CS 350 Algorithms and Complexity

Winter 2019

Lecture 4: Analyzing Recursive Algorithms

Andrew P. Black

Department of Computer Science

Portland State University

General Plan for Analysis of Recursive algorithms

- Decide on parameter *n* indicating input size
- Identify algorithm’s basic operation
- Determine worst, average, and best cases for input of size *n*
- Set up a recurrence relation, with initial condition, for the number of times the basic operation is executed
- Solve the recurrence, or at least ascertain the order of growth of the solution (see Levitin Appendix B)

Ex 2.4, Problem 1(a)

- Use a piece of paper and do this now, individually.
  - Solve this recurrence relation:

$$x(n) = x(n-1) + 5 \quad \text{for } n > 1$$

$$x(1) = 0$$

Individual Problem (Q1):

Solve the recurrence

$$x(n) = x(n-1) + 5 \qquad \text{for } n > 1$$

$$x(1) = 0$$

What's the answer?

Individual Problem (Q1):

Solve the recurrence

$$x(n) = x(n-1) + 5 \quad \text{for } n > 1$$

$$x(1) = 0$$

What's the answer?

A. $x(n) = n - 1$

B. $x(n) = 5n$

C. $x(n) = 5n - 5$

D. None of the above

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1:

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n$–$1)$:

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n$–$1)$: $$x(n) = x(n-2) + 5 + 5$$

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by $n-1$: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n-1)$: $$x(n) = x(n-2) + 5 + 5$$

substitute for $x(n-2)$:

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n$–$1)$: $$x(n) = x(n-2) + 5 + 5$$

substitute for $x(n$–$2)$: $$= x(n-3) + 5 + 5 + 5$$

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $x(n-1) = x(n-2) + 5$

substitute for $x(n$–$1)$: $x(n) = x(n-2) + 5 + 5$

substitute for $x(n$–$2)$: $= x(n-3) + 5 + 5 + 5$

generalize:

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n\text{–}1)$: $$x(n) = x(n-2) + 5 + 5$$

substitute for $x(n\text{–}2)$: $$= x(n-3) + 5 + 5 + 5$$

generalize: $$= x(n-i) + 5i \quad \forall i < n$$

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n$–$1)$: $$x(n) = x(n-2) + 5 + 5$$

substitute for $x(n$–$2)$: $$= x(n-3) + 5 + 5 + 5$$

generalize: $$= x(n-i) + 5i \quad \forall i < n$$

put i = $(n$–$1)$:

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by n–1: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n$–$1)$: $$x(n) = x(n-2) + 5 + 5$$

substitute for $x(n$–$2)$: $$= x(n-3) + 5 + 5 + 5$$

generalize: $$= x(n-i) + 5i \quad \forall i < n$$

put i = $(n$–$1)$: $$= x(n-(n-1)) + 5(n-1)$$

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by $n{-}1$: $$x(n-1) = x(n-2) + 5$$

substitute for $x(n{-}1)$: $$x(n) = x(n-2) + 5 + 5$$

substitute for $x(n{-}2)$: $$= x(n-3) + 5 + 5 + 5$$

generalize: $$= x(n-i) + 5i \quad \forall i < n$$

put i = $(n{-}1)$: $$= x(n-(n-1)) + 5(n-1)$$

substitute for $x(1)$:

My Solution

$$x(n) = x(n-1) + 5 \quad \text{for all } n > 1$$
$$x(1) = 0$$

replace n by $n–1$: $x(n-1) = x(n-2) + 5$

substitute for $x(n–1)$: $x(n) = x(n-2) + 5 + 5$

substitute for $x(n–2)$: $= x(n-3) + 5 + 5 + 5$

generalize: $= x(n-i) + 5i \quad \forall i < n$

put i = $(n–1)$: $= x(n-(n-1)) + 5(n-1)$

substitute for $x(1)$: $= 5(n-1)$

Ex 2.4, Problem 1(c)

- Use a piece of paper and do this now, individually.
  - Solve this recurrence relation:

$$x(n) = x(n-1) + n \quad \text{for } n > 0$$

$$x(0) = 0$$

Ex 2.4, Problem 1(c)

- Use a piece of paper and do this now, individually.
  - Solve this recurrence relation:

$$x(n) = x(n-1) + n \quad \text{for } n > 0$$

$$x(0) = 0$$

Answer?

A. $x(n) = n^2$

B. $x(n) = n^2/2$

C. $x(n) = n(n+1)/2$

D. None of the above

Ex 2.4, Problem 1(d)

- Use a piece of paper and do this now, individually.
  - Solve this recurrence relation for $n = 2^k$:

$$x(n) = x(n/2) + n \quad \text{for } n > 1$$
$$x(1) = 1$$

Ex 2.4, Problem 1(d)

- Use a piece of paper and do this now, individually.
  - Solve this recurrence relation for $n = 2^k$:

$$x(n) = x(n/2) + n \quad \text{for } n > 1$$
$$x(1) = 1$$

Answer?

A. $x(n) = 2^{n+1}$

B. $x(n) = 2n - 1$

C. $x(n) = n(n+1)$

D. None of the above

What does that mean?

What does that mean?

- "Solving a Recurrence relation" means:

What does that mean?

- "Solving a Recurrence relation" means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.

What does that mean?

- “Solving a Recurrence relation” means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
  - For example, for the relation

    $$x(n) \;=\; 3x(n-1) \;\; \text{for } n > 1,\;\; x(1) = 4$$

    the solution is

    $$x(n) \;=\; 4 \times 3^{n-1}$$

What does that mean?

- "Solving a Recurrence relation" means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
  - For example, for the relation

    $$x(n) \; = \; 3x(n-1) \;\; \text{for } n > 1, \;\; x(1) = 4$$

    the solution is

    $$x(n) \; = \; 4 \times 3^{n-1}$$
  - Check:

What does that mean?

- "Solving a Recurrence relation" means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
  - For example, for the relation

    $$x(n) \;=\; 3x(n-1) \;\text{ for } n > 1,\; x(1) = 4$$

    the solution is

    $$x(n) \;=\; 4 \times 3^{n-1}$$

  - Check:

    $$x(1) \;=\; 4 \times 3^0 \;= 4 \times 1 \;=\; 4$$

What does that mean?

- "Solving a Recurrence relation" means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
  - For example, for the relation

    $$x(n) \;=\; 3x(n-1) \;\; \text{for } n > 1, \;\; x(1) = 4$$

    the solution is

    $$x(n) \;=\; 4 \times 3^{n-1}$$

  - Check:

    $$x(1) \;=\; 4 \times 3^0 \;= 4 \times 1 \;=\; 4$$

    $$x(n) \;=\; 3x(n-1) \qquad \text{definition of recurrence}$$

What does that mean?

- "Solving a Recurrence relation" means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
  - For example, for the relation

    $$x(n) \;=\; 3x(n-1) \;\; \text{for } n > 1, \;\; x(1) = 4$$

    the solution is

    $$x(n) \;=\; 4 \times 3^{n-1}$$

  - Check:

    $$x(1) \;=\; 4 \times 3^0 \;= 4 \times 1 \;=\; 4$$

    $$\begin{aligned} x(n) &= 3x(n-1) && \text{definition of recurrence} \\ &= 3 \times [4 \times 3^{(n-1)-1}] && \text{substitute solution} \end{aligned}$$

What does that mean?

- "Solving a Recurrence relation" means:
  - find an explicit (non-recursive) formula that satisfies the relation and the initial condition.
  - For example, for the relation

    $$x(n) \;=\; 3x(n-1) \;\; \text{for } n > 1, \;\; x(1) = 4$$

    the solution is

    $$x(n) \;=\; 4 \times 3^{n-1}$$

  - Check:

    $$x(1) \;=\; 4 \times 3^0 \;= 4 \times 1 \;=\; 4$$

    $$\begin{aligned} x(n) &= 3x(n-1) && \text{definition of recurrence} \\ &= 3 \times \left[4 \times 3^{(n-1)-1}\right] && \text{substitute solution} \\ &= 4 \times 3^{n-1} \;=\; x(n) \end{aligned}$$

Ex 2.4, Problem 2

2. Set up and solve a recurrence relation for the number of calls made by $F(n)$, the recursive algorithm for computing $n!$.

$F(n) \stackrel{\text{def}}{=}$ **if** $n = 0$
then return 1
else return $F(n-1) \times n$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0$$
$$\textbf{then return } 1$$
$$\textbf{else return } F(n-1) \times n$$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \textbf{ then return } 1 \textbf{ else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

my solution

$F(n) \stackrel{\text{def}}{=}$ **if** $n = 0$
then return 1
else return $F(n-1) \times n$

Let $C(n)$ be the number of calls made in computing $F(n)$

$C(n) = C(n-1) + 1$

$C(0) = 1$ (when $n = 0$, there is 1 call)

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0$$
$$\textbf{then return } 1$$
$$\textbf{else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

$C(n) = C(n-1) + 1$

$C(0) = 1$ (when $n = 0$, there is 1 call)

$$\begin{aligned} C(n) &= C(n-1) \qquad + 1 \\ &= [C(n-2) + 1] + 1 \end{aligned}$$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0$$
$$\textbf{then return } 1$$
$$\textbf{else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

$C(n) = C(n-1) + 1$

$C(0) = 1$ (when $n = 0$, there is 1 call)

$$\begin{aligned} C(n) &= C(n-1) \qquad + 1 \\ &= [C(n-2) + 1] + 1 \\ &= C(n-2) + 2 \end{aligned}$$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \\ \textbf{then return } 1 \\ \textbf{else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

$$C(n) = C(n-1) + 1$$

$$C(0) = 1 \qquad \text{(when } n = 0\text{, there is 1 call)}$$

$$\begin{aligned} C(n) &= \ C(n-1) \qquad\quad + 1 \\ &= [C(n-2) + 1] + 1 \\ &= \ C(n-2) + 2 \\ &= \ C(n-i) + i \quad \forall i < n \qquad \text{(generalize)} \end{aligned}$$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \quad \textbf{then return } 1 \quad \textbf{else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

$C(n) = C(n-1) + 1$

$C(0) = 1$ (when $n = 0$, there is 1 call)

$$\begin{aligned}
C(n) &= C(n-1) + 1 \\
&= [C(n-2) + 1] + 1 \\
&= C(n-2) + 2 \\
&= C(n-i) + i \quad \forall i < n \qquad \text{(generalize)} \\
\text{Put } i = n\text{:} \quad &= C(0) + n
\end{aligned}$$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \quad \textbf{then return } 1 \quad \textbf{else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

$C(n) = C(n-1) + 1$

$C(0) = 1$ (when $n = 0$, there is 1 call)

$$\begin{aligned} C(n) &= C(n-1) + 1 \\ &= [C(n-2) + 1] + 1 \\ &= C(n-2) + 2 \\ &= C(n-i) + i \quad \forall i < n \qquad \text{(generalize)} \\ \text{Put } i = n: \quad &= C(0) + n \\ &= 1 + n \end{aligned}$$

my solution

$$F(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \quad \textbf{then return } 1 \quad \textbf{else return } F(n-1) \times n$$

Let $C(n)$ be the number of calls made in computing $F(n)$

$C(n) = C(n-1) + 1$

$C(0) = 1$ (when $n = 0$, there is 1 call)

$$\begin{aligned} C(n) &= C(n-1) \qquad\quad +1 \\ &= [C(n-2) + 1] + 1 \\ &= C(n-2) + 2 \\ &= C(n-i) + i \quad \forall i < n \qquad \text{(generalize)} \\ \text{Put } i = n\text{:} \qquad &= C(0) + n \\ &= 1 + n \end{aligned}$$

Now check!

Ex 2.4, Problem 3

3. Consider the following recursive algorithm for computing the sum of the first n cubes: $S(n) = 1^3 + 2^3 + ... + n^3$.

   Algorithm $S(n)$
   //Input: A positive integer n
   //Output: The sum of the first n cubes
   if $n = 1$ **return** 1
   else return $S(n-1) + n * n * n$

   a. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

Ex 2.4, Problem 3

3. Consider the following recursive algorithm for computing the sum of the first n cubes: $S(n) = 1^3 + 2^3 + ... + n^3$.

   Algorithm $S(n)$
   //Input: A positive integer n
   //Output: The sum of the first n cubes
   if $n = 1$ **return** 1
   else return $S(n-1) + n * n * n$

   a. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

   b. How does this algorithm compare with the straightforward nonrecursive algorithm for computing this function?

   $S \leftarrow 1$
   for $i \leftarrow 2$ **to** n **do**
   $\quad S \leftarrow S + i \times i \times i$
   return S

Ex 2.4, Problem 4(a)

Consider the following recursive algorithm.

Algorithm $Q(n)$
//Input: A positive integer n
if $n = 1$ **return** 1
else return $Q(n-1) + 2 * n - 1$

a. Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.

Ex 2.4, Problem 4(a)

Consider the following recursive algorithm.

Algorithm $Q(n)$
//Input: A positive integer n
if $n = 1$ **return** 1
else return $Q(n-1) + 2 * n - 1$

a. Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.

b. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.

Ex 2.4, Problem 4(a)

Consider the following recursive algorithm.

Algorithm $Q(n)$
//Input: A positive integer n
if $n = 1$ **return** 1
else return $Q(n-1) + 2 * n - 1$

a. Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.

b. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.

c. Set up a recurrence relation for the number of additions/subtractions made by this algorithm and solve it.

Ex 2.4, Problem 8

a. Design a recursive algorithm for computing 2^n for any nonnegative integer n that is based on the formula: $2^n = 2^{n-1} + 2^{n-1}$.

b. Set up a recurrence relation for the number of additions made by the algorithm and solve it.

c. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.

d. Is it a good algorithm for solving this problem?

Solution to Problem 8

a. **Algorithm** $Power(n)$
//Computes 2^n recursively by the formula $2^n = 2^{n-1} + 2^{n-1}$
//Input: A nonnegative integer n
//Output: Returns 2^n
if $n = 0$ **return** 1
else return $Power(n-1) + Power(n-1)$

We didn't simplify!

Solution to Problem 8

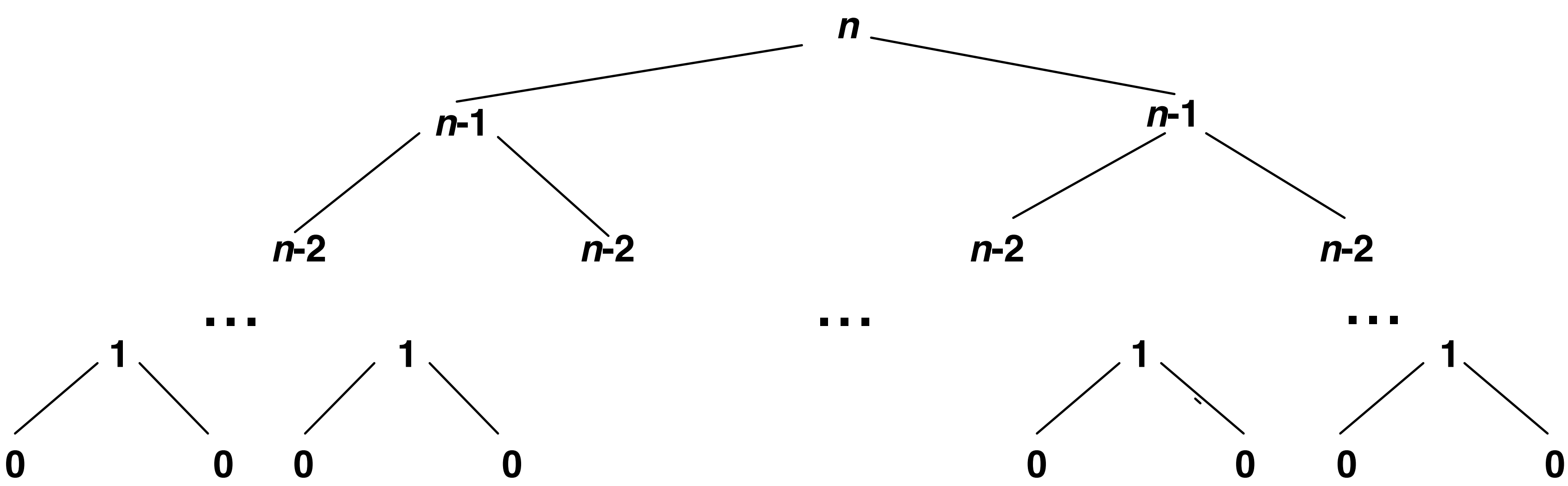
a. **Algorithm** $Power(n)$
//Computes 2^n recursively by the formula $2^n = 2^{n-1} + 2^{n-1}$
//Input: A nonnegative integer n
//Output: Returns 2^n
if $n = 0$ **return** 1
else return $Power(n-1) + Power(n-1)$

We didn't simplify!

b. $A(n) = 2A(n-1) + 1,\ \ A(0) = 0.$

$$
\begin{aligned}
A(n) &= 2A(n-1)+1 \\
&= 2[2A(n-2)+1]+1 = 2^2A(n-2)+2+1 \\
&= 2^2[2A(n-3)+1]+2+1 = 2^3A(n-3)+2^2+2+1 \\
&= \ldots \\
&= 2^iA(n-i)+2^{i-1}+2^{i-2}+\ldots+1 \\
&= \ldots \\
&= 2^nA(0)+2^{n-1}+2^{n-2}+\ldots+1 = 2^{n-1}+2^{n-2}+\ldots+1 = 2^n-1.
\end{aligned}
$$

Solution to Problem 8



Ex 2.4, Problem 11

The determinant of an n-by-n matrix

$$A = \begin{bmatrix} a_{11} & & a_{1n} \\ a_{21} & & a_{2n} \\ & & \\ a_{n1} & & a_{nn} \end{bmatrix},$$

denoted $\det A$, can be defined as a_{11} for $n = 1$ and, for $n > 1$, by the recursive formula

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j,$$

where s_j is +1 if j is odd and -1 if j is even, a_{1j} is the element in row 1 and column j, and A_j is the $(n-1)$-by-$(n-1)$ matrix obtained from matrix A by deleting its row 1 and column j.

a. Set up a recurrence relation for the number of multiplications made by the algorithm implementing this recursive definition. (Ignore multiplications by s_j.)

Ex 2.4, Problem 11

The determinant of an n-by-n matrix

$$A = \begin{bmatrix} a_{11} & & a_{1n} \\ a_{21} & & a_{2n} \\ & & \\ a_{n1} & & a_{nn} \end{bmatrix},$$

denoted $\det A$, can be defined as a_{11} for $n = 1$ and, for $n > 1$, by the recursive formula

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j,$$

where s_j is +1 if j is odd and -1 if j is even, a_{1j} is the element in row 1 and column j, and A_j is the $(n-1)$-by-$(n-1)$ matrix obtained from matrix A by deleting its row 1 and column j.

a. Set up a recurrence relation for the number of multiplications made by the algorithm implementing this recursive definition. (Ignore multiplications by s_j.)

b. Without solving the recurrence, what can you say about the solution's order of growth as compared to $n!$?

my solution

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j$$

my solution

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j$$

Let $M(n)$ be the number of multiplications made in computing the determinant of an $n \times n$ matrix.

$M(1) = 0$

$$M(n) = \sum_{j=1}^{n} (1 + M(n-1)) \qquad \forall n > 1$$

my solution

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j$$

Let $M(n)$ be the number of multiplications made in computing the determinant of an $n \times n$ matrix.

$M(1) = 0$

$$M(n) = \sum_{j=1}^{n} (1 + M(n-1)) \qquad \forall n > 1$$

$$M(n) = n \times (1 + M(n-1)) \qquad \forall n > 1$$

$$M(n) = n + nM(n-1)$$

my solution

$$\det A = \sum_{j=1}^{n} s_j a_{1j} \det A_j$$

Let $M(n)$ be the number of multiplications made in computing the determinant of an $n \times n$ matrix.

$M(1) = 0$

$$M(n) = \sum_{j=1}^{n} (1 + M(n-1)) \qquad \forall n > 1$$

$$M(n) = n \times (1 + M(n-1)) \qquad \forall n > 1$$

$$M(n) = n + nM(n-1)$$

Without solving this relation, what can you say about M's order of growth, compared to $n!$?

Problem

Problem 3, Levitin 2e §2.5

The maximum values of the Java primitive type **int** is $2^{31} - 1$. Find the smallest n for which the nth Fibonacci number is not going to fit in a variable of type **int**.

Recall Eqn 2.10:

$$Fib(n) \;=\; \frac{1}{\sqrt{5}}\phi^n \;\text{ rounded to the nearest integer}$$

where $\phi \;=\; \frac{1}{2}(1+\sqrt{5})$

Solution

Solution

- We need the smallest n s.t. $\text{Fib}(n) > 2^{31} - 1$

Solution

- We need the smallest n s.t. Fib(n) $> 2^{31} - 1$

$$\frac{1}{\sqrt{5}}\phi^n > 2^{31} - 1 \;\; \text{or} \;\; \phi^n > \sqrt{5}(2^{31} - 1).$$

Solution

- We need the smallest n s.t. $\mathrm{Fib}(n) > 2^{31} - 1$

$$\frac{1}{\sqrt{5}}\phi^n > 2^{31} - 1 \ \text{ or } \ \phi^n > \sqrt{5}(2^{31} - 1).$$

where $\phi = \frac{1}{2}(1 + \sqrt{5})$

Solution

- We need the smallest n s.t. Fib(n) $> 2^{31} - 1$

$$\frac{1}{\sqrt{5}}\phi^n > 2^{31} - 1 \ \text{ or } \ \phi^n > \sqrt{5}(2^{31} - 1).$$

where $\phi = \frac{1}{2}(1 + \sqrt{5})$

- Take natural logs of both sides:

Solution

- We need the smallest n s.t. $\mathrm{Fib}(n) > 2^{31} - 1$

$$\frac{1}{\sqrt{5}}\phi^n > 2^{31} - 1 \quad \text{or} \quad \phi^n > \sqrt{5}(2^{31} - 1).$$

where $\phi = \frac{1}{2}(1 + \sqrt{5})$

- Take natural logs of both sides:

$$n > \frac{\ln(\sqrt{5}(2^{31} - 1))}{\ln \phi}$$

Solution

- We need the smallest n s.t. Fib(n) $> 2^{31} - 1$

$$\frac{1}{\sqrt{5}}\phi^n > 2^{31} - 1 \quad \text{or} \quad \phi^n > \sqrt{5}(2^{31} - 1).$$

where $\phi = \frac{1}{2}(1 + \sqrt{5})$

- Take natural logs of both sides:

$$n > \frac{\ln(\sqrt{5}(2^{31} - 1))}{\ln \phi} \approx 46.3.$$

Solution

- We need the smallest n s.t. Fib(n) $> 2^{31} - 1$

$$\frac{1}{\sqrt{5}}\phi^n > 2^{31} - 1 \quad \text{or} \quad \phi^n > \sqrt{5}(2^{31} - 1).$$

where $\phi \;=\; \frac{1}{2}(1 + \sqrt{5})$

- Take natural logs of both sides:

$$n > \frac{\ln(\sqrt{5}(2^{31} - 1))}{\ln \phi} \approx 46.3.$$

- Thus, the answer is $n = 47$