Recursion Trees
Recursion Trees

- Visualize call structure
Recursion Trees

• Visualize call structure

• Example: \texttt{fnc(8)}

```c
unsigned int fnc (unsigned int n) {
    ...
    return fnc(n/2) + fnc(n/2);
}
```
Fibonacci Tree Problem
Fibonacci - Recursion Tree

```cpp
// POST: return value is the n-th
//       Fibonacci number F(n)
ifmp::integer fib (const unsigned int n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n-1) + fib(n-2); // n > 1
}
```

`fib(4)`: 3
// POST: return value is the n-th Fibonacci number F(n)
ifmp::integer fib (const unsigned int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-1) + fib(n-2); // n > 1
}

fib(4):
Fibonacci - Recursion Tree

Fibonacci-number VS function calls

n=4:

fib(4):

```
      f(4)
     /   \
f(3)   f(2)
     /   /  \
   f(2) f(1)  f(2)
      /   /  \
  f(1) 1  f(1) 1  f(0)
      /   \
  1 0   
```
Fibonacci - Recursion Tree

Fibonacci-number vs function calls

n=4: 3

\textbf{fib}(4):

```
  f(4)
 /   \
/     \n\textbf{f}(3)
   /   \n  /     \n\textbf{f}(2) f(1) f(2)
 / /    /   /    \
/ /  f(1) f(1) f(0)
```

1 1 0
Fibonacci - Recursion Tree

Fibonacci-number VS function calls

n=4: 3 9

fib(4):

```
f(4)
  /   \
 /     \ 
/       \
|         |
f(3)------f(2)
  |        |
  |        |
f(2)------f(1)
  |        |
  |        |
f(1)------f(0)
```

1 0

1 1 0
Fibonacci - Recursion Tree

Fibonacci-number VS function calls

n=4: 3 vs 9

```
fib(4):

   f(4)
   /    /
  f(3)  f(1)
 /    |
f(2)  f(1)  f(1)
        |
  f(1)  1  f(0)  0
    |
  1   1
```

Many more function calls!
Fibonacci - Recursion Tree

Fibonacci number vs. function calls

\[
fib(4):
\]

Fibonacci growth:

\[
fib(n) \sim c^n
\]

(for \(n\) large, \(c\) golden ratio)

<table>
<thead>
<tr>
<th>(n)</th>
<th>(fib(n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
</tr>
<tr>
<td>20</td>
<td>6,765</td>
</tr>
<tr>
<td>40</td>
<td>102,334,155</td>
</tr>
<tr>
<td>80</td>
<td>23,416,728,348,467,685</td>
</tr>
</tbody>
</table>
Fibonacci - Recursion Tree

Fibonacci number VS function calls

\[ \text{fib}(n) \sim c^n \]
(for \( n \) large, \( c \) golden ratio)

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \text{fib}(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
</tr>
<tr>
<td>20</td>
<td>6,765</td>
</tr>
<tr>
<td>40</td>
<td>102,334,155</td>
</tr>
<tr>
<td>80</td>
<td>23,416,728,348,467,685</td>
</tr>
</tbody>
</table>

Requires unbelievably many recursive calls!
The Problem

• Problem: Same computation multiple times
The Problem

• Problem: Same computation multiple times
The Problem

• Problem:  Same computation multiple times
• Gets worse as $n$ increases  :-(

Diagram:

```
  f(4)
  /   \
/     \         
/       \        
/         \       
/           \      
/             \    
f(3)             f(2)
/   \
/     \
/       \        
/         \       
/           \      
/             \    
f(2)             f(1)
  /   \
  /     \
  /       \        
  /         \       
  /           \      
  /             \    
f(1)             f(1)
  /   \
  /     \
  /       \        
  /         \       
  /           \      
  /             \    
f(1)             f(0)
```
The Problem

- Not all recursive functions are this inefficient.
The Problem

• Not all recursive functions are this inefficient.

• Example:

```c
unsigned int fnc (unsigned int n) {
    ...
    return fnc(n/2) + fnc(n/2);
}
```

![Recursive function call tree](image)
The Problem

• Not all recursive functions are this inefficient.

Example:

```c
unsigned int fnc (unsigned int n) {
    ...  
    return fnc(n/2) + fnc(n/2);
}
```

2n - 1 recursive calls in total
The Problem

• Not all recursive functions are this inefficient.

Example:

```c
unsigned int fnc(unsigned int n)
{
    if (n <= 1)
        return 1;
    return fnc(n/2) + fnc(n/2);
}
```

2^n recursive calls in total

<table>
<thead>
<tr>
<th>n</th>
<th>fib</th>
<th>fnc</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&gt;5</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>&gt;55</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>&gt;6765</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>&gt;102'334'155</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>&gt;23'416'728'348'467'685</td>
<td>159</td>
</tr>
</tbody>
</table>
The Problem

- Not all recursive functions are this inefficient.

Example:

```cpp
20 unsigned int fnc(unsigned int n)
{
... 
return fnc(n/2) + fnc(n/2);
}
```

Number of Recursive Calls

<table>
<thead>
<tr>
<th>n</th>
<th>fib</th>
<th>fnc</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&gt; 5</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>&gt; 55</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>&gt; 6,765</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>&gt; 102,334,155</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>&gt; 23,416,728,348,467,685</td>
<td>159</td>
</tr>
</tbody>
</table>

Mindblowing difference!
The Problem

• Reason: \( n/2 \) falls much faster than \( n - 1 \) and \( n - 2 \)
  • \( n/2 \) \( \rightarrow \) sub-tree of height: \( \log_2(n) \)
  • \( n - 1, n - 2 \) \( \rightarrow \) sub-tree of height: \( n - 1, n - 2 \)