

Lecture 4: Recurrences and Master Theorem

Dr. Khalil Yousef

Adopted from the Slides of the ECE 608 Computational Models and Methods Course at Purdue University

Read Chapter 4 of *Introduction to Algorithms*

Recurrences

A recurrence is an equation or inequality that describes a function in terms of its value(s) on smaller inputs. For example, the following recurrence describes the worst-case running time of MERGE-SORT:

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

The solution to this equation is $\Theta(n \lg n)$. An important question is how can we find such a closed-form expression (exact or asymptotic) for recurrences?

We will discuss 3 methods:

- **Substitution method:** Guess the bound and prove by induction.
- **Iteration method:** Treat it as summation.
- **Master method:** A “cookbook” method for solving $T(n) = aT(\frac{n}{b}) + f(n)$, $a \geq 1, b > 1$, and $f(n)$ is a given function.

Recurrences *continued*

In practice, we often neglect certain details when we state and solve recurrences. Sometimes we gloss over the assumption that the functions take integer arguments. For example, the MERGE-SORT worst-case running time is really:

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

We also typically don't state the boundary conditions. For sufficiently small n , generally $T(n) = \Theta(1)$. Hence, we often report the recurrence simply as:

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

We omit floors, ceilings, and boundary conditions when they don't matter. We also often solve for powers of 2 rather than all n (and in that case, we really haven't shown the bound for all n).

The Substitution Method

Example: Solve the recurrence: $T(n) = 4T(\frac{n}{2}) + n$, $T(1) = 1$.

Guess to Verify by Induction: $T(n) = O(n^3)$, i.e., $T(n) \leq cn^3$.

Base Case: $T(1) = 1 \leq c1^3 = c$, if $c \geq 1$.

Assume: $T(k) \leq ck^3$, for $k < n$.

Inductive Step:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4c\left(\frac{n}{2}\right)^3 + n \\ &= \frac{c}{2}n^3 + n \\ &= cn^3 - \frac{c}{2}n^3 + n \\ &= cn^3 - \left(\frac{c}{2}n^3 - n\right) \\ &\leq cn^3, \text{ if } c \geq 2, n \geq 1 \end{aligned}$$

The Substitution Method *continued*

How to make a good guess:

- Similarity to known function, e.g., $T(n) = 2T(n/2) + n$
- Start with a loose upper and lower bound and try to narrow it down.

There are times when you can guess a bound but the math doesn't work out in the induction. Sometimes the inductive assumption isn't strong enough to prove a detailed bound; try subtracting a lower-order term. We will next show an example where this technique will help.

The Substitution Method *continued*

$O(n^3)$ is **not** a tight bound for $T(n) = 4T(\frac{n}{2}) + n$; we can show that $T(n) = O(n^2)$.

Check previous lecture (lecture 3).

The Substitution Method_{continued}

Sometimes a little algebraic manipulation can make an unknown recurrence into one we have seen before. For example,

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

Renaming $n = 2^m$ (so $m = \lg n$) yields:

$$T(2^m) = 2T(2^{\frac{m}{2}}) + m$$

Now we can rename $S(m) = T(2^m)$:

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

We know $S(m) = O(m \lg m)$. Changing back from $S(m)$ to $T(n)$ yields:

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$$

The Iteration Method

The idea is to expand the recurrence and express the recurrence as a summation dependent only on n and the initial conditions. Then techniques for evaluating summations can be used to provide a bound on the solution. For example, recall the recurrence equation: $T(n) = 4T(\frac{n}{2}) + n$. Let us expand $T(n)$ a few times looking for patterns:

$$\begin{aligned} T(n) &= n + 4T\left(\frac{n}{2}\right) \\ &= n + 4\left(\frac{n}{2} + 4T\left(\frac{n}{4}\right)\right) \\ &= n + 4\left(\frac{n}{2} + 4\left(\frac{n}{4} + 4T\left(\frac{n}{8}\right)\right)\right) = n + 4\frac{n}{2} + 4^2\frac{n}{4} + 4^3T\left(\frac{n}{8}\right) \\ &\vdots \\ &= n + 4\frac{n}{2} + 4^2\frac{n}{4} + 4^3\frac{n}{8} + \dots + 4^kT\left(\frac{n}{2^k}\right) \\ &: \triangleright \text{Iterate until } \frac{n}{2^k} = 1 \text{ or } \lg n \text{ times.} \\ &= n + 4\frac{n}{2} + 4^2\frac{n}{4} + 4^3\frac{n}{8} + \dots + 4^{\lg n}T(1), \text{ where } 4^{\lg n} = n^2 \\ &= \sum_{i=0}^{\lg n - 1} 4^i \frac{n}{2^i} + \Theta(n^2) \\ &= n \left(\sum_{i=0}^{\lg n - 1} 2^i \right) + \Theta(n^2) = n \left(\frac{2^{\lg n} - 1}{2 - 1} \right) + \Theta(n^2) \\ &= n(n - 1) + \Theta(n^2) = \Theta(n^2) + \Theta(n^2) = \Theta(n^2) \end{aligned}$$

The Iteration Method *continued*

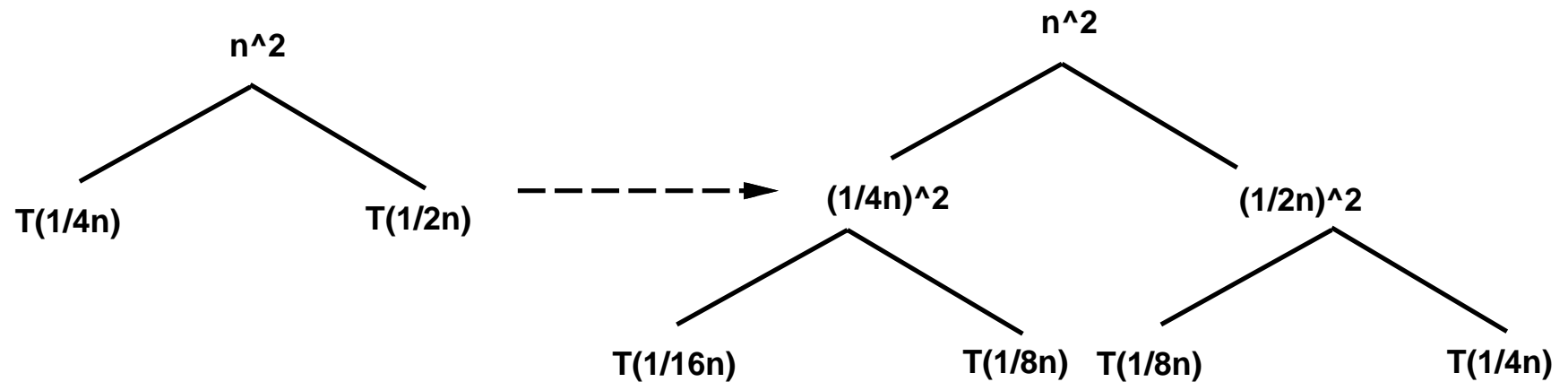
Let's obtain an exact solution for: $T(n) = 2T(\frac{n}{2}) + 2$. Assume that n is a power of 2 and $T(2)=1$;

$$\begin{aligned} T(n) &= 2 + 2T\left(\frac{n}{2}\right) \\ &= 2 + 2\left(2 + 2T\left(\frac{n}{4}\right)\right) = 2 + 4 + 4T\left(\frac{n}{4}\right) \\ &= 2 + 4 + 4\left(2 + 2T\left(\frac{n}{8}\right)\right) = 2 + 4 + 8 + 8T\left(\frac{n}{8}\right) \\ &\vdots \\ &= 2 + 4 + 8 + 16 + \dots + 2^k + 2^k T\left(\frac{n}{2^k}\right) \\ &\vdash T(2) \text{ is the base case, so solve } \frac{n}{2^k} = 2. \\ &= \sum_{i=1}^{\lg n - 1} 2^i + 2^{\lg n - 1} T\left(\frac{n}{2^{\lg n - 1}}\right) \\ &= \sum_{i=0}^{\lg n - 1} 2^i - 1 + \frac{n}{2} T\left(\frac{n}{2}\right) \\ &= \frac{2^{\lg n} - 1}{2 - 1} - 1 + \frac{n}{2} T(2) \\ &= n - 2 + \frac{n}{2} = \frac{3}{2}n - 2, \text{ since } T(2) = 1. \end{aligned}$$

Recursion Trees

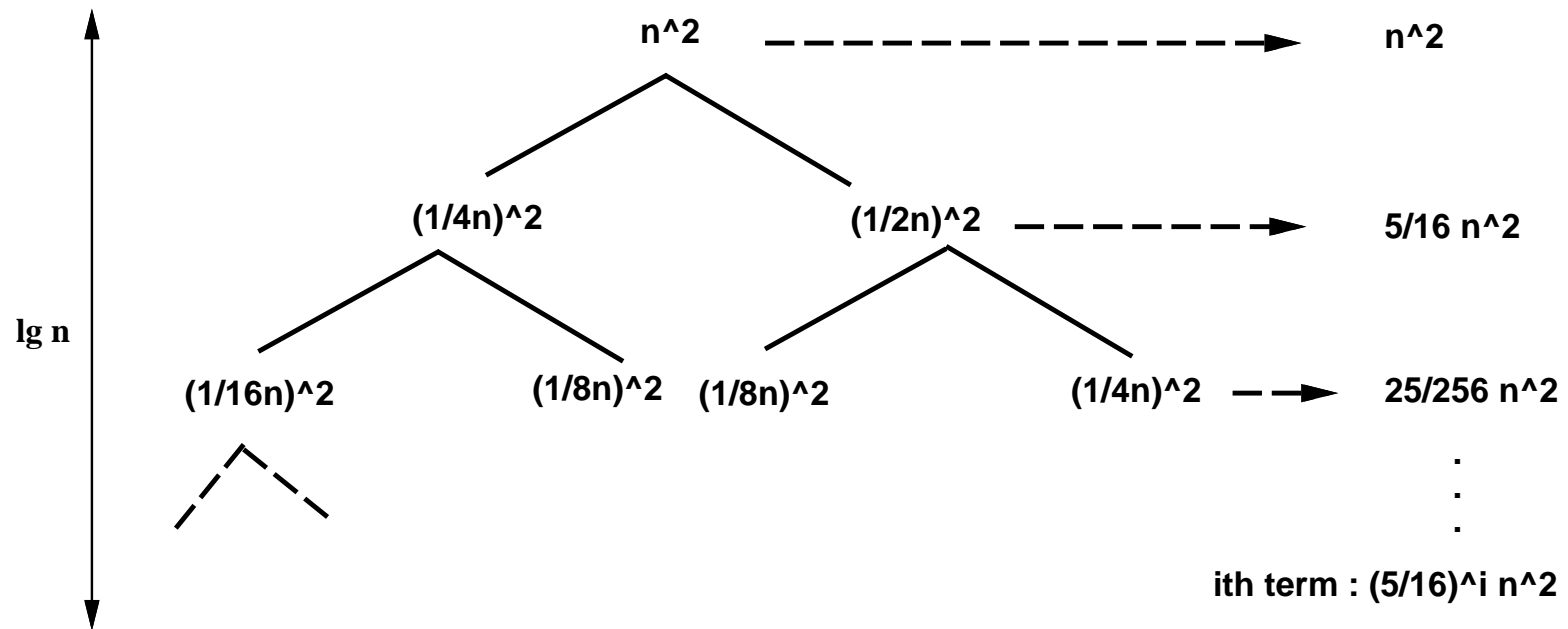
For certain forms of recursions a graphical representation is convenient for visualizing the iteration of a recurrence. For example:

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$$



Recursion trees *continued*

Recursion Tree for $T(n) = T(n/4) + T(n/2) + n^2$



Summing over levels gives a decreasing geometric series:

$$\sum_{i=0}^{\lg n - 1} \left(\frac{5}{16}\right)^i n^2 \leq n^2 \sum_{i=0}^{\infty} \left(\frac{5}{16}\right)^i = n^2 \frac{1}{1 - \frac{5}{16}} = O(n^2)$$

The Master Method

“Cookbook method” for solving recurrences of the type:

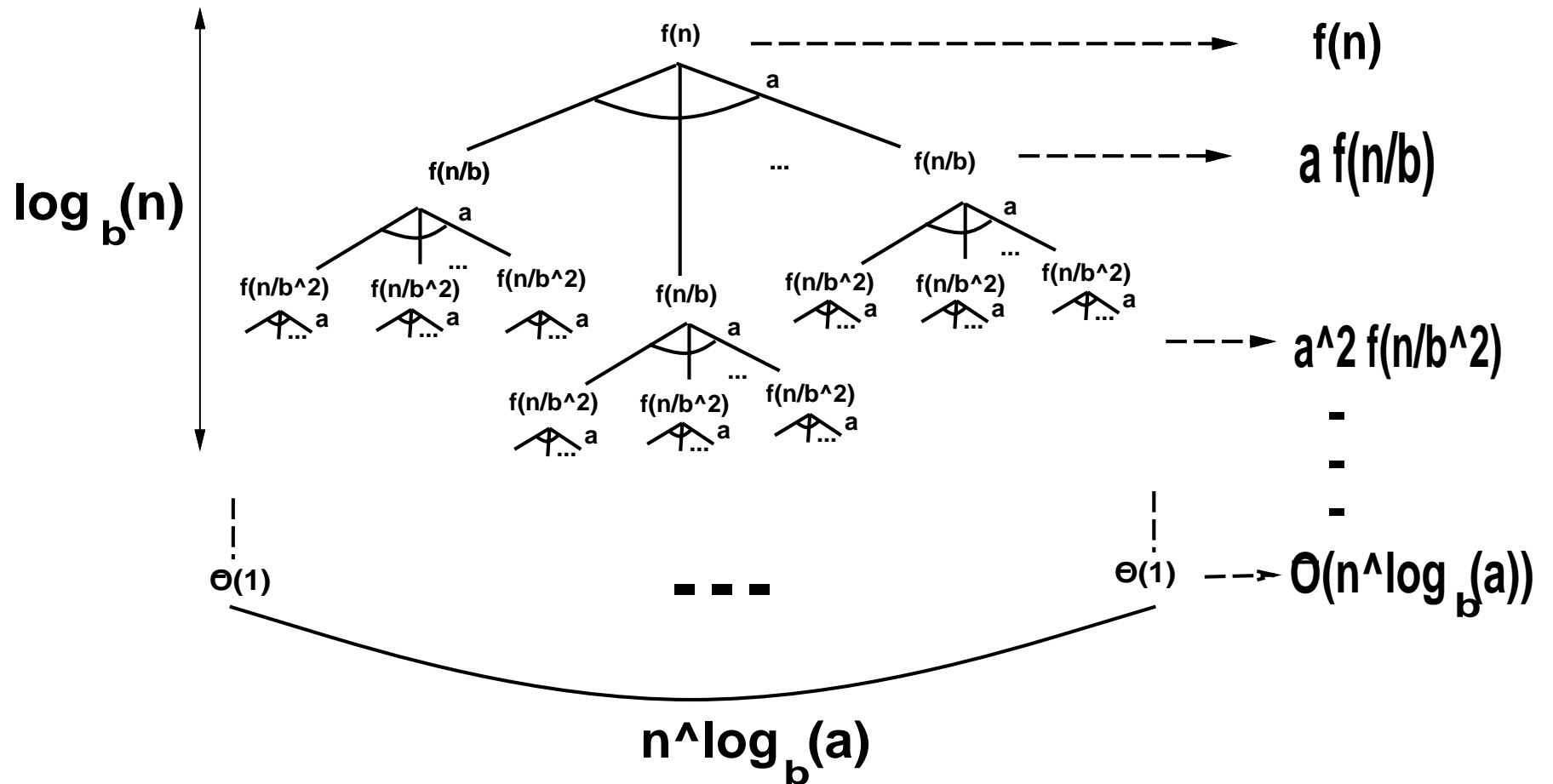
$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $a \geq 1, b > 1$, and f is asymptotically positive.

- Divide the problem of size n into a subproblems of size $\frac{n}{b}$.
- Solve them recursively.
- Cost to divide the original problem and recombine the results is $f(n)$, which is asymptotically positive.

The Master Method *continued*

Recursion Tree for $T(n) = a T(n/b) + f(n)$



The Master Theorem

Theorem 4.1 The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be a function, and $T(n)$ be defined on the nonnegative integers by the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), T(1) = \Theta(1)$$

where we interpret $\frac{n}{b}$ to mean either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$. Or alternatively, $\frac{n^{\log_b a}}{f(n)} = \Omega(n^\varepsilon), \varepsilon > 0$ (equivalently, $\frac{f(n)}{n^{\log_b a}} = O(n^{-\varepsilon}), \varepsilon > 0$). In other words, $n^{\log_b a}$ is polynomially larger than $f(n)$. The running time is dominated by the costs in the leaves.

The Master Theorem *continued*

2. If $f(n) = \Theta(n^{\log_b a} \lg^k n)$, for $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ (as in Exercise 4.4-2). Or alternatively, $\frac{f(n)}{n^{\log_b a}} = \Theta(\lg^k n)$ for some constant $k \geq 0$. In other words, $f(n)$ and $n^{\log_b a}$ are within a polylogarithmic factor. The cost is evenly distributed across levels.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$. Or alternatively, $\frac{f(n)}{n^{\log_b a}} = \Omega(n^\varepsilon)$, $\varepsilon > 0$ and $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n . In other words, $f(n)$ is polynomially larger than $n^{\log_b a}$. Hence, the time is dominated by the cost of the root.

The Master Theorem *continued*

The following equation represents the running time for the class of problems for which the Master Method applies:

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

Note that the number of small problems to solve (leaves in the tree) is $a^{\log_b n} = n^{\log_b a}$.

$$\begin{aligned} a^{\log_b n} &= (b^{\log_b a})^{\log_b n} \\ &= b^{\log_b a \log_b n} \\ &= b^{\log_b n \log_b a} \\ &= (b^{\log_b n})^{\log_b a} \\ &= n^{\log_b a} \end{aligned}$$

Applying the Master Method

Example 1: $T(n) = 9T(\frac{n}{3}) + n$

Test: $a = 9, b = 3, f(n) = n$ so $n^{\log_b a} = n^{\log_3 9} = n^2$.

Because $f(n) = n = O(n^{2-\epsilon})$ for $0 < \epsilon \leq 1$ (or $\frac{n^2}{n} = \Omega(n^\epsilon)$ for $\epsilon > 0$), case 1 applies and so $T(n) = \Theta(n^2)$. Confirm by induction: $T(n) \leq cn^2$.

Base Case: Assuming $T(1) = \Theta(1) \leq c1^2 = c$, if $c > 1$.

Assume: $T(k) = ck^2 - k$ for $k < n$

Inductive Step:

$$\begin{aligned} T(n) &= 9T(\frac{n}{3}) + n \leq 9(c(\frac{n}{3})^2 - \frac{n}{3}) + n \\ &= cn^2 - 3n + n = cn^2 - 2n \\ &\leq cn^2 - n \end{aligned}$$

Applying the Master Method *continued*

Example 2: $T(n) = 2T(\frac{n}{2}) + 2$

Test: $a = 2, b = 2, f(n) = 2$ so $n^{\log_b a} = n^{\log_2 2} = n$.

Because $f(n) = \Theta(1) = O(n^{1-\epsilon})$ for $0 < \epsilon \leq 1$ (or $\frac{n}{2} = \Omega(n^\epsilon), 0 < \epsilon \leq 1$), case 1 applies and so $T(n) = \Theta(n)$.

Example 3: $T(n) = 2T(\frac{n}{2}) + n$

Test: $a = 2, b = 2, f(n) = n$ so $n^{\log_b a} = n^{\log_2 2} = n$.

Because $f(n) = \Theta(n^{\log_b a} \lg^0 n)$ (or $\frac{n}{n} = \Theta(1) = \Theta(\lg^0 n)$), case 2 applies; hence, $T(n) = \Theta(n \lg n)$.

Applying the Master Method *continued*

Example 4: $T(n) = 4T(\frac{n}{2}) + n^3$

Test: $a = 4, b = 2, f(n) = n^3$ so $n^{\log_b a} = n^{\log_2 4} = n^2$.

Because $f(n) = n^3 = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{2+\varepsilon})$ (or $\frac{n^3}{n^{\log_2 4}} = n = \Omega(n^\varepsilon)$) for $0 < \varepsilon \leq 1$, and $4f(\frac{n}{2}) \leq c f(n)$ for $c < 1$ (say $c = \frac{1}{2}$); hence case 3 applies and $T(n) = \Theta(n^3)$.

Example 5: $T(n) = 4T(\frac{n}{2}) + \frac{n^2}{\lg n}$

Test: $a = 4, b = 2, f(n) = \frac{n^2}{\lg n}$ so $n^{\log_b a} = n^{\log_2 4} = n^2$.

Because $\frac{n^2}{\lg n} = \frac{1}{\lg n}$, which is not $O(n^{-\varepsilon}), \varepsilon > 0$ (i.e., $\lg n \neq \Omega(n^\varepsilon)$), $\Theta(\lg^k n), k \geq 0$, or $\Omega(n^\varepsilon), \varepsilon > 0$, no case applies (must use another method).