18 DP#1: Rod Cutting

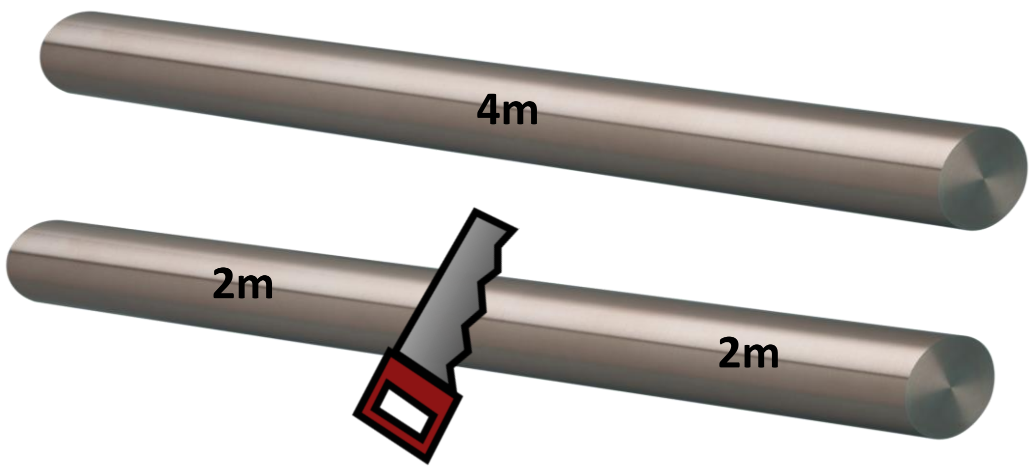
Textbook Chapter 15.1 – Rod Cutting

Rod Cutting Problem

- Input: a rod of length n and a table of prices p_i for $i = 1, \dots, n$

length i (m)	1	2	3	4	5
price p_i	1	5	8	9	10

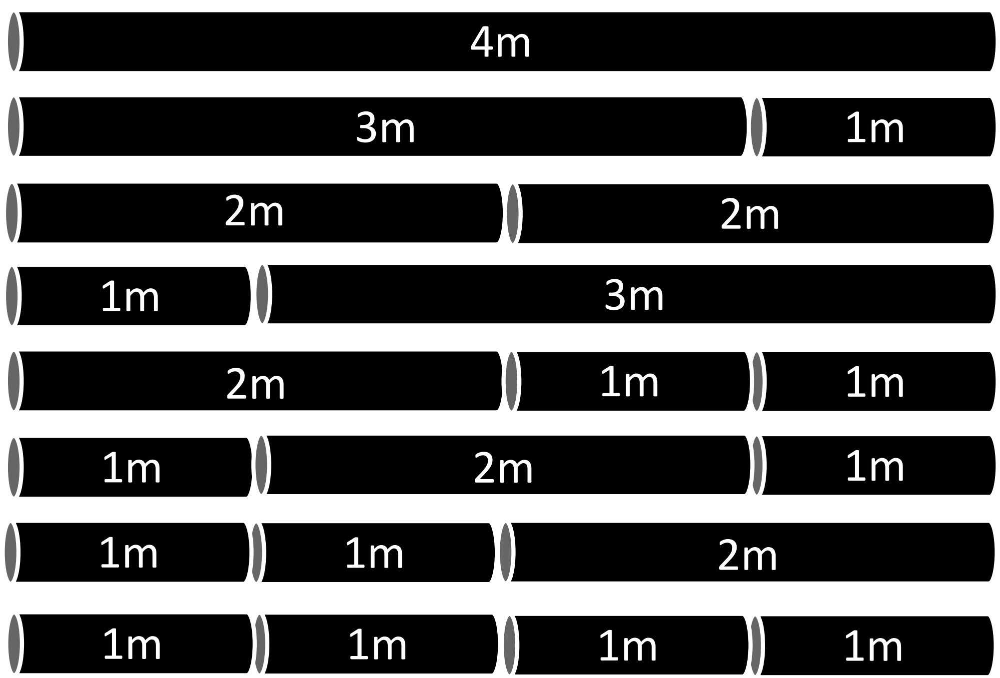
- Output: the maximum revenue r_n obtainable by cutting up the rod and selling the pieces



Brute-Force Algorithm

length i (m)	1	2	3	4	5
price p_i	1	5	8	9	10

- A rod with the length = 4



→ 9

→ 8 + 1 = 9

→ 5 + 5 = $\boxed{10}$

→ 1 + 8 = 9

→ 5 + 1 + 1 = 7

→ 1 + 5 + 1 = 7

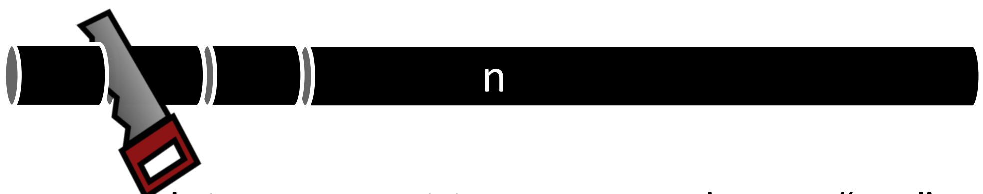
→ 1 + 1 + 5 = 7

→ 1 + 1 + 1 + 1 = 4

Brute-Force Algorithm

length i (m)	1	2	3	4	5
price p_i	1	5	8	9	10

- A rod with the length = n

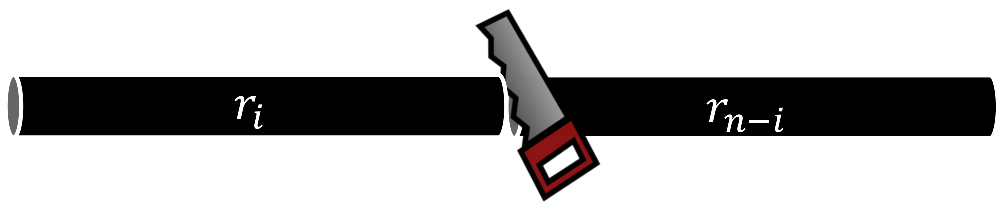


  - For each integer position, we can choose “cut” or “not cut”
  - There are $n -\ 1$ positions for consideration
- The total number of cutting results is $2^{n-1} = \Theta(2^{n-1})$

Recursive Thinking

r_n: the maximum revenue obtainable for a rod of length n

- We use a *recursive* function to solve the subproblems
- If we know the answer to the subproblem, can we get the answer to the original problem?



$$r_n = \max(\underbrace{p_n}_{\text{no cut}}, \underbrace{r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1}_{\text{cut at the i-th position (from left to right)}})$$

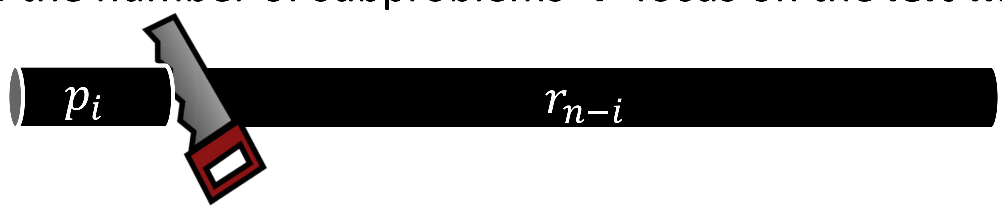
- Optimal substructure – an optimal solution can be constructed from optimal solutions to subproblems

Recursive Algorithms

- Version 1

$$r_n = \max(\underbrace{p_n}_{\text{no cut}}, \underbrace{r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1}_{\text{cut at the i-th position (from left to right)}})$$

- Version 2
  - try to reduce the number of subproblems → focus on the **left-most** cut



$$r_n = \max_{1 \le i \le n} (\underbrace{p_i}_{\text{left-most value}} + \underbrace{r_{n-i}}_{\text{maximum value obtainable from the remaining part}})$$

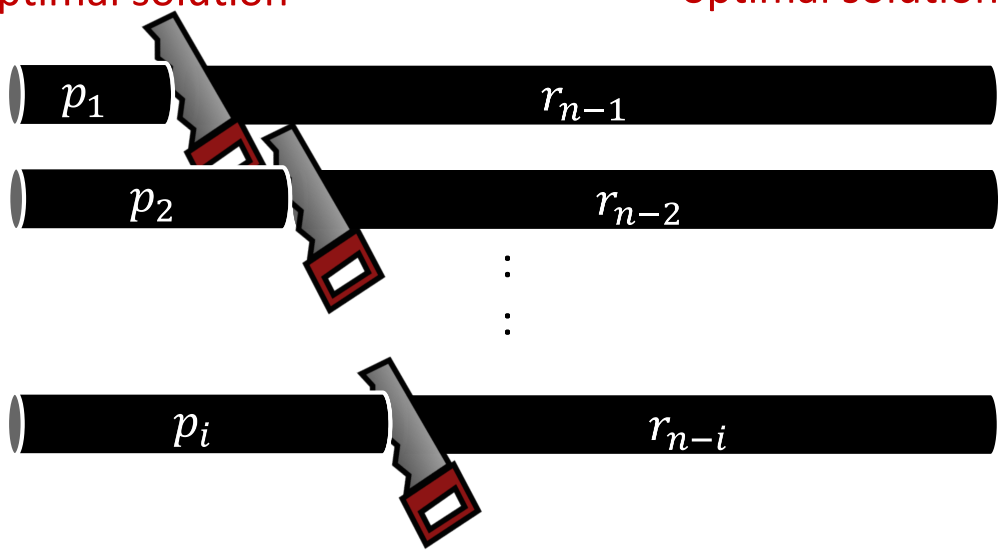
Recursive Procedure

- Focus on the left-most cut
  - assume that we always cut **from left to right** → the **first cut**

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

optimal solution

optimal solution to subproblems



Rod cutting problem has optimal substructure

Naïve Recursion Algorithm

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

```
Cut-Rod(p, n)
  // base case
  if n == 0
    return 0
  // recursive case
  q = -∞
  for i = 1 to n
    q = max(q, p[i] + Cut-Rod(p, n - i))
  return q
```

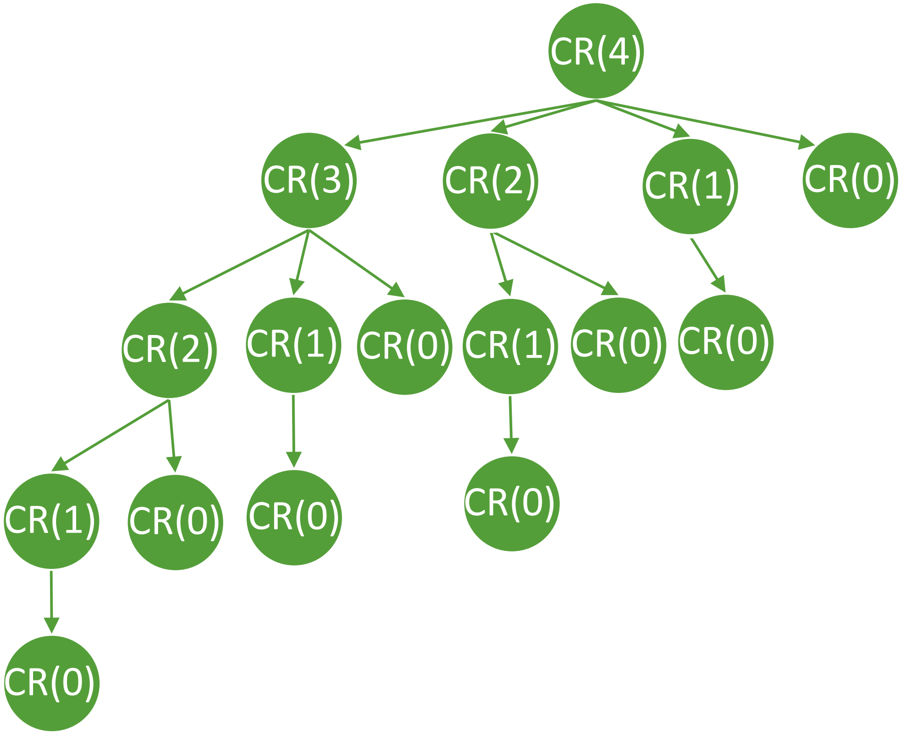
- $T(n)$ = time for running `Cut-Rod(p, n)`

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \Theta(1) + \sum_{i=0}^{n} T(n-i) & \text{if } n \ge 2 \end{cases} \Rightarrow T(n) = \Theta(2^n)$$

Naïve Recursion Algorithm

- Rod cutting problem

```
Cut-Rod(p, n)
  // base case
  if n == 0
    return 0
  // recursive case
  q = -∞
  for i = 1 to n
    q = max(q, p[i] + Cut-Rod(p, n - i))
  return q
```



Calling overlapping subproblems result in poor efficiency

Dynamic Programming

- Idea: use space for better time efficiency
- Rod cutting problem has **overlapping subproblems** and **optimal substructures** → can be solved by DP
- When the number of subproblems is polynomial, the time complexity is polynomial using DP
- DP algorithm
  - Top-down: solve overlapping subproblems recursively with memoization
  - Bottom-up: build up solutions to larger and larger subproblems

Dynamic Programming

- Top-Down with Memoization
  - Solve recursively and memo the subsolutions (跳著填表)
  - Suitable that **not all subproblems should be solved**

f(0)	f(1)	f(2)	…	f(n)

- Bottom-Up with Tabulation
  - Fill the table **from small to large**
  - Suitable that **each small problem should be solved**

f(0)	f(1)	f(2)	…	f(n)

Algorithm for Rod Cutting Problem

Top-Down with Memoization

```
Memoized-Cut-Rod(p, n)
  // initialize memo (an array r[] to keep max revenue)
  r[0] = 0
  for i = 1 to n
    r[i] = -∞ // r[i] = max revenue for rod with length=i
  return Memorized-Cut-Rod-Aux(p, n, r)

Memoized-Cut-Rod-Aux(p, n, r)
  if r[n] >= 0
    return r[n] // return the saved solution
  q = -∞
  for i = 1 to n
    q = max(q, p[i] + Memoized-Cut-Rod-Aux(p, n-i, r))
  r[n] = q // update memo
  return q
```

$\Theta(n)$

$\Theta(1)$

$\Theta(n^2)$

- $T(n)$ = time for running `Memoized-Cut-Rod(p, n)` ➡ $T(n) = \Theta(n^2)$

Algorithm for Rod Cutting Problem

Bottom-Up with Tabulation

```
Bottom-Up-Cut-Rod(p, n)
  r[0] = 0
  for j = 1 to n // compute r[1], r[2], ... in order
    q = -∞
    for i = 1 to j
      q = max(q, p[i] + r[j - i])
    r[j] = q
  return r[n]
```

$\Theta(n^2)$

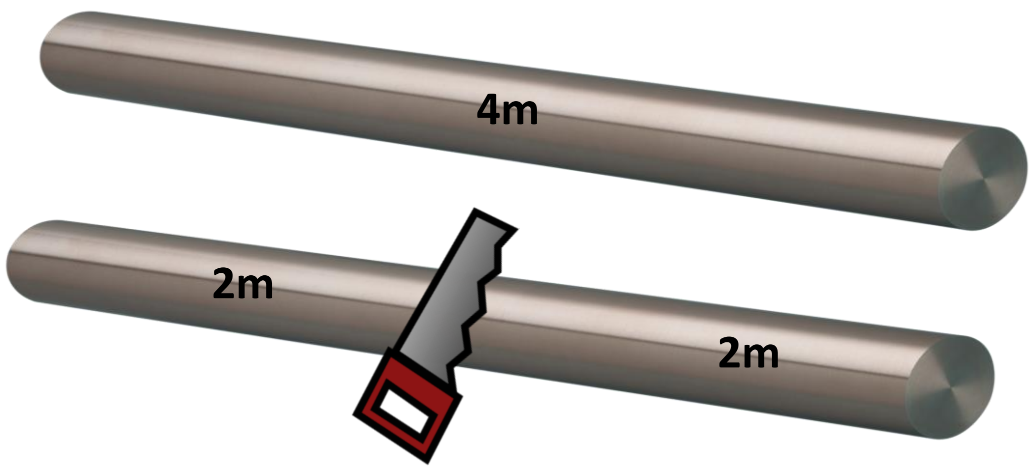
- $T(n)$ = time for running `Bottom-Up-Cut-Rod(p, n)` ➡ $T(n) = \Theta(n^2)$

Rod Cutting Problem

- Input: a rod of length n and a table of prices p_i for $i = 1, \dots, n$

length i (m)	1	2	3	4	5
price p_i	1	5	8	9	10

- Output: the maximum revenue r_n obtainable and **the list of cut pieces**



Algorithm for Rod Cutting Problem
Bottom-Up with Tabulation

- Add an array to keep the cutting positions **cut**

```
Extended-Bottom-Up-Cut-Rod(p, n)
  r[0] = 0
  for j = 1 to n //compute r[1], r[2], ... in order
  q = -∞
    for i = 1 to j
      if q < p[i] + r[j - i]
        q = p[i] + r[j - i]
        cut[j] = i // the best first cut for len j rod
    r[i] = q
  return r[n], cut
```

```
Print-Cut-Rod-Solution(p, n)
  (r, cut) = Extended-Bottom-up-Cut-Rod(p, n)
  while n > 0
    print cut[n]
    n = n - cut[n] // remove the first piece
```