CS 381 – Spring 2021

Week 2, Lecture 1

Part 1

The divide-and-conquer (DC) algorithm design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

About D&C

- **Running time** is captured by a recurrence relation
  - E.g. $T(n) \leq 3T\left(\frac{n}{3}\right) + 4\mathrm{n}^2$
    - break a problem of size n into 3 problems of size n/3 each
    - time for creating the subproblems and combining is $O(n^2)$.
- **Correctness**
  - of the recurrence: proof by induction, use Master theorem
  - of the algorithm: inductive argument

Problem: Given a, n>0, compute a^n.
Minimize number of multiplications.

Naive algorithm: $\Theta(n)$ multiplications

Problem: Given a, n>0, compute a^n.

Minimize number of multiplications.

Naive algorithm: $\Theta(n)$ multiplications

Can we do better?

Problem: Given a, n>0, compute a^n.
Minimize number of multiplications.

Naive algorithm: $\Theta(n)$ multiplications

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd} \end{cases}$$

Problem: Given a, n>0, compute a^n.
Minimize number of multiplications.

Naive algorithm: $\Theta(n)$ multiplications

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd} \end{cases}$$

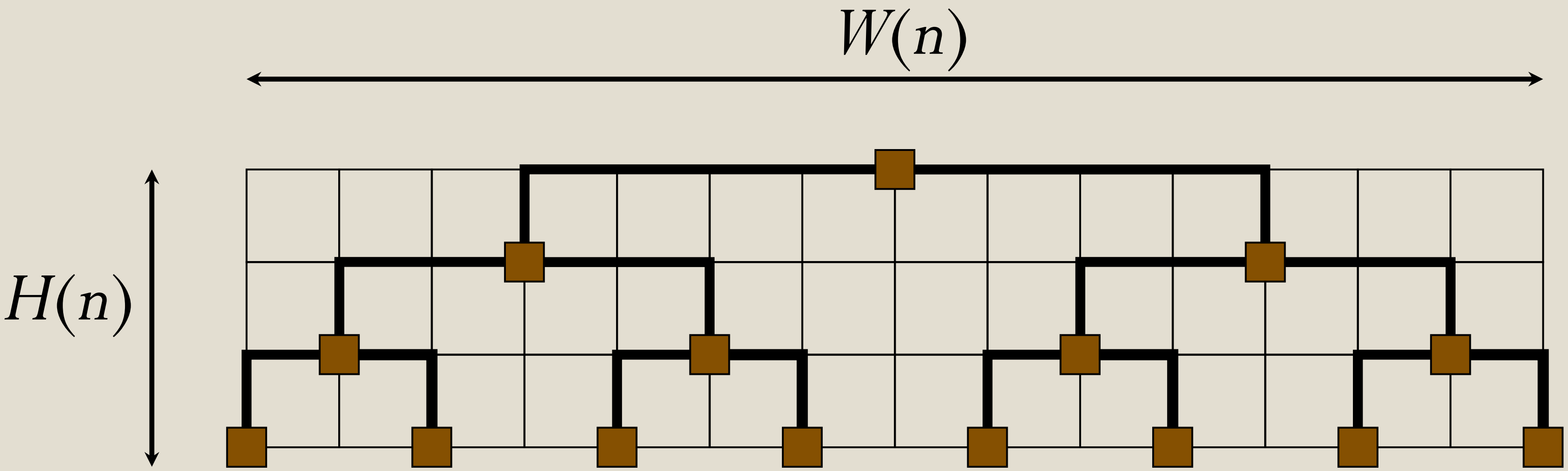
$$T(n) = T(n/2) + c \Rightarrow T(n) = \Theta(\log n)$$

D&C algorithm for computing a^n

```
Exp(a,n) {
    if n=0 return 1
    else if n=1 return a
    else if n is even
        b ← Exp(a,n/2)
        return b × b
    else                // n>1 is odd
        b ← Exp(a,(n-1)/2)
        return b × b × a
}
```

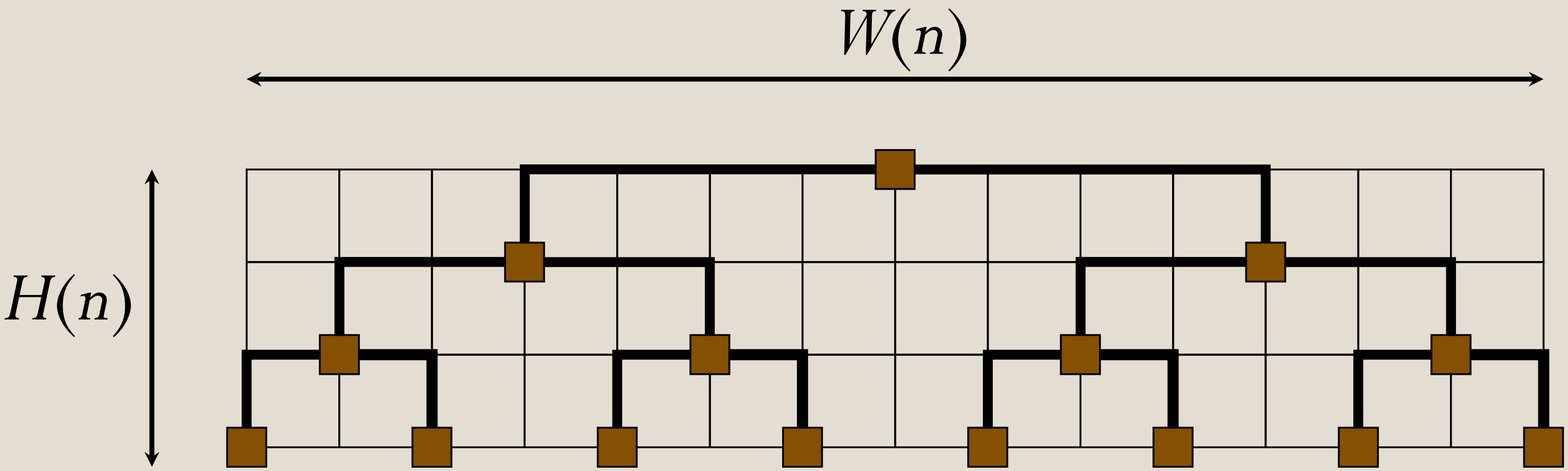
$$T(n) = T(n/2) + c \Rightarrow T(n) = \Theta(\lg n)$$

Problem: Draw a complete binary tree with n leaves on a grid minimizing the area.



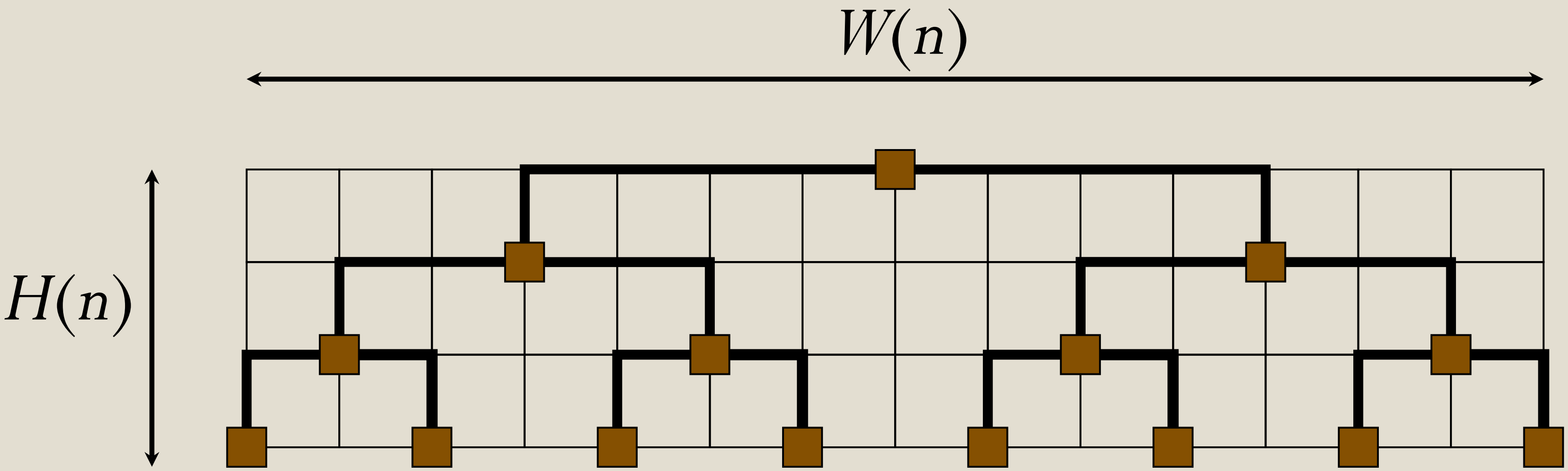
What are $H(n)$ and $W(n)$?

Problem: Draw a complete binary tree with n leaves on a grid minimizing the area.



$$H(n) = H(n/2) + \Theta(1)$$

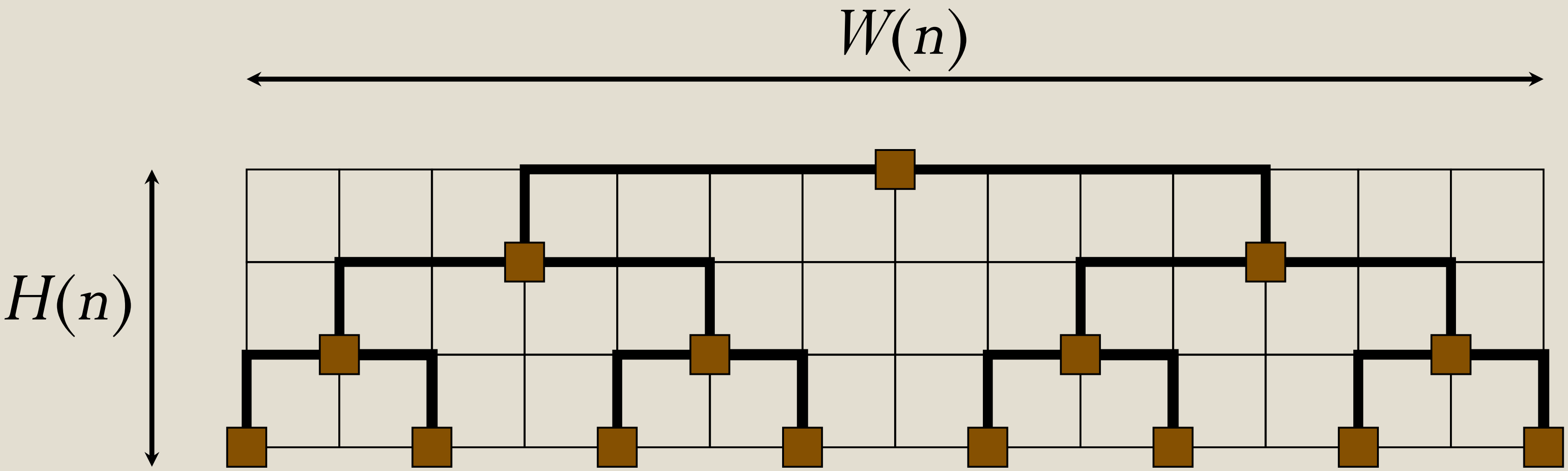
Problem: Draw a complete binary tree with n leaves on a grid minimizing the area.



$$H(n) = H(n/2) + \Theta(1) = \Theta(\lg n)$$

$$W(n) = 2\,W(n/2) + \Theta(1)$$

Problem: Draw a complete binary tree with n leaves on a grid minimizing the area.



$$H(n) = H(n/2) + \Theta(1) = \Theta(\lg n)$$

$$W(n) = 2\,W(n/2) + \Theta(1) = \Theta(n)$$

Drawing a complete binary tree on a grid with leaves on “one line” results in $O(n \log n)$ area.

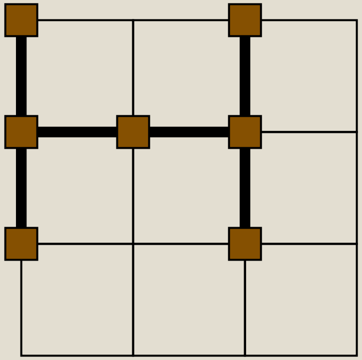
Can we do better?

Drawing a complete binary tree on a grid with leaves on "one line" results in $O(n \log n)$ area.

Can we do better?

- Area needs to be at least n (i.e., area is $\Omega(n)$)

n = 1



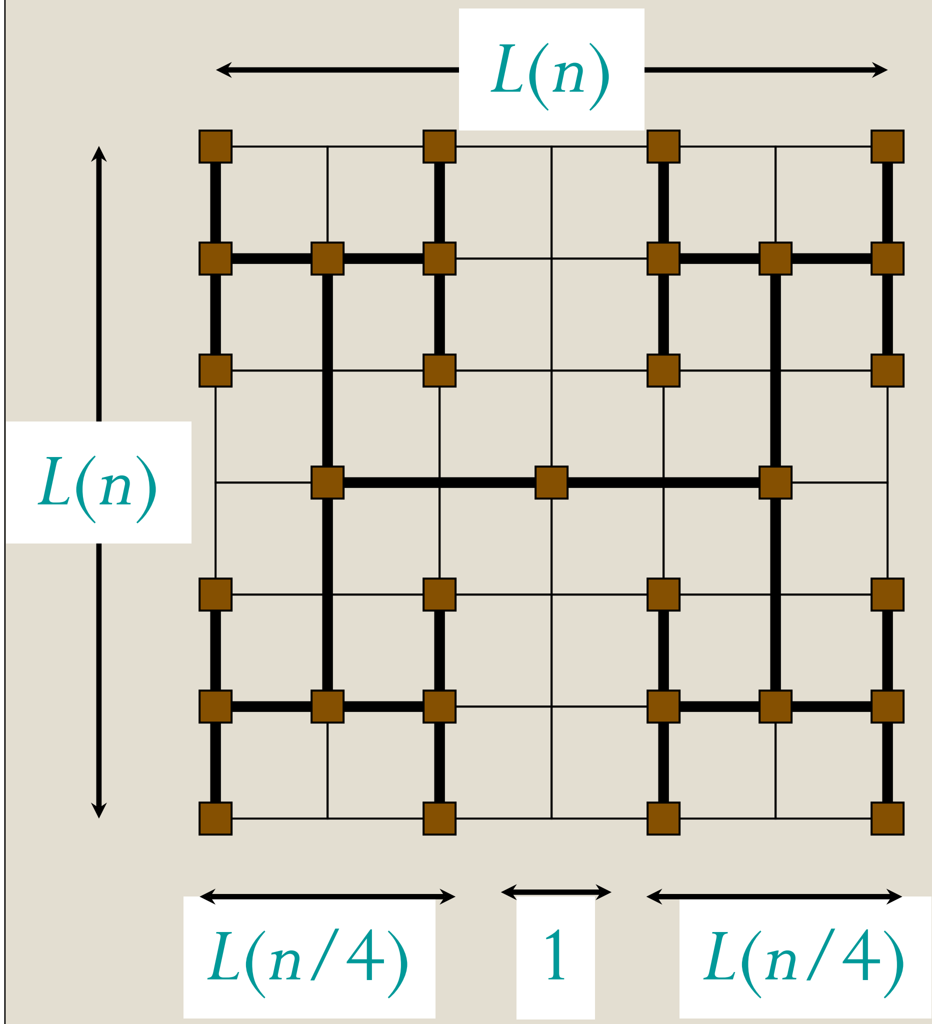
n = 7

For $n = 2^k - 1$ and k odd

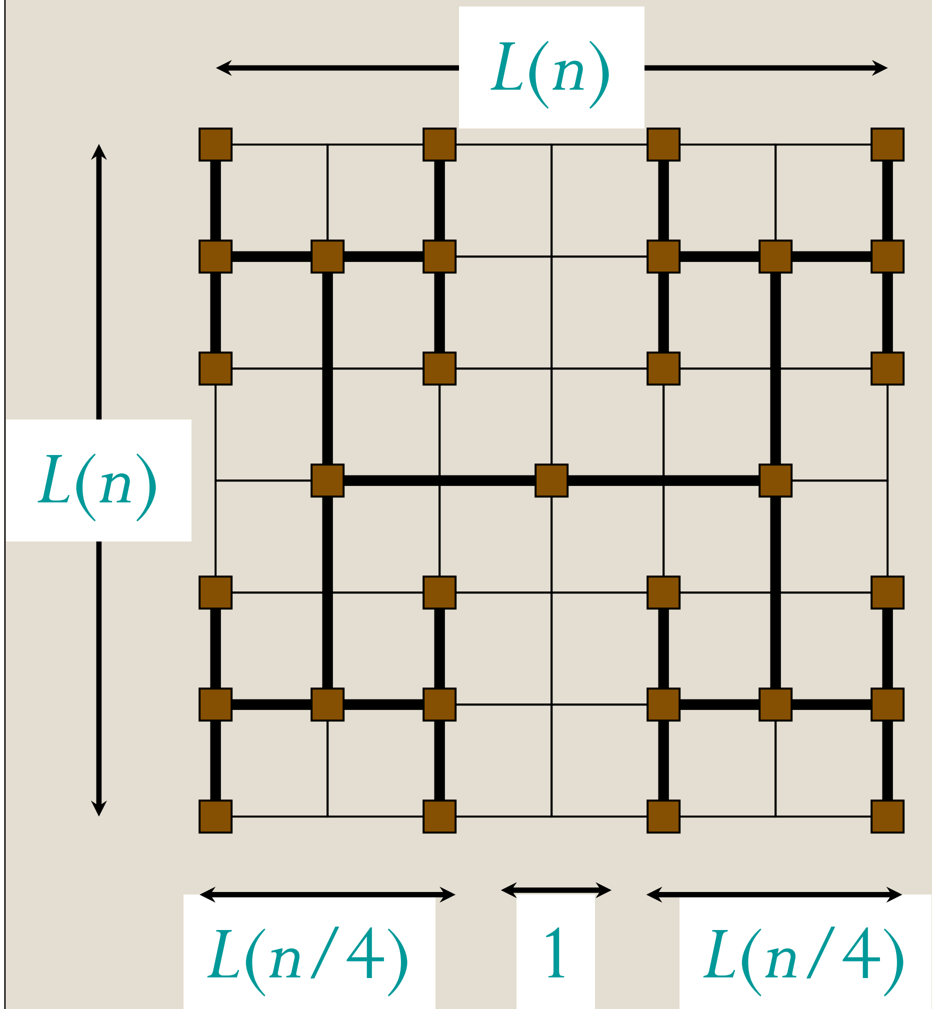
L(n) is the side length of the embedding

- L(1) = 1
- L(7) = 3
- L(63) = 7

H-tree drawing: Idea



H-tree drawing: Idea

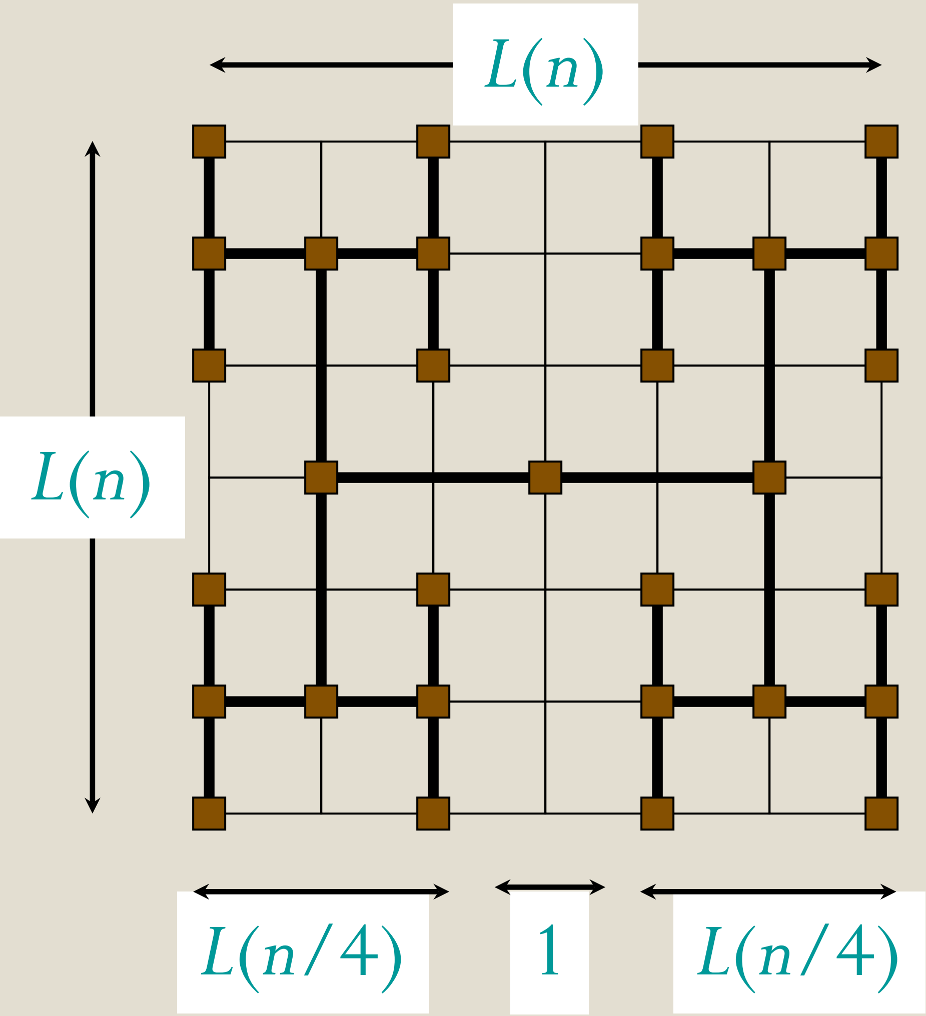


$n = 2^k - 1$ and k odd:

L(63) = 7

What is $L(n)$?

H-tree drawing: Idea



$n = 2^k - 1$ and k odd:

$L(63) = 7$

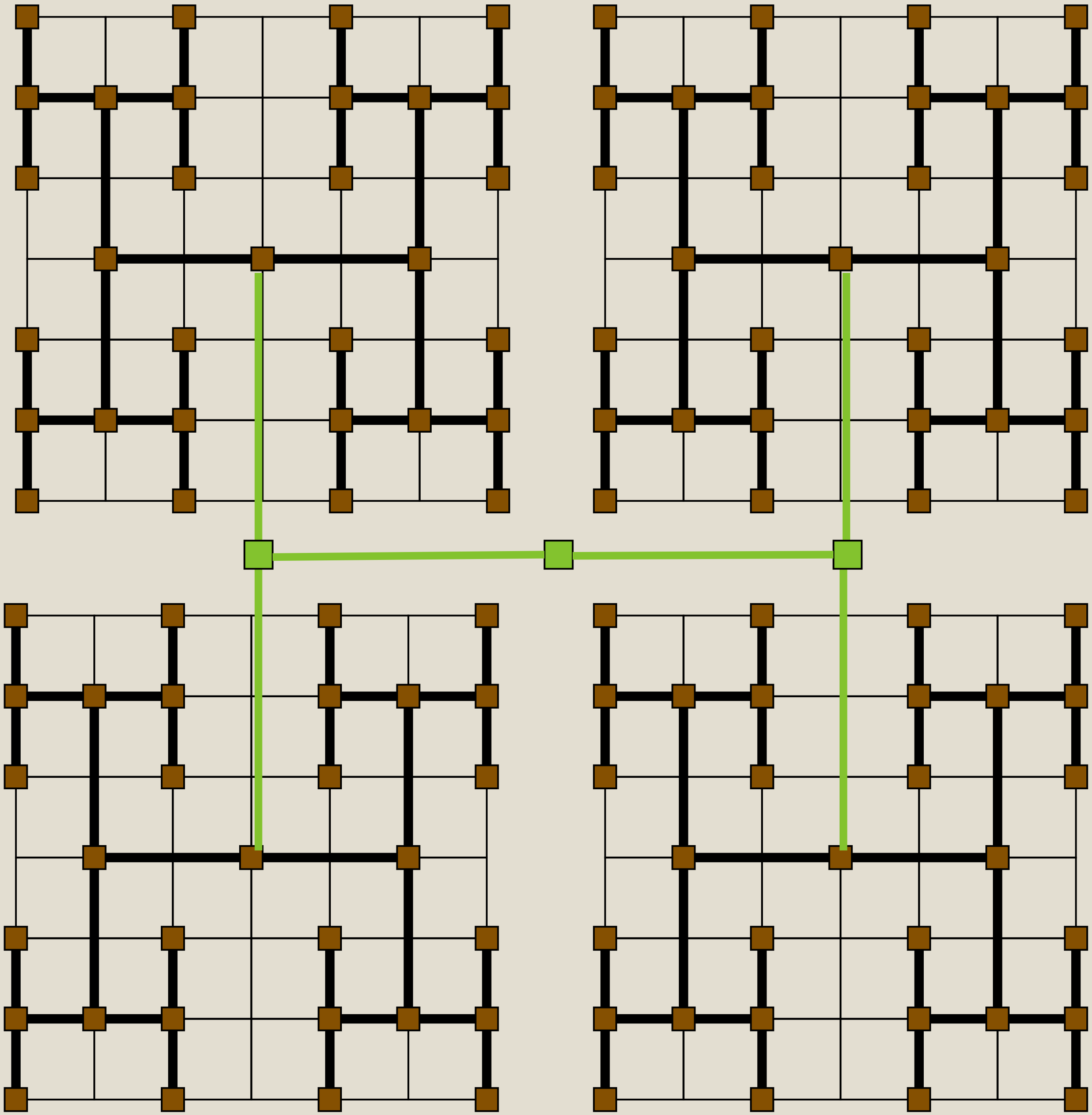
$$L(n) = 2L\left(\frac{n+1}{4} - 1\right) + 1$$

$$= 2L\left(\frac{n-3}{4}\right) + 1$$

$$\cong 2L(n/4) + 1$$

$$= \Theta(\sqrt{n})$$

$$\text{Area} = \Theta(n)$$



$n = 255$

How to determine the closed form of a recurrence relation?

For example, $T(n) = 4T(n/2) + n$

General method

(1) "Guess" the solution.

(in closed exact form or in asymptotic form)

(2) Prove correctness by induction.

If the guessed solution is incorrect, the induction will fall apart somewhere.

$$T(n) = 4T(n/2) + n \quad (n = 2^k)$$

$$T(1) = 0$$

$$T(n) = 4T(n/2) + n \quad (n = 2^k)$$
$$T(1) = 0$$

Claim: $T(n) = O(n^3)$

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

$T(1) = 0$

Claim: $T(n) = O(n^3)$

Induction Hypothesis:

Assume $T(m) \leq cm^3$ for all $m<n$, $m = 2^r$, for some constant c

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

$$\leq 4\, c\, (n/2)^3 + n$$

$$= cn^3/2 + n$$

$$= cn^3 - (cn^3/2 - n)$$

$$\mathbf{\leq cn^3}$$

Need to show $cn^3/2 - n \geq 0$. True for $c \geq 2$

Is $T(n) = \Theta(n^3)$?

$$\mathbf{T(n) = 4T(n/2) + n} \quad (n = 2^k)$$
$$\mathbf{T(1) = 0}$$

Claim: $\mathbf{T(n) = \Omega(n^3)}$

$\mathbf{T(n) = 4T(n/2) + n}$ $(n = 2^k)$

$\mathbf{T(1) = 0}$

Claim: $\mathbf{T(n) = \Omega(n^3)}$

Induction Hypothesis:

Assume $T(k) \geq ck^3$ for all $k<n$, for some constant c

$$\mathbf{T(n) = 4T(n/2) + n}$$
$$\geq 4\,c\,(n/2)^3 + n$$
$$= cn^3/2 + n$$
$$\geq \mathbf{cn^3}$$

Would need $n \geq \dfrac{cn^3}{2}$ for some constant c – Not True!

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

$T(1) = 0$

~~**Claim: $T(n) = \Omega(n^3)$**~~ **FALSE!**

Induction Hypothesis:

Assume $T(k) \geq ck^3$ for all $k<n$, for some constant c

$$
\begin{aligned}
\mathbf{T(n)} &\mathbf{= 4T(n/2) + n} \\
&\geq 4\,c\,(n/2)^3 + n \\
&= cn^3/2 + n \\
&\geq \mathbf{cn^3}
\end{aligned}
$$

Would need $n \geq \frac{cn^3}{2}$ for some constant c – Not True!

Show that $T(n) = O(n^2)$

Claim: $T(n) \leq cn^2$

Show that $T(n) = O(n^2)$

Claim: $T(n) \le cn^2$

$$\mathbf{T(n) = 4T(n/2) + n \le cn^2}$$
$$\le 4\, c\, (n/2)^2 + n$$
$$= cn^2 + n$$
$$\le \mathbf{cn^2} \qquad \textbf{\textit{NO!}}$$

Wrong argument: *we just made the constant c larger*

But .. $T(n) = O(n^2)$

What went wrong?

To show $O(n^2)$, we need to subtract lower order terms in the IH!

Claim: $\mathbf{T(n) \le c_1 n^2 - c_2 n}$, for some constants c_1 and c_2

Inductive hypothesis: $T(k) \le c_1 k^2 - c_2 k$ for all $k<n$

$$
\begin{aligned}
\mathbf{T(n)} &\mathbf{= 4T(n/2) + n} \\
&\le 4\,\mathbf{(c_1(n/2)^2 - c_2(n/2))} + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= \mathbf{c_1 n^2 - c_2 n} - c_2 n + n \\
&\le \mathbf{c_1 n^2 - c_2 n}
\end{aligned}
$$

Need $-c_2 n + n \le 0$:

true if $c_2 \ge 1$

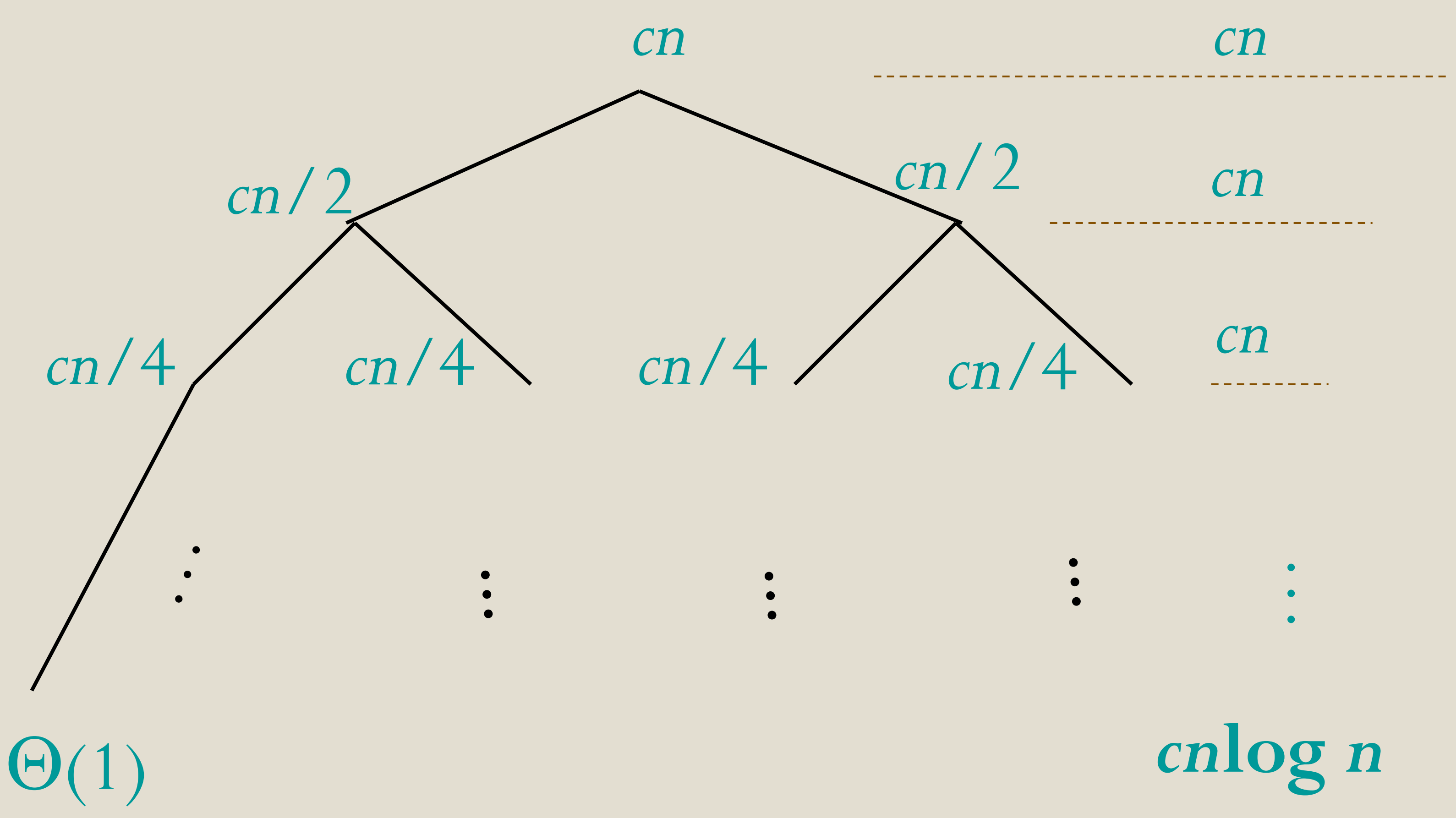
Recursion Tree Method

Use the recursion tree to determine the solution to a recurrence

- Tree represents a model of the cost of the recursive algorithm
- Getting an exact closed form can be messy
- Insight obtained from tree can give a good initial guess to be used in an induction

Mergesort: $T(n) = 2T(n/2) + cn$

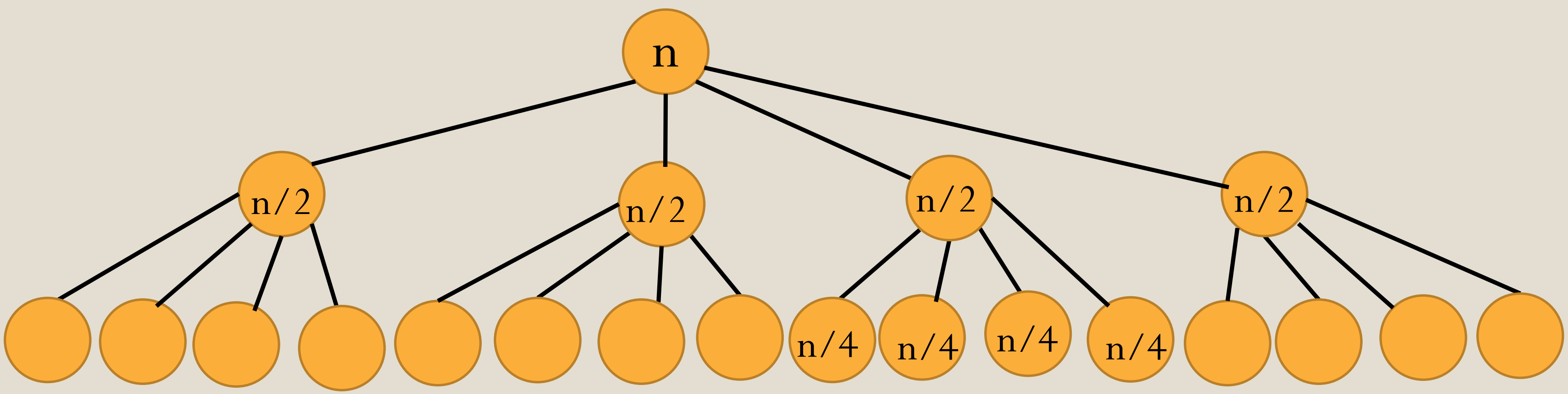
Prove O(n*log*n) by summing up total work using the recursion tree



$T(n) = 4T(n/2) + n, n=2^k$

Work done at each level

- Level 0: n
- Level 1: 2n (4 instances of size n/2 each)
- Level 2: 4n ($4^2 = 16$ instances of size n/4 each)
- Level 3: 8n (4^3 instances of size $n/2^3$ each)
- at level i, there are 4^i instances of size $n/2^i$ results in $2^i n$ total work for level i



Total work done at all levels of the recursion tree

at level i of the tree

- there are 4^i nodes, each doing work of size $n/2^i$
- results in $n2^i$ total work for level i

$$\sum_{i=0}^{k} n\, 2^i = n\left(2^{k+1} - 1\right) = n(2n - 1) = 2n^2 - n$$

This gives $T(n) = \Theta(n^2)$

Recurrences of divide and conquer algorithms

Assume the basis is $T(1) = \Theta(1)$

- $T(n) = T(n/2) + c$
- $T(n) = T(n/2) + cn$
- $T(n) = 2T(n/2) + cn$
- $T(n) = 2T(n-1) + 1$
- $T(n) = 4T(n/2) + n$
- $T(n) = 2T(n/4) + 1$
- $T(n) = T(n/4) + T(n/2) + n^2$
- $T(n) = T(2n/3) + n$
- $T(n) = T(\sqrt{n}) + c$

Recurrences of divide and conquer algorithms

Assume the basis is $T(1) = \Theta(1)$

- $T(n) = T(n/2) + c$ $\theta(\log n)$
- $T(n) = T(n/2) + cn$ $\theta(n)$
- $T(n) = 2T(n/2) + cn$ $\theta\ (n \log n)$
- $T(n) = 2T(n-1) + 1$ $\theta\ (2^n)$
- $T(n) = 4T(n/2) + n$ $\theta\ (n^2)$
- $T(n) = 2T(n/4) + n$ $\Theta(\sqrt{n})$
- $T(n) = T(n/4) + T(n/2) + n^2$ $\theta\ (n^2)$
- $T(n) = T(2n/3) + n$ $\theta\ (n)$
- $T(n) = T(\sqrt{n}) + c$ $\theta\ (\log \log n)$