A stack of three colored rings (orange, green, pink) is placed on a wooden table. A thin wooden stick is inserted through the center of the rings. In the background, a red robot with a camera and sensors is visible on the table. The robot has a black camera mounted on top and several sensors or lights on its front. The scene is set in a laboratory or workshop environment.

Solving Recurrences

Lecture 22
CS2110 – Fall 2008

Prelim tonight!

Analysis of Merge-Sort

```
public static Comparable[] mergeSort(Comparable[] A, int low, int high) {
    if (low < high) { //at least 2 elements?           cost = c
        int mid = (low + high)/2;                       cost = d
        Comparable[] A1 = mergeSort(A, low, mid);       cost = T(n/2) + e
        Comparable[] A2 = mergeSort(A, mid+1, high);   cost = T(n/2) + f
        return merge(A1,A2);                           cost = gn + h
    }                                                    cost = i
    ....
}
```

Recurrence:

$$T(n) = c + d + e + f + 2T(n/2) + gn + h \quad \leftarrow \text{recurrence}$$

$$T(1) = i \quad \leftarrow \text{base case}$$

How do we solve this recurrence?

Analysis of Merge-Sort

Recurrence:

$$T(n) = c + d + e + f + 2T(n/2) + gn + h$$

$$T(1) = i$$

First, simplify by dropping lower-order terms

Simplified recurrence:

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

$$T(1) = d$$

How do we find the solution?

Solving Recurrences

- Unfortunately, solving recurrences is like solving differential equations
 - No general technique works for all recurrences
- Luckily, can get by with a few common patterns
- You will learn some more techniques in CS 2800

Analysis of Merge-Sort

- Recurrence for MergeSort

- $T(n) = 2T(n/2) + cn$
- $T(2) = 2c$

- Solution is $T(n) = O(n \log n)$

- Proof: strong induction on n

- Show that

$$T(2) \leq 2c$$

$$T(n) \leq 2T(n/2) + cn$$

imply

$$T(n) \leq cn \log n$$

- Basis

$$T(2) \leq 2c = c \cdot 2 \log 2$$

- Induction step

$$T(n) \leq 2T(n/2) + cn$$

$$\leq 2(cn/2 \log n/2) + cn \quad (\text{IH})$$

$$= cn (\log n - 1) + cn$$

$$= cn \log n$$

Solving Recurrences

- Recurrences are important when using divide & conquer to design an algorithm
- Solution techniques:
 - Can sometimes change variables to get a simpler recurrence
 - Make a guess, then prove the guess correct by induction
 - Build a recursion tree and use it to determine solution
 - Can use the *Master Method*
 - ◆ A “cookbook” scheme that handles many common recurrences

To solve $T(n) = aT(n/b) + f(n)$
compare $f(n)$ with $n^{\log_b a}$

- Solution is $T(n) = O(f(n))$
if $f(n)$ grows more rapidly
- Solution is $T(n) = O(n^{\log_b a})$
if $n^{\log_b a}$ grows more rapidly
- Solution is $T(n) = O(f(n) \log n)$
if both grow at same rate
- Not an exact statement of the theorem – $f(n)$ must be “well-behaved”

Recurrence Examples

- $T(n) = T(n - 1) + 1$ \rightarrow $T(n) = O(n)$ Linear Search
- $T(n) = T(n - 1) + n$ \rightarrow $T(n) = O(n^2)$ QuickSort worst-case
- $T(n) = T(n/2) + 1$ \rightarrow $T(n) = O(\log n)$ Binary Search
- $T(n) = T(n/2) + n$ \rightarrow $T(n) = O(n)$
- $T(n) = 2 T(n/2) + n$ \rightarrow $T(n) = O(n \log n)$ MergeSort
- $T(n) = 2 T(n - 1)$ \rightarrow $T(n) = O(2^n)$

	10	50	100	300	1000
$5n$	50	250	500	1500	5000
$n \log n$	33	282	665	2469	9966
n^2	100	2500	10,000	90,000	1,000,000
n^3	1000	125,000	1,000,000	27 million	1 billion
2^n	1024	a 16-digit number	a 31-digit number	a 91-digit number	a 302-digit number
$n!$	3.6 million	a 65-digit number	a 161-digit number	a 623-digit number	unimaginably large
n^n	10 billion	an 85-digit number	a 201-digit number	a 744-digit number	unimaginably large

- protons in the known universe ~ 126 digits
- μ sec since the big bang ~ 24 digits

- Source: D. Harel, *Algorithmics*

How long would it take @ 1 instruction / μsec ?

	10	20	50	100	300
n^2	1/10,000 sec	1/2500 sec	1/400 sec	1/100 sec	9/100 sec
n^5	1/10 sec	3.2 sec	5.2 min	2.8 hr	28.1 days
2^n	1/1000 sec	1 sec	35.7 yr	400 trillion centuries	a 75-digit number of centuries
n^n	2.8 hr	3.3 trillion years	a 70-digit number of centuries	a 185-digit number of centuries	a 728-digit number of centuries

- The big bang was 15 billion years ago ($5 \cdot 10^{17}$ secs)

- Source: D. Harel, *Algorithmics*

The Fibonacci Function

- Mathematical definition:

$$\text{fib}(0) = 0$$

$$\text{fib}(1) = 1$$

$$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2), \quad n \geq 2$$

- Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, ...

```
int fib(int n) {  
    if (n == 0 || n == 1) return n;  
    else return fib(n-1) + fib(n-2);  
}
```



Fibonacci (Leonardo
Pisano) 1170–1240?

Statue in Pisa, Italy
Giovanni Paganucci
1863

The Fibonacci Recurrence

```
int fib(int n) {  
    if (n == 0 || n == 1) return n;  
    else return fib(n-1) + fib(n-2);  
}
```

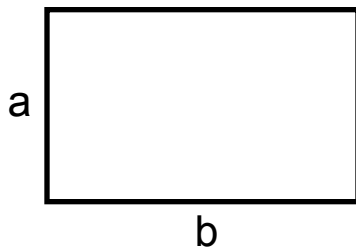
$$T(0) = c$$

$$T(1) = c$$

$$T(n) = T(n - 1) + T(n - 2) + c$$

- Solution is exponential in n
- But not quite $O(2^n)$...

The Golden Ratio



ratio of sum of sides (a+b)
to longer side (b)

=

ratio of longer side (b) to
shorter side (a)

$$\varphi = (a+b)/b = b/a$$

$$\varphi^2 = \varphi + 1$$

$$\varphi = \frac{1 + \sqrt{5}}{2}$$

$$= 1.618\dots$$

Fibonacci Recurrence is $O(\varphi^n)$

- want to show $T(n) \leq c\varphi^n$
- have $\varphi^2 = \varphi + 1$
- multiplying by $c\varphi^n$, $c\varphi^{n+2} = c\varphi^{n+1} + c\varphi^n$

- Basis:
 - $T(0) = c = c\varphi^0$
 - $T(1) = c \leq c\varphi^1$

- Induction step:
 - $T(n+2) = T(n+1) + T(n) \leq c\varphi^{n+1} + c\varphi^n = c\varphi^{n+2}$

Can We Do Better?

```
if (n <= 1) return n;
int parent = 0;
int current = 1;
for (int i = 2; i ≤ n; i++) {
    int next = current + parent;
    parent = current;
    current = next;
}
return (current);
```

- Number of times loop is executed? $n - 1$
- Number of basic steps per loop? **Constant**
- Complexity of iterative algorithm = $O(n)$
- Much, much, much, much, much, better than $O(\varphi^n)$!

...But We Can Do Even Better!

- Let f_n denote the n^{th} Fibonacci number

- $f_0 = 0$
- $f_1 = 1$
- $f_{n+2} = f_{n+1} + f_n, n \geq 0$

- Note that $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix} = \begin{pmatrix} f_{n+1} \\ f_{n+2} \end{pmatrix}$, thus $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} f_0 \\ f_1 \end{pmatrix} = \begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix}$

- Can compute the n th power of a matrix by repeated squaring in $O(\log n)$ time
- Gives complexity $O(\log n)$
- Just a little cleverness got us from exponential to logarithmic!

But We Are Not Done Yet...

- Would you believe constant time?

$$f_n = \frac{\varphi^n - \varphi'^n}{\sqrt{5}}$$

where $\varphi = \frac{1 + \sqrt{5}}{2}$ $\varphi' = \frac{1 - \sqrt{5}}{2}$

Matrix Multiplication in Less Than $O(n^3)$ (Strassen's Algorithm)

- Idea: naive 2 x 2 matrix multiplication takes 8 scalar multiplications, but we can do it in 7:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} s_1 + s_2 - s_4 + s_6 & s_4 + s_5 \\ s_6 + s_7 & s_2 - s_3 + s_5 - s_7 \end{pmatrix}$$

where

$$s_1 = (b - d)(g + h)$$

$$s_5 = a(f - h)$$

$$s_2 = (a + d)(e + h)$$

$$s_6 = d(g - e)$$

$$s_3 = (a - c)(e + f)$$

$$s_7 = e(c + d)$$

$$s_4 = h(a + b)$$

Now Apply This Recursively – Divide and Conquer!

- Break $2^{n+1} \times 2^{n+1}$ matrices up into 4 $2^n \times 2^n$ submatrices
- Multiply them the same way

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{pmatrix}$$

where

$$S_1 = (B - D)(G + H)$$

$$S_5 = A(F - H)$$

$$S_2 = (A + D)(E + H)$$

$$S_6 = D(G - E)$$

$$S_3 = (A - C)(E + F)$$

$$S_7 = E(C + D)$$

$$S_4 = H(A + B)$$

Now Apply This Recursively – Divide and Conquer!

- Gives recurrence $M(n) = 7 M(n/2) + cn^2$ for the number of multiplications
- Solution is $M(n) = O(n^{\log 7}) = O(n^{2.81\dots})$
- Number of additions is $O(n^2)$, bound separately

Is That the Best You Can Do?

- How about 3 x 3 for a base case?
 - best known is 23 multiplications
 - not good enough to beat Strassen
- In 1978, Victor Pan discovered how to multiply 70 x 70 matrices with 143640 multiplications, giving $O(n^{2.795\dots})$
- Best bound to date (obtained by entirely different methods) is $O(n^{2.376\dots})$ (Coppersmith & Winograd 1987)
- Best known lower bound is still $\Omega(n^2)$

Moral: Complexity Matters!

- But you are acquiring the best tools to deal with it!