Lec03 分治法

Divide and Conquer

1. divide the problem (more precisely, the instance of that problem) into subproblems
   should be smaller in some sens
2. conquer each subproblem recursively
3. combine those solutions into a solution for the whole problem

Example: Merge Sort

1. divide the array into two halves
2. conquer recursively sort each subarray
3. combine those solutions (merge two sorted arrays) in linear time

"size of subproblems"

running time: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

"number of subproblems" "extra work"

"in case 2" $= \Theta(n\log_2 n)$

Example: Binary Search // to find x in a sorted array

1. divide: compare x with the middle element in your array
2. conquer: recurse in one subarray
3. combine: trivial (do nothing)

$T(n) = T(\frac{n}{2}) + \Theta(1)$

$= \Theta(\log_2 n)$

Example: Powering a Number // given number x, integer $n \geq 0$, to compute x^n

naive algorithm: $\underbrace{x \cdot x \cdot \cdots \cdot x}_{n \text{ copies of } x \text{ totally}} = x^n$, $\Theta(n)$ time

divide-and-conquer algorithm:

$$x^n = \begin{cases} x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} & , \text{if } x \text{ is even} \\ x^{\frac{n-1}{2}} \cdot x^{\frac{n-1}{2}} \cdot x & , \text{if } x \text{ is odd} \end{cases}$$

$T(n) = T(\frac{n}{2}) + \Theta(1)$

$= T(\log_2 n)$

Example: Fibonacci Numbers // $F_n = \begin{cases} 0 & , \text{if } n=0 \\ 1 & , \text{if } n=1 \\ F_{n-1} + F_{n-2} & , \text{if } n \geq 2 \end{cases}$

naive algorithm: recursive algorithm

$T(n) = \Omega(\varphi^n)$, $\varphi = \frac{1+\sqrt{5}}{2}$

"exponential time" – BAD

"polynomial time – GOOD"

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

"you are solving two subproblem of almost the same size" bottom-up algorithm: (better、带记忆表的、cache)

"if you build up the recursion tree for Fibonacci of n, you will see that there are lots of common subtrees"

to compute $F_0, F_1, F_2, F_3, \cdots, F_{n-2}, F_{n-1}, F_n$

$T(n) = \Theta(n)$

naive recursive squaring (mathematical trick)

$F_n = \varphi^n / \sqrt{5}$ rounded to the nearest integer

$T(n) = \Theta(\log_2 n)$

recursive squaring

Theorem: $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n}_{\text{similar to "powering a number"}}$

$T(n) = \Theta(\log_2 n)$

proof (induction):

base $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1 = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix}$ ✓

step $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix}}_{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ ✓

Example: Matrix Multiplication

input: $A = [a_{ij}]_{n \times n}$, $B = [b_{ij}]_{n \times n}$

output: $C = [c_{ij}]_{n \times n} = AB$

standard algorithm: $\Theta(n^3)$

divide-and-conquer algorithm:

an idea: $n \times n$ matrix = 2×2 block matrix of $\frac{n}{2} \times \frac{n}{2}$ sub-metrices

$$\underset{C}{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}} = \underset{A}{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}} \cdot \underset{B}{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}$$

8 recursive multiplications of $\frac{n}{2} \times \frac{n}{2}$ sub-metrices, and 4 matrix-sum ($\Theta(n^2)$)

$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$

$= \Theta(n^3)$ "That kind of sucks!"

Strassen's algorithm:

the idea is that we have got to somehow reduce the number of multiplications (from 8 to 7)

$$P_1 = A_{11} \cdot (B_{12} - B_{22})$$
$$P_2 = (A_{11} + A_{12}) \cdot B_{22}$$
$$P_3 = (A_{21} + A_{22}) \cdot B_{11}$$
$$P_4 = A_{22} \cdot (B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$P_6 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$
$$P_7 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$
$$C_{12} = P_1 + P_2$$
$$C_{21} = P_3 + P_4$$
$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$T(n) = 7T(\tfrac{n}{2}) + \Theta(n^2)$$
$$= \Theta(n^{\log_2 7}) \qquad \log_2 7 \approx 2.81$$

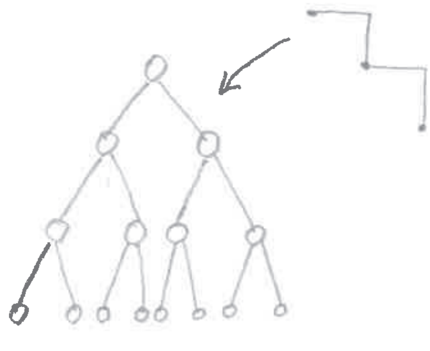
Strassen's algorithm is still not the best algorithm for matrix multiplication, the best $\underbrace{\text{so far}}_{2005}$ is like $n^{2.376}$, getting closer to n^2

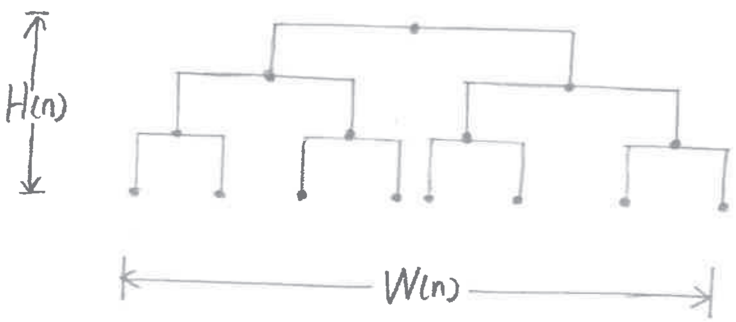
Example: VLSI layout (very large scale integration)

// embed a complete binary tree of n nodes,

// in a grid with minimum area

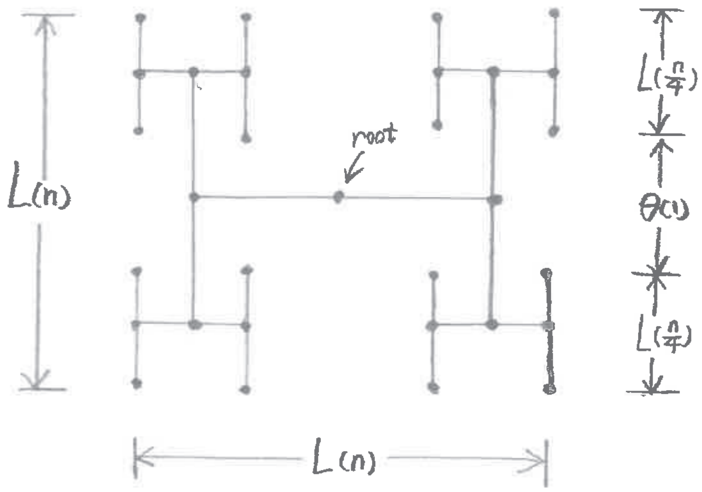
△ naive embedding:





$$H(n) = H(\tfrac{n}{2}) + \Theta(1) \qquad W(n) = 2W(\tfrac{n}{2}) + O(1)$$
$$= \Theta(\log_2 n) \qquad = \Theta(n)$$
$$\text{Area} = \Theta(n \log_2 n)$$

△ the H layout



Goal:

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

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

$\Rightarrow Area = \Theta(n)$

Then:

$\log_4 2 = \frac{1}{2}$

$\Rightarrow T(n) = 2T(\frac{n}{4}) + O(n^{\frac{1}{2}-\varepsilon})$

$L(n) = 2L(\frac{n}{4}) + \Theta(1)$

"case 1" $= \Theta(\sqrt{n})$