursion Tree

- Recursion tree
  - Each node represents the cost of a single sub-problem in the set of recursive function invocations
  - Sum the costs within each level of the tree to obtain a set of per-level costs
  - Sum all the per-level costs to determine the total cost of all levels of the recursion
- Useful for generating a good guess for the substitution method

Recursion tree for T(n) = 3T(n/4) + cn^2



$T(n) = 3T(n/4) + cn^2$

- The sub-problem size for a node at depth i is $n/4^i$
  - When the sub-problem size is 1 → $n/4^i = 1$ → $i=\log_4 n$
  - The tree has $\log_4 n+1$ levels (0, 1, 2,.., $\log_4 n$)
- The cost at each level of the tree (0, 1, 2,.., $\log_4(n-1)$)
  - Number of nodes at depth i is 3^i
  - Each node at depth i has a cost of $c(n/4^i)^2$
  - The total cost over all nodes at depth i is $3^i c(n/4^i)^2=(3/16)^i cn^2$
- The cost at depth $\log_4 n$
  - Number of nodes is $3^{\log_4 n} = n^{\log_4 3}$
  - Each contributing cost T(1)
  - The total cost $n^{\log_4 3}T(1) = \Theta(n^{\log_4 3})$

T(n) = 3T(n/4) + cn^2

$$T(n) = cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \ldots + (\frac{3}{16})^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1 - \frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) = \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

Master Theorem

- Provides a cookbook method for solving recurrences of the form
  $T(n)=aT(n/b) + f(n)$
  where $a \geq 1$ and $b > 1$ and $f(n)$ is an asymptotically positive function
- The form of the master theorem is very convenient because divide and conquer algorithms have recurrences of the form
  $T(n) = aT(n/b)+D(n)+C(n)$

Master Theorem

- Provides solutions to the recurrences of the form:

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

$$\text{case1) if } f(n) = O(n^{\log_b a-\varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{case2) if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{case3) if } f(n) = \Omega(n^{\log_b a+\varepsilon}) \text{ for } \varepsilon > 0 \text{ and}$$
$$af(n/b) \le cf(n) \text{ for } c < 1 \quad \text{then } T(n) = \Theta(f(n))$$

What does the Master Theorem say?

- Compare two functions:
  f(n) and $n^{\log_b a}$
- When *f(n)* grows asymptotically slower (Case 1)
  $$T(n) = \Theta(n^{\log_b a})$$
- When the growth rates are the same (Case 2)
  $$T(n) = \Theta(n^{\log_b a} \log n)$$
- When *f(n)* grows asymptotically faster (Case 3)
  $$T(n) = \Theta(f(n))$$

Some Examples of master theorem

$$T(n) = 16T(n/4) + n$$

- Compare two function: *f(n)* and $n^{\log_b a}$
- *f(n)= n*
- *a=16 b= 4* $n^{\log_b a} = n^{\log_4 16} = n^2$
- *f(n)=n* grows asymptotically slower than n^2
  case 1 $T(n) = \Theta(n^{\log_b a})$
  $= \Theta(n^2)$

Some Examples of master theorem

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

- Compare two function: *f(n)* and $n^{\log_b a}$
- *f(n)= n*
- *a=2 b=2*

$$n^{\log_b a} = n^{\log_2 2}$$
$$= n^1$$

- *f(n)=n* grows asymptotically same with n^1
  case 2 $T(n) = \Theta(n^{\log_b a} \log n)$
  $= \Theta(n \log n)$

Some Examples of master theorem

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

- Compare two function: *f(n)* and $n^{\log_b a}$
- $f(n) = 2^n$
- $a=1$ $b=2$

$$\begin{aligned} n^{\log_b a} &= n^{\log_2 1} \\ &= n^0 \end{aligned}$$

- $f(n) = 2^n$ grows asymptotically faster with n^0

  case 3 $T(n) = \Theta(f(n))$

  $= \Theta(2^n)$

Summary

- We talked about:
  - The substitution method
  - The recursion-tree method
  - The master method
- Be able to solve recurrences using all three of these methods