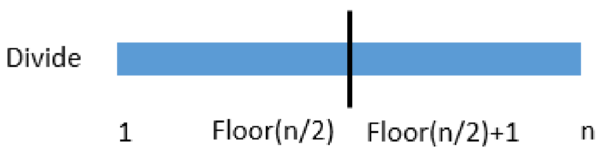
Lec3 Divide-and-Conquer 分治法

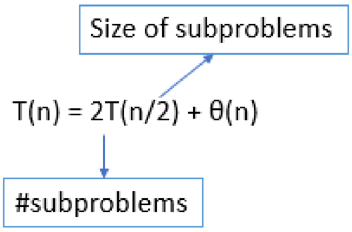
Divide and conquer paradigm(步骤)

1. **Divide** the problem into subproblems
2. **Conquer** each subproblem recursively
3. **Combine** those solutions for the whole problem

Merge sort

1. Divide an array into half
2. Conquer: recursively sort each subarray
3. Combine the solutions (process time = $\theta(n)$)





Running time: $T(n) = 2T(n/2) + \theta(n)$

利用主定理法，比较n和$n^{\log_b a} = n^{\log_2 2} = n$

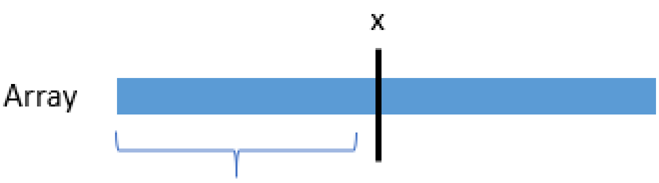
case 2: $k = 0, T(n) = \theta(n \log(n))$

Binary Search

Problem definition: Find x in a sorted array

Steps: Time T(n)

1. Divide: compare x with middle element in the array θ(1)
2. Conquer: recurse in one subarray T(n/2)
3. Combine: nothing



Running time: $T(n) = T(n/2) + \theta(1) = \theta(\log(n))$

Powering a number

Problem: Given a number x, integer $n \geq 0$, compute x^n 乘方问题

Sol1: Naive solution:

multiply x by n times, $x * x * x * .... x$

Running time = $\theta(n)$

Sol2: Divide-and-Conquer:

需要考虑奇偶情况

$$f(x) = \begin{cases} x^{n/2} * x^{n/2} & \text{if x is even} \\ x^{\frac{(n-1)}{2}} * x^{\frac{(n-1)}{2}} * x & \text{if x is odd} \end{cases}$$

Steps: Time T(n)

1. **Divide**: 分解x
2. **Conquer**: recursively solve $x^{n/2}$ or $x^{(n-1)/2}$ T(n/2)
3. **Combine**: multiple θ(1) (两次或一次乘法运算)

Running time: $T(n) = T(n/2) + \theta(1) = \theta(log(n))$

Fibonacci numbers

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n > 1 \end{cases}$$

Sol 1: Naive solution (recursive solution)

Recursively compute F_{n-1} and F_{n-2}, until $n = 1$ or $n = 0$

Disadvantage: get a branching tree, solve two subproblems of almost the same size, with each time decrease by one

Running time: $T(n) = \Omega(\phi^n), \phi = \frac{1+\sqrt{5}}{2}$ (phi = golden ratio) exponential time

Sol2: Bottom up solution

Computer $F_0, F_1, \ldots, F_n$ in order

Running time: $T(n) = \theta(n)$

Sol3: Naive recursive square (theorectically)

根据Fibonacci数列的一个特性：$F_n = \frac{\phi^n}{\sqrt{5}}$ rounded to nearest integer

Running time: $T(n) = log(n)$ 根据squaring 的时间复杂度

This method cannot work on a real machine, where uses floating numbers. Those numbers have a fixed amount of precise bits. When doing computations you might lose some important bits and cannot get a right answer.

Sol4: recursive squaring solution

Theorm :

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

通过计算矩阵的幂得到F_n

Running time: $T(n) = \theta(log(n))$ (takes constant time to do a two-by-two matrix multiplication)

Proof: by induction 归纳法

Base:

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

Steps:

$$\begin{aligned} \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} &= \begin{pmatrix} F_{n-1} + F_n & F_n \\ F_{n-1} + F_{n-2} & F_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} * \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \end{aligned}$$

Matrix multiplication

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

Output: $C = [c_{ij}] = A * B, c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

Sol1: Naive solution

根据公式c_{ij} =blabla计算，计算c_{ij}需要$2n - 1$步, θ(n) 时间

矩阵C里一共有n^2 项需要计算

Running time: $T(n) = \theta(n^3)$

Pseudo code:

```
for i = 1 to n
do for j = 1 to n
    cij = 0
    do for k = 1 to n
        do cij = cij + aik*bkj
```

Sol2: Divide-and-conquer

Idea: see $n * n$ matrix as a $2 * 2$ matrix of $n/2 * n/2$ sub matrices

$$\left[\begin{array}{c|c} C_1 & C_2 \\ \hline C_3 & C_4 \end{array}\right] = \left[\begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & A_4 \end{array}\right] * \left[\begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array}\right]$$
$$\text{where}$$
$$C_1 = A_1 * B_1 + A_2 * B_3$$
$$C_2 = A_1 * B_2 + A_2 * B_4$$
$$C_3 = A_3 * B_1 + A_4 * B_3$$
$$C_4 = A_3 * B_2 + A_4 * B_4$$
$$8 \text{ recursive multiplications of } n/2 * n/2 \text{ matrices}$$

Steps:

1. **Divide** each matrix into 4 blocks T(n/2)
2. **Conquer**: recursively divide matrix
3. **Combine**: matrix addition θ(n^2)

Running time: $T(n) = 8T(n/2) + \theta(n^2)$ 不是$T(n/4)$不是因为拆成4块 因为n是行or列

$n^{log_b a} = n^{log_2 8} = n^3$

$f(n) = o(n^3)$ => case 1: $T(n) = \theta(n^3)$

Sol3: Strassen's algorithm

Idea: addition is cheap, reduce #multiplications 8 => 7

$$P_1 = A_1 * (B_2 - B_4)$$
$$P_2 = (A_1 + A_2) * B_4$$
$$P_3 = (A_3 + A_4) * B_1$$
$$P_4 = A_4 * (B_3 - B_1)$$
$$P_5 = (A_1 + A_4) * (B_1 + B_4)$$
$$P_6 = (A_2 - A_4) * (B_3 + B_4)$$
$$P_7 = (A_1 - A_3) * (B_1 + B_2)$$
$$C_1 = P_5 + P_4 - P_2 + P_6$$
$$C_2 = P_1 + P_2$$
$$C_3 = P_3 + P_4$$
$$C_4 = P_5 + P_1 - P_3 - P_7$$

Steps:

1. **Divide** A and B, compute terms(P_i) for product θ(n^2)
2. Conquer: recursively compute each P_i
3. Combine products C_i into a full matrix θ(n^2)

Running time: $T(n) = 7T(n/2) + \theta(n^2)$

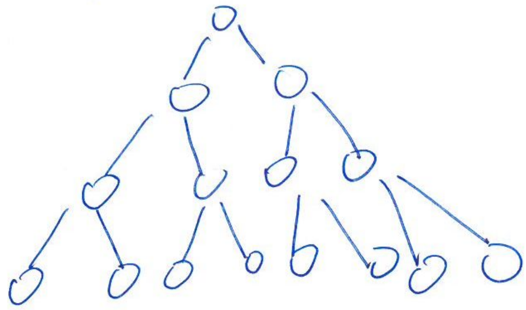
$n^{log_b a} = n^{log_2 7}$

$f(n) = o(n^{log_2 7})$ => case 1: $T(n) = \theta(n^{log_2 7})$

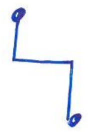
VLSI Layout

Problem: suppose a circuit is a complete binary tree, embed the circuit (with n leaves) on a grid. What is the minimum area for the grid?

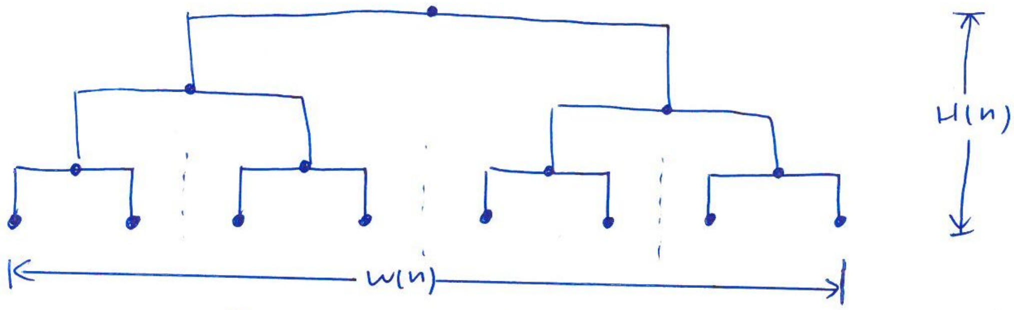
complete binary tree



$\Rightarrow$ want



Sol 1: naive embedding



$H(n) = H(\frac{n}{2}) + \theta(1) \quad = \theta(\lg n)$

$W(n) = 2W(\frac{n}{2}) + O(1) \quad = \theta(n)$

$S = H(n) \cdot W(n) = \theta(n \lg n)$

$\Rightarrow$ want to linear.

Goal: $W(n) = \theta(\sqrt{n}) \quad H(n) = \theta(\sqrt{n})$

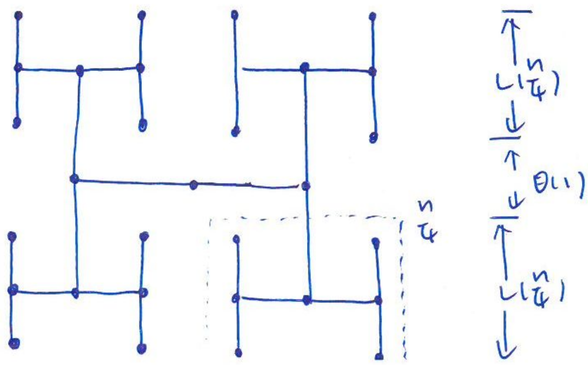
$S = \theta(n)$

sol 2:

从结果倒推

$n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}}$

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



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

$= \theta(\sqrt{n}) \rightarrow$ case 1