Introduction to Algorithm Design

Lecture Notes 2

ROAD MAP

- **Recurrence Relations**
  - **Exact Solution**
    - Forward substitution
    - Backward substitution
    - Methods similar to those used in solving differential equations
  - **Asymptotic Solution**
    - Guess and Prove
    - Recursion Tree
    - Master theorem

Recurrence Relations

- Generally arise in recursive algorithms
- A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs

Examples :

$$f(n) = f(n/2) + 1$$
$$f(0) = 1$$

$$f(n) + f(n-1) - 6f(n-2) = 2^n - 1$$
$$\underbrace{f(0) = 1/4 \qquad f(1) = 2}_{initial \quad conditions}$$

Analysis of Recursive Algorithms

General plan for analyzing efficiency of recursive algorithm

- Decide on parameters indicating input's size
- Identify the algorithm's basic operation
- Calculate the number of times the basic operation is executed at each recursive call
  - If it varies on different inputs of the same size
    - worst-case, average-case, and best-case efficiencies must be investigated separately
- Set up a recurrence relation for the number of times the basic operation is executed
  - Identify the recursive calls and the input size for each recursive call
  - Identify appropriate initial condition
- Solve the recurrence
  - at least ascertain the order of growth of its solution

Examples

- Finding the largest element in an array
  - One recursive call
  - Two recursive calls
- Insertion sort
- Binary search

Recurrence Relations

- **Solution Methods**
  - **Exact**
    - Forward substitution
    - Backward substitution
    - Methods similar to those used in solving differential equations
  - **Asymptotic**
    - Guess and Prove
    - Recursion Tree
    - Master theorem

Example 1 – Factorial Calculation

- Goal:

  Computing the factoral function **`F(n)=n!`** for an arbitrary non-negative integer **`n`**.

  Since

  $$n! = n \times (n-1) \times \ldots \times 1 = n \times (n-1)! \quad \text{for} \quad n \geq 1 \quad \text{and} \quad 0! = 1$$

  by definition.

  $$F(n) = F(n-1) \times n \quad \text{for} \quad n > 0$$

  $$F(0) = 1$$

Factorial Calculation

Algorithm :

```
if n=0 return 1
else return F(n-1)*n
```

Analysis:

$$M(n) = \underbrace{M(n-1)}_{\substack{\text{to calculate} \\ F(n-1)}} + \underbrace{1}_{\substack{\text{to multiply} \\ F(n-1) \text{ by } n}}$$

$$M(0) = 0$$

Forward Substitution

- Start with the initial condition
- Generate first few terms of the solution
  - Using the recurrence equation and values of previous terms
- Try to guess a pattern
- Form a closed-form formula
- Check the validity of the formula
  - Using induction or
  - Substituting in the recurrence equation and initial condition

Factorial Function

Using forward substitution:

$$
\begin{aligned}
&M(0) = 0 \\
&M(1) = M(0) + 1 = 1 \\
&M(2) = M(1) + 1 = 2 \\
&M(3) = M(2) + 1 = 3 \\
&\vdots \\
&M(n) = n
\end{aligned}
$$

Need to prove the resulting formula

Backward Substitution

- Start with the recurrence equation
- Substitute *f(n-1)*
  - with its value using recurrence equation
- Perform similar substitution few more times
  - for *f(n-2), f(n-3)* etc..
- Try to guess a pattern for *f(n)* in terms of *f(n-i)*
- Check the validity of the pattern
  - usually using induction
- Pick an *i* which makes *n-i* to reach the initial condition
- Obtain a closed-form formula

Factorial Function

Using back substitution:

$$M(n) = M(n-1) + 1$$

$$M(n) = M(n-2) + 1 + 1$$

$$M(n) = M(n-3) + 1 + 1 + 1$$

$$\vdots$$

$$M(n) = M(n-i) + i$$

for $n = i$

$$M(n) = M(0) + n = n$$

Example 2- Towers of Hanoi

Goal:

Transfer *n* disks from peg *A* to peg C using peg B

Approach: (recursive)

- transfer n-1 disks from *A* to *B* using *C*
- move largest disk from *A* to *C*
- transfer n-1 disks from *B* to *C* using *A*

Total number of moves

T(n) = 2T(n-1) + 1

T(1) = 1

Towers of Hanoi

$$T(n) = 2T(n-1) + 1$$
$$T(1) = 1$$

Solution by backward substitution

$$T(n) = 2T(n-1) + 1$$
$$T(n) = 4\big(T(n-2)\big) + 2 + 1$$
$$T(n) = 4\big(2T(n-3) + 1\big) + 2 + 1$$
$$T(n) = 8T(n-3) + 4 + 2 + 1$$
$$\vdots$$
$$T(n) = 2^{i}T(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^{1} + 1$$ **← needs proving**

$$when \qquad i = n-1$$
$$T(n) = 2^{n-1}T(1) + 2^{n-2} + \ldots + 2^{1} + 1$$
$$T(n) = 2^{n} - 1 = \theta(2^{n})$$

Example

$$T(n) = 2T(\sqrt{n}) + 1 \qquad T(2) = 0$$

$$T(n) = 2T(n^{1/2}) + 1$$
$$T(n) = 2\left(2T(n^{1/4}) + 1\right) + 1$$
$$T(n) = 4T((n^{1/4}) + 1 + 2$$
$$T(n) = 8T(n^{1/8}) + 1 + 2 + 3$$
$$\vdots$$
$$T(n) = 2^i T(n^{1/2^i}) + 2^0 + 2^1 + \ldots + 2^{i-1}$$

$$n^{1/2^i} = 2 \quad \Rightarrow \quad i = \log\log(n)$$
$$T(n) = 2^0 + \ldots + 2^{\log\log n - 1} = \theta(\log n)$$

ROAD MAP

- **Recurrence Relations**
  - **Exact Solution**
    - Forward substitution
    - Backward substitution
    - Methods similar to those used in solving differential equations
  - **Asymptotic Solution**
    - Guess and Prove
    - Recursion Tree
    - Master theorem

Linear Recurrences with Constant Coefficients

General form

$$f(n) - \sum_{i=1}^{k} c_i f(n-i) = g(n)$$

where

c_i ‘s are some fixed numbers

k is any integer

Linear Recurrences with Constant Coefficients

- Example

$$f(n) = -f(n-1) + 6f(n-2) + 2^n - 1$$

$$f(0) = 1/4 \qquad f(1) = 1/4$$

1. Write the equations as

$$f(n) - \sum_{i=1}^{k} c_i f(n-i) = g(n)$$

$$f(n) + f(n-1) - 6f(n-2) = \underbrace{2^n - 1}_{g(n)}$$

Linear Recurrences with Constant Coefficients

2. Write the characteristics polynomial for homogeneous part and factor it

$$p(r) = r^2 + r - 6 = (r+3)(r-2)$$

3. Find the simplest equation for which *g(n)* is solution

$$g(n) = 2^n - 1^n$$

$$q(r) = (r-2)(r-1)$$

Linear Recurrences with Constant Coefficients

In general

$$g(n) = b^n s(n) \ldots \Rightarrow q(r) = (r - b)^{d+1}$$

$$d = \text{degree of } s(n)$$

Linear Recurrences with Constant Coefficients

4. $P(r) = p(r).q(r)$

   find the solution for *P(r)* => *H(n)*

$$P(r) = (r+3)(r-2)^2(r-1)$$

$$H(n) = A(-3)^n + B2^n + Cn2^n + D$$

5. Delete the part of H(n) coming from p(r)

$$p(r) = (r+3)(r-2)$$

$$D(n) = A(-3)^n + B2^n$$

$$R(n) = Cn2^n + D$$

Linear Recurrences with Constant Coefficients

6. Plug *R(n)* into original equation and solve the constants

$$(Cn2^n + D) + (C(n-1)2^{n-1} + D).6(C(n-2)2^{n-2} + D) = 2^n - 1$$

$$C = 2/5 \qquad D = 1/4$$

7. R(n) is a particular solution

$$R(n) = \frac{2}{5}n2^n + \frac{1}{4}$$

8. Append R(n) and D(n)

$$A(-3)^n + B2^n + \frac{2}{5}n2^n + \frac{1}{4}$$

Linear Recurrences with Constant Coefficients

9. Use initial conditions to find other constants

For *n=0*

$$A(-3)^0 + B2^0 + 0 + \frac{1}{4} = \frac{1}{4}$$

$$A + B = 0$$

For *n=1*

$$A(-3)^1 + B2^1 + \frac{2}{5}2^1 + \frac{1}{4} = \frac{1}{4}$$

$$-3A + 2B + \frac{4}{5} = 0$$

$$A = -\frac{4}{5} \qquad B = \frac{4}{5}$$

Linear Recurrences with Constant Coefficients

Example

$$T(n) = 2T(n-1) + n + 2^n \qquad\qquad T(0) = 0$$

1) $t_n - 2t_{n-1} = n + 2^n$

2) $p(r) = r - 2$

3) $q(r) = (r-1)^2 (r-2)$

4) $p(r) = (r-1)^2 (r-2)^2$

$$T(n) = A1^n + Bn1^n + C2^n + Dn2^n$$

Linear Recurrences with Constant Coefficients

5) $p(r) = r - 2$

$D(n) = C2^n$

$R(n) = A1^n + Bn1^n + Dn2^n$

6) $(Dn2^n + Bn + A) - 2(D(n-1)2^{n-1} + B(n-1) + A) = n + 2^n$

$(Dn2^n - Dn2^n + D2^n + Bn - 2Bn + A + 2B - 2A = n + 2^n$

$D = 1 \qquad D2^n = 2^n$

$B = -1 \qquad -Bn = n$

$A = -2 \qquad 2B - A = 0$

Linear Recurrences with Constant Coefficients

7) $R(n) = n2^n - n - 2$

8) $T(n) = n2^n + c2^n - n - 2$

9) $T(0) = 0$

$$H(a) = 02^0 + C2^0 - 0 - 2 = 0$$

$$C - 2 = 0 \qquad C = 2$$

$$T(n) = n2^n + 2^{n+1} - 2 - n$$

Example

$$f(n) = f(n-1) + f(n-2) \qquad f(0) = 0,\, f(1) = 1$$

$$f(n) - f(n-1) - f(n-2) = 0$$

$$r^2 - r - 1 = \left(r - \frac{1-\sqrt{5}}{2} \right)\left(r - \frac{1+\sqrt{5}}{2} \right)$$

$$A\left(\frac{1+\sqrt{5}}{2} \right)^n + B\left(\frac{1-\sqrt{5}}{2} \right)^n = f(n)$$

$$f(0) = A + B = 0$$

$$f(1) = A\left(\frac{1+\sqrt{5}}{2}\right) + B\left(\frac{1-\sqrt{5}}{2}\right) = 1$$

$$f(1) = A + \frac{A\sqrt{5}}{2} + B - \frac{B\sqrt{5}}{2} = 1$$

$$f(1) = \left(A - B\right)\frac{\sqrt{5}}{2} = 1$$

$$A - B = \frac{2}{\sqrt{5}} \qquad A = \frac{1}{\sqrt{5}} \qquad B = \frac{-1}{\sqrt{5}}$$

$$f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$for \quad l\arg e \quad n \qquad \left(\frac{1-\sqrt{5}}{2}\right)^n \rightarrow 0$$

$$f(n) \leq \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

$$f(n) \leq \frac{1}{\sqrt{5}} 2^n$$

$$f(n) = O(2^n)$$

- Example

$$f(n) = f(n-1) * f(n-2)$$

take log

- Example

$$(n-1)T(n) = nT(n-1) + (n-1)$$

divide by n(n-1)

- Example

$$T(n) = T(n/2) + T(n/4) + n$$

substitute n=2^m

$$S(m) = S(m-1) + S(m-2) + 2^m$$

ROAD MAP

- **Recurrence Relations**
  - Exact Solution
    - Unfolding
    - Guess and proove
    - Generating functions
    - Methods similar to those used in solving difference equations
  - **Asymptotic Solution**
    - **Guess and Prove**
    - **Recursion Tree**
    - **Master theorem**

Guess and Prove

- Two steps
  - Guess the form of the solution
  - Use mathematical induction to show the solution works
- Useful when it is easy to guess the form of the answer

- Example: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

T(n) = O(n lg n) ?

Guess and Prove

- Example: $T(n) = 2T(\lfloor n/2 \rfloor) + n$ if $n > 1$ and $T(1) = 1$
- Guess T(n) = O(*n* lg *n*) → prove T(n) ≤ *cn* lg *n* for some c
  - By mathematical induction
  - Inductive base: prove the inequality holds for some small n
    - n = 1? → T(1) ≤ c * 1 * log 1 = 0 ➔ but T(1) = 1
    - start from T(2) = 4 or T(3) = 5 ➔ choose any c ≥ 2
      - OK because asymptotic notation requires us to prove T(n) ≤ *cn* lg *n for n* ≥ n_0
    - Trick: extend boundary conditions to make the inductive assumption work for small n
  - Induction assumption: assume the bound holds for $\lfloor n/2 \rfloor$

$$T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor)$$

Guess and Prove

- Induction : holds for n ?

$$
\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\le 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n \\
&\le cn \lg \lfloor n/2 \rfloor + n \\
&\le cn(\lg n - \lg 2) + n \\
&\le cn \lg n - cn + n \\
&\le cn \lg n
\end{aligned}
$$

(holds as long as c ≥1)

Making a good guess

- Need experience and creativity
- Use recursion trees to generate good guesses
- If a recurrence is similar to one you are familiar, then guessing a similar solution is reasonable
  - $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ ➔T(n) = O(n lg n) ➔ Why?
  - The additional term (17) cannot substantially affect the solution to the recurrence (when n is large)
- Prove loose upper and lower bounds and then reduce the range of uncertainty

Subtleties

- Sometimes you may guess right but the math does not seem to work out in the induction
  - Inductive assumption is not strong enough
  - Revise the guess by subtracting a lower-order term

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$ ➔ O(n) ??? Show T(n) ≤ cn

$$T(n) \le c(\lfloor n/2 \rfloor) + c(\lceil n/2 \rceil) + 1 = cn + 1$$ Seem wrong!!

New guess T(n) ≤ cn-b

$$T(n) \le c(\lfloor n/2 \rfloor - b) + c(\lceil n/2 \rceil - b) + 1 = cn - 2b + 1$$

$\le cn - b$ (as long as b ≥1)

Avoiding Pitfalls

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

- Falsely prove T(n)=O(n) by guessing T(n) ≤ cn and…

$$T(n) \leq 2(c\lfloor n/2 \rfloor) + n \leq cn + n = O(n)$$ ← Wrong

You have to prove the exact form of the inductive hypothesis

ROAD MAP

- **Recurrence Relations**
  - Exact Solution
    - Unfolding
    - Guess and proove
    - Generating functions
    - Methods similar to those used in solving difference equations
  - **Asymptotic Solution**
    - **Guess and Prove**
    - **Recursion Tree**
    - **Master theorem**

Recursion Trees

- Recursion tree
  - Each node represents the cost of a single subproblem somewhere 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 when the recurrence describes the running time of a divide-and-conquer algorithm
- Useful for generating a good guess, which is then verified by the guess and prove method
  - Sloppiness are allowed

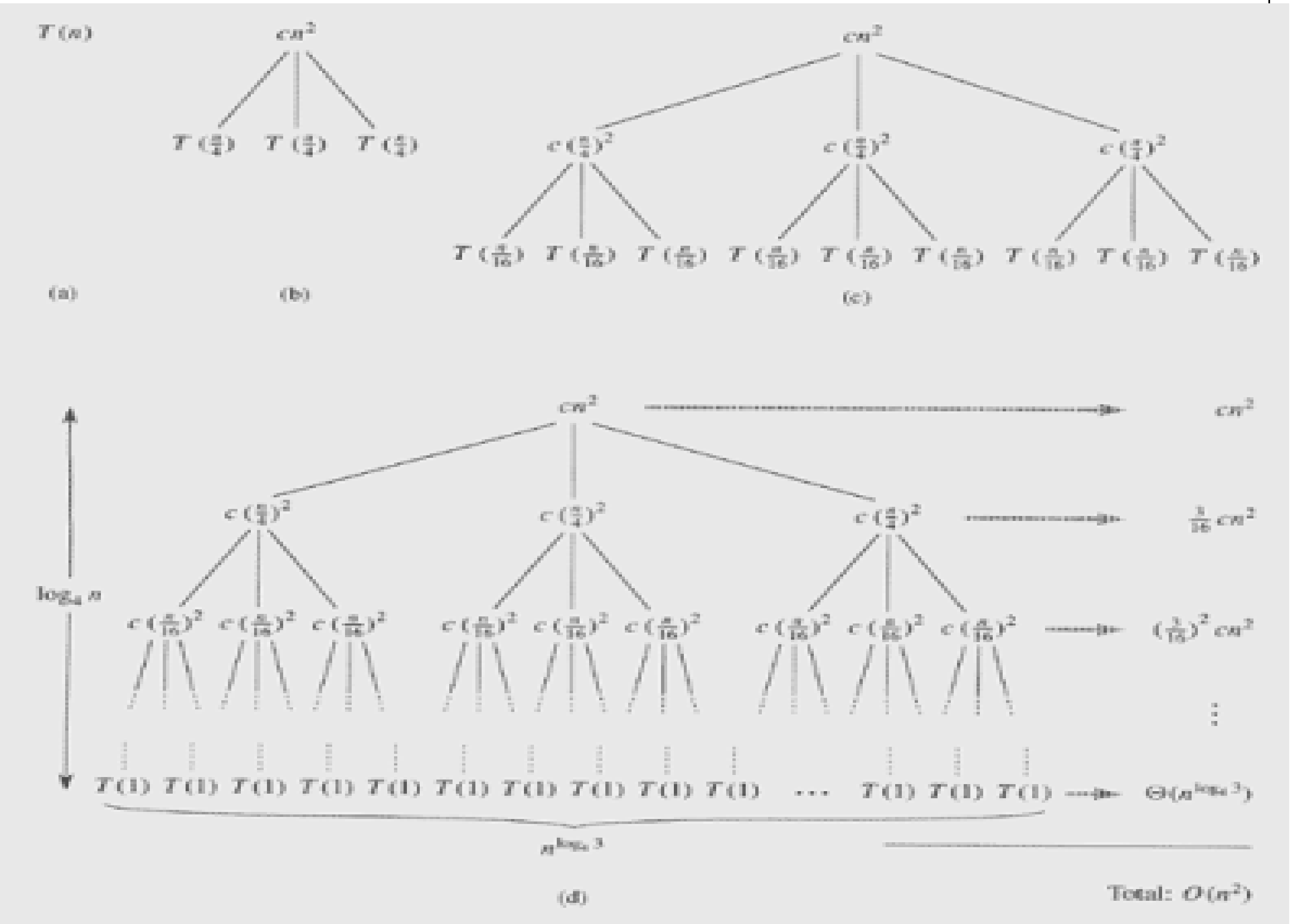
Recursion Trees

- Example:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

  - Ignore the floors and write out the implicit coefficient c > 0 (sloppiness)

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



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

- The subproblem size for a node at depth i is $n/4^i$
  - When the subproblem size is 1 $\rightarrow$ $n/4^i = 1$ $\rightarrow$ $i=\log_4 n$
  - 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 (Cont.)

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

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

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

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

$$= O(n^2)$$

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

Prove $T(n)=O(n^2)$ is an upper bound by the guess and prove method

➔ $T(n) \le dn^2$ for some constant $d > 0$

$$T(n) \le 3T(\lfloor n/4 \rfloor) + cn^2$$

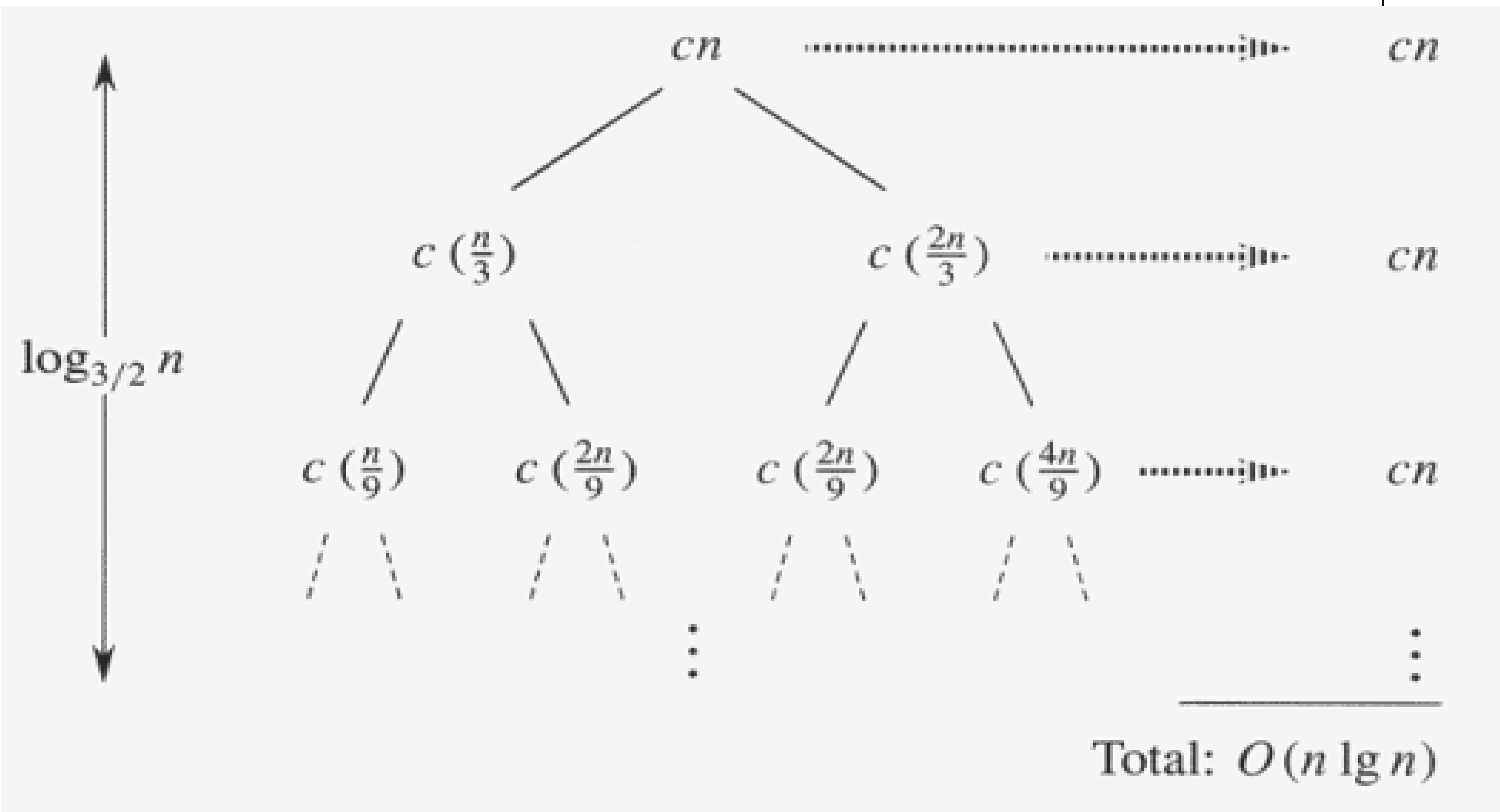
$$\le 3d\lfloor n/4 \rfloor^2 + cn^2$$

$$\le 3d(n/4)^2 + cn^2$$

$$= \frac{3}{16}dn^2 + cn^2$$

$$\le dn^2$$ (as long as $d \ge (16/13)c$)

A recursion tree for T(n)=T(n/3) + T(2n/3) + cn



$$\left(\frac{2}{3}\right)^x n = 1 \quad \Rightarrow \quad n = (3/2)^x \quad \Rightarrow \quad x = log_{(3/2)} n$$

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

- # of levels * cost of each level = O($cn \log_{3/2} n$)=O($n \lg n$)
- Complication
  - If the tree is complete binary tree, # of leaves = $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$
  - But the tree is not complete
  - Go down from the root, more and more internal nodes are absent
- Verify O($n \lg n$) is an upper bound by the substitution method

ROAD MAP

- **Recurrence Relations**
  - Exact Solution
    - Unfolding
    - Guess and proove
    - Generating functions
    - Methods similar to those used in solving difference equations
  - **Asymptotic Solution**
    - **Guess and Prove**
    - **Recursion Tree**
    - **Master theorem**

Master Theorem

- Cooked up solution for some recurrences
  - Describing the running time of an algorithm that divides a problem of size *n* into *a* subproblems, each of size *n/b*

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

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$\text{where} \qquad \frac{n}{b} = \left\lfloor \frac{n}{b} \right\rfloor \text{ or } \left\lceil \frac{n}{b} \right\rceil \qquad \text{then}$$

Master Theorem

1. $If \quad f(n) = O(n^{\log_b a - \in}), \quad \in > 0$

   $then \quad T(n) = \theta(n^{\log_b a})$

2. $If \quad f(n) = \theta(n^{\log_b a})$

   $then \quad T(n) = \theta(n^{\log_b a} \log n)$

3. $If \quad f(n) = \Omega(n^{\log_b a + \in}), \quad \in > 0$

   $and \quad if \quad af(n/b) \le cf(n) \quad c > 1 \quad n > m_0$

   $then \quad T(n) = \theta(f(n))$

Master Theorem

- The solution is determined by the larger of f(n) and $n^{\log_b a}$
  - Case 1: *f(n)* must be polynomially smaller than $n^{\log_b a}$
    - *f(n)* must be asymptotically smaller than $n^{\log_b a}$ by a factor of n^{ε}
  - Case 3: *f(n)* must be polynomially larger than $n^{\log_b a}$, and in addition satisfy the regularity condition that $af(n/b) \leq cf(n)$
- The three cases do not cover all possibilities for *f(n)*

Using the master method

- Example 1: T(n)=9T(n/3) + n (Case 1)
  - a=9, b=3, f(n) = n
  - $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2), f(n) = O(n^{\log_3 9-1})$
  - T(n) = θ(n^2)
- Example 2: T(n) = T(2n/3) + 1 (Case 2)
  - a=1, b=3/2, f(n)=1
  - $n^{\log_b a} = n^{\log_{3/2} 1} = 1, f(n) = 1 = \Theta(1)$
  - T(n) = θ(lg *n*)

Using the master method (Cont.)

- Example 3: T(n) = 3T(n/4) + n lg n (Case 3)
  - a=3, b=4, f(n) = n lg n
  - $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$ $f(n) = \Omega(n^{\log_4 3+\varepsilon})$ ($\varepsilon \approx 0.2$)
  - For sufficiently large n, af(n/b) = 3(n/4)lg(n/4) ≤ (3/4)n lg n=cf(n) for c=3/4
  - T(n) = θ(n lg n)
- Example 4: T(n) = 2T(n/2) + n lg n
  - a=2, b=2, f(n) = n lg n
  - $f(n)/n^{\log_b a} = n\lg n/n = \lg n$ (Asymptotically less than n^{ε} for any ε)
  - $n^{\log_b a} = n^{\log_2 2} = n$
  - Falls into the gap between case 2 and case 3

Example

- Binary search

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

$$a = 1 \qquad b = 2 \qquad f(n) = c$$

$$\log_b a = 0 \qquad f(n) = \theta(n^0) = \theta(1)$$

$$T(n) = \theta(n^0 \log n) = \theta(\log n)$$

Example

- Merge sort

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

$$a = 2 \qquad b = 2 \qquad f(n) = cn$$

$$\log_b a = 1 \qquad f(n) = \theta(n^1) = \theta(n)$$

$$T(n) = \theta(n \log n)$$

Example

$$T(n) = 7T(n/2) + 18n^2$$

$a = 7$ $\qquad$ $b = 2$ $\qquad$ $f(n) = 18n^2$

$$\log_b a \cong 2.81$$

$18n^2 = O(n^{2.81-\in})$ $\qquad$ $\in > 0$

$$T(n) = \theta(n^{\log_2 7})$$

Example

$$T(n) = 9T(n/3) + 4n^6$$

$$a = 9 \qquad b = 3 \qquad f(n) = 4n^6$$

$$\log_b a = 2$$

$$4n^6 = \Omega(n^{\log_b a + \in}) \qquad \in > 0$$

$$af(n/b) = 9 * 4(\frac{n}{3})^6 \leq ?\, c4n^6$$

$$\frac{9}{3^6} n^6 \leq cn^6$$

$$c = 2 \qquad m_0 = 1$$

$$T(n) = \theta(n^6)$$

Exact Solution

$$Let \qquad n = 3^m$$

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

$$T(n) = 9(9T(n/3^2) + n/3 + n$$

$$T(n) = 9^2 T(n/3^2) + 3n + n$$

$$T(n) = 9^2 (9T(n/3^3) + n/3^2) + 3n + n$$

$$T(n) = 9^3 T(n/3^3) + 3^2 n + 3n + n$$

$$m = \log_3 n$$

$$T(n) = 9^m T(1) + n(3^{m-1} + ... + 1)$$

$$T(n) = 9^{\log_3 n} T(1) + n\frac{3^m - 1}{3 - 1}$$

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

$$T(n) = \theta(n^2)$$