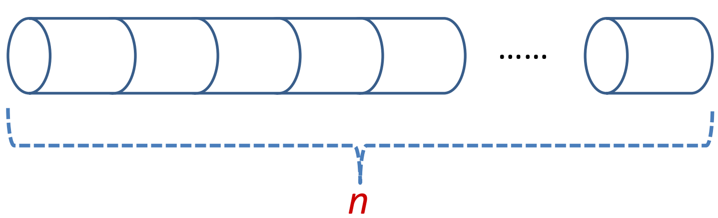
Rod Cutting - Example

The rod-cutting problem

- Given a rod of length *n* inches and a price table, determine the maximum revenue

price table *p*

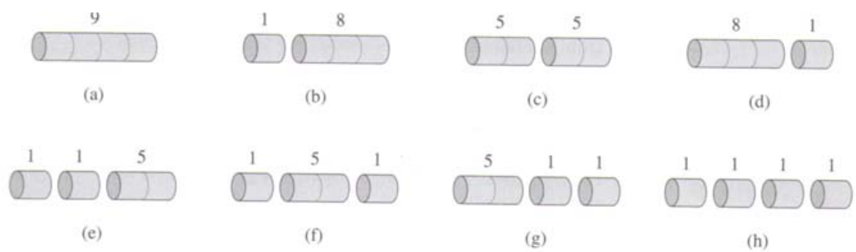
Length	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Price p_i	1	5	8	9	10	17	17	20	24	30



Rod Cutting

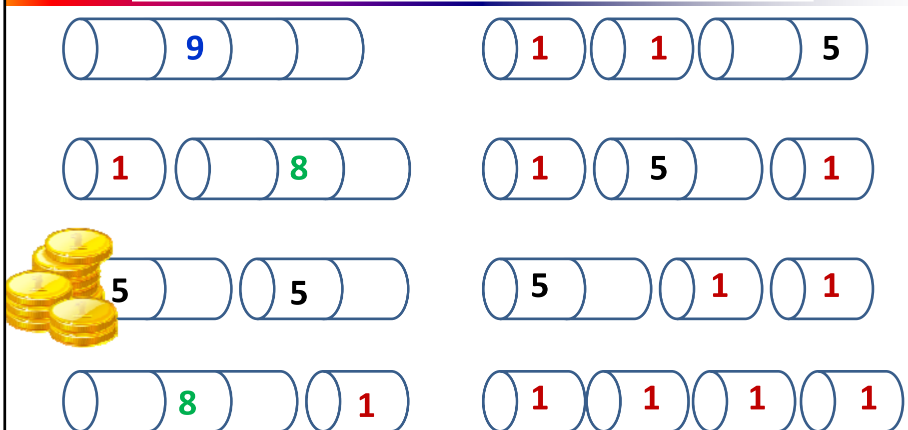
- The rod-cutting problem
  - We can cut up a rod of length 2^{n-1} different ways.

Length i	1	2	3	4	5	6	7	8	9	10
Price pi	1	5	8	9	10	17	17	20	24	30



Example: n=4

Which cut gives the maximal revenue?

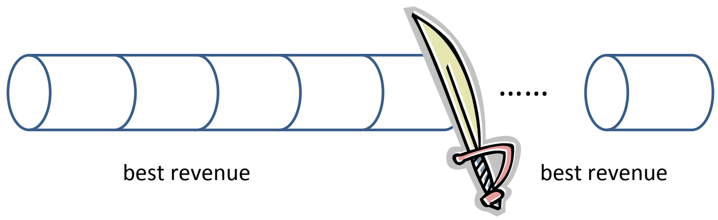


Length i	1	2	3	4
Price p_i	1	5	8	9

An optimal decomposition

- Cut into k pieces
  - $n = i_1 + i_2 + \ldots + i_k$
  - $r_n = p_{i1} + p_{i2} + \ldots + p_{ik}$

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



Maximum Revenue

- The maximum revenue
  - $n=i_1+i_2+\ldots i_k$
  - $r_n=p_{i1}+p_{i2}+\ldots p_{ik}$

$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)}, \\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)}, \\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)}, \\
r_4 &= 10 && \text{from solution } 4 = 2 + 2, \\
r_5 &= 13 && \text{from solution } 5 = 2 + 3, \\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)}, \\
r_7 &= 18 && \text{from solution } 7 = 1 + 6 \text{ or } 7 = 2 + 2 + 3, \\
r_8 &= 22 && \text{from solution } 8 = 2 + 6, \\
r_9 &= 25 && \text{from solution } 9 = 3 + 6, \\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}.
\end{aligned}
$$

Recursive Top-down Implementation

- General equation

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

- Simpler version

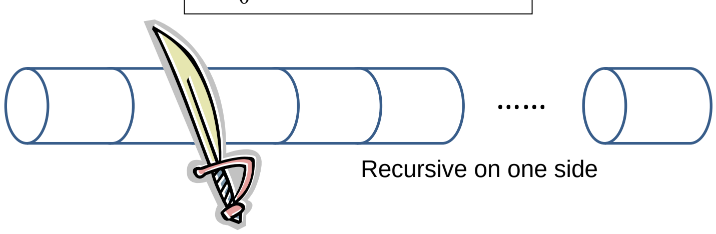
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

A simpler way

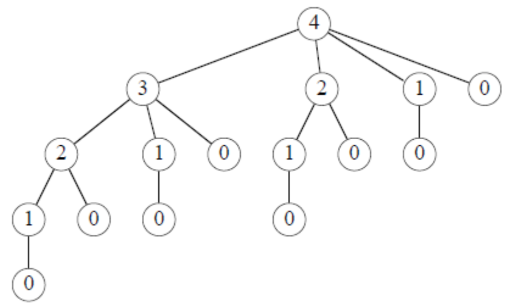
- Cut into a piece of length *i* and a remainder of length *n-i*
- Only the remainder may be further divided

$$\boxed{\begin{aligned} r_n &= \max_{1 \le i \le n}(p_i + r_{n-i}) \\ r_0 &= 0 \end{aligned}}$$



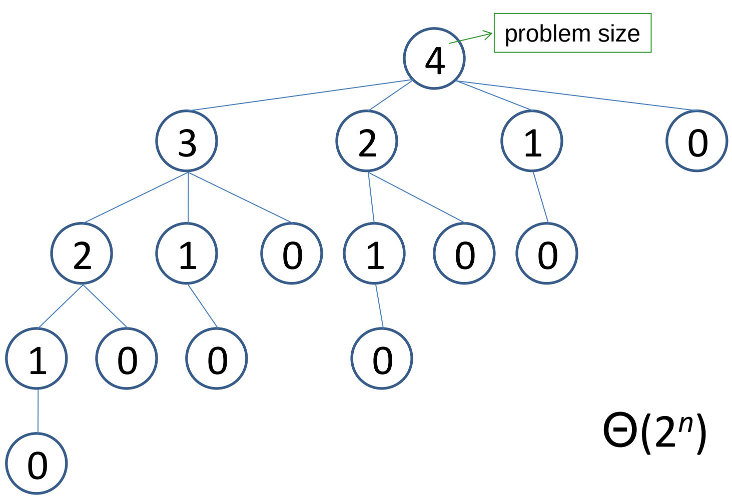
Time Complexity

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$



The result: $T(n) = 2^n$

CUT-ROD(p, 4)



$\Theta(2^n)$

Using Dynamic Programming

- Time-memory trade-off
- Two approaches
  - Top-down with memorization
  - Bottom-up method
    - Often has much better constant factors

CUT-ROD with Memorization

```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = −∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
r[n] = q
return q
```

Bottom-up Method

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

Time complexity: $\Theta(n^2)$

Bottom-up approach

Length	1	2	3	4
Price p_i	1	5	8	9

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$
$$r_0 = 0$$

- $r_0 = 0$
- $r_1 = p_1 + r_0 = 1$
- $r_2 = \max(p_1 + r_1, p_2) = 5$
- $r_3 = \max(p_1 + r_2, p_2 + r_1, p_3) = 8$
- $r_4 = \max(p_1 + r_3, p_2 + r_2, p_3 + r_1, p_4) = 10$
- …

The End