Chapter 4: Recurrences

Chi-Yeh Chen
陳奇業
成功大學資訊工程學系

Overview

- Recall that in divide-and-conquer, we solve a problem recursively, applying three steps at each level of the recursion:
  - Divide the problem into a number of subproblems that are smaller instances of same problem.
  - Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
  - Combine the solutions to the subproblems into the solution for the original problem.

Overview (cond’t)

- When the subproblems are large enough to solve recursively, we call that the recursive case.

- Once the subproblems become small enough that we no longer recurse, we call that the base case.

Overview (cond't)

- A recurrence is an equation or inequality that describes a function in terms of
  - one or more base cases, and
  - itself, with smaller arguments
- For example, the worst-case running time $T(n)$ of the MEGRE-SORT procedure is the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Solution: $T(n) = \Theta(n\lg n)$.

- Example
  - $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$ A recursive version of linear search

    Solution: $T(n) = n$.

  - $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$

    Solution: $T(n) = n\lg n + n$.

  - $T(n) = \begin{cases} 0 & \text{if } n = 2 \\ T(\sqrt{n}) + 1 & \text{if } n > 2 \end{cases}$

    Solution: $T(n) = \lg\lg n$.

  - $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/3) + T(2n/3) + n & \text{if } n > 1 \end{cases}$ A recursive algorithm might divide subproblems into unequal size.

    Solution: $T(n) = \Theta(n\lg n)$.

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

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 1 + 1 \\ &= T(1) + 1 + \cdots + 1 \\ &= n \end{aligned}$$

- Example

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

$$
\begin{aligned}
T(n) &= T(n-1) + \frac{1}{n} \\
&= T(n-2) + \frac{1}{n-1} + \frac{1}{n} \\
&= 1 + \frac{1}{2} + \cdots + \frac{1}{n} \\
&= \mathrm{O}(\lg n)
\end{aligned}
$$

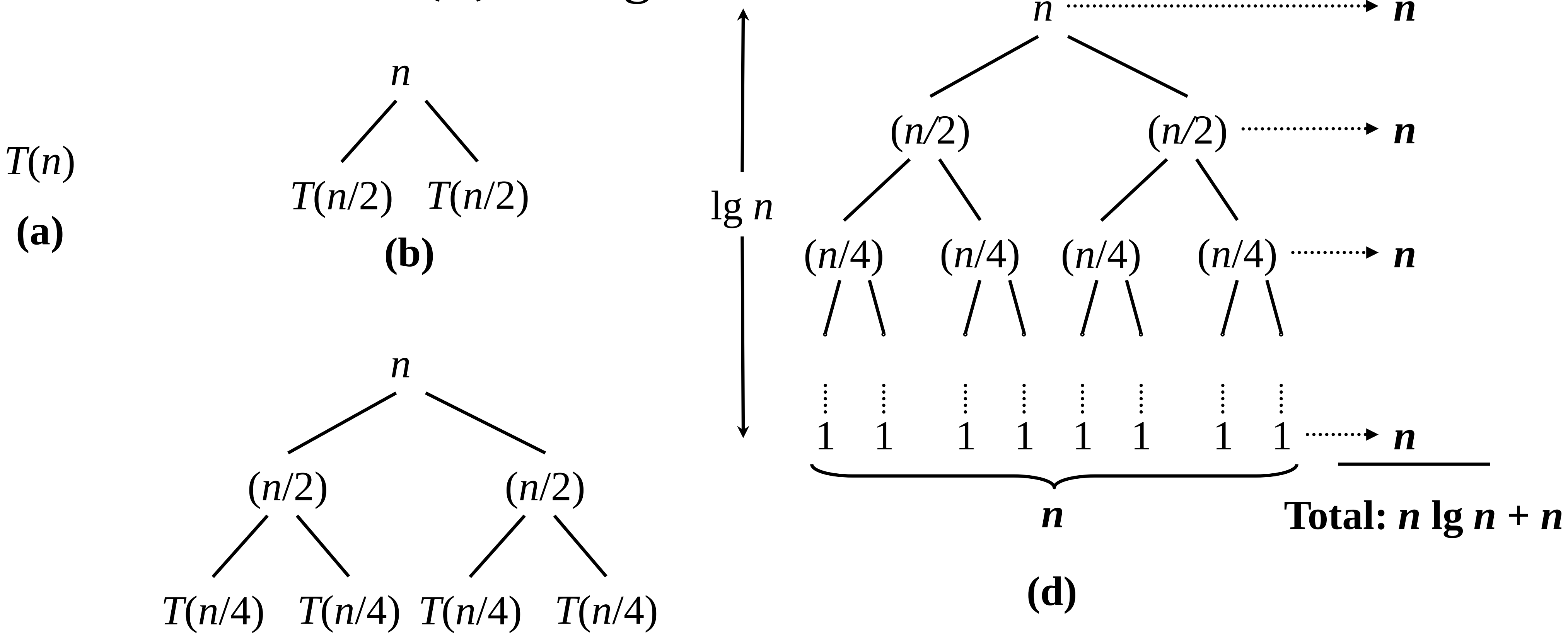
- Euler’ s constant γ is defined by

$$\gamma = \lim_{n \to \infty} \left(\sum_{k=1}^{n} \frac{1}{k} - \log n \right)$$

- $\sum_{k=1}^{n} \frac{1}{k} \approx \log n + \gamma + \cdots$

- Example
  - $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$

    Solution: $T(n) = n\lg n + n$.



- Example
  - $T(n) = \begin{cases} 0 & \text{if } n = 2 \\ T(\sqrt{n}) + 1 & \text{if } n > 2 \end{cases}$

    Solution: $T(n) = \lg\lg n$.

    Let $m = \lg n$ and $S(m) = T(2^m)$.

    $T(2^m) = T\left(2^{m/2}\right) + 1 \Rightarrow S(m) = S(m/2) + 1$

    Using the master theorem, $m^{\log_b a} = m^{\log_2 1} = m^0 = 1$ and $f(m) = 1$. Since $f(m) = \Theta(m^{\log_b a})$, case 2 applies and $S(m) = \Theta(\lg m)$.

    Therefore, $T(n) = \Theta(\lg\lg n)$.

Overview (cond't)

- This chapter offers three methods for solving recurrences:
  - Substitution method: We guess a bound and then use mathematical induction to prove our guess correct.
  - Recursion-tree method: converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion. We use techniques for bounding summations to solve the recurrence.
  - Master method: It provides bounds for recurrences of the form

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

    where $a \geq 1$, $b > 1$, and $f(n)$ is a given function.

Overview (cond't)

- We shall see recurrences that are not equalities but rather inequalities
  - $T(n) \leq 2T(n/2) + \Theta(n)$. We will couch its solution using O-notation rather than Θ-notation.
  - $T(n) \geq 2T(n/2) + \Theta(n)$. We will use Ω-notation in its solution

- Many technical issues:
  - Floors and ceilings
    [Floors and ceilings can easily be removed and don't affect the solution to the recurrence.]

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1. \end{cases}$$

  - Exact vs. asymptotic functions
  - Boundary condition (ignore)

In algorithm analysis, we usually **express both the recurrence and its solution using asymptotic notation**.

- E.g. $T(n) = 2T(n/2) + \Theta(n)$, with solution $T(n) = \Theta(n \lg n)$
- The boundary conditions are usually expressed as "$T(n) = O(1)$ for sufficiently small n."
- When we desire an exact, rather than an asymptotic, solution, we need to deal with boundary conditions.
- In practice, we just use asymptotic most of the time, and we ignore boundary conditions.

The maximum-subarray problem

- You are allowed to buy one unit of stock only one time and the sell it at a later date.

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

The maximum-subarray problem (cond't)

- A brute-force solution
  - Just try every possible pair of buy and sell dates in which the buy date precedes the sell date.
  - A period of n day has $\binom{n}{2}$ such pairs of dates
  - $\binom{n}{2} = \Theta(n^2)$

The maximum-subarray problem (cond’t)

- A transformation
  - We want to find the nonempty, contiguous subarray of A whose values have the largest sum.
  - We call this contiguous subarray the **maximum subarray**.
  - We stall need to check $\binom{n-1}{2} = \Theta(n^2)$ subarrays for a period of n days.

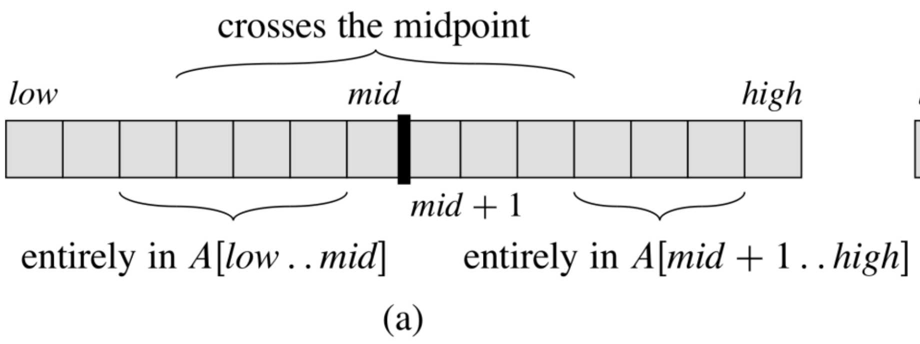
buy (day 7) sell (day 11)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray (entries 8–11)

The maximum-subarray problem (cond't)

- A solution using divide-and-conquer
  - Suppose we want to find a maximum subarray of the subarray $A[low \ldots high]$
  - We find the midpoint, say mid, of the subarray, and consider the subarrays $A[low \ldots mid]$ and $A[mid + 1 \ldots high]$



FIND-MAX-CROSSING-SUBARRAY(A, low, mid, $high$)

```
left-sum = -∞
sum = 0
for i = mid downto low do
    sum = sum + A[i]
    if sum > left-sum then
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high do
    sum = sum + A[j]
    if sum > right-sum then
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

FIND-MAXIMUM-SUBARRAY(A, low, $high$)

```
if high == low then
    return (low, high, A[low])
else
    mid = ⌊(low + high)/2⌋
    (left-low,left-high,left-sum) =
     FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low,right-high,right-sum) =
     FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low,cross-high,cross-sum) =
     FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum then
        return (left-low, left-high, left-sum)
    else if right-sum ≥ left-sum and right-sum ≥ cross-sum then
        return (right-low,right-high,right-sum)
    else
        return (cross-low,cross-high,cross-sum)
```

The maximum-subarray problem (cond't)

- Analyzing the divide-and conquer algorithm
  - We denote by $T(n)$ the running time of FIND-MAXIMUM-SUBARRAY on a sub array of n elements.
  - The base case, when $n = 1$: line 2 takes constant time, and so $T(1) = \Theta(1)$.
  - line 5 and 6: is on a sub array of $n/2$ elements, and so we spend $T(n/2)$ time solving each of them.
  - FIND-MAX-CROSSING-SUBARRAY takes $\Theta(n)$.

The maximum-subarray problem (cond't)

- Analyzing the divide-and conquer algorithm
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$
  - $T(n) = \Theta(n\lg n)$

Matrix Multiplication

- If $A = (a_{ij})$ and $\mathrm{B} = (b_{ij})$ are square $n \times n$ matrices, then in the product $C = A \cdot B$, we define the entry c_{ij}, for $i, j = 1,2, \dots, n$, by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Matrix Multiplication (cond’t)

SQUARE-MATRIC-MULTIPLY(A, B)

```
n = A.rows
let C be a new n × n matrix
for i = 1 to n do
    for j = 1 to n do
        c_ij = 0
        for k = 1 to n do
            c_ij = c_ij + a_ik · b_kj
return (C)
```

Divide-and-Conquer Algorithm

- Suppose that we partition each of A, B, and C into four $n/2 \times n/2$ matrices
  $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$, $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$
  so that we rewrite the equation $C = A \cdot B$ as
  $$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Divide-and-Conquer Algorithm (cond't)

- $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$
- $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
- $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$
- $C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$

SQUARE-MATRIC-MULTIPLY-RECURSIVE(A, B)

$n = A.rows$
let C be a new $n \times n$ matrix
if $n == 1$ **then**
    $c_{11} = a_{11} \cdot b_{11}$
else
    partition A, B, and C as in equations (4.9)
    $C_{11} =$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{11}, B_{11}) $T(n/2)$
$\Theta(n^2)$ $\boxed{+}$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{12}, B_{21}) $T(n/2)$
    $C_{12} =$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{11}, B_{12}) $T(n/2)$
$\Theta(n^2)$ $\boxed{+}$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{12}, B_{22}) $T(n/2)$
    $C_{21} =$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{21}, B_{11}) $T(n/2)$
$\Theta(n^2)$ $\boxed{+}$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{22}, B_{21}) $T(n/2)$
    $C_{22} =$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{21}, B_{12}) $T(n/2)$
$\Theta(n^2)$ $\boxed{+}$ SQUARE-MATRIC-MULTIPLY-RECURSIVE(A_{22}, B_{22}) $T(n/2)$
return (C)

Divide-and-Conquer Algorithm (cond't)

- Analyzing the divide-and conquer algorithm
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$
  - Using the master theorem, $n^{\log_2 8} = n^3$ and $f(n) = \Theta(n^2)$. Since $f(n) = O(n^3)$, case 1 applies. Therefore, $T(n) = \Theta(n^3)$.

Strassen's method

- Divide the input matrices A and B and output matrix C into $n/2 \times n/2$ submatrices.
- Create 10 matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1.
- Using the submatrices created in step 1 and the 10 matrices created in step2, recursively compute seven matrix products $P_1, P_2, \ldots, P_7$. Each matrix P_i is $n/2 \times n/2$.
- Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix $C$by adding and subtraction various combinations of the P_i matrices.

- $S_1 = B_{12} - B_{22}$
- $S_2 = A_{11} + A_{12}$
- $S_3 = A_{21} + A_{22}$
- $S_4 = B_{21} - B_{11}$
- $S_5 = A_{11} + A_{22}$
- $S_6 = B_{11} + B_{22}$
- $S_7 = A_{12} - A_{22}$
- $S_8 = B_{21} + B_{22}$
- $S_9 = A_{11} - A_{21}$
- $S_{10} = B_{11} + B_{12}$

Strassen's method (cond't)

- $P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$
- $P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$
- $P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$
- $P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$
- $P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$
- $P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$
- $P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$

Strassen's method (cond't)

- $C_{11} = P_5 + P_4 - P_2 + P_6$
- $C_{12} = P_1 + P_2$
- $C_{21} = P_3 + P_4$
- $C_{22} = P_5 + P_1 - P_3 - P_7$

Strassen's method (cond't)

- $C_{11} = P_5 + P_4 - P_2 + P_6 = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$
  $A_{11} \cdot B_{11} + \cancel{A_{11} \cdot B_{22}} + \cancel{A_{22} \cdot B_{11}} + \cancel{A_{22} \cdot B_{22}}$
  $\cancel{A_{22} \cdot B_{21}} - \cancel{A_{22} \cdot B_{11}}$
  $-\cancel{A_{11} \cdot B_{22}} - \cancel{A_{12} \cdot B_{22}}$
  $A_{12} \cdot B_{21} + \cancel{A_{12} \cdot B_{22}} - \cancel{A_{22} \cdot B_{21}} - \cancel{A_{22} \cdot B_{22}}$

Strassen's method (cond't)

- $C_{12} = P_1 + P_2 = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$

  $A_{11} \cdot B_{12} - \cancel{A_{11} \cdot B_{22}}$

  $\cancel{A_{11} \cdot B_{22}} + A_{12} \cdot B_{22}$

Strassen's method (cond't)

- $C_{21} = P_3 + P_4 = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$
  $A_{21} \cdot B_{11} + \cancel{A_{22} \cdot B_{11}}$
  $A_{22} \cdot B_{21} - \cancel{A_{22} \cdot B_{11}}$

Strassen's method (cond't)

- $C_{22} = P_5 + P_1 - P_3 - P_7 = A_{22} \cdot B_{22} + A_{21} \cdot B_{12}$
  $\cancel{A_{11} \cdot B_{11}} + \cancel{A_{11} \cdot B_{22}} + \cancel{A_{22} \cdot B_{11}} + A_{22} \cdot B_{22}$
  $\cancel{A_{11} \cdot B_{12}} - \cancel{A_{11} \cdot B_{22}}$
  $-\cancel{A_{21} \cdot B_{11}} - \cancel{A_{22} \cdot B_{11}}$
  $-\cancel{A_{11} \cdot B_{11}} - \cancel{A_{11} \cdot B_{12}} + \cancel{A_{21} \cdot B_{11}} + A_{21} \cdot B_{12}$

STRASSEN-MATRIC-MULTIPLY-RECURSIVE(A, B)

$n = A.rows$
Let C be a new $n \times n$ matrix
if $n == 1$ **then**
    $c_{11} = a_{11} \cdot b_{11}$
else
    Partition A, B, and C into $n/2 \times n/2$ submatrices
    Let $S_1, S_2, \ldots, S_{10}$ be $n/2 \times n/2$ matrices $\Theta(n^2)$
    Compute $S_1, S_2, \ldots, S_{10}$ by sum or difference of two submatrices of A and B $\Theta(n^2)$
    P_1 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(A_{11}, S_1) $T(n/2)$
    P_2 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(S_2, B_{22}) $T(n/2)$
    P_3 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(S_3, B_{11}) $T(n/2)$
    P_4 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(A_{22}, S_4) $T(n/2)$
    P_5 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(S_5, S_6) $T(n/2)$
    P_6 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(S_7, S_8) $T(n/2)$
    P_7 =STRASSEN-MATRIC-MULTIPLY-RECURSIVE(S_9, S_{10}) $T(n/2)$
    $C_{11} = P_5 + P_4 - P_2 + P_6$ $\Theta(n^2)$
    $C_{12} = P_1 + P_2$ $\Theta(n^2)$
    $C_{21} = P_3 + P_4$ $\Theta(n^2)$
    $C_{22} = P_5 + P_1 - P_3 - P_7$ $\Theta(n^2)$
return (C)

Strassen's method (cond't)

- Analyzing the divide-and conquer algorithm
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$
  - Using the master theorem, $n^{\log_2 7} = n^{\lg 7}$ and $f(n) = \Theta(n^2)$. Since $f(n) = O\left(n^{\lg 7}\right)$, case 1 applies. Therefore, $T(n) = \Theta\left(n^{\lg 7}\right)$.

Substitution Method

1. Guess the solution.
2. Use induction to find the constants and show that the solution works.

- *E.g.*

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1. \end{cases}$$

1. *Guess:* $T(n) = n\lg n + n$. [Here, we have a recurrence with an exact function, rather than asymptotic notation, and the solution is also exact rather than asymptotic. We'll have to check boundary conditions and the base case.]

Substitution Method (cont'd)

2. *Induction*:

Bass: $n = 1 \rightarrow n\lg n + 1 = 1 = T(n)$

Inductive step: Inductive hypothesis is that $T(k) = k\lg k + k$ for all $k < n.$

We'll use this inductive hypothesis for $T(n/2).$

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n \\
&= 2\left(\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}\right) + n \\
&= n\lg\frac{n}{2} + n + n \\
&= n(\lg n - \lg 2) + n + n \\
&= n\lg n - n + n + n \\
&= n\lg n + n
\end{aligned}
$$

Substitution Method (cont'd)

- Generally, we use asymptotic notation:

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

- Assume $T(n) = O(1)$ for sufficiently small n
- Express the solution by asymptotic notation: $T(n) = \Theta(n\lg n)$
- Don't worry about boundary cases, nor do we show base cases in the substitution proof.

Substitution Method (cont’d)

- $T(n)$ is always constant for any constant n.
- Since we are ultimately interested in asymptotic solution to a recurrence, it will always be possible to choose base cases that work
- When we want an asymptotic solution to a recurrence, we don’ t worry about the base cases in our proofs.
- When we want an exact solution, then we have to deal with base cases.

Substitution Method (cont'd)

For the substitution method:

- Name the constant in the additive term
- Show the upper (O) and lower (Ω) bounds separately. Might need to use different constants for each notation

E.g.: $T(n) = T(n/2) + \Theta(n)$. If we want to show an upper bound of $T(n)$, we write $T(n) \leq T(n/2) + cn$ for some positive constant c.

1. Upper bound:

Guess: $T(n) \leq dn\lg n$ for some positive constant d. We are given c in the recurrence, and we get to choose d as any positive constant. It's OK for d to depend on c.

Substitution:

$$\begin{aligned} T(n) &\leq 2T\left(\frac{n}{2}\right) + cn \\ &\leq 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\ &= dn\lg\frac{n}{2} + cn \\ &= dn\lg n - dn + cn \\ &\leq dn\lg n \qquad \text{if } -dn + cn \leq 0, d \geq c \end{aligned}$$

Therefore, $T(n) = O(n\lg n)$

2. Lower bound:

Write $T(n) \geq T(n/2) + cn$ for some positive constant c.

Guess: $T(n) \geq dn\lg n$ for some positive constant d.

Substitution:

$$\begin{aligned} T(n) &\geq 2T\left(\frac{n}{2}\right) + cn \\ &\geq 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\ &= dn\lg\frac{n}{2} + cn \\ &= dn\lg n - dn + cn \\ &\geq dn\lg n \qquad\qquad \text{if } -dn + cn \geq 0, d \leq c \end{aligned}$$

Therefore, $T(n) = \Omega(n\lg n)$.

Therefore, $T(n) = \Theta(n\lg n)$ [For this particular recurrence, we can use $d = c$ for both the upper-bound and lower-bound proofs. That won' t always be the case.]

Substitution Method (cont'd)

- Make sure you show the same exact form when doing a substitution proof.
- Consider the recurrence

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

- For an upper bound: $T(n) \leq 8T(n/2) + cn^2$
- Guess: $T(n) \leq dn^3$

$$T(n) \leq 8d\left(\frac{n}{2}\right)^3 + cn^2 = 8d\left(\frac{n^3}{8}\right) + cn^2 = dn^3 + cn^2 \leq dn^3$$

Doesn' t work!

Substitution Method

Remedy: Subtract off a lower-order term.

Guess: $T(n) \le dn^3 - d'n^2$

$$
\begin{aligned}
T(n) &\le 8\left(d\left(\frac{n}{2}\right)^3 - d'\left(\frac{n}{2}\right)^2\right) + cn^2 \\
&= 8d\left(\frac{n^3}{8}\right) - 8d'\left(\frac{n^2}{4}\right) + cn^2 \\
&= dn^3 - 2d'n^2 + cn^2 \\
&\le dn^3 - d'n^2 \qquad \text{if } -2d'n^2 + cn^2 \le -d'n^2,
\end{aligned}
$$

$d' \ge c$

Substitution Method (cont'd)

- Be careful when using asymptotic notation.
- The false proof for the recurrence $T(n) = 4T(n/4) + n$, that $T(n) = O(n)$:

$$T(n) \le 4\left(c\left(\frac{n}{4}\right)\right) + n \le cn + n = O(n)$$ Wrong!

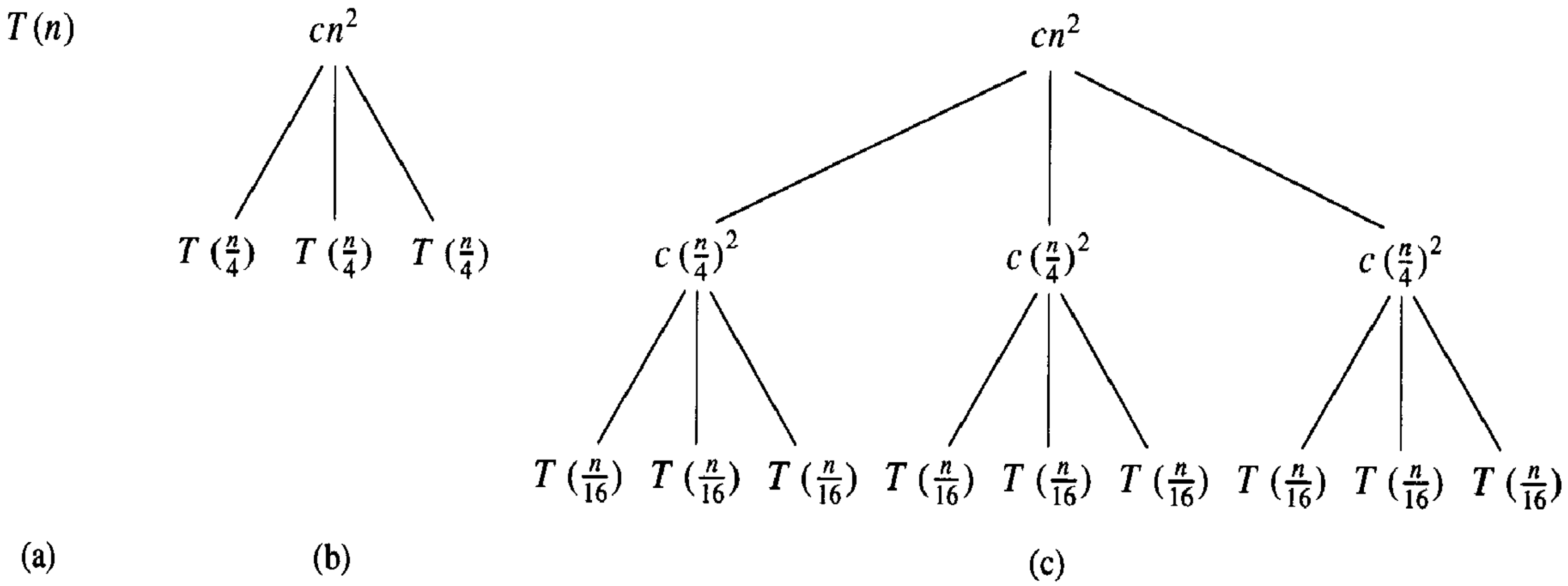
- Because we haven' t proven the exact from of our inductive hypothesis (which is that $T(n) \le cn$), this proof is false.

Recurrence Trees

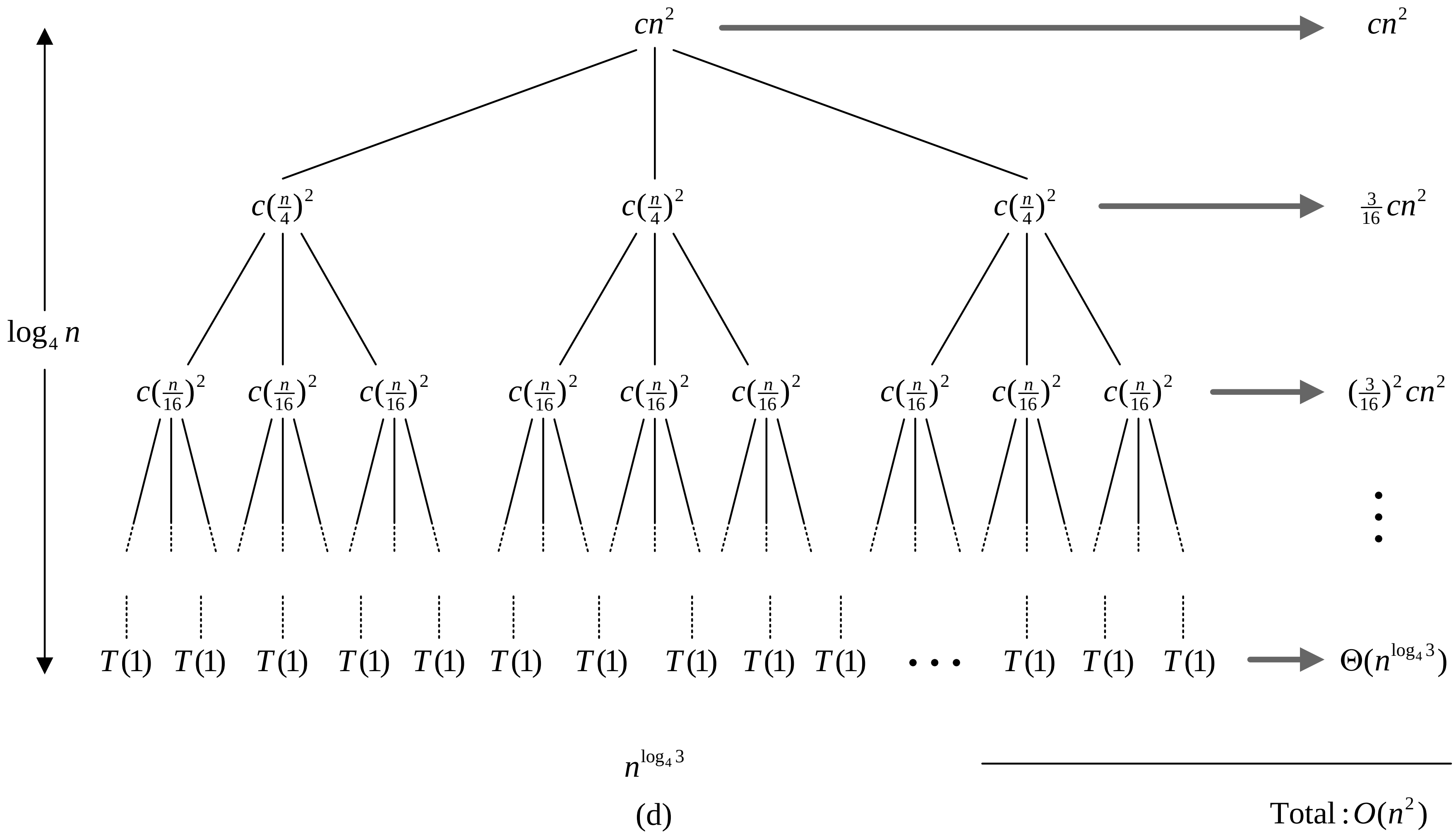
- Goal of the recursion-tree method
  - a good guess for the substitution method
  - a direct proof of a solution to a recurrence (provided by carefully drawing a recursion tree)

Recurrence Trees (cont'd)

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



Recurrence Trees (cont'd)



(d)

Recurrence Trees (cont'd)

- The cost of the entire tree

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \\
&= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}).
\end{aligned}
$$

Recurrence Trees (cont'd)

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta\left(n^{\log_4 3}\right) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

Recurrence Trees (cont'd)

- Verify by the substitution method
  - Show that $T(n) \le dn^2$ for some constant $d > 0$

$$T(n) \le 3T\left(\left\lfloor\frac{n}{4}\right\rfloor\right) + cn^2 \le 3d\left(\left\lfloor\frac{n}{4}\right\rfloor\right)^2 + cn^2 \le 3d\left(\frac{n}{4}\right)^2 + cn^2$$

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

where the last step holds as long as $d \ge \frac{16}{13}c$.

Recurrence Trees (cont'd)

Use to generate a guess. Then verify by substitution method.

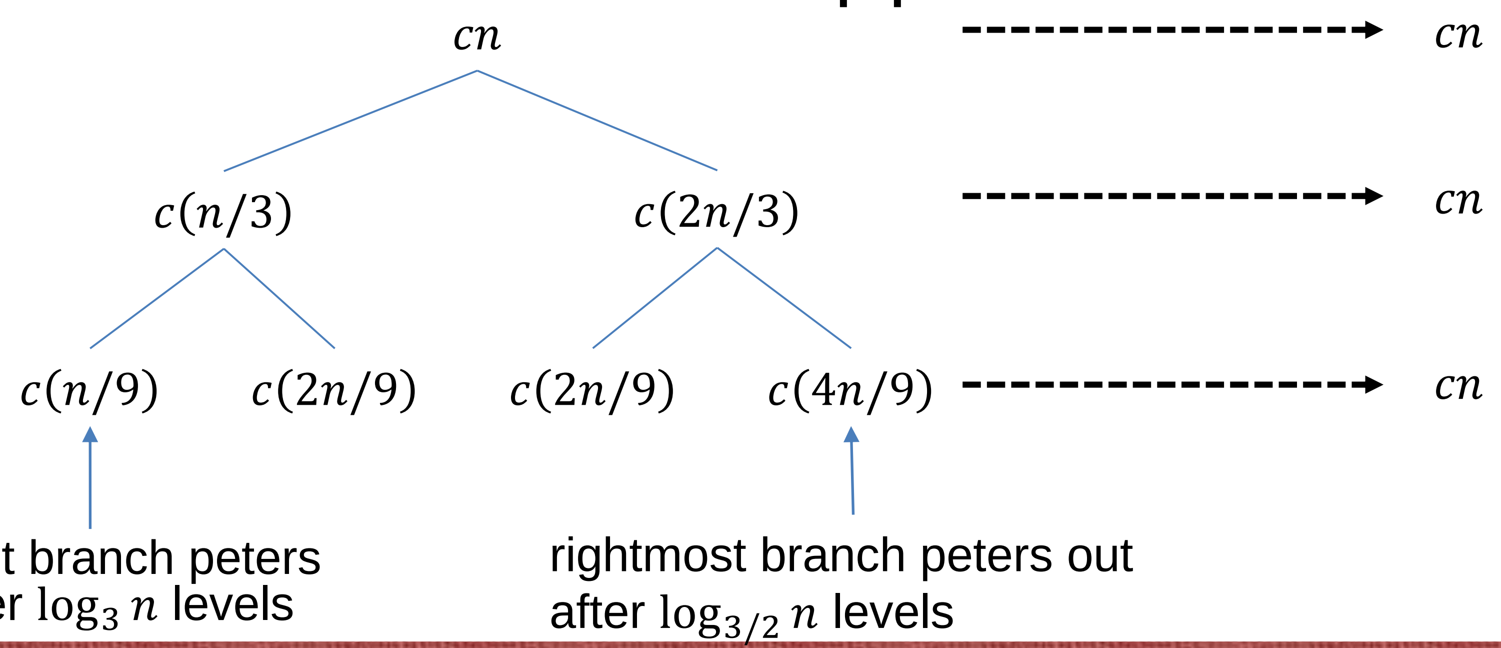
E.g.: $T(n) = T(n/3) + T(2n/3) + \Theta(n)$.

For upper bound, as $T(n) \leq T(n/3) + T(2n/3) + cn$

For lower bound, as $T(n) \geq T(n/3) + T(2n/3) + cn$

Recurrence Trees (cont'd)

- By summing across each level, the recursion tree shows the cost at each level of recursion (minus the costs of recursive calls, which appear in subtrees):



Recurrence Trees (cont'd)

- There are $\log_3 n$ full levels, and after $\log_{3/2} n$ levels, the problem size is down to 1.
- Each level contributes $\leq cn$.
- Lower bound guess: $\geq dn \log_3 n = \Omega(n \log n)$ for some positive constant d.
- Upper bound guess: $\leq dn \log_{3/2} n = O(n \log n)$ for some positive constant d.
- Then prove by substitution.

Recurrence Trees (cont'd)

- Upper bound:
  Guess: $T(n) \leq dn\lg n$.
  Substitution:
  $$\begin{aligned} T(n) &\leq T(n/3) + T(2n/3) + cn \\ &\leq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\ &= (d(n/3)\lg n - d(n/3)\lg 3) + \big(d(2n/3)\lg n - d(2n/3)\lg(3/2)\big) + cn \\ &= dn\lg n - d\big((n/3)\lg 3 + (2n/3)\lg(3/2)\big) + cn \\ &= dn\lg n - d\big((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2\big) + cn \\ &= dn\lg n - dn(\lg 3 - 2/3) + cn \\ &\leq dn\lg n \qquad \text{if } -dn(\lg 3 - 2/3) + cn \leq 0,\ d \geq \frac{c}{\lg 3 - 2/3} \end{aligned}$$
  Therefore, $T(n) = O(n\log n)$.

Note: Make sure that symbolic constants used in the recurrence (e.g.,c) and the guess (e.g.,d) are different.

Recurrence Trees (cont'd)

- Lower bound:
  Guess: $T(n) \geq dn\lg n$.
  Substitution: Same as for the upper bound, but replacing $\leq$ by $\geq$. End up needing
  $$0 \leq d \leq \frac{c}{\lg 3 - \frac{2}{3}}$$
  Therefore, $T(n) = \Omega(n \log n)$.
  Since $T(n) = O(n \log n)$ and $T(n) = \Omega(n \log n)$, we conclude that $T(n) = \Theta(n \log n)$

Master Theorem

- Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be an asymptotically positive function, and let $T(n)$ be defined on the nonnegative integers by the recurrence
$$T(n) = aT(n/b) + f(n)$$
where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Master Theorem (cond't)

- Then $T(n)$ has the following asymptotic bounds:
  - If $f(n) = O\left(n^{\log_b a-\varepsilon}\right)$ for some constant $\varepsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
  - If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\lg n\right)$.
  - If $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$ for some constant $\varepsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

Master Theorem (cond’t)

- **What’ s with the Case 3 regularity condition?**
  - Generally not a problem.
  - It always holds whenever $f(n) = n^k$ and $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$ for constant $\varepsilon > 0$. So you don’ t need to check it when $f(n)$ is a polynomial.

- $T(n) = 9T(n/3) + n$
  Using the master theorem, $n^{\log_3 9} = n^2$ and $f(n) = n$. Since $f(n) = O\left(n^{\log_3 9-\varepsilon}\right)$, case 1 applies. Therefore, $T(n) = \Theta(n^2)$.

- $T(n) = T(2n/3) + 1$
  Using the master theorem, $n^{\log_{3/2} 1} = n^0 = 1$ and $f(n) = 1$. Since $f(n) = \Theta\left(n^{\log_b a}\right)$, case 2 applies. Therefore, $T(n) = \Theta(\lg n)$.

- $T(n) = 3T(n/4) + n\lg n$
  Using the master theorem, $n^{\log_4 3} = O(n^{0.793})$ and $f(n) = n\lg n$. Since $f(n) = \Omega\left(n^{\log_4 3+\varepsilon}\right)$ where $\varepsilon \approx 0.2$ and $af(n/b) = 3(n/$

- $T(n) = 2T(n/2) + n\lg n$
  $a = 2, b = 2, f(n) = n\lg n$ and $n^{\log_b a} = n$
  Case 3 should apply, since $f(n) = n\lg n$ is asymptotically larger than $n^{\log_b a} = n$. Wrong!
  The ratio $\frac{f(n)}{n^{\log_b a}} = \lg n$ is asymptotically less than n^{ε} for any positive constant ε (not polymomially larger).
- The recurrence falls into the gap between case 2 and case 3. (Using Extended Master Theorem)

Extended Master Theorem

- Then $T(n)$ has the following asymptotic bounds:
  - If $f(n) = O\left(n^{\log_b a}(\log_b n)^k\right)$ with $k < -1$, then $T(n) = \Theta\left(n^{\log_b a}\right)$. (includes case 1 of the Master Theorem)
  - If $f(n) = \Theta\left(n^{\log_b a}(\log_b n)^{-1}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\log_b \log_b n\right)$.
  - If $f(n) = \Theta\left(n^{\log_b a}(\log_b n)^k\right)$ with $k > -1$, then $T(n) = \Theta\left(n^{\log_b a}(\log_b n)^{k+1}\right)$. (with $k = 0$ is case 2 in the Master Theorem)
  - If $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$ for some constant $\varepsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.

Extended Master Theorem (cond't)

- Examples:
  - $T(n) = 5T(n/2) + \Theta(n^2)$
    $n^{\log_2 5}$ vs. n^2
    Since $\log_2 5 - \varepsilon = 2$ for some constant $\varepsilon > 0$, use Case 1⇒
    $T(n) = \Theta\left(n^{\lg 5}\right)$
  - $T(n) = 27T(n/3) + \Theta(n^3 \lg n)$
    $n^{\log_3 27} = n^3$ vs. $n^3 \lg n$
    Use Case 3 with $k = 1 \Rightarrow T(n) = \Theta(n^3 \lg^2 n)$

Extended Master Theorem (cond't)

– $T(n) = 5T(n/2) + \Theta(n^3)$
  $n^{\log_2 5}$ vs. n^3
  Now $\log_2 5 + \varepsilon = 3$ for some constant$\varepsilon > 0$
  Check regularity condition (don' t really need to since $f(n)$ is a polynomial):
  $af(n/b) = 5(n/2)^3 = 5n^3/8 \leq cn^3$ for $c = \frac{5}{8} < 1$
  Use Case 4: $T(n) = \Theta(n^3)$

Extended Master Theorem (cond't)

– $T(n) = 27T(n/3) + \Theta(n^3/\text{lg}n)$

$n^{\log_3 27}$ vs. $n^3/\text{lg}n = n^3\text{lg}^{-1}n$

Since $f(n) = \Theta\left(n^{\log_b a}(\log_b n)^{-1}\right)$, use Case 2. Therefore, $T(n) = \Theta\left(n^{\log_b a}\log_b \log_b n\right) = \Theta(n^3 \log_3 \log_3 n)$.