Divide and Conquer Algorithms

CPSC 320 2023W2

Definitions

A divide and conquer algorithm proceeds by…

- Dividing the input into two or more **smaller instances** of the same problems
  - We call these **subproblems**
- Solving the subproblems recursively
- Combining the subproblem solutions to obtain a solution to the original problem

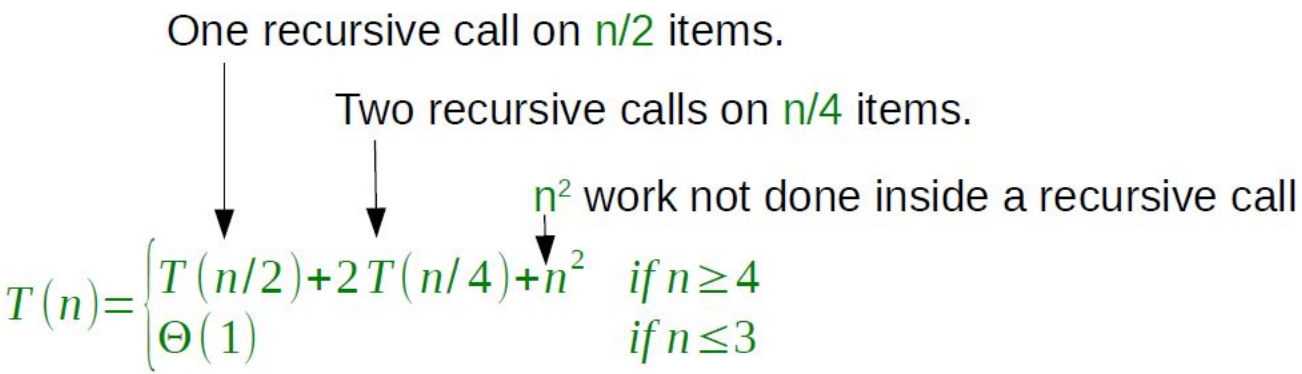
Examples

Some divide and conquer algorithms you are already familiar with:

- QuickSort
- MergeSort

Recurrence relations

- The running time **T(n)** of a recursive function can be described using a **recurrence relation:**
  - **T(n)** is defined in terms of one or more terms of the form **T(something smaller than n)**
  - Example:



Recursion Trees

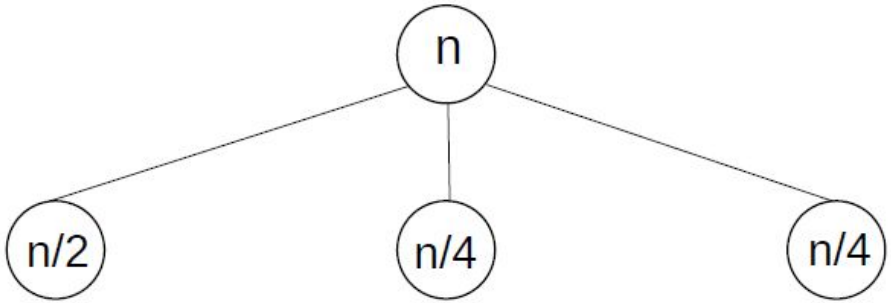
- One way to solve a recurrence relation is to draw a **recursion tree**
  - Represent the recursion with a tree where each node represents a recursive subproblem
  - Inside each node, write the size of the subproblem this call to the function solves
  - Next to each node, write the amount of work done by the call to the function, **not including** any time spent in recursive calls
  - Compute the total amount of non-recursive work done on each row
  - Then add up the work done over all rows

$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: drawing the tree

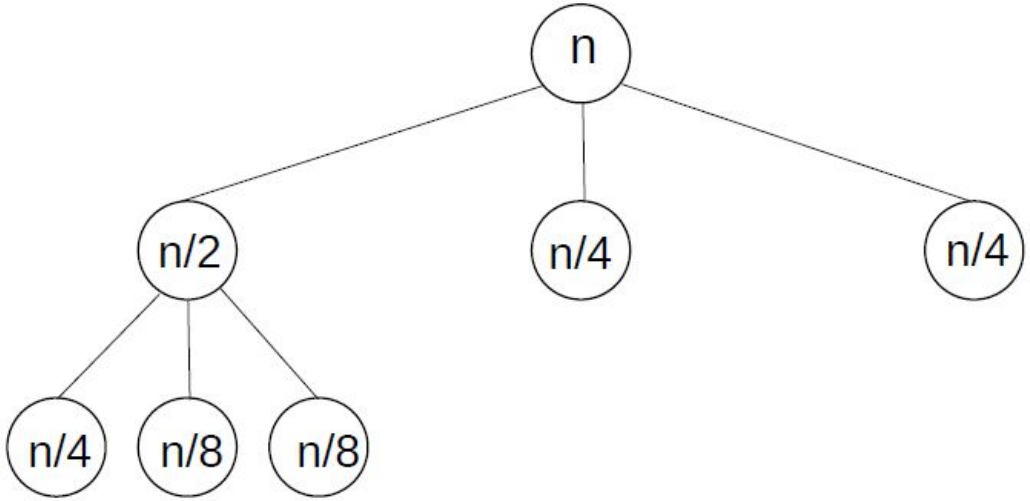
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: drawing the tree (continued)



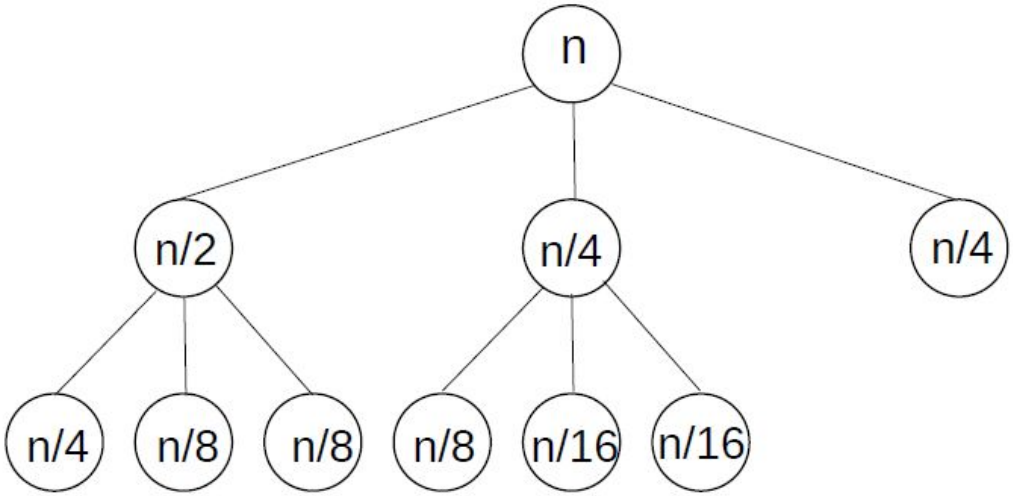
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: drawing the tree (continued)



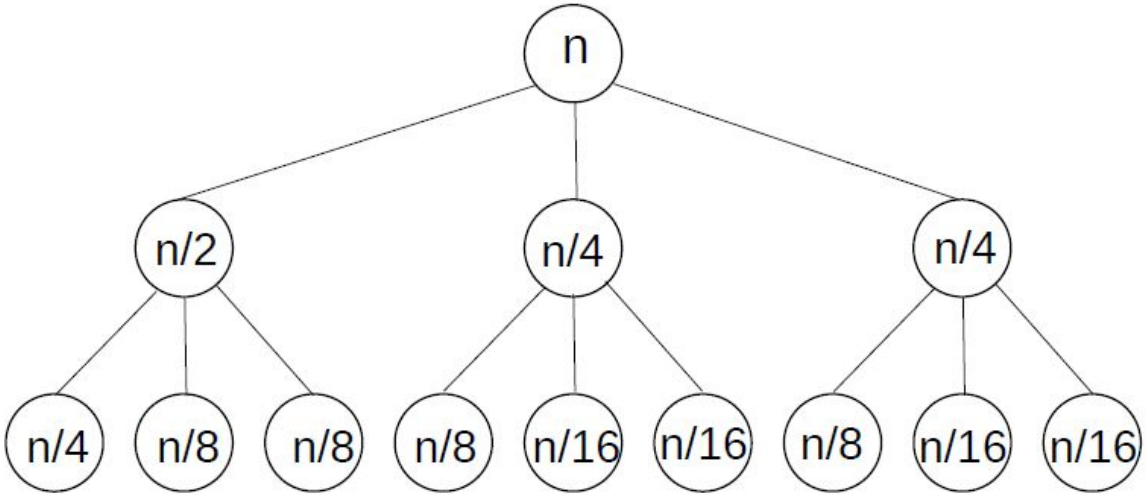
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: drawing the tree (continued)



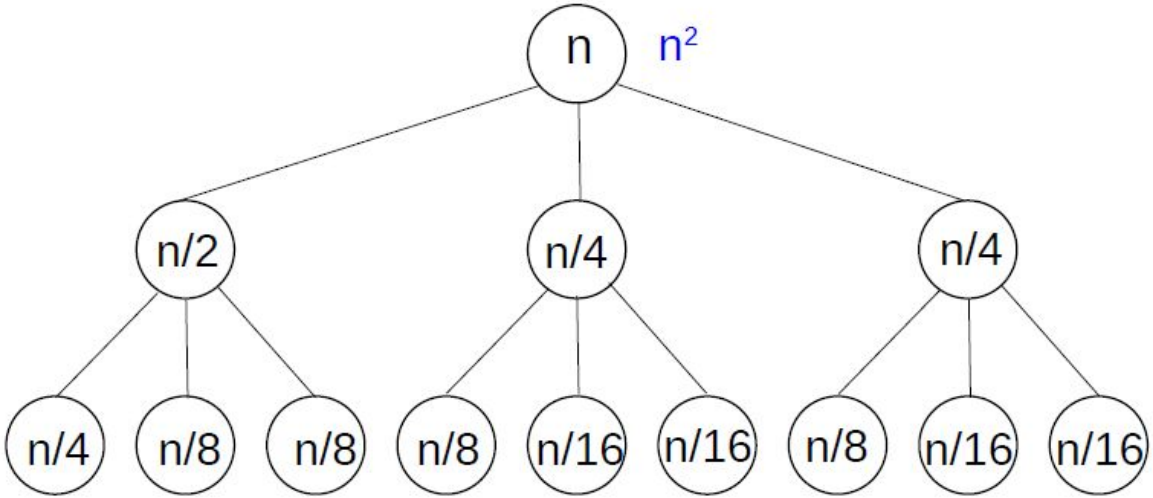
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: drawing the tree (continued)



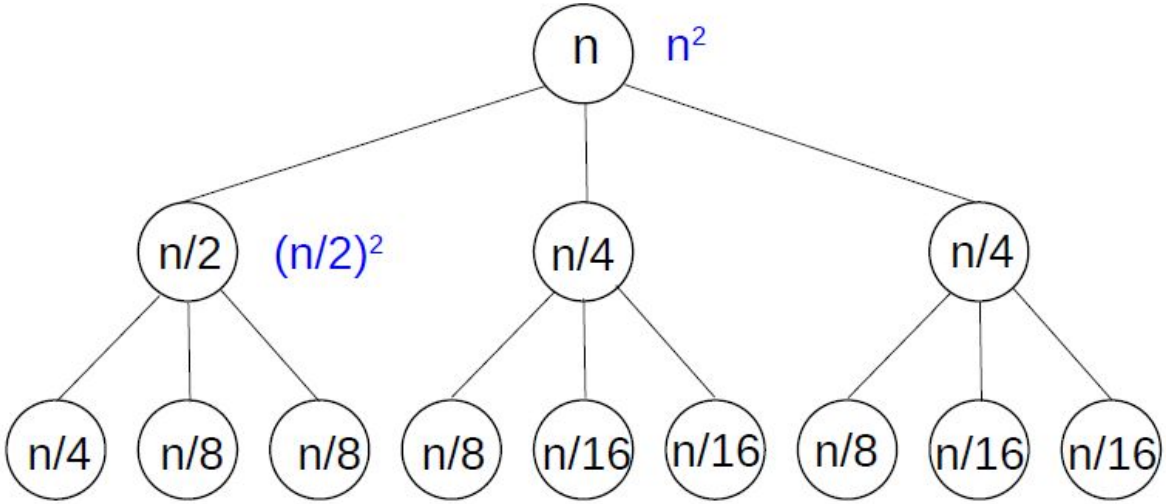
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: work done at each node



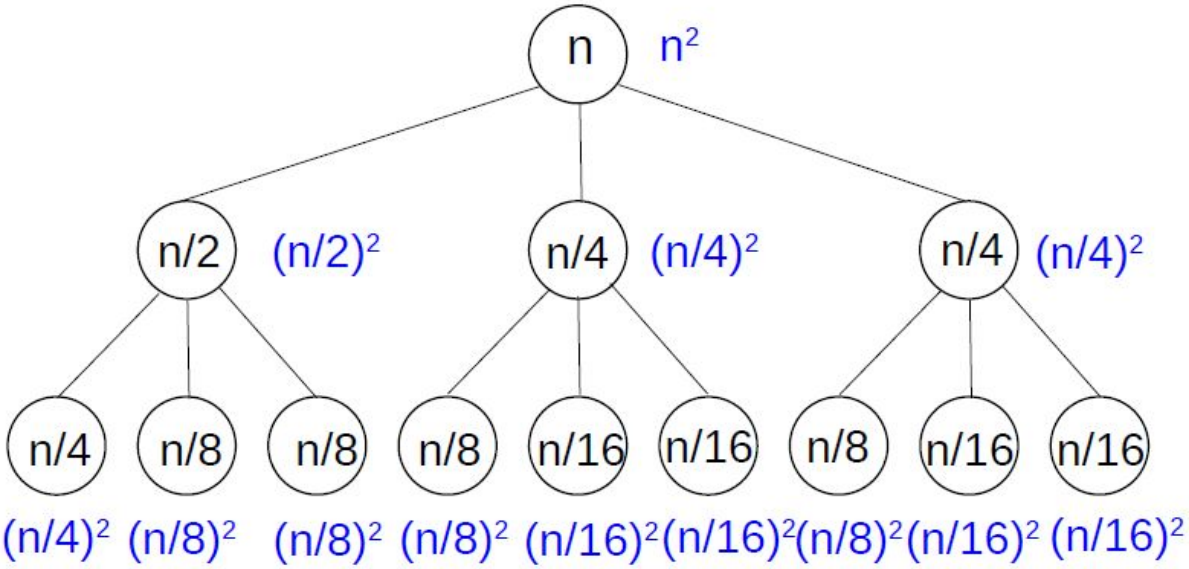
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: work done at each node (continued)



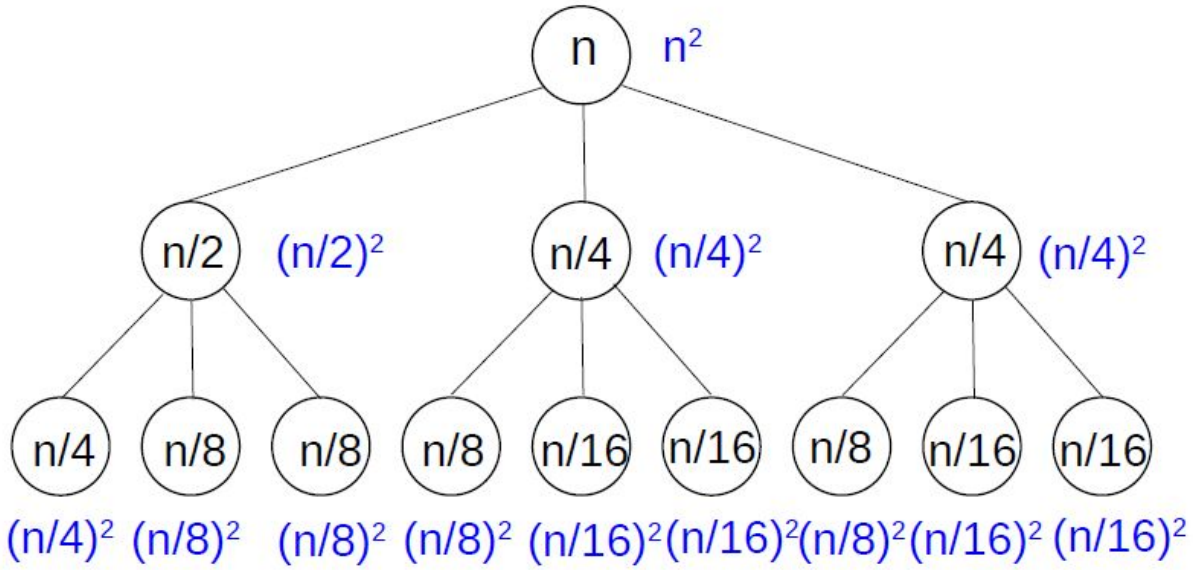
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: work done at each node (continued)



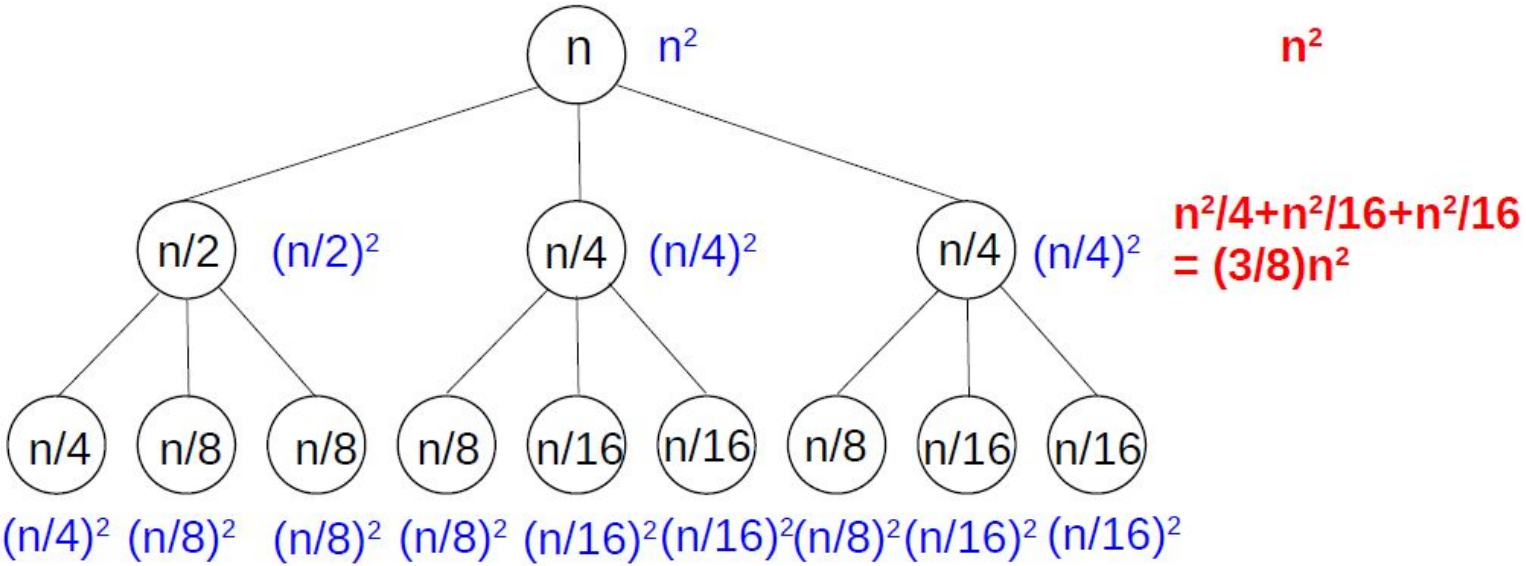
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: work done on each row



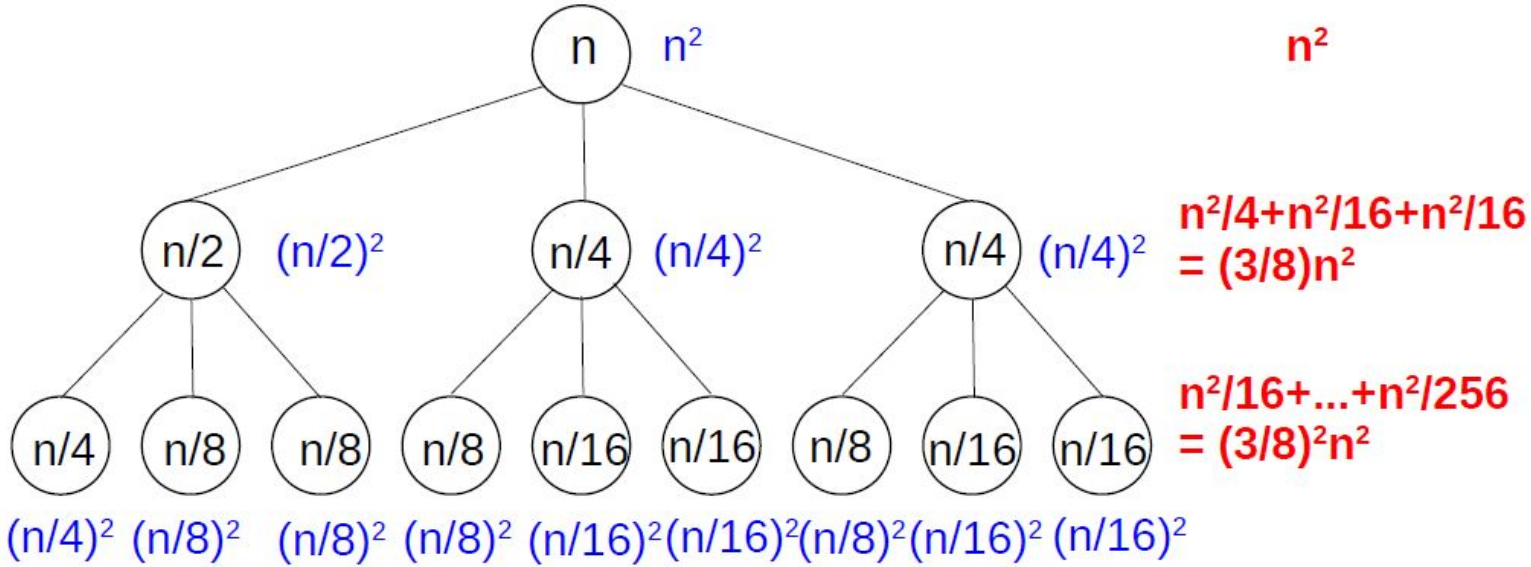
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: work done on each row



$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: work done on each row



$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: summing up the work on all the rows
  - The total work is $n^2 + (3/8)n^2 + (3/8)^2n^2 + \ldots$
  - This is a geometric series, and $3/8 < 1$.
  - So the sum converges to $\frac{1}{1-3/8}n^2$
  - Hence $T(n)\in\Theta(n^2)$

The Master Theorem

- Most divide and conquer algorithms split the input into equal-size subproblems
- Most recursion trees fall into one of three categories:
  - The work per level increases geometrically
  - The work per level is constant (e.g., MergeSort)
  - The work per level decreases geometrically (e.g., the previous example)

The Master Theorem [Bentley, Haken, Saxe]

- Theorem: Let $a \geq 1$, $b > 1$ be real constants, let $f: N \rightarrow R^+$, and let $T(n)$ be defined by:

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{if } n \geq n_0 \\ \Theta(1) & \text{if } n < n_0 \end{cases}$$

where n/b might be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then

1. If $f(n) \in O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $af(n/b) < \delta f(n)$ for some $0 < \delta < 1$ and all n large enough, then $T(n) \in \Theta(f(n))$.

regularity condition

The Master Theorem [Bentley, Haken, and Saxe]

How to apply the theorem:

- Compute $\mathbf{\log_b a}$
- Compare it to the exponent of **n** in **f(n)**
  - If $\mathbf{\log_b a}$ is larger: case 1.
  - If they are equal: maybe case 2.
  - If $\mathbf{\log_b a}$ is smaller: check regularity condition, and if it holds then case 3.