Recursion

- An algorithm is recursive if in the process of solving the problem, it calls itself one or more times.
- Example:

  **Algorithm power(a,n)**

  **Input:** non-negative integers a, n
  **Output:** a^n

  ```
  product ← 1
  for i = 1 to n do
    product ← product * a
  return product
  ```

  Challenge question: Can you compute a^n without using loops?

Iterative algorithms

- Definition: Algorithm where a problem is solved by iterating (going step-by-step) through a set of commands, often using loops.

  **Algorithm power(a,n)**

  **Input:** non-negative integers a, n
  **Output:** a^n

  ```
  product ← 1
  for i = 1 to n do
    product ← product * a
  return product
  ```

  Challenge question: Can you compute a^n without using loops?

Recursive binary search

**Algorithm binarySearch(array, start, stop, key)**

**Input:**
- A sorted array
- the region start...stop (inclusively) of indices to be searched
- the key to be found

**Output:** returns the index at which the key k has been found, or -1 if it is not in array[start...stop].

Simulating power(a,n)

When you call power(7,4), what happens?

1. power(7,4) calls power(7,3)
2. power(7,3) calls power(7,2)
3. power(7,2) calls power(7,1)
4. power(7,1) returns 7
5. power(7,2) returns 7*7 = 49
6. power(7,3) returns 7*49 = 343
7. power(7,4) returns 7*343 = 2401

Recursive binary search

**Algorithm binarySearch(array, start, stop, key)**

**Input:**
- A sorted array
- the region start...stop (inclusively) of indices to be searched
- the key to be found

**Output:** returns the index at which the key k has been found, or -1 if it is not in array[start...stop].
The Fibonacci sequence is a sequence of numbers where each number is the sum of the two that precede it:

\[
\begin{align*}
F(0) &= 0 \\
F(1) &= 1 \\
F(n) &= F(n-1) + F(n-2) \quad \text{if } n \geq 2
\end{align*}
\]

<table>
<thead>
<tr>
<th>n</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(n)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
<td>34</td>
<td>55</td>
<td>89</td>
</tr>
</tbody>
</table>

### Iterative Fibonacci

**Algorithm** \texttt{IterFib}(n)

- **Input**: an non-negative integer \( n \)
- **Output**: the \( n \)-th Fibonacci number

\[
\begin{align*}
\text{if } (n=0) & \text{ then return } 0; \\
\text{if } (n=1) & \text{ then return } 1; \\
\text{previous} & \leftarrow 0 \quad \text{// used to store previous Fib term} \\
\text{current} & \leftarrow 1 \quad \text{// used to store current Fib term} \\
\text{for } i = 2 \text{ to } n & \text{ do} \\
& \quad \text{tmpCurrent} \leftarrow \text{current} \\
& \quad \text{current} \leftarrow \text{current} + \text{previous} \\
& \quad \text{previous} \leftarrow \text{tmpCurrent} \\
\text{return} \text{ current}
\end{align*}
\]

### Recursive Fibonacci

**Algorithm** \texttt{Fib}(n)

- **Input**: an non-negative integer \( n \)
- **Output**: the \( n \)-th Fibonacci number

\[
\begin{align*}
\text{Fib}(5)
\end{align*}
\]

**Question**: When computing \( \text{Fib}(n) \), how many times are \( F(0) \) or \( F(1) \) called?