Analysis and Design of Algorithms

CS3230

Week 3

Divide and Conquer

Diptarka Chakraborty

Wing-Kin Sung, Ken

Divide-and-conquer design paradigm

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

```
MERGE-SORT $A[1\,..\,n]$
    1. If $n = 1$, done.
    2. Recursively sort $A[\,1\,..\,\lceil n/2\rceil\,]$
       and $A[\,\lceil n/2\rceil+1\,..\,n\,]$ .
    3. “*Merge*” the 2 sorted lists.
```

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.

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

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

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

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

Design Paradigm

Divide, **conquer**, **combine**.
Consider Master Theorem recurrence

$$T(n) = \mathbf{a}T(\mathbf{n/b}) + \mathbf{f(n)}$$

Reduce #sub-problems

Reduce sub-problem size

Reduce time to divide and combine

Find an element in a sorted array

Divide-and-conquer solution

Find an element in a sorted array:

O(1) *1. Divide:* Check middle element.

2T(n/2) *2. Conquer:* Search in left subarray and right subarray.

O(1) *3. Combine:* Trivial.

- T(n) = 2 T(n/2) + 1
- $a=2$, $b=2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n^1$
- $1 \in O(n^1) \Rightarrow$ CASE 1
- $\Rightarrow$ $T(n) = \Theta(n)$.

This is the same as linear search!

Idea to improve

- $T(n) = \boxed{2}\, T(n/2) + 1$

Can we reduce 2 to 1?

Binary search

Find an element in a sorted array:

$O(1)$	1. ***Divide:*** Check middle element.
$T(n/2)$	2. ***Conquer:*** Recursively search 1 subarray.
$O(1)$	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

Powering a number

- Problem: Compute $f(n) = a^n$ for any integer n.

- Observation: $f(x+y) = f(x)*f(y)$.

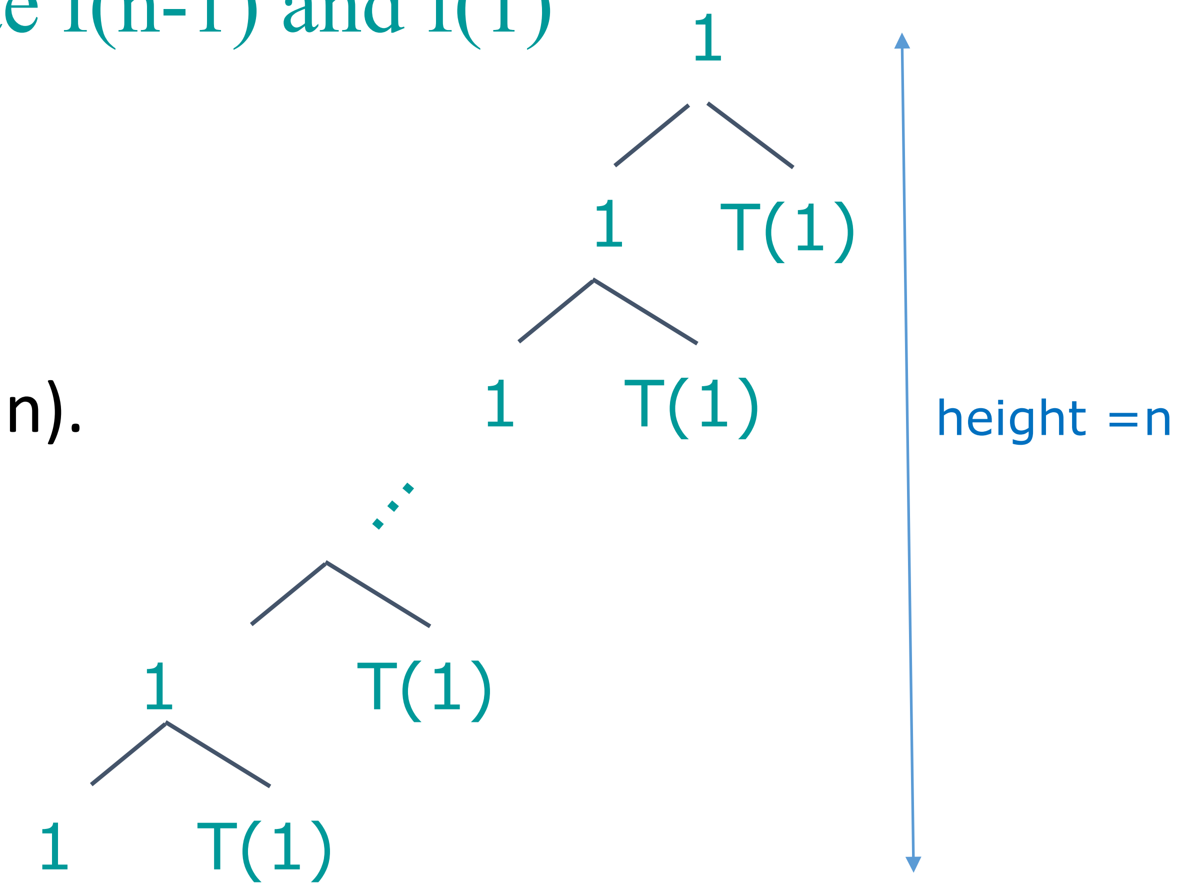
- Naïve solution:

1. Divide: Trivial.
2. Conquer: Recursively compute f(n-1) and f(1)
3. Combine: f(n-1)*f(1)

Running time of naïve solution

1. Divide: Trivial.
2. Conquer: Recursively compute f(n-1) and f(1)
3. Combine: f(n-1)*f(1)

- $T(n) = T(n-1) + T(1) + \Theta(1)$
- By recursion tree, we have $T(n) = \Theta(n)$.



Can we improve the algorithm?

- We can change the algorithm to:

1. Divide: Trivial.
2. Conquer: Recursively compute f(x) and f(n-x)
3. Combine: f(x)*f(n-x)

- Then, the running time is $T(n) = T(x) + T(n-x) + \Theta(1)$.
- We can show that $T(n) = \Theta(n)$ time. [Why?]
- We cannot improve!

Observation

- Previous method is slow since we need to recursively compute both f(x) and f(n-x).
- When x = n-x, we only need to recursively compute one value, which save the computational time.
- Let x=$\lfloor n/2 \rfloor$.
- When n is even, $f(n) = f(\lfloor n/2 \rfloor) * f(\lfloor n/2 \rfloor)$.
- When n is odd, $f(n) = f(1) * f(\lfloor n/2 \rfloor) * f(\lfloor n/2 \rfloor)$.

A better algorithm for powering a number

1. Divide: Trivial.
2. Conquer: Recursively compute $f(\lfloor n/2 \rfloor)$
3. Combine: $f(n) = f(\lfloor n/2 \rfloor)^2$ if n is even; $f(n) = f(1) * f(\lfloor n/2 \rfloor)^2$ if n is odd.

- T(n) = T(n/2) + Θ(1).
- By master theorem, we have T(n) = Θ(log n).

Computing Fibonacci number

Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

$$0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad \Lambda$$

Computing Fibonacci numbers

- **Bottom-up:**
- Compute F_0, F_1, F_2, …, F_n in order, forming each number by summing the two previous.
- Running time: $\Theta(n)$.

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

Computing Fibonacci numbers

- From CS1231, you learned a method to solve a second-order linear homogeneous recurrence.

$$F_n = F_{n-1} + F_{n-2}$$

- We can show that F_n has a closed form:

  $F_n = \frac{1}{\sqrt{5}}(\phi^n - (-\phi)^n)$ where $\phi = (1 + \sqrt{5})/2$.

A fast solution for computing Fibonacci numbers

- By the technique of powering a number, we compute ϕ^n and $(-\phi)^n$.
    - Takes O(log n) time.
- Then, $F_n = \frac{1}{\sqrt{5}}(\phi^n - (-\phi)^n)$ can be computed in O(1) time.

- This solution takes O(log n) time.
- However, this solution is not good since floating point arithmetic is prone to round-off errors.

Observation

- We can formula the computation of Fibonacci number as the multiplication of two matrices:
  - $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n + F_{n-1} & F_n \\ F_{n-1} + F_{n-2} & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$

- Hence, we have the following theorem:
  - $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$
  - Exercise: Show the correctness of this theorem by mathematical induction.

A better algorithm for computing Fibonacci number

- Let f(n) $= \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$.

1. Divide: Trivial.
2. Conquer: Recursively compute $f(\lfloor n/2 \rfloor)$
3. Combine: $f(n) = f(\lfloor n/2 \rfloor) * f(\lfloor n/2 \rfloor)$ if n is even;
   $f(n) = f(1) * f(\lfloor n/2 \rfloor) * f(\lfloor n/2 \rfloor)$ if n is odd.

- T(n) = T(n/2) + Θ(1).
- Hence, T(n) = Θ(log n).

Matrix multiplication

Matrix multiplication

$$\left.\begin{array}{ll}\textbf{Input:} & A = [a_{ij}],\, B = [b_{ij}]. \\ \textbf{Output:} & C = [c_{ij}] = A \cdot B. \end{array}\right\} \quad i, j = 1, 2, \ldots, n.$$

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

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

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

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

Divide-and-conquer algorithm

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

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

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

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

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

Example

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

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

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

- $\left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right] \left[\begin{array}{cc|cc} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ \hline 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{array}\right] = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$

- where $a = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, b = \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}, c = \begin{bmatrix} 9 & 10 \\ 13 & 14 \end{bmatrix}, d = \begin{bmatrix} 11 & 12 \\ 15 & 16 \end{bmatrix},$

  $e = \begin{bmatrix} 17 & 18 \\ 21 & 22 \end{bmatrix}, f = \begin{bmatrix} 19 & 20 \\ 23 & 24 \end{bmatrix}, g = \begin{bmatrix} 25 & 26 \\ 29 & 30 \end{bmatrix}, h = \begin{bmatrix} 27 & 28 \\ 31 & 32 \end{bmatrix}.$

Analysis of D&C algorithm

$$T(n) = \underbrace{8}_{\textit{\# submatrices}}\, T(\underbrace{n/2}_{\textit{submatrix size}}) + \underbrace{\Theta(n^2)}_{\textit{work adding submatrices}}$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \quad \begin{array}{l} 8 \text{ mults of } (n/2)\times(n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2)\times(n/2) \text{ submatrices} \end{array}$$

Analysis of D&C algorithm

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

# submatrices

submatrix size

work adding submatrices

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

No better than the ordinary algorithm.

Can we reduce # submatrices?

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.
- $P_1 = a \cdot (f - h)$
- $P_2 = (a + b) \cdot h$
- $P_3 = (c + d) \cdot e$
- $P_4 = d \cdot (g - e)$
- $P_5 = (a + d) \cdot (e + h)$
- $P_6 = (b - d) \cdot (g + h)$
- $P_7 = (a - c) \cdot (e + f)$

We can show that:

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

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

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

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!

Prove: $s = P_1 + P_2$

- LHS = s = $af + bh$
- RHS = $P_1 + P_2 = a \cdot (f - h) + (a + b) \cdot h$

  $= af - ah + ah + bh$

  $= af + bh =$ LHS

- For r, t, u, please give a proof by yourself.

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 the seven $(n/2)\times(n/2)$ submatrices.

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

Analysis of Strassen

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

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

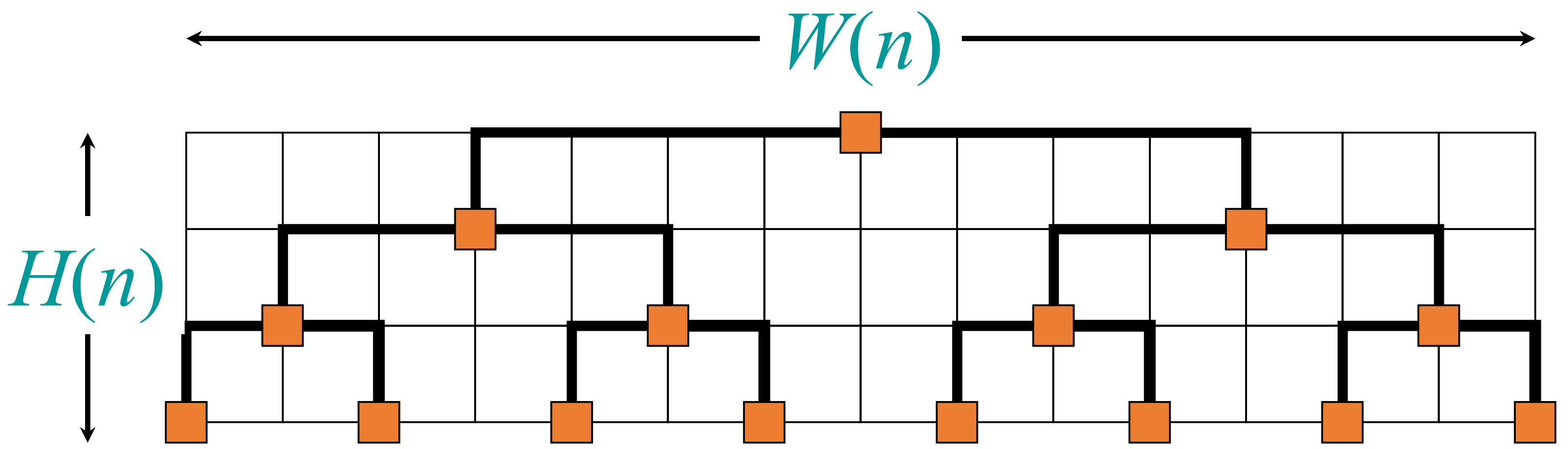
The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.373\cdots})$.

VLSI layout

VLSI layout

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

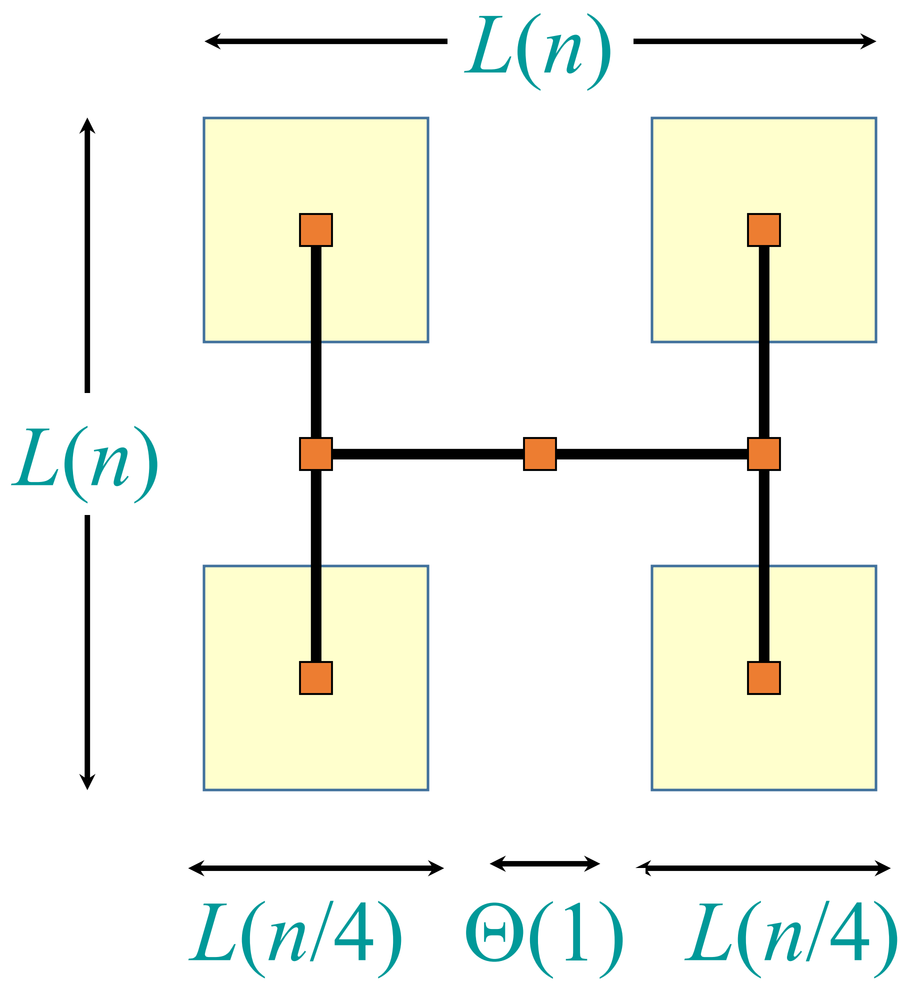


$$H(n) = H(n/2) + \Theta(1) = \Theta(\lg n)$$

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

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

H-tree embedding

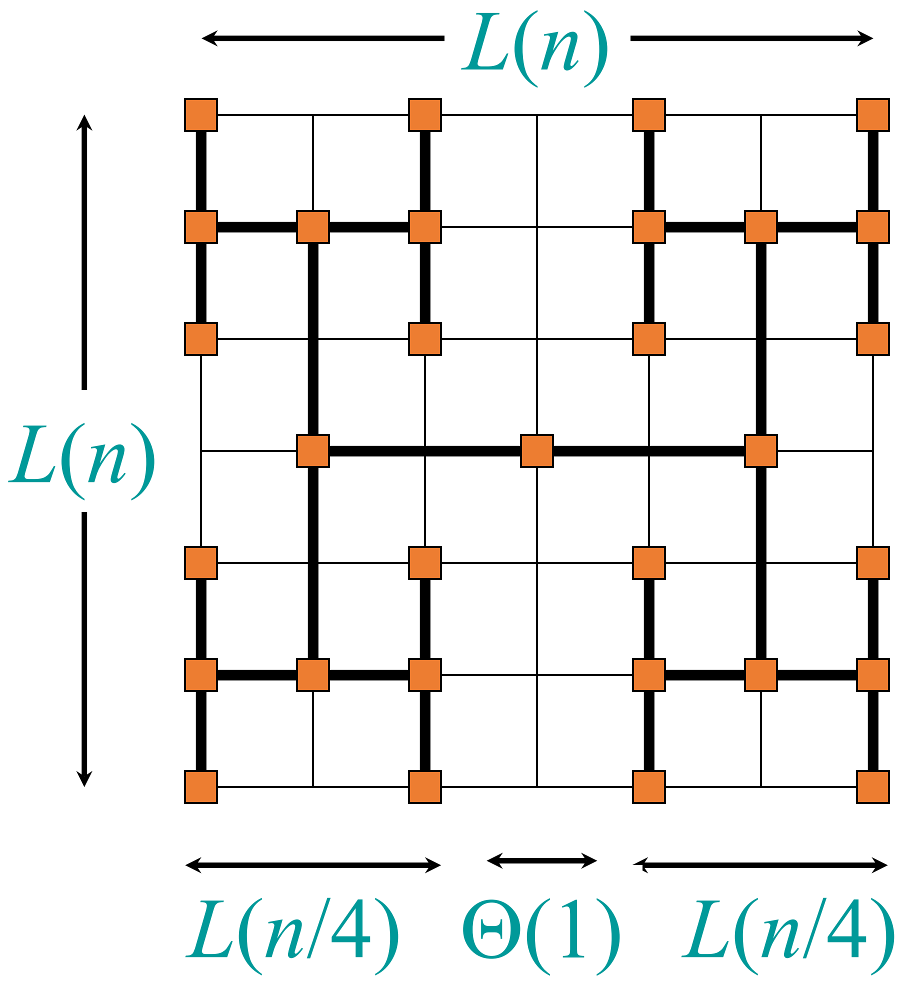


Instead of arranging the leaves in 1D, we arrange the leaves in 2D.

We use the H-tree to partition the n leaves into 4 subproblems.

Let L(n) be the length of the VLSI layout.

H-tree embedding



$$
\begin{aligned}
L(n) &= 2L(n/4) + \Theta(1) \\
&= \Theta(\sqrt{n})
\end{aligned}
$$

Area $= \Theta(n)$

Summary

Binary search: $T(n) = \Theta(\lg n)$

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

Powering, Fibonacci Num: $T(n) = \Theta(\lg n)$

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

Matrix Mult: $T(n) = \Theta(n^{\log(2)\ 7})$

$T(n) = 8T(n/2) + \Theta(n^2)$ $\Rightarrow$ $T(n) = 7T(n/2) + \Theta(n^2)$

VLSI Layout: $W(n) = \Theta(\sqrt{n})$

$W(n) = 2W(n/2) + \Theta(1)$ $\Rightarrow$ $W(n) = 2W(n/4)+ \Theta(1)$

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.

Acknowledgement

- The slides are modified from
  - the slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - the slides from Prof. Leong Hon Wai
  - the slides from Prof. Lee Wee Sun