3. Divide and Conquer Algorithms

Definitions

- A divide and conquer algorithm proceeds by
  - Dividing the input into two or more **smaller** instances of the same problem.
    - 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 algorithm can be expressed using a *recurrence relation*:
  - T(n) is defined in terms of one or more T(something smaller than n).
  - Example:

One recursive call on n/2 items.

Two recursive calls on n/4 items.

n^2 work not done inside a recursive call

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

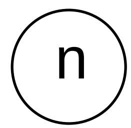
Recursion trees

- One way to solve a recurrence relation is to draw a recursion tree.
  - You draw a tree that represents the recursion.
  - 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 inside recursive alls.
  - Compute the total amount of work on each row.
  - Then add up the work done by the rows.

Recursion trees

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

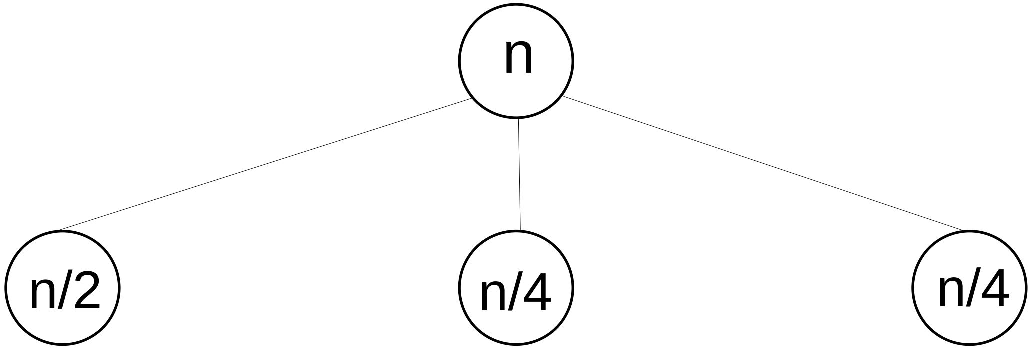
- Example: drawing the tree



Recursion trees

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

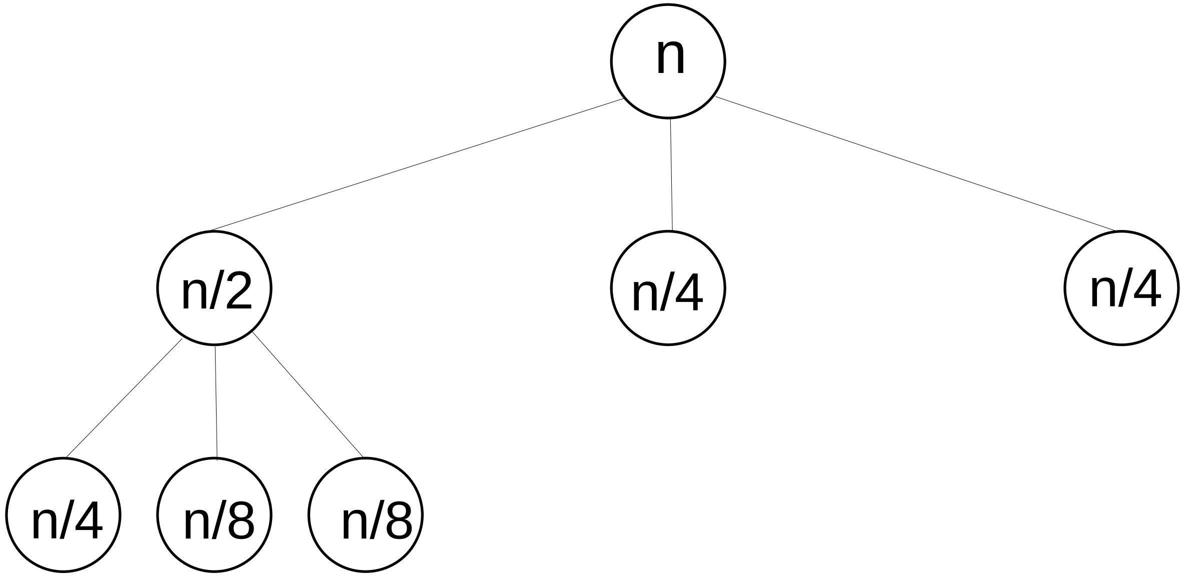
- Example: drawing the tree (continued)



Recursion trees

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

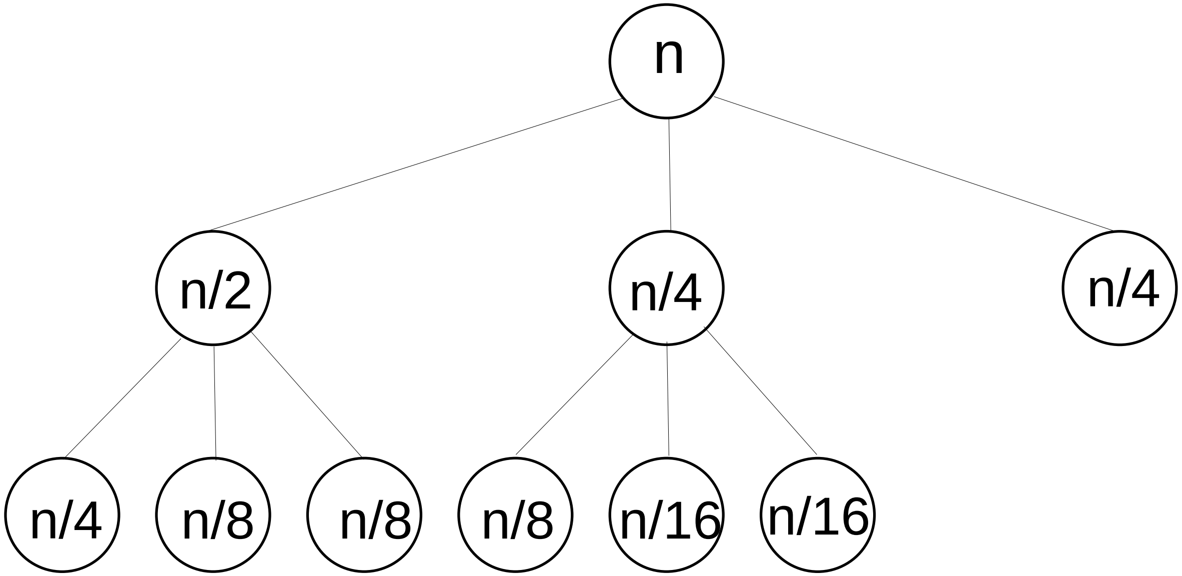
Example: drawing the tree (continued)



Recursion trees

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

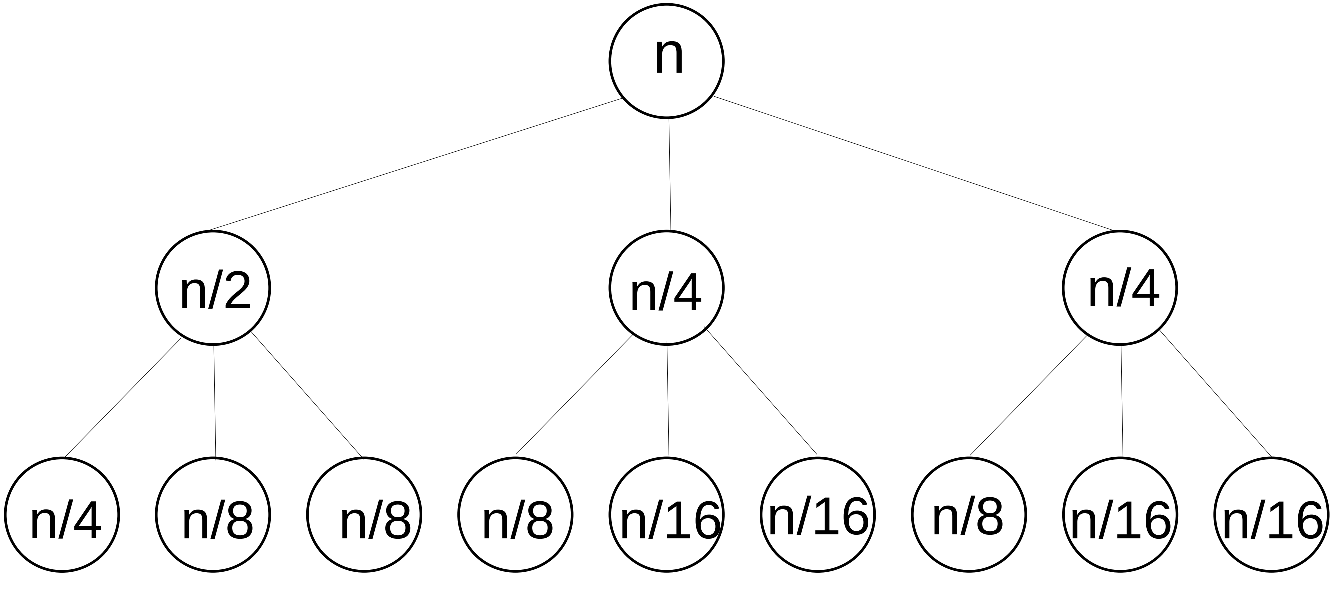
Example: drawing the tree (continued)



Recursion trees

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

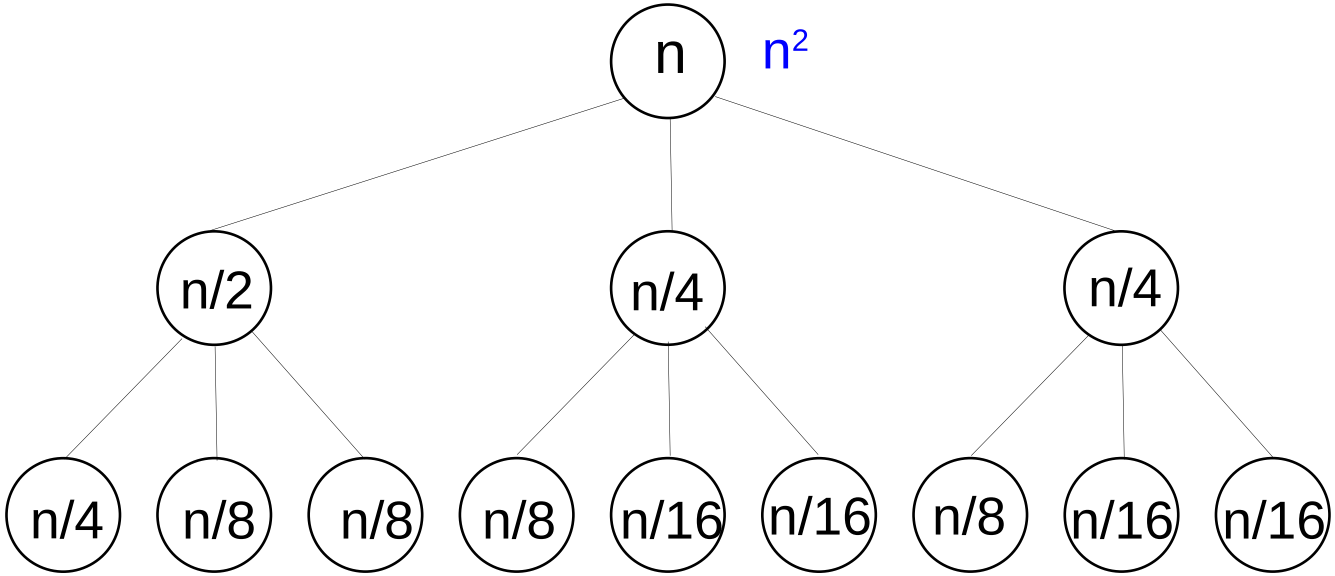
Example: drawing the tree (continued)



Recursion trees

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

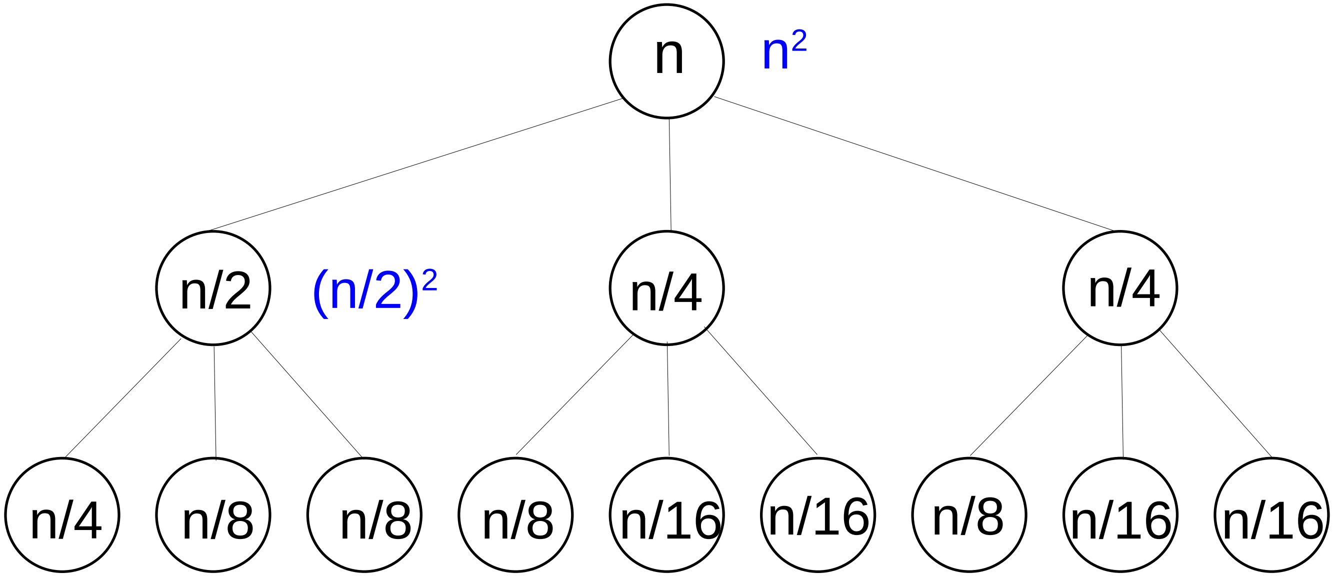
Example: work done at each node



Recursion trees

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

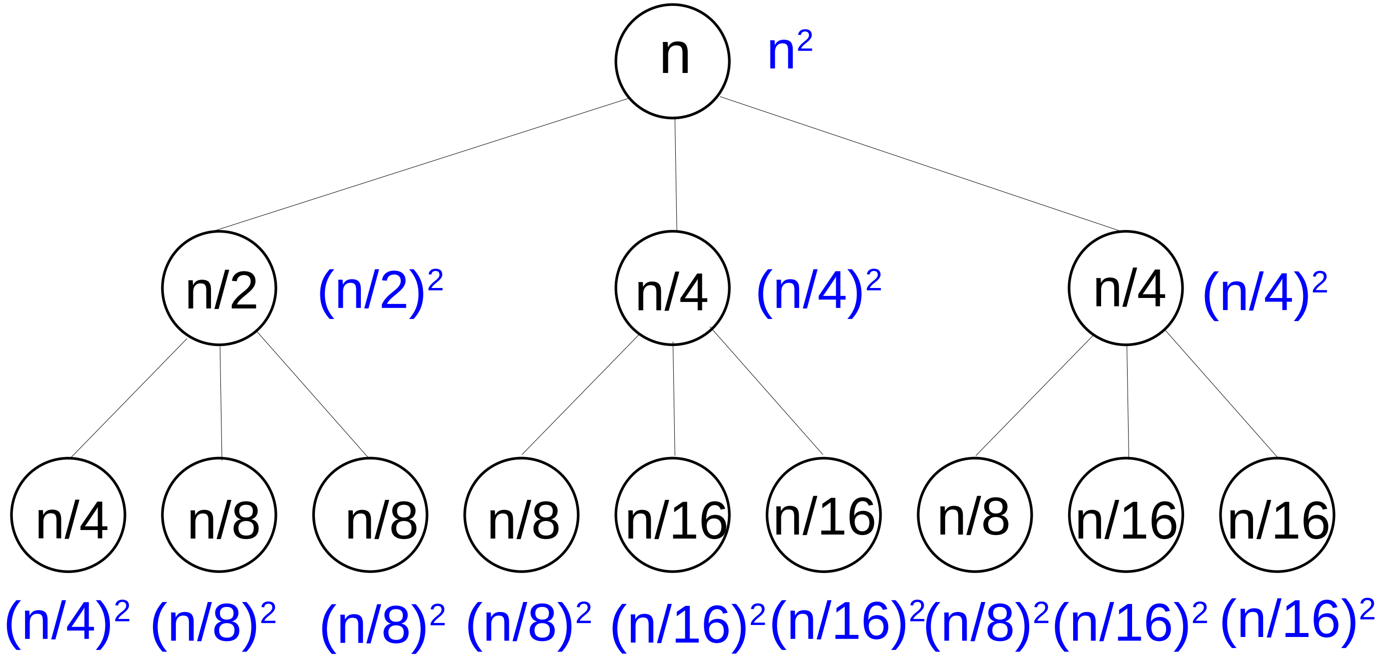
Example: work done at each node (continued)



Recursion trees

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

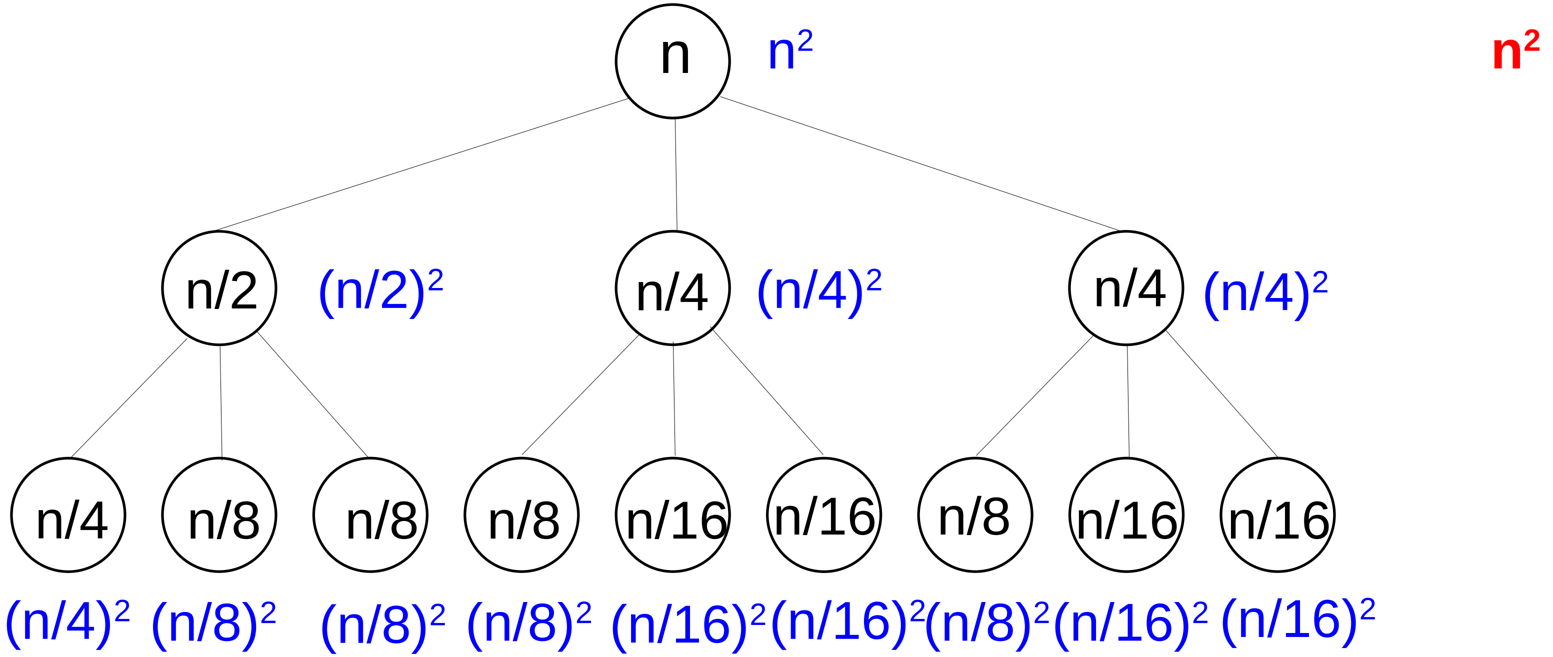
Example: work done at each node (continued)



Recursion trees

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

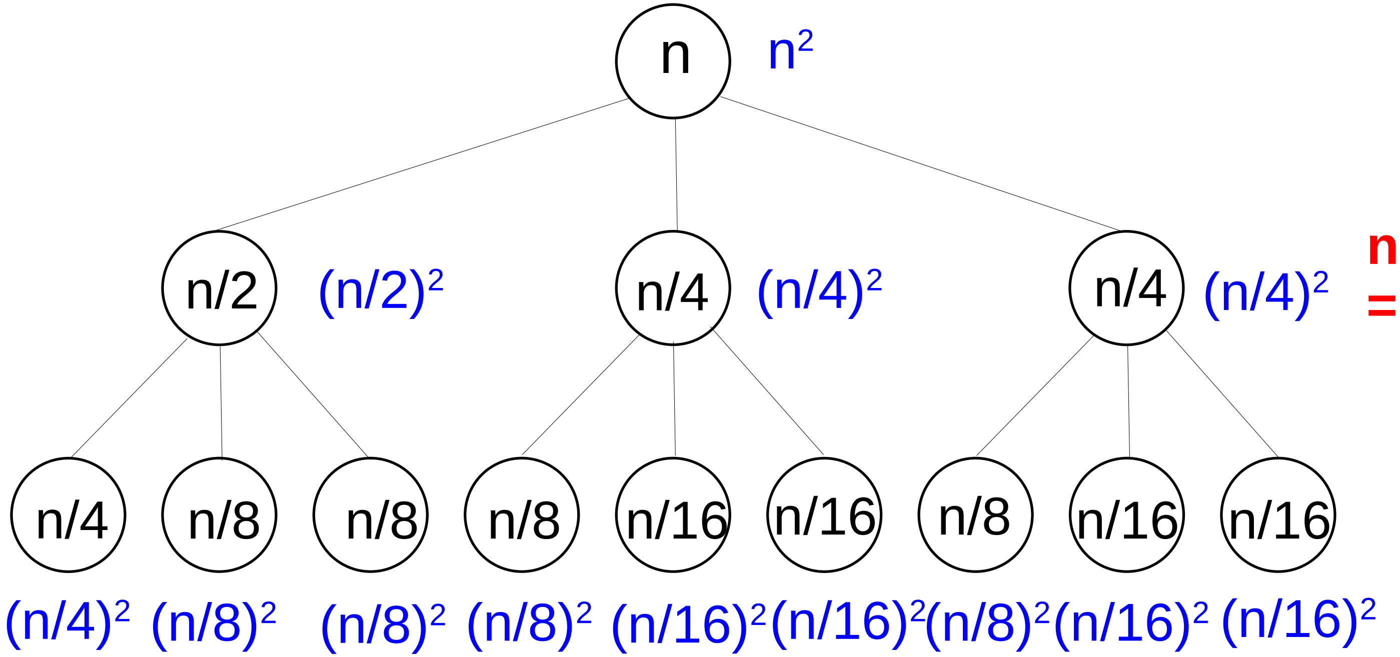
- Example: work done on each row



Recursion trees

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

- Example: work done on each row



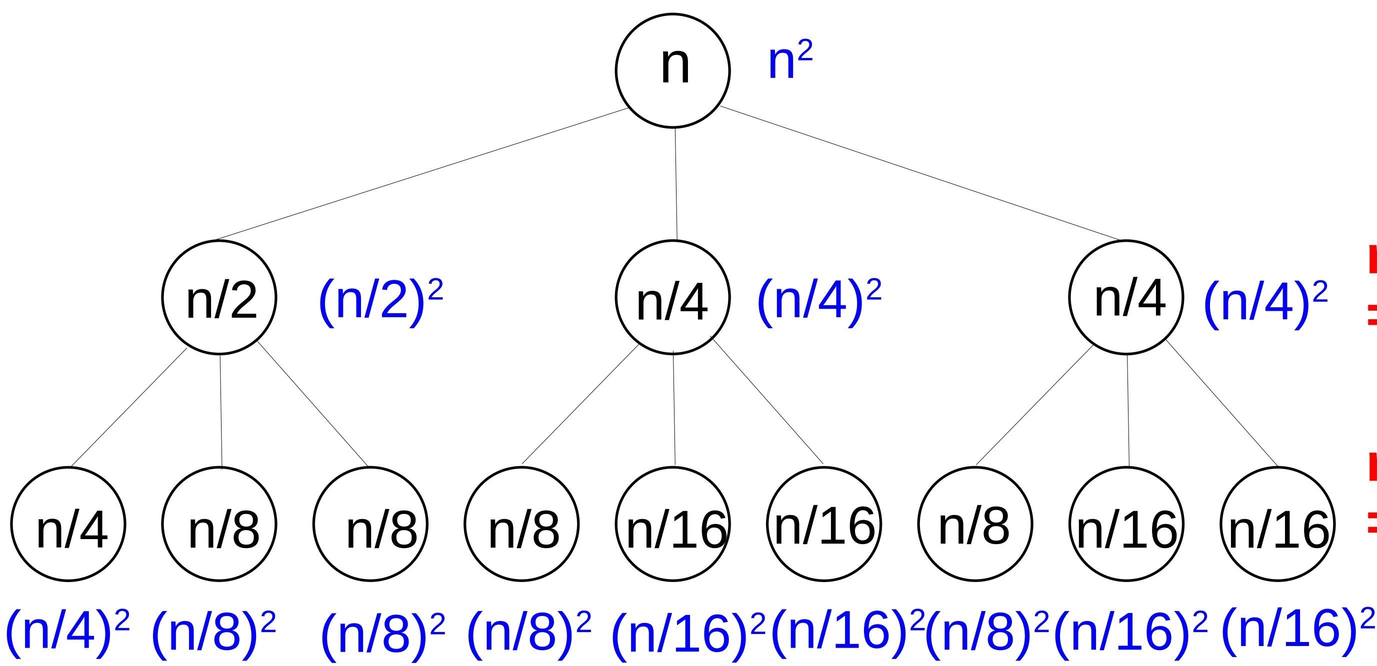
n^2

$n^2/4+n^2/16+n^2/16$
$= (3/8)n^2$

Recursion trees

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

- Example: work done on each row



n^2

$n^2/4+n^2/16+n^2/16$
$= (3/8)n^2$

$n^2/16+\ldots+n^2/256$
$= (3/8)^2n^2$

Recursion trees

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

- 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 algorithm split the input into equal-size subproblems.
- Most recursion trees fall in one of 3 cases:
  - The work per level increases geometrically (“The case of $q > 2$ subproblems”).
  - The work per level is constant (“Mergesort”).
  - The work per level decreases geometrically.

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, Saxe]

- Applying the theorem:
  - Compute $\log_b a$.
  - Compare it to the exponent of n in $f(n)$.
    - If $\log_b a$ is larger: case 1.
    - If they are equal: maybe case 2.
    - If $\log_b a$ is smaller: check regularity condition, and if it works it's case 3.