095946- ADVANCED ALGORITHMS AND PARALLEL PROGRAMMING

Fabrizio Ferrandi

a.a. 2021-2022

- Material adapted from Erik D. Demaine and Charles E. Leiserson slides

DIVIDE AND CONQUER

BINARY SEARCH
POWERING A NUMBER
MATRIX MULTIPLICATION
STRASSEN'S ALGORITHM
VLSI TREE LAYOUT

THE DIVIDE-AND-CONQUER DESIGN PARADIGM

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

MERGE SORT

1. *Divide:* Trivial.
2. *Conquer:* Recursively sort 2 subarrays.
3. *Combine:* Linear-time merge.

MERGE SORT

1. *Divide:* Trivial.
2. *Conquer:* Recursively sort 2 subarrays.
3. *Combine:* Linear-time merge.

$$\underbrace{T(n) = 2}_{\#\ \text{subproblems}}\ T\underbrace{(n/2)}_{\text{subproblem size}} + \underbrace{\Theta(n)}_{\text{work dividing and combining}}$$

due to the fact we have to combine all the "sub"solutions

 # subproblems

subproblem size

work dividing and combining

MASTER THEOREM (REPRISE)

nel caso del mergesort f(n) = theta(n)
sicuramente posso associarlo al caso 2 perchè il caso 1 e 3 hanno O e omega -> lo associo al case 2 con k=0 perchè non ho log nella f(n)

$$T(n) = a\,T(n/b) + f(n)$$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$.

Master Theorem (reprise)

$$T(n) = a\,T(n/b) + f(n)$$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$.

Merge sort: $a = 2, b = 2 \;\Rightarrow\; n^{\log_b a} = n^{\log_2 2} = n$

$\Rightarrow$ CASE 2 $(k = 0) \;\Rightarrow\; T(n) = \Theta(n \lg n)$.

BINARY SEARCH

prendi l'elemento centrale, se il valore da cercare è maggiore allora cerchiamo a dx
altrimenti a sx

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

BINARY SEARCH

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

Example: Find 9

3	5	7	8	9	12	15

BINARY SEARCH

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

Example: Find 9

3	5	7	8	9	12	15

BINARY SEARCH

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

Example: Find 9

3	5	7	8	9	12	15

BINARY SEARCH

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

Example: Find 9

3	5	7	8	9	12	15

BINARY SEARCH

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

Example: Find 9

3	5	7	8	9	12	15

BINARY SEARCH

Find an element in a sorted array:

1. *Divide:* Check middle element.
2. *Conquer:* Recursively search 1 subarray.
3. *Combine:* Trivial.

Example: Find 9

3 5 7 8 9 12 15

RECURRENCE FOR BINARY SEARCH

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

 # subproblems

subproblem size

work dividing and combining

RECURRENCE FOR BINARY SEARCH

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

\# subproblems

subproblem size

work dividing
and combining

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{Case 2 } (k = 0)$$

$$\Rightarrow T(n) = \Theta(\lg n)\,.$$

POWERING A NUMBER

Problem: Compute a^n, where $n \in N$.

Naive algorithm: $\Theta(n)$.

POWERING A NUMBER

Problem: Compute a^n, where $n \in N$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

infatti il prodotto fa a

POWERING A NUMBER

Problem: Compute a^n, where $n \in N$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n)\,.$$ (by the master theorem)

sto lavorando sulla metà di n (NB->non devo mettere 2*T(N/2) perchè sto lavorando sulla stessa identica porzione di dati-> IMPORTANTISSIMO SPESSO CI SI SBAGLIA) e poi per il combine spendo solo theta(1) perchè devo solo fare una moltiplicazione che viene considerata costante

MATRIX MULTIPLICATION

Input: $A = [a_{ij}], B = [b_{ij}].$

Output: $C = [c_{ij}] = A \cdot B.$

$$\left.\begin{array}{l} A = [a_{ij}], B = [b_{ij}]. \\ C = [c_{ij}] = A \cdot B. \end{array}\right\} \; i, j = 1, 2, \ldots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

STANDARD ALGORITHM

for $i \leftarrow 1$ to n

do for $j \leftarrow 1$ to n

do $c_{ij} \leftarrow 0$

for $k \leftarrow 1$ to n

do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

STANDARD ALGORITHM

for $i \leftarrow 1$ to n

do for $j \leftarrow 1$ to n

do $c_{ij} \leftarrow 0$

for $k \leftarrow 1$ to n

do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$

DIVIDE-AND-CONQUER ALGORITHM

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices

DIVIDE-AND-CONQUER ALGORITHM

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

recursive

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices

ANALYSIS OF D&C ALGORITHM

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

Analysis of D&C Algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \text{Case 1} \;\Rightarrow\; T(n) = \Theta(n^3).$$

ANALYSIS OF D&C ALGORITHM

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

\# submatrices

submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \text{Case 1} \;\Rightarrow\; T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.

STRASSEN'S IDEA

- Multiply 2×2 matrices with only 7 recursive mults.

STRASSEN'S IDEA

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

STRASSEN'S IDEA

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

Strassen's Idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

STRASSEN'S IDEA

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$

STRASSEN'S ALGORITHM

1. *Divide:* Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. *Conquer:* Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. *Combine:* Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

STRASSEN'S ALGORITHM

1. *Divide:* Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using + and – .
2. *Conquer:* Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. *Combine:* Form C using + and – on $(n/2)\times(n/2)$ submatrices.

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

ANALYSIS OF STRASSEN

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \text{Case 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

ANALYSIS OF STRASSEN

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

ANALYSIS OF STRASSEN

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^{\lg 7}).$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

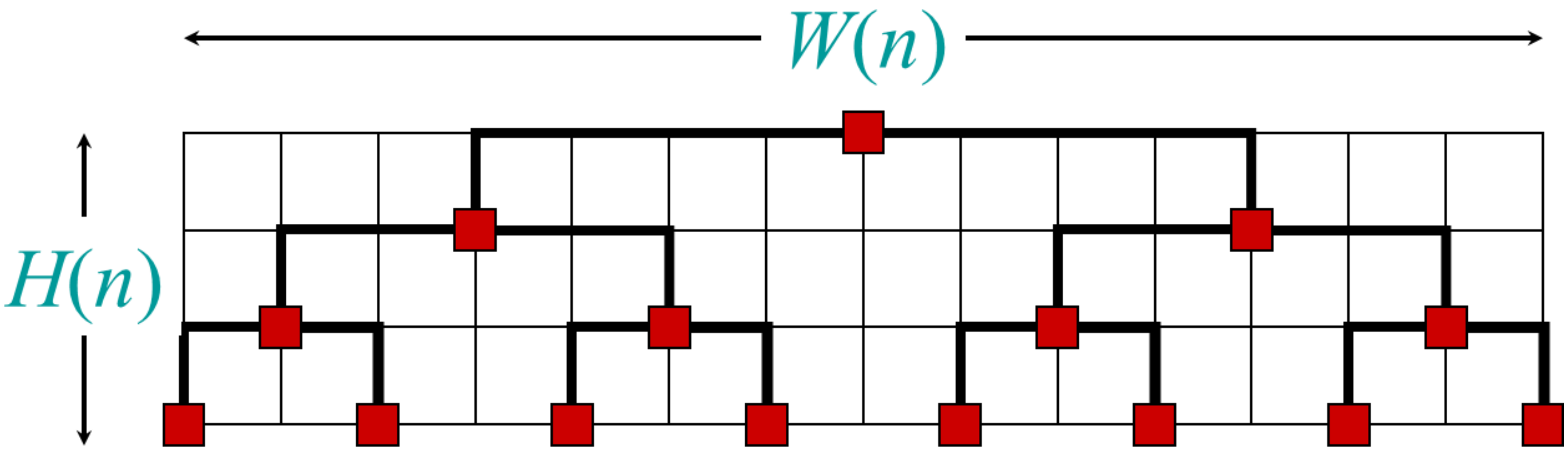
Best to date (of theoretical interest only): $\Theta(n^{2.376\cdots})$.

VLSI LAYOUT

Problem: Embed a complete binary tree with n leaves in a grid using minimal area.

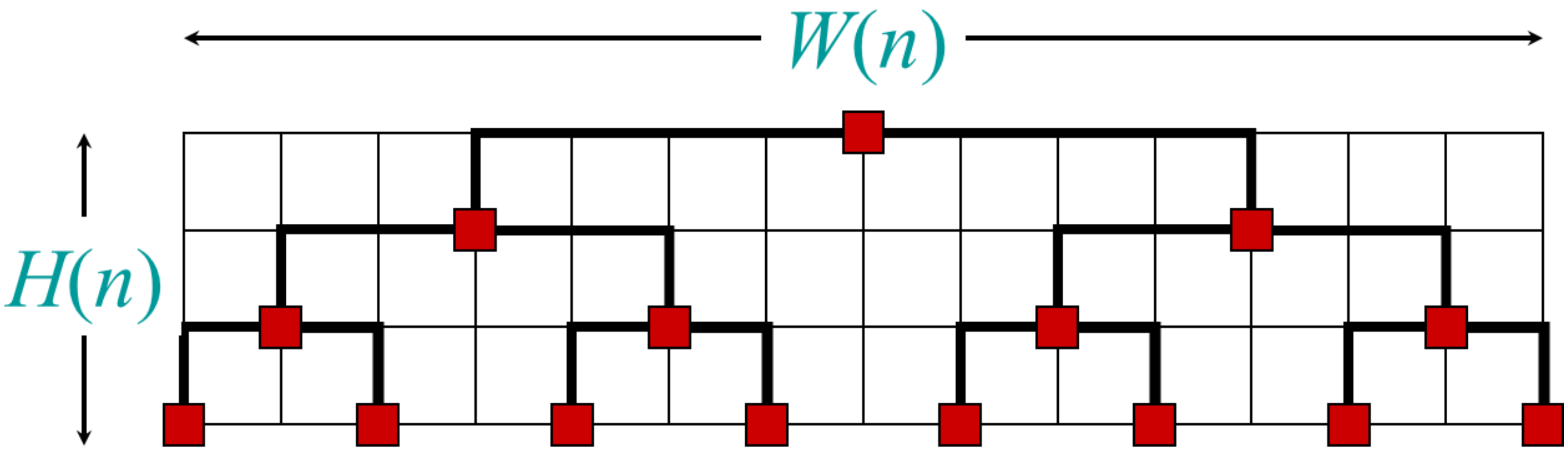
VLSI LAYOUT

Problem: Embed a complete binary tree with n leaves in a grid using minimal area.



VLSI LAYOUT

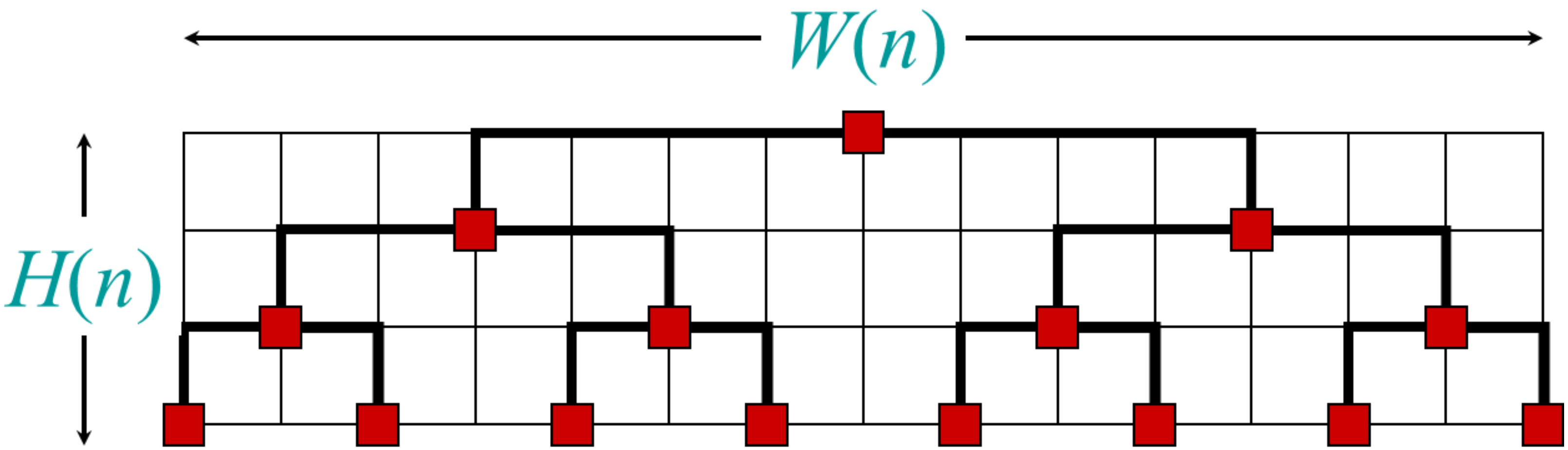
Problem: Embed a complete binary tree with n leaves in a grid using minimal area.



$$
\begin{aligned}
H(n) &= H(n/2) + \Theta(1) \\
&= \Theta(\lg n)
\end{aligned}
$$

VLSI LAYOUT

Problem: Embed a complete binary tree with n leaves in a grid using minimal area.



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

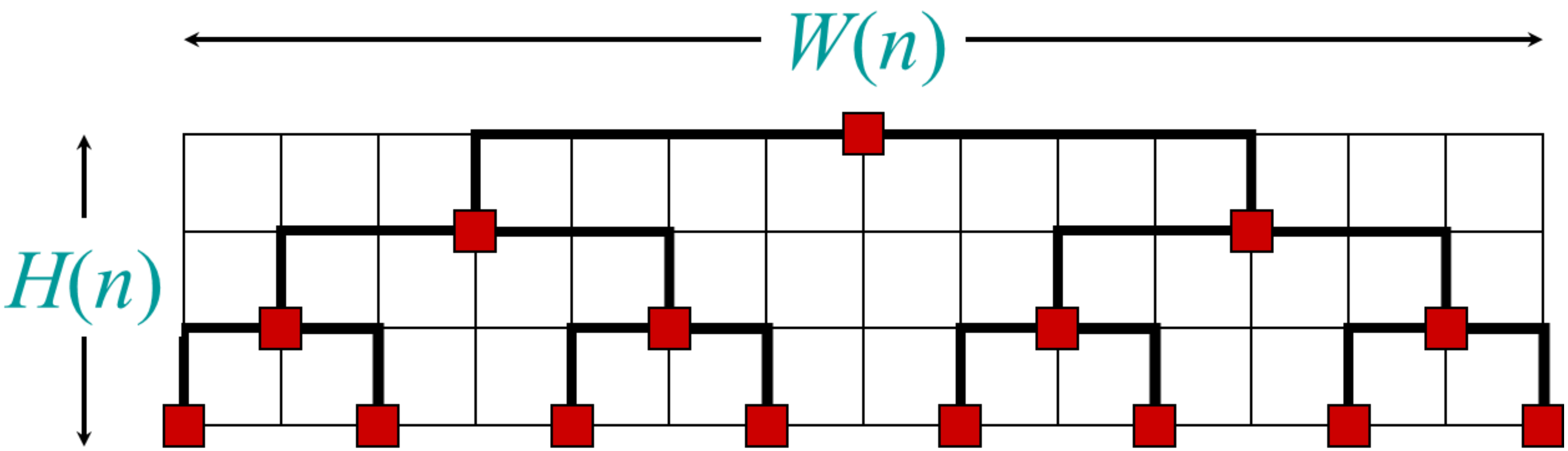
guardando la H(n) sto prendendo in esame n/2 nodi, + la radice

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

stessa cosa qui, solo che prendo 2 metà (sottoalbero sx e dx + root)

VLSI Layout

Problem: Embed a complete binary tree with n leaves in a grid using minimal area.

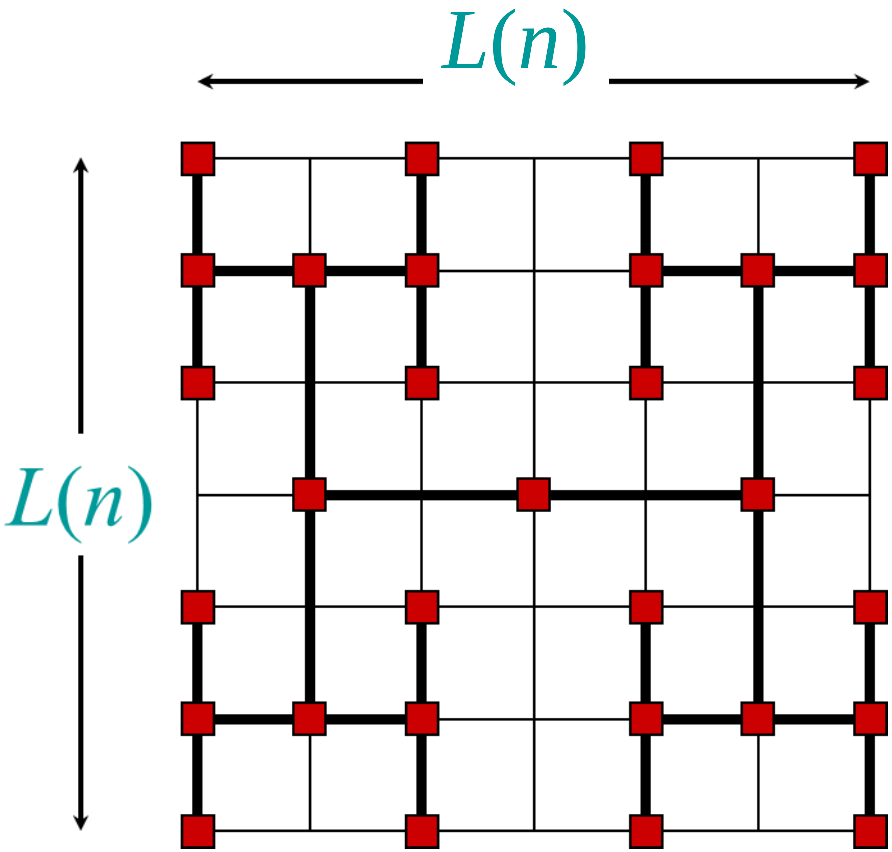


$H(n) = H(n/2) + \Theta(1)$
$= \Theta(\lg n)$

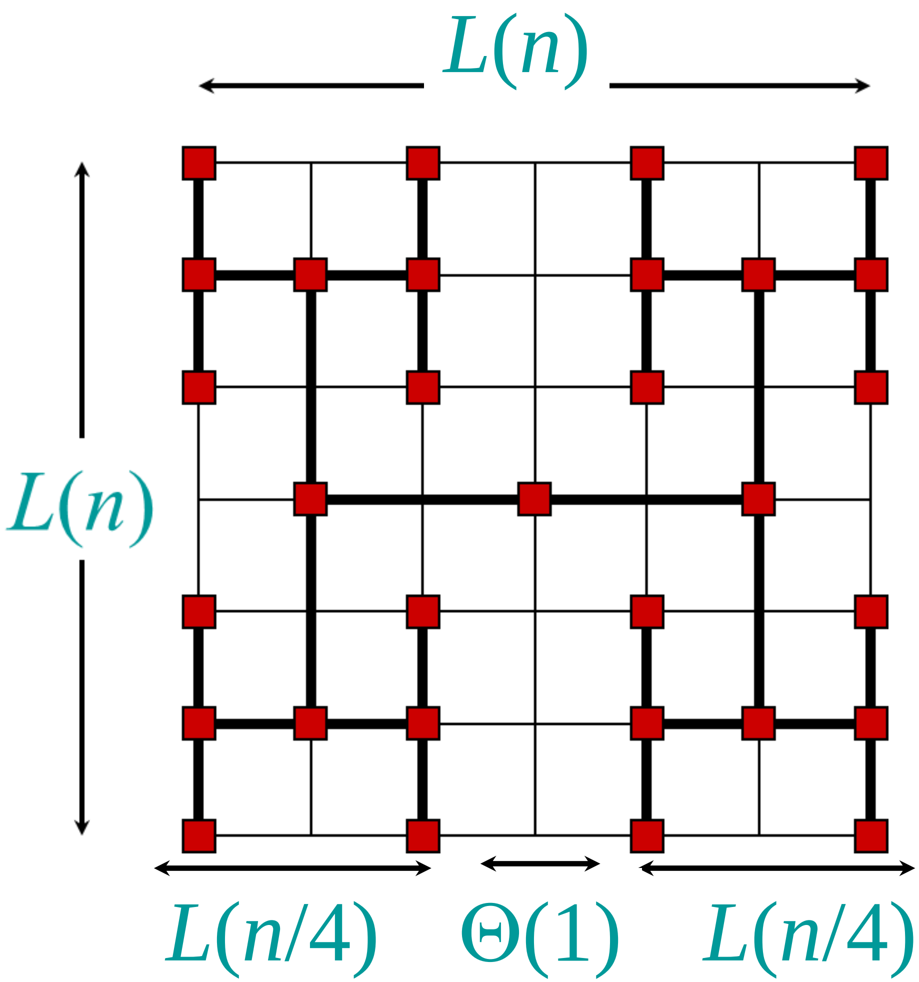
$W(n) = 2\,W(n/2) + \Theta(1)$
$= \Theta(n)$

Area = $\Theta(n \lg n)$

H-TREE EMBEDDING

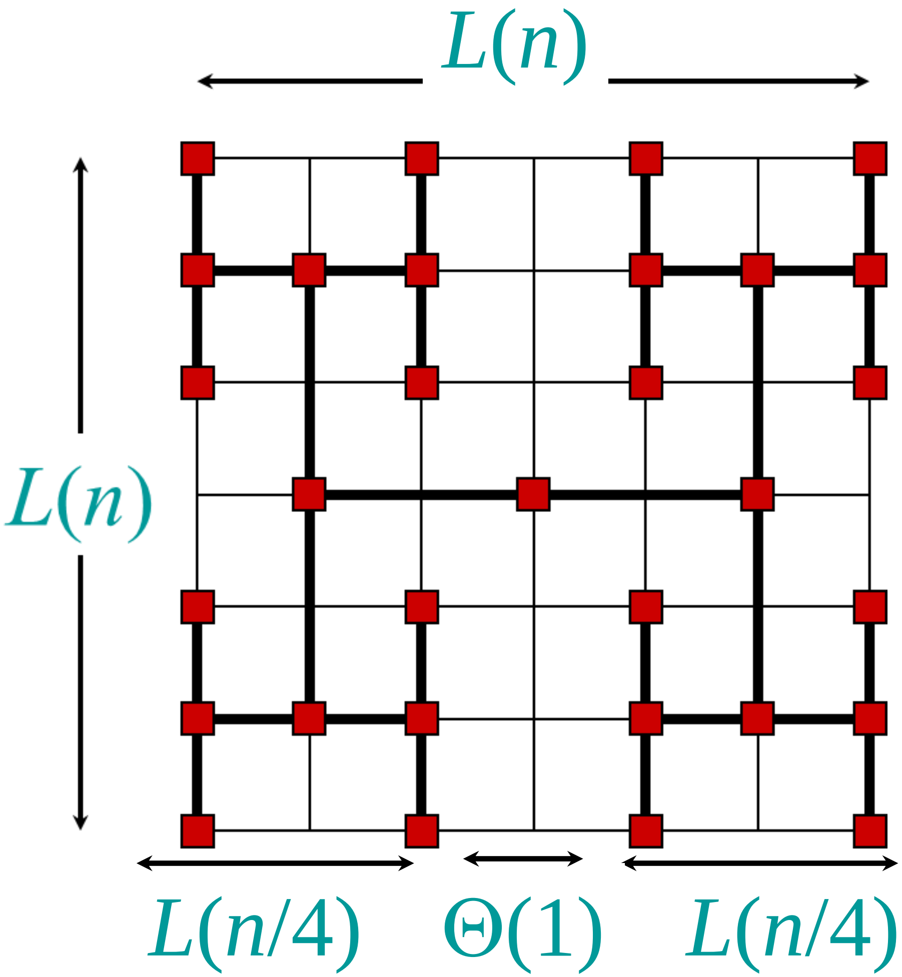


H-TREE EMBEDDING



basta che ragioni sulle porzioni dell'albero. ad esempio in orizzontale hai 1/4 dei nodi per entrambi i lati e al centro un solo nodo

H-TREE EMBEDDING



$$L(n) = 2L(n/4) + \Theta(1)$$
$$= \Theta(\sqrt{n})$$ (sto applicando il master theorem)

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

CONCLUSION

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- The divide-and-conquer strategy often leads to efficient algorithms.