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:

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

Powering a number

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

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

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

Fibonacci numbers

Recursive definition:

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

0 1 1 2 3 5 8 13 21 34 L

Fibonacci numbers

Recursive definition:

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

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

Naive recursive algorithm: $\Omega(\phi^n)$ (exponential time), where $\phi = (1+\sqrt{5})/2$ is the ***golden ratio***.

Computing Fibonacci numbers

Bottom-up:

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

Computing Fibonacci numbers

Bottom-up:

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

Naive recursive squaring:

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

- Recursive squaring: $\Theta(\lg n)$ time.
- This method is unreliable, since floating-point arithmetic is prone to round-off errors.

Recursive squaring

Theorem: $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$.

Recursive squaring

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

Algorithm: Recursive squaring.

Time = $\Theta(\lg n)$.

Recursive squaring

Theorem: $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$.

Algorithm: Recursive squaring.

Time = $\Theta(\lg n)$.

Proof of theorem. (Induction on n.)

Base ($n = 1$): $\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$.

Recursive squaring

Inductive step ($n \geq 2$):

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

$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n} \qquad \blacksquare$$

Thank You