My demo application solves the Rod Cutting Problem in 3 ways, and it prints how much time it took each of the 3 ways.

**Business:** Selling metal rods.

<table>
<thead>
<tr>
<th>Table Length (inches)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit from selling</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

**Data:**

<table>
<thead>
<tr>
<th>1 rod</th>
</tr>
</thead>
</table>

**Prob:**

Rods are cast with length L = 36".

**Question:** What is the most profitable way to cut the 36" rod into pieces? Assumption: We can sell any number of rods of standard lengths and always earn the profit from each sale written in the table!

(Might not be true 😐)

Any model might not be accurate!
To solve the problem:
Use "great idea" Recursion:

\[
\text{sum: } f^\star
\]

What if \( L > 1 \)?

\[
\text{Recursions:}
\]

Divide and conquer.

Loop: for all first cuts (\( i^\star \) \( f = 1 \); \( f \leq L; f++ \))

\[
\text{profit} = P[i\star] +
\]

Recursion (\( L - f \));

of all these calculated profit values (there are \( L \) of them!)

return the MAX

```
inputted table - profits for single pieces

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
```

\[
\text{cut1=1, cut2=2, cut3=3}
\]

\[
\text{bestProfit}(2) = 4 + \text{bestProfit}(1) = 4 + 0 = 4
\]

\[
\text{bestProfit}(3) = 3 + \text{bestProfit}(1) = 3 + 0 = 3
\]

\[
\text{bestProfit}(2) < \text{bestProfit}(3)
\]

\[
\text{cut1=1, cut2=2}
\]

\[
\text{bestProfit}(1) = 1 + \text{bestProfit}(0) = 1 + 0 = 1
\]

\[
\text{bestProfit}(0)
\]

\[
1 + 4 + 3 + 3
\]

\[
2 = 3 - 1
\]

\[
1 = 3 - 2
\]

\[
f^\star = 3
\]
Worrying about Recursion

Choice 1: Don't do it!

Worry only here.

Don't worry!

about only the top level activation.

Trust these activations to correctly solve smaller problems.

- Draw the whole activation history tree: L = curvLength
- L = 0
- L = 1
- L = 2
- L = 3

Free

Only Time