Recurrence Relations.

Recurrence Relations

- Equation or an inequality that characterizes a function by its values on smaller inputs.
- **Solution Methods**
  - Substitution Method.
  - Recursion-tree Method.
  - Master Method.
- Recurrence relations **arise when we analyze the running time of iterative or recursive algorithms**.
  - **Ex:** Divide and Conquer.

$$T(n) = \Theta(1) \qquad \text{if } n \leq c$$

$$T(n) = a\,T(n/b) + D(n) + C(n) \qquad \text{otherwise}$$

Substitution Method

- **Guess** the form of the solution, then **use mathematical induction** to show it is correct.
  - Substitute guessed answer for the function when the inductive hypothesis is applied to smaller values – hence, the name.
- Works well when the solution is easy to guess.
- No general way to guess the correct solution.

Example – Exact Function

Recurrence: $T(n) = 1 \quad \text{if } n = 1$

$$T(n) = 2T(n/2) + n \quad \text{if } n > 1$$

- Guess: $T(n) = n \lg n + n$.
- Induction:
  - **Basis:** $n = 1 \Rightarrow n \lg n + n = 1 = T(n)$.
  - **Hypothesis:** $T(k) = k \lg k + k$ for all $k < n$.
  - **Inductive Step:** $T(n) = 2\,T(n/2) + n$

$$= 2\,((n/2)\lg(n/2) + (n/2)) + n$$

$$= n\,(\lg(n/2)) + 2n$$

$$= n \lg n - n + 2n$$

$$= n \lg n + n$$

Example – With Asymptotics

To Solve: $T(n) = 3T(\lfloor n/3 \rfloor) + n$

- Guess: $T(n) = O(n \lg n)$
- Need to prove: $T(n) \le cn \lg n$, for some $c > 0$.
- Hypothesis: $T(k) \le ck \lg k$, for all $k < n$.
- Calculate:

$$
\begin{aligned}
T(n) &\le 3c \lfloor n/3 \rfloor \lg \lfloor n/3 \rfloor + n \\
&\le c\, n \lg (n/3) + n \\
&= c\, n \lg n - c\, n \lg 3 + n \\
&= c\, n \lg n - n\, (c \lg 3 - 1) \\
&\le c\, n \lg n
\end{aligned}
$$

(The last step is true for $c \ge 1 / \lg 3$.)

Example – With Asymptotics

To Solve: $T(n) = 3T(\lfloor n/3 \rfloor) + n$

- To show $T(n) = \Theta(n \lg n)$, must show both upper and lower bounds, i.e., $T(n) = O(n \lg n)$ **AND** $T(n) = \Omega(n \lg n)$
- Show: $T(n) = \Omega(n \lg n)$
- Calculate:

$$\begin{aligned} T(n) &\geq 3c \lfloor n/3 \rfloor \lg \lfloor n/3 \rfloor + n \\ &\geq c\, n \lg (n/3) + n \\ &= c\, n \lg n - c\, n \lg 3 + n \\ &= c\, n \lg n - n\, (c \lg 3 - 1) \\ &\geq c\, n \lg n \end{aligned}$$

(The last step is true for $c \leq 1 / \lg 3$.)

Example – With Asymptotics

If $T(n) = 3T(\lfloor n/3 \rfloor) + O(n)$, as opposed to $T(n) = 3T(\lfloor n/3 \rfloor) + n$, then rewrite $T(n) \le 3T(\lfloor n/3 \rfloor) + cn$, $c > 0$.

- To show $T(n) = O(n \lg n)$, use second constant $\boldsymbol{d}$, different from $\boldsymbol{c}$.
- Calculate:

$$\begin{aligned} T(n) &\le 3d \lfloor n/3 \rfloor \lg \lfloor n/3 \rfloor + c\,n \\ &\le d\,n \lg (n/3) + cn \\ &= d\,n \lg n - d\,n \lg 3 + cn \\ &= d\,n \lg n - n\,(d \lg 3 - \mathrm{c}) \\ &\le d\,n \lg n \end{aligned}$$

(The last step is true for $d \ge c / \lg 3$.)

It is <u>OK</u> for d to depend on c.

Practice Examples

1. $T(n) = T(n-1) + T(n-2) + c$, for any $c > 0$.

- Guess: T(n) = $O(2^n)$

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

- Guess: T(n) = $O(n^2 \lg n)$

3. $T(n) = T(n-2) + n^2$

- Guess: T(n) = $O(n^3)$

Making a Good Guess

- If a recurrence is similar to one seen before, then guess a similar solution.
  - $T(n) = 3T(\lfloor n/3 \rfloor + 5) + n$ (Similar to $T(n) = 3T(\lfloor n/3 \rfloor) + n$)
    - When n is large, the difference between $n/3$ and $(n/3 + 5)$ is insignificant.
    - Hence, can guess $O(n \lg n)$.
- Method 2: Prove loose upper and lower bounds on the recurrence and then reduce the range of uncertainty.
  - E.g., start with $T(n) = \Omega(n)$ & $T(n) = O(n^2)$.
  - Then lower the upper bound and raise the lower bound.

Recursion-tree Method

- Making a good guess is sometimes difficult with the substitution method.
- Use **recursion trees** to devise good guesses.
- Recursion Trees
  - Show successive expansions of recurrences using trees.
  - Keep track of the time spent on the subproblems of a divide and conquer algorithm.
  - Help organize the algebraic bookkeeping necessary to solve a recurrence.

Recursion Tree – Example

- Running time of Merge Sort:

$$T(n) = \Theta(1) \quad \text{if } n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) \quad \text{if } n > 1$$

- Rewrite the recurrence as

$$T(n) = c \quad \text{if } n = 1$$

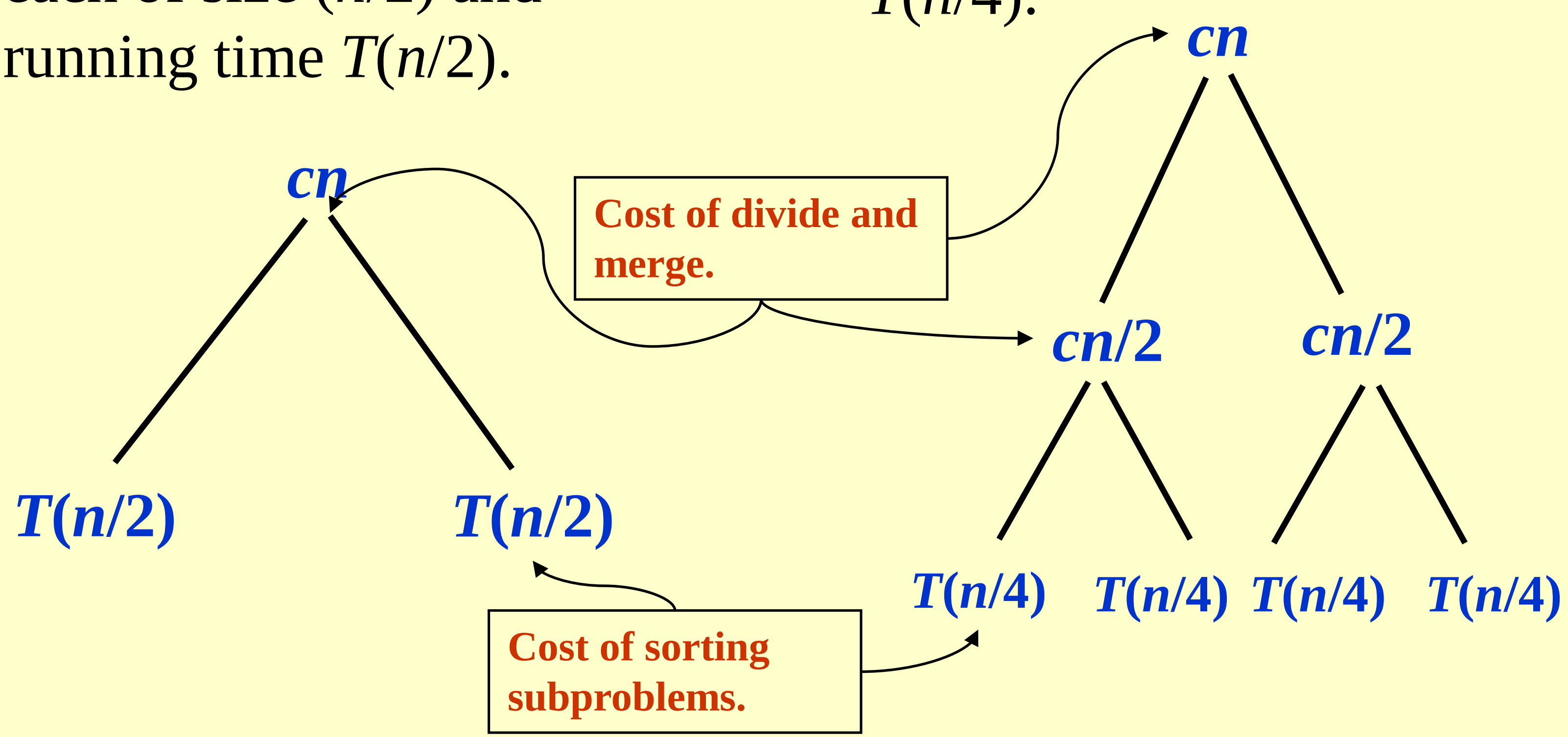
$$T(n) = 2T(n/2) + cn \quad \text{if } n > 1$$

$c > 0$: Running time for the base case and time per array element for the divide and combine steps.

Recursion Tree for Merge Sort

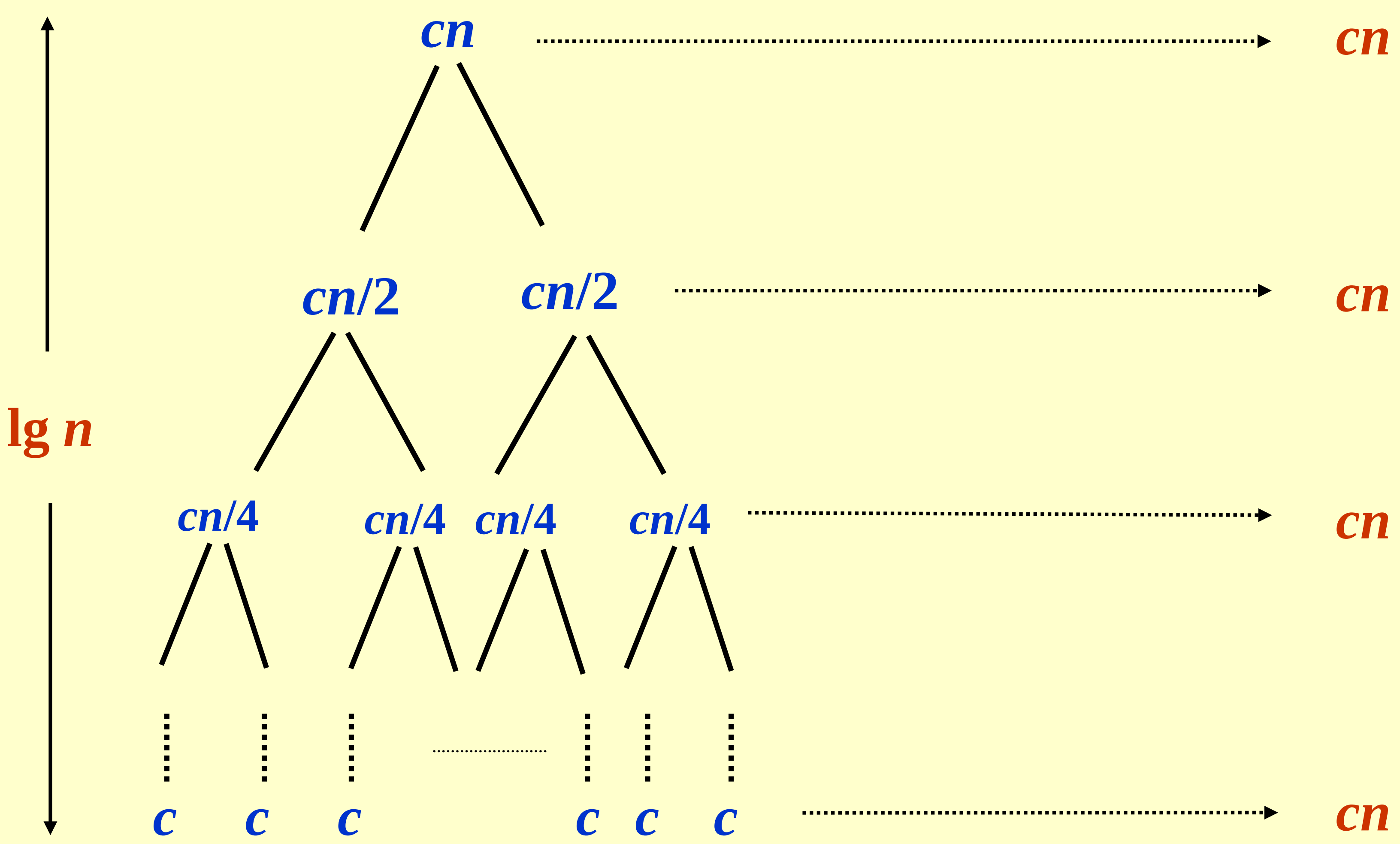
For the original problem, we have a cost of *cn*, plus two subproblems each of size ($n/2$) and running time $T(n/2)$.

Each of the size $n/2$ problems has a cost of $cn/2$ plus two subproblems, each costing $T(n/4)$.



Recursion Tree for Merge Sort

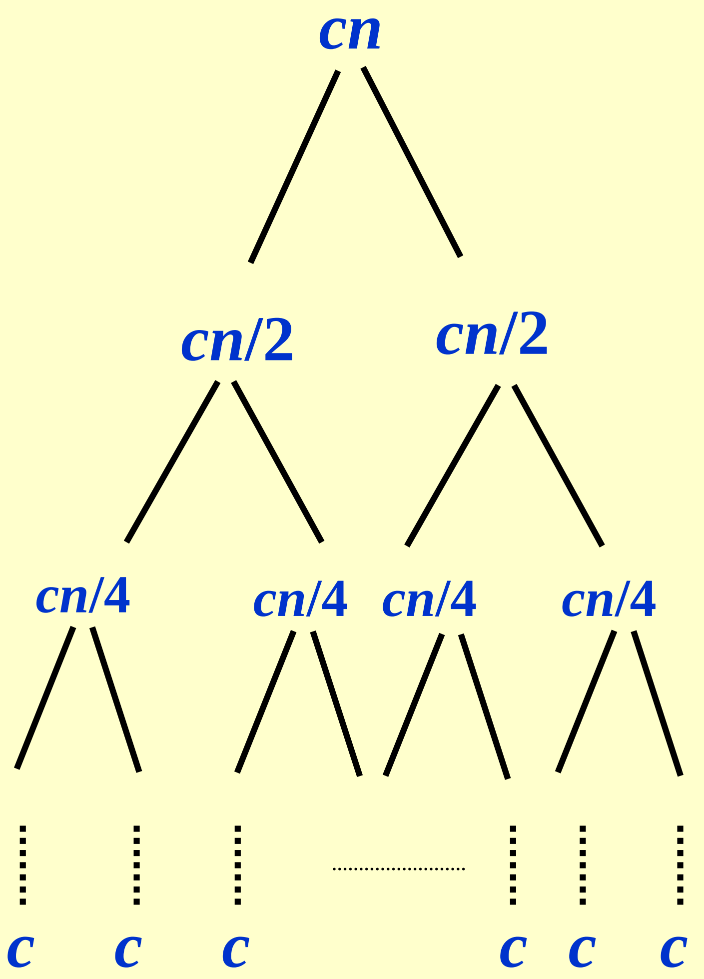
Continue expanding until the problem size reduces to 1.



Total : $cn\lg n + cn$

Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



- Each level has total cost ***cn***.
- Each time we go down one level, the number of subproblems doubles, but the cost per subproblem halves $\Rightarrow$ *cost per level remains the same*.
- There are lg n + 1 levels, height is lg n. (Assuming n is a power of 2.)
  - Can be proved by induction.
- Total cost = sum of costs at each level = $(\lg n + 1)cn = cn\lg n + cn = \Theta(n \lg n)$.

Practice Examples

- Use the recursion-tree method to determine a guess for the recurrences
  - $T(n) = 2T(n/2) + 1$
  - $T(n) = 2T(n/2) + n^2$
  - $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.
  - $T(n) = T(n/3) + T(2n/3) + O(n)$.
  - $T(n) = T(n/10) + T(9n/10) + \Theta(n)$.
  - $T(n) = T(n - 1) + T(n - 2) + \Theta(1)$.

The Master Method

- Based on the Master theorem.
- "Cookbook" approach for solving recurrences of the form

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

  - $a \geq 1$, $b > 1$ are constants.
  - $f(n)$ is asymptotically positive.
  - n/b may not be an integer, but we ignore floors and ceilings.
- Requires memorization of three cases.

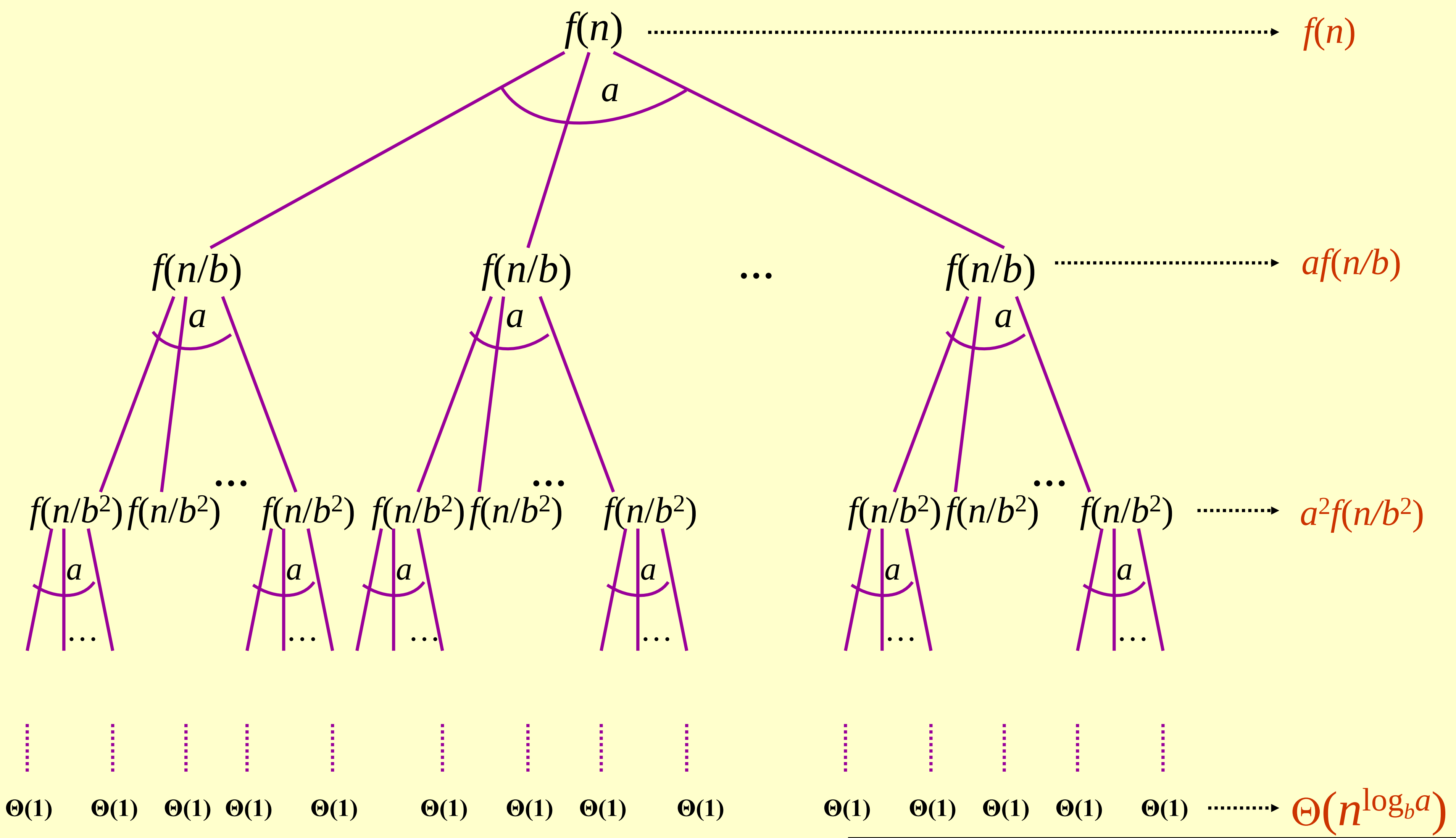
The Master Theorem

Theorem:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and
Let $T(n)$ be defined on nonnegative integers by the recurrence
$T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.
$T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$,
   and if, for some constant $c < 1$ and all sufficiently large n,
   we have $a \cdot f(n/b) \leq c\, f(n)$, then $T(n) = \Theta(f(n))$.

Recursion tree view



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

The Master Theorem

Theorem:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and

Let $T(n)$ be defined on nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. $T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if, for some constant $c < 1$ and all sufficiently large n, we have $a \cdot f(n/b) \leq c\, f(n)$, then $T(n) = \Theta(f(n))$.

Master Method – Examples

- $T(n) = 16T(n/4)+n$
  - $a = 16$, $b = 4$, $n^{\log_b a} = n^{\log_4 16} = n^2$.
  - $f(n) = n = O(n^{\log_b a - \varepsilon}) = O(n^{2-\varepsilon})$, where $\varepsilon = 1$ $\Rightarrow$ **Case 1.**
  - Hence, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

- $T(n) = T(3n/7) + 1$
  - $a = 1$, $b=7/3$, and $n^{\log_b a} = n^{\log_{7/3} 1} = n^0 = 1$
  - $f(n) = 1 = \Theta(n^{\log_b a})$ $\Rightarrow$ **Case 2**.
  - Therefore, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

Master Method – Examples

- $T(n) = 3T(n/4) + n \lg n$
  - $a = 3$, $b=4$, thus $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
  - $f(n) = n \lg n = \Omega(n^{\log_4 3 + \varepsilon})$ where $\varepsilon \approx 0.2$ $\Rightarrow$ **Case 3**.
  - Therefore, $T(n) = \Theta(f(n)) = \Theta(n \lg n)$.

- $T(n) = 2T(n/2) + n \lg n$
  - $a = 2$, $b=2$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_2 2} = n$
  - ***f*(*n*)** is asymptotically larger than $n^{\log_b a}$, but not polynomially larger. The ratio $\lg n$ is asymptotically less than n^{ε} for any positive ε. Thus, the Master Theorem ***doesn't*** apply here.

Master Theorem – What it means?

- **Case 1:** If $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  - $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
  - $f(n) = O(n^{\log_b a-\varepsilon}) \Rightarrow$ Sum of the cost of the nodes at each internal level asymptotically smaller than the cost of leaves by a *polynomial* factor.
  - Cost of the problem dominated by leaves, hence cost is $\Theta(n^{\log_b a})$.

Master Theorem – What it means?

- **Case 2:** If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
  - $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
  - $f(n) = \Theta(n^{\log_b a}) \Rightarrow$ Sum of the cost of the nodes at each level asymptotically the same as the cost of leaves.
  - There are $\Theta(\lg n)$ levels.
  - Hence, total cost is $\Theta(n^{\log_b a} \lg n)$.

Master Theorem – What it means?

- **Case 3:** If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$, and if, for some constant $c < 1$ and all sufficiently large n, we have $a \cdot f(n/b) \leq c\, f(n)$, then $\boldsymbol{T(n) = \Theta(f(n))}$.

  - $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
  - $f(n) = \Omega(n^{\log_b a+\varepsilon}) \Rightarrow$ Cost is dominated by the root. Cost of the root is asymptotically larger than the sum of the cost of the leaves by a polynomial factor.
  - Hence, cost is $\Theta(f(n))$.

Practice Examples

- Use the Master's method to solve the following recurrences. If a recurrence can not be solved by the master's method, state the appropriate reason for it.
  - $T(n) = 4T(n/2) + \mathrm{n}$
  - $T(n) = 4T(n/2) + n^2$
  - $T(n) = 16T(n/2) + n^2$
  - $T(n) = 7T(n/2) + n^2$
  - $T(n) = 2T(\lfloor n/4 \rfloor) + \Theta(1)$
  - $T(n) = 2T(\lfloor n/4 \rfloor) + \sqrt{n}$
  - $T(n) = 2T(\lfloor n/4 \rfloor) + \Theta(n)$
  - $T(n) = 2T(\lfloor n/4 \rfloor) + n^2$

Practice Examples

- Use the Master's method to solve the following recurrences. If a recurrence can not be solved by the master's method, state the appropriate reason for it.
  - $T(n) = T(3n/7) + \Theta(1)$
  - $T(n) = 3T(\lfloor n/4 \rfloor) + n \log n$
  - $T(n) = T(n/2) + \Theta(1)$
  - $T(n) = 3T(\lfloor n/2 \rfloor) + \Theta(n)$
  - $T(n) = 4T(\lfloor n/2 \rfloor) + n^2 \log n$

Iteration Method

- A "brute force" method of solving a recurrence relation.
- It is quite similar to the recursion tree method (summation is same)
- Usually considered for recurrences involving subtraction to reduce the problem size
- The general idea is to iteratively substitute the value of the recurrent part of the equation until a pattern (usually a summation) is noticed which can be used to evaluate the recurrence.

Example: Iteration Method

Given recurrence, $T(n) = T(n-1) + 1$ and $T(1) = \theta(1)$.

$$T(n) = T(n-1) + 1$$
$$= (T(n-2) + 1) + 1 = (T(n-3) + 1) + 1 + 1$$
$$= T(n-4) + 4 = T(n-5) + 1 + 4$$
$$= T(n-5) + 5 = T(n-k) + k$$

When $k = n-1$

$$T(n-k) = T(1) = \theta(1)$$
$$T(n) = \theta(1) + (n-1)$$
$$= 1 + n - 1$$
$$= n = \theta(n).$$

Iteration Method

- **T (n) = 1 if n=1**
- **= 2T (n-1) if n>1**
- T (n) = 2T (n-1)
- = 2[2T (n-2)] = 2^2 T (n-2)
- = 4[2T (n-3)] = 2^3T (n-3)
- = 8[2T (n-4)] = 2^4T (n-4)   (Eq.1)
- Repeat the procedure for i times
- T (n) = 2^i T (n-i)
- Put n - i = 1 or i = n - 1 in   (Eq.1)
- T (n) = 2^{n-1} T (1)
- = 2^{n-1} .1   {T (1) =1 .....given}

Iteration Method

- $T(n) = T(n-1) + n$
- $T(n) = n + (n-1) + T(n-2)$
- $\quad = n + (n-1) + (n-2) + T(n-3)$
- … … …
- … … …
- $T(n) = n + (n-1) + (n-2) + \ldots + [n-(n-2)] + [n-(n-1)]$
- $T(n) = n*n-(1+2+\ldots+n-1)$
- $\quad = n^2 - (n^2 - n)/2$
- $\quad = \frac{1}{2}(n^2 + n)$
- $\quad = O(n^2)$

Changing the variable

Sometimes reucurrences can be reduced to simpler ones by *changing variables*

- Example: Solve $T(n) = 2T(\sqrt{n}) + \log n$

  Let $m = \log n \Rightarrow 2^m = n \Rightarrow \sqrt{n} = 2^{m/2}$

  $T(n) = 2T(\sqrt{n}) + \log n \Rightarrow T(2^m) = 2T(2^{m/2}) + m$

  Let $S(m) = T(2^m)$

  $$
  \begin{aligned}
  T(2^m) = 2T(2^{m/2}) + m \; &\Rightarrow S(m) = 2S(m/2) + m \\
  &\Rightarrow S(m) = O(m \log m) \\
  &\Rightarrow T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n)
  \end{aligned}
  $$

Practice Examples: $T(n) = 2T(\sqrt{n}) + 1$

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

Some more recurrences

Some important/typical bounds on recurrences not covered by master method:

- Logarithmic: $\Theta(\log n)$
  - Recurrence: $T(n) = 1 + T(n/2)$
  - Typical example: Recurse on half the input (and throw half away)
  - Variations: $T(n) = 1 + T(99n/100)$
- Linear: $\Theta(N)$
  - Recurrence: $T(n) = 1 + T(n-1)$
  - Typical example: Single loop
  - Variations: $T(n) = 1 + 2T(n/2), T(n) = n + T(n/2), T(n) = T(n/5) + T(7n/10 + 6) + n$
- Quadratic: $\Theta(n^2)$
  - Recurrence: $T(n) = n + T(n-1)$
  - Typical example: Nested loops
- Exponential: $\Theta(2^n)$
  - Recurrence: $T(n) = 2T(n-1)$