Dynamic Programming for Rod Cutting
(excerpts)
Example: Rod Cutting (text)

- You are given a rod of length \( n \geq 0 \) (\( n \) in inches)
- A rod of length \( i \) inches will be sold for \( p_i \) dollars
- Cutting is free (simplifying assumption)
- **Problem**: given a table of prices \( p_i \) determine the maximum revenue \( r_n \) obtainable by cutting up the rod and selling the pieces.

<table>
<thead>
<tr>
<th>Length ( i )</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>
</tr>
</thead>
<tbody>
<tr>
<td>Price ( p_i )</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>
Example: Rod Cutting

Step 1: Characterizing an Optimal Solution

**Question:** in how many different ways can we cut a rod of length $n$?

For a rod of length 4:

For a rod of length $n$: $2^{n-1}$. **Exponential:** we cannot try all possibilities for $n$ "large". The obvious exhaustive approach won't work.
Example: Rod Cutting

Step 1: Characterizing an Optimal Solution

**Question:** in how many different ways can we cut a rod of length $n$?

**Proof Details:** a rod of length $n$ can have exactly $n-1$ possible cut positions – choose $0 \leq k \leq n-1$ actual cuts. We can choose the $k$ cuts (without repetition) anywhere we want, so that for each such $k$ the number of different choices is 

$$\binom{n-1}{k}$$

When we sum up over all possibilities ($k = 0$ to $k = n-1$):

$$\sum_{k=0}^{n-1} \binom{n-1}{k} = \sum_{k=0}^{n-1} \frac{(n-1)!}{k!(n-1-k)!} = (1+1)^{n-1} = 2^{n-1}.$$ 

For a rod of length $n$: $2^{n-1}$. 
Example: Rod Cutting

Characterizing an Optimal Solution

Let us find a way to solve the problem recursively (we might be able to modify the solution so that the maximum can be actually computed): assume we have cut a rod of length $n$ into $0 \leq k \leq n$ pieces of length $i_1, \ldots, i_k$,

\[ n = i_1 + \ldots + i_k, \]
with revenue

\[ r_n = p_{i_1} + \ldots + p_{i_k} \]

Assume further that this solution is optimal.

How can we construct it?

Advice: when you don’t know what to do next, start with a simple example and hope something will occur to you…
# Example: Rod Cutting

## Characterizing an Optimal Solution

<table>
<thead>
<tr>
<th>Length $i$</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>
</tr>
</thead>
<tbody>
<tr>
<td>Price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

We begin by constructing (by hand) the optimal solutions for $i = 1, \ldots, 10$:

- $r_1 = 1$ from sln. $1 = 1$ (no cuts)
- $r_2 = 5$ from sln. $2 = 2$ (no cuts)
- $r_3 = 8$ from sln. $3 = 3$ (no cuts)
- $r_4 = 10$ from sln. $4 = 2 + 2$
- $r_5 = 13$ from sln. $5 = 2 + 3$
- $r_6 = 17$ from sln. $6 = 6$ (no cuts)
- $r_7 = 18$ from sln. $7 = 1 + 6$ or $7 = 2 + 2 + 3$
- $r_8 = 22$ from sln. $8 = 2 + 6$
- $r_9 = 25$ from sln. $9 = 3 + 6$
- $r_{10} = 30$ from sln. $10 = 10$ (no cuts)
Example: Rod Cutting

Characterizing an Optimal Solution

Notice that in some cases \( r_n = p_n \), while in other cases the optimal revenue \( r_n \) is obtained by cutting the rod into smaller pieces.

In ALL cases we have the recursion

\[
r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)
\]

exhibiting optimal substructure (meaning?)

A slightly different way of stating the same recursion, which avoids repeating some computations, is

\[
r_n = \max_{1\leq i \leq n}(p_i + r_{n-i})
\]

And this latter relation can be implemented as a simple top-down recursive procedure:

```
function CUT-ROD(p, n)
    if n == 0
        return 0
    q = -\infty
    for i = 1 to n
        q = max(q, p[i] + CUT-ROD(p, n - i))
    return q
```
We can also notice that all the items we choose the maximum of are optimal in their own right: each substructure (max revenue for rods of lengths 1, …, \(n-1\)) is also optimal (again, optimal substructure property).

Nevertheless, we are still in trouble: computing the recursion leads to recomputing a number of values – how many?
Example: Rod Cutting

Characterizing an Optimal Solution

Let’s call Cut-Rod(p, 4), to see the effects on a simple case:

```
CUT-ROD(p, n)
1  if n == 0
2     return 0
3  q = -\infty
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

The number of nodes for a tree corresponding to a rod of size \( n \) is:

\[
T(0) = 1, \quad T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 2^n, \quad n \geq 1.
\]
We have a problem: “reasonable size” problems are not solvable in “reasonable time” (but, in this case, they are solvable in “reasonable space”).

Specifically:

• Note that navigating the whole tree requires $2^n$ stack-frame activations.
• Note also that no more than $n + 1$ stack-frames are active at any one time and that no more than $n + 1$ different values need to be computed or used.

Can we exploit these observations?

A standard solution method involves saving the values associated with each $T(j)$, so that we compute each value only once (called “memoizing” = writing yourself a memo).
We introduce two procedures:

\begin{verbatim}
MEMOIZED-CUT-ROD(p, n)
1    let r[0..n] be a new array
2    for i = 0 to n
3        r[i] = -\infty
4    return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
1    if r[n] \geq 0
2        return r[n]
3    if n == 0
4        q = 0
5    else q = -\infty
6        for i = 1 to n
7            q = \max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8    r[n] = q
9    return q
\end{verbatim}


We now remove some unnecessary complications:

\begin{verbatim}
BOTTOM-UP-CUT-ROD (p, n)
1   let r[0..n] be a new array
2   r[0] = 0
3   for j = 1 to n
4       q = -\infty
5       for i = 1 to j
6           q = max(q, p[i] + r[j - i])
7       r[j] = q
8   return r[n]
\end{verbatim}
Whether we solve the problem in a top-down or bottom-up manner the asymptotic time is $\Theta(n^2)$, the major difference being recursive calls as compared to loop iterations.

Why??