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

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

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 \( x(n-1) \):
\[ x(n) = x(n - 2) + 5 + 5 \]

substitute for \( x(n-2) \):
\[ = x(n - 3) + 5 + 5 + 5 \]

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

\[ x(n) = 3x(n - 1) \quad \text{for } n > 1, \quad x(1) = 4 \]

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
    \[ x(n) = 3x(n - 1) \text{ for } n > 1, \quad x(1) = 4 \]
    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
  \[ x(n) = 3x(n - 1) \text{ for } n > 1, \quad x(1) = 4 \]
  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
    \[ x(n) = 3x(n - 1) \quad \text{for } n > 1, \quad x(1) = 4 \]
    the solution is
    \[ x(n) = 4 \times 3^{n-1} \]
  - Check:
    \[ x(1) = 4 \times 3^0 = 4 \times 1 = 4 \]
    \[ x(n) = 3x(n - 1) \quad \text{definition of recurrence} \]
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
    \[ x(n) = 3x(n - 1) \] for \( n > 1 \), \( x(1) = 4 \)
    the solution is
    \[ x(n) = 4 \times 3^{n-1} \]
  ▪ Check:
    \[ x(1) = 4 \times 3^0 = 4 \times 1 = 4 \]
    \[ x(n) = 3x(n - 1) \] definition of recurrence
    \[ = 3 \times [4 \times 3^{(n-1)-1}] \] substitute solution
“Solving a Recurrence relation” means:

- find an explicit (non-recursive) formula that satisfies the relation and the initial condition.

For example, for the relation

\[ x(n) = 3x(n - 1) \quad \text{for} \quad n > 1, \quad x(1) = 4 \]

the solution is

\[ x(n) = 4 \times 3^{n - 1} \]

Check:

\[ x(1) = 4 \times 3^0 = 4 \times 1 = 4 \]

\[ x(n) = 3x(n - 1) \quad \text{definition of recurrence} \]

\[ = 3 \times [4 \times 3^{(n - 1) - 1}] \quad \text{substitute solution} \]

\[ = 4 \times 3^{n - 1} = x(n) \]
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 $n!$.

$$F(n) \overset{\text{def}}{=} \text{if } n = 0 \text{ then return 1 }$$
$$\text{else return } F(n - 1) \times n$$
my solution

\[ F(n) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } n = 0 \\ F(n - 1) \times n & \text{else} \end{cases} \]
my solution

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

$$F(n) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } n = 0 \\ F(n-1) \times n & \text{else} \end{cases}$$
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)

$$F(n) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } n = 0 \\ F(n - 1) \times n & \text{else} \end{cases}$$
Let $C(n)$ be the number of calls made in computing $F(n)$

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

$C(0) = 1$  

(my solution)

$F(n) \overset{\text{def}}{=} \begin{cases} 
1 & \text{if } n = 0 \\
F(n - 1) \times n & \text{else return } F(n - 1) \times n 
\end{cases}$

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

$= [C(n - 2) + 1] + 1$

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

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)

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

$$= [C(n - 2) + 1] + 1$$

$$= C(n - 2) + 2$$

$F(n) \overset{\text{def}}{=} \text{if } n = 0$ \text{ then return } 1 \text{ else return } F(n - 1) \times n$
my solution

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

\[ C(n) = C(n - 1) + 1 \]

\[ C(0) = 1 \quad \text{(when } n = 0, \text{ there is 1 call)} \]

\[
C(n) = C(n - 1) + 1 \\
= [C(n - 2) + 1] + 1 \\
= C(n - 2) + 2 \\
= C(n - i) + i \quad \forall i < n \quad \text{(generalize)}
\]

\[
F(n) \overset{\text{def}}{=} \begin{cases} 
\text{if } n = 0 & \text{then return } 1 \\
\text{else return } F(n - 1) \times n 
\end{cases}
\]
Let $C(n)$ be the number of calls made in computing $F(n)$

\[
C(n) = C(n - 1) + 1 \\
C(0) = 1 \\
\text{(when } n = 0, \text{ there is 1 call)}
\]

\[
C(n) = C(n - 1) + 1 \\
= [C(n - 2) + 1] + 1 \\
= C(n - 2) + 2 \\
= C(n - i) + i \quad \forall i < n \quad \text{(generalize)}
\]

Put $i = n$:

\[
= C(0) + n
\]
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)

\[ C(n) = C(n - 1) + 1 \]
\[ = [C(n - 2) + 1] + 1 \]
\[ = C(n - 2) + 2 \]
\[ = C(n - i) + i \quad \forall i < n \quad \text{(generalize)} \]

Put $i = n$:

\[ = C(0) + n \]
\[ = 1 + n \]
my solution

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)

$F(n) \overset{\text{def}}{=} \begin{cases} 
1 & \text{if } n = 0 \\
F(n - 1) \times n & \text{else return}
\end{cases}$

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

$= [C(n - 2) + 1] + 1$

$= C(n - 2) + 2$

$= C(n - i) + i \quad \forall i < n$  

(generalize)

Put $i = n$:  

$= C(0) + n$

$= 1 + n$

Now check!
3. Consider the following recursive algorithm for computing the sum of the first \( n \) cubes: \( S(n) = 1^3 + 2^3 + \ldots + n^3 \).

**Algorithm** \( S(n) \)

//Input: A positive integer \( n \)
//Output: The sum of the first \( n \) cubes

if \( n = 1 \) return 1
else return \( S(n - 1) + n \times n \times n \)

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 $S(n - 1) + n \times n \times n$

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
    $S \leftarrow S + i \times i \times i$
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 $Q(n - 1) + 2 \times n - 1$

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 $Q(n - 1) + 2 \times n - 1$

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 $Q(n - 1) + 2 \times n - 1$

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)*

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

   $A(n) = 2A(n − 1) + 1$
   = 2[2A(n − 2) + 1] + 1 = 2^2 A(n − 2) + 2 + 1
   = 2^2[2A(n − 3) + 1] + 2 + 1 = 2^3 A(n − 3) + 2^2 + 2 + 1
   = ...
   = $2^n A(0) + 2^{n−1} + 2^{n-2} + ... + 1 = 2^{n-1} + 2^{n-2} + ... + 1 = 2^n − 1$.  

*We didn’t simplify!*
Solution to Problem 8

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

c. The tree of recursive calls for this algorithm looks as follows:

Note that it has one extra level compared to the similar tree for the Tower of Hanoi puzzle.

d. It's a very bad algorithm because it is vastly inferior to the algorithm that simply multiplies an accumulator by 2 \(n\) times, not to mention much more efficient algorithms discussed later in the book. Even if only additions are allowed, adding two \(2^{n-1}\) times is better than this algorithm.

8. a. The algorithm computes the value of the smallest element in a given array.

b. The recurrence for the number of key comparisons is

\[C(n) = C(n-1) + 1\] for \(n > 1\), \(C(1) = 0\).

Solving it by backward substitutions yields

\[C(n) = n - 1\].

9. a. The recurrence for the number of key comparisons is

\[C(n) = C\left(\lceil n/2 \rceil\right) + C\left(\lfloor n/2 \rfloor\right) + 1\] for \(n > 1\), \(C(1) = 0\).
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

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

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

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

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

\[ \text{det } A = \sum_{j=1}^{n} s_j a_{1j} \text{ det } A_j \]
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.

\[ M(1) = 0 \]

\[ M(n) = \sum_{j=1}^{n} (1 + M(n - 1)) \quad \forall n > 1 \]
Let $M(n)$ be the number of multiplications made in computing the determinant of an $n \times n$ matrix.

$M(1) = 0$

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

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

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

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

$M(1) = 0$

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

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

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

Without solving this relation, what can you say about $M$'s order of growth, compared to $n!$?
The maximum values of the Java primitive type \textbf{int} is $2^{31} - 1$. Find the smallest $n$ for which the $n$th Fibonacci number is not going to fit in a variable of type \textbf{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. $\text{Fib}(n) > 2^{31} - 1$

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

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

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

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

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

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

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

✧ Take natural logs of both sides:
Solution

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

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

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

✧ Take natural logs of both sides:

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

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

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

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. $\text{Fib}(n) > 2^{31} - 1$

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

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

✧ Take natural logs of both sides:

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

✧ Thus, the answer is $n = 47$