Rod Cutting Problem

Rod Cutting Problem

- Some company buys long steel rods and cuts them into shorter rods, which it then sells. Each cut is free. The management of company wants to know the best way to cut up the rods.

- We assume that we know the price p_i in dollars (for i = 1,2,...n), that the company charges for a rod of length i inches. Rod lengths are always an integral number of inches.

Rod Cutting Problem

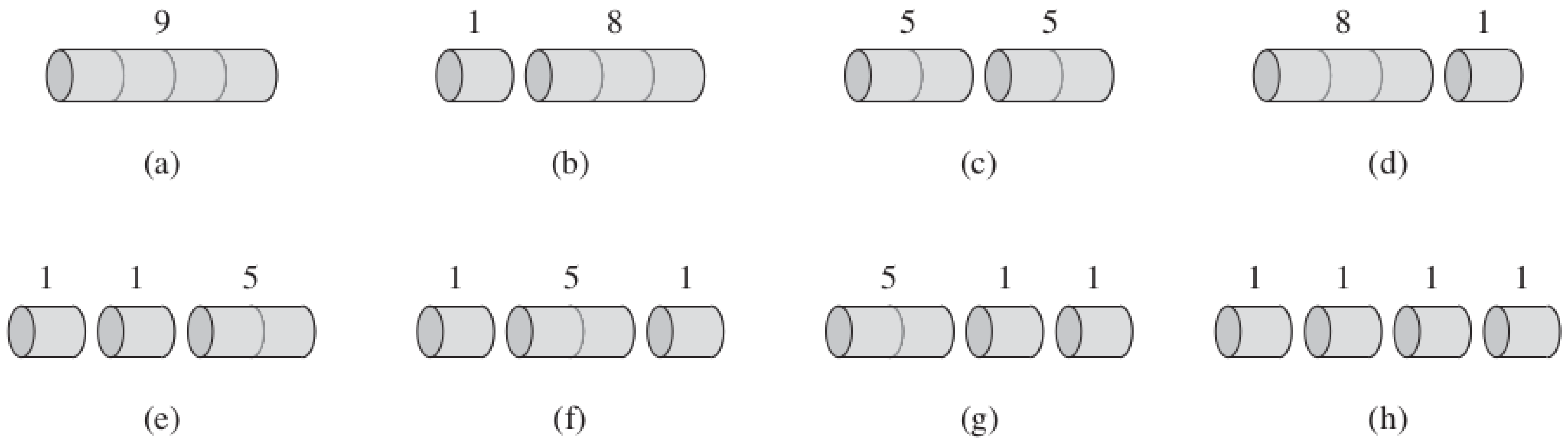
- **Input:** We are given a rod of length n and a table of prices p_i for i = 1…… n; p_i is the price of a rod of length i .

- **Goal:** to determine the maximum revenue r_n, obtainable by cutting up the rod and selling the pieces

Rod Cutting Problem (Example)

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

A sample price table for each size of rod



Above figure shows all the ways to cut up a rod of 4 inches in length, including the way with no cuts at all.

All possible solutions

Which one provides the maximum revenue r(4)?

- 4 rods of size 1 (achieved by 3 cuts) = 4 x p(1) = 4 x 1 = **4**
- 2 rods of size 1 + 1 rod of size 2 (achieved by 2 cuts) = 2 x p(1) + 1 x p(2)
  = 2 x 1 + 5 = **7**
- 2 rods of size 2 (achieved by 1 cut)= 2 x p(2) = 2 x 5 = **10**
- 1 rod of size 1 + 1 rod of size 3 (achieved by 2 cuts)= 1 x p(1) + 1 x p(3)
  = 1 + 8= **9**
- original rod of size 4 (achieved by no cuts)= 1 x p(4) = **9**

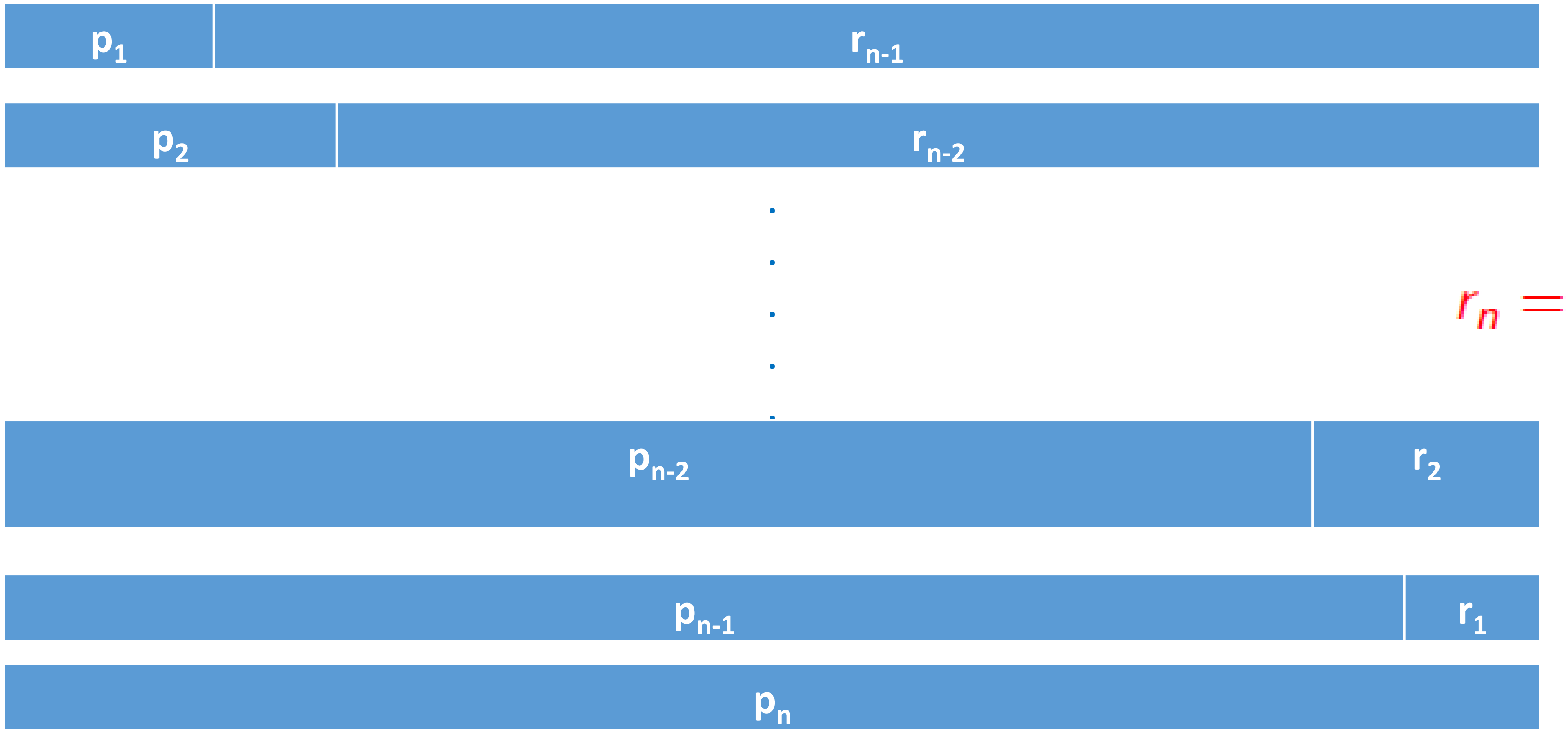
Optimal solution

- We see that cutting a 4-inch rod into two 2-inch pieces produces revenue $p_2 + p_2 = 5 + 5 = 10$, which is optimal.

Brute Force Solution to Rod Cutting Problem

Recursive Solution for Rod cutting problem

- Suppose r_n denotes optimal solution for rod of length n and p_i denotes price of rod of length i
- For a rod of length n, there are n-1 ways to make the first cut as follows:



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

Recursive Solution for Rod cutting problem

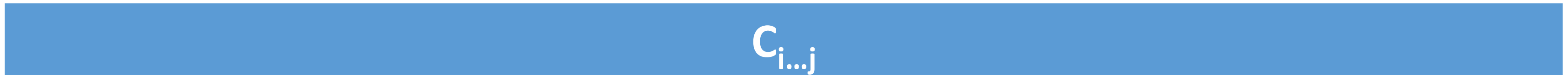
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i}).$$

- Cut a piece of length i , with remainder of length n
- Only the remainder, and not the rest piece, may be further divided

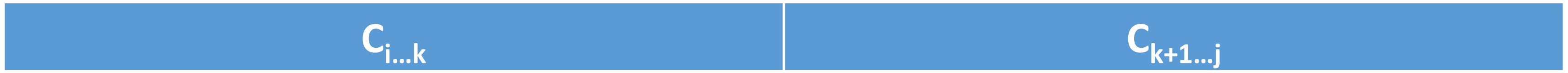
Optimal Substructure in Rod Cutting

Proof of Optimal Substructure

- Lets say we had the optimal solution (revenue 25) for cutting the rod $C_{i…j}$ where C_i is the first piece and C_j is the last piece



- If we take one of the cuts from this solution, somewhere in the middle, say k, and split it so we have two sub problems $C_{i…k}$ and $C_{k+1…j}$ (assuming our optimal is not just a single piece) suppose the revenue in optimal solution for $C_{i…k}$ is 10 and for $C_{k+1…j}$ is 15



- Lets assume we had more optimal way of cutting $C_{i…k}$ with higher revenue 11
- We would swap the old $C_{i…k}$, and replace it with the more optimal $C_{i…k}$
- Overall, the entire problem will now have an even more optimal solution! (with revenue 11+15 = 26)
- But we already had stated that we already have the optimal solution. This is a **contradiction!**
- Therefore our original optimal solution is the optimal solution and this problem exhibits optimal substructure

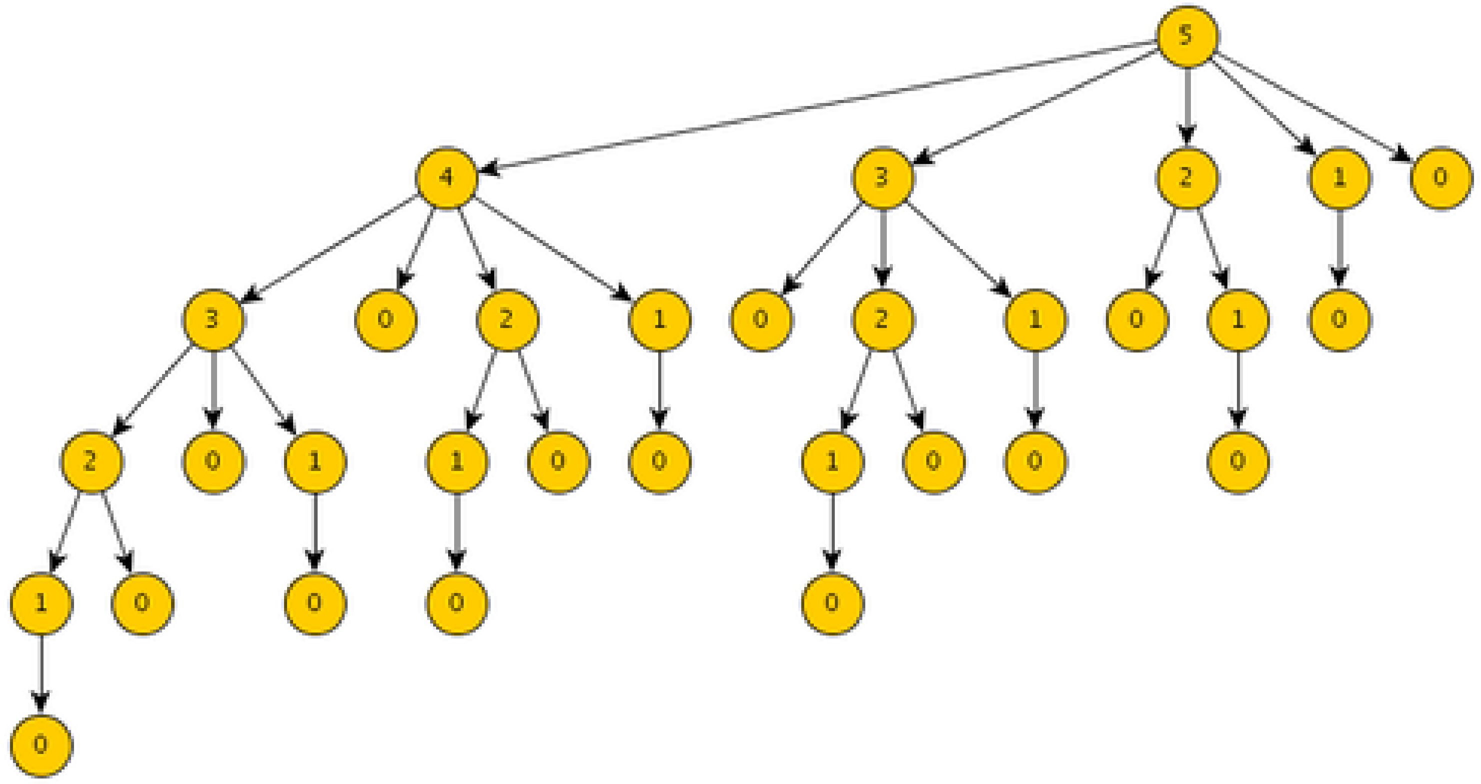
Recursive Solution (Brute Force) for Rod cutting problem

The following procedure implements the method. The inputs of the procedure are the length n and the price p[1….. n]

```
procedure CUT-ROD($p, n$)
    if $n == 0$ then
        return 0
    end if
    $q = -\infty$
    for $i = 1$ to $n$ do
        $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$
    end for
    return $q$
end procedure
```

To see why the procedure is inefficient, we draw the recursion tree of Cut-Rod for n= 5 in this Figure. In the tree, each vertex is a procedure calling. The number in the vertex is the parameter n.

Recursion tree for Cut-Rod(*p*, 5)



Time Complexity of Recursive Solution

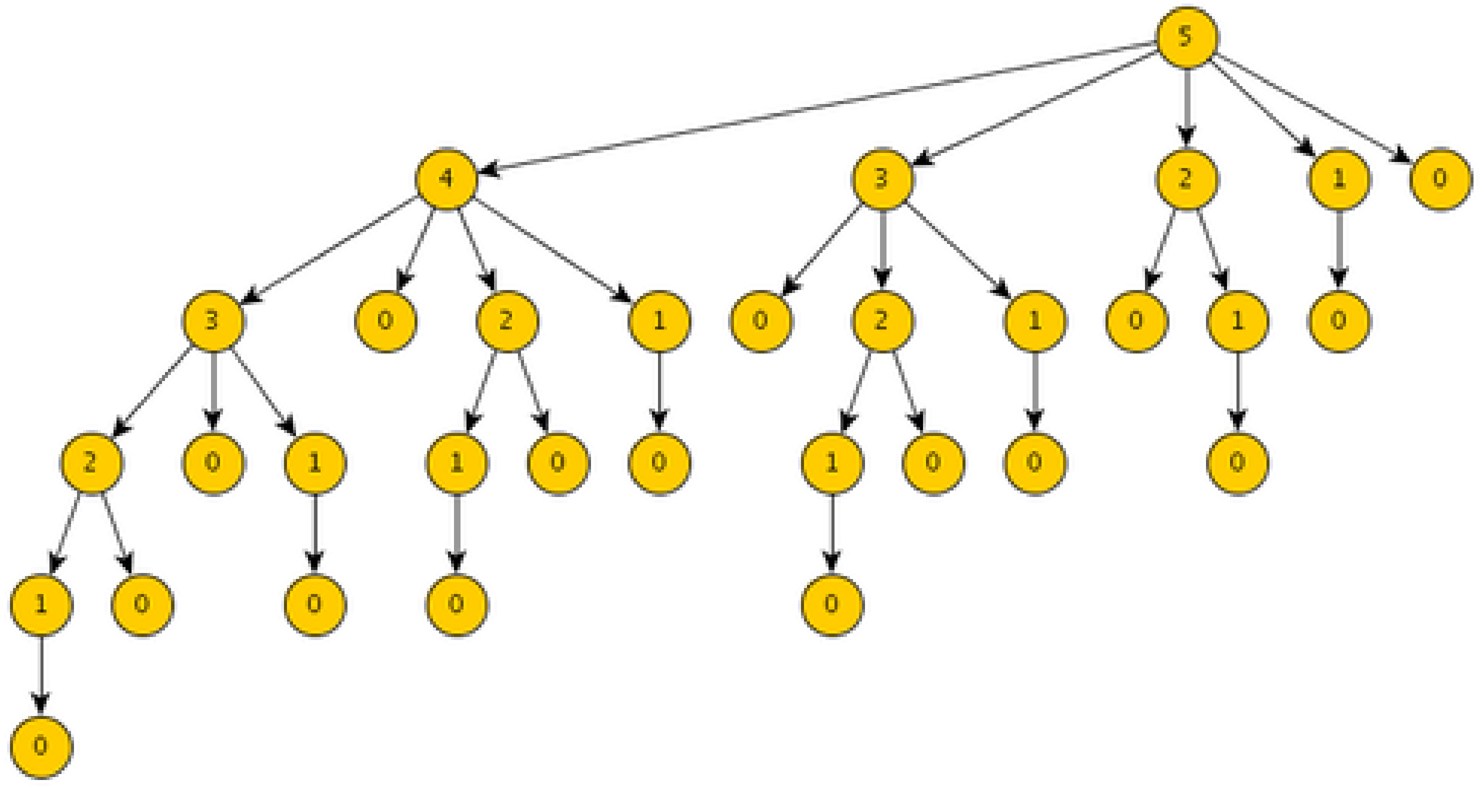
- This procedure is very inefficient. This is because the Cut-Rod procedure calls itself recursively again and again. Suppose the running time of the procedure is T(n). Then we have the recurrence

$$T(n) = \begin{cases} 1 + \sum_{0 \le j \le n-1} T(j), & \text{if } n > 0, \\ 1, & \text{if } n = 0. \end{cases}$$

- ***$T(n) = 2^n$***

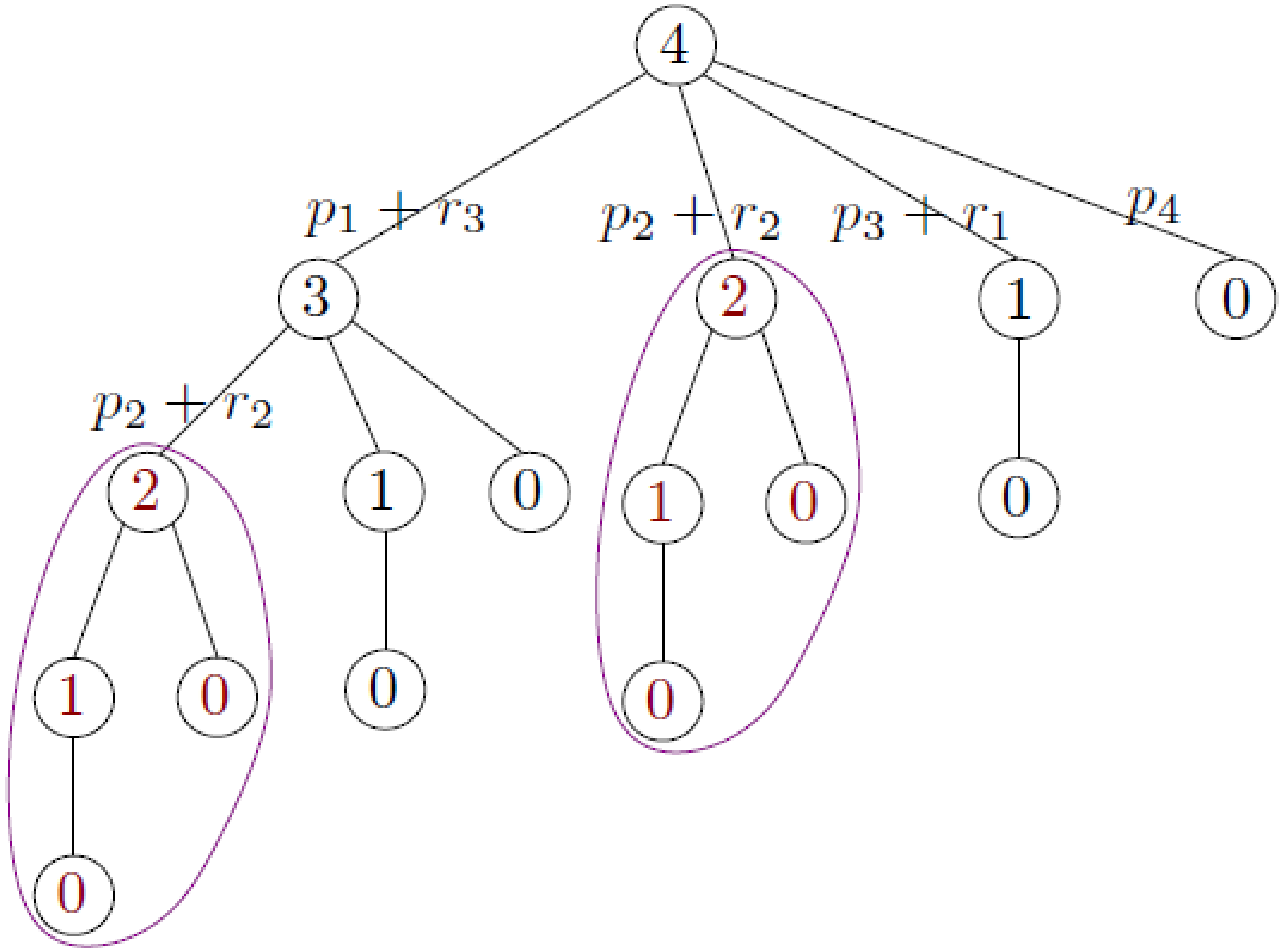
Dynamic Programming Solution to Rod Cutting Problem

Overlapping sub problems in Rod Cutting Problem



The recursion tree of Cut-Rod for n= 5 . The number in the vertex is the parameter n.

Algorithm calls same sub problem many times



Rod Cutting DP

- Lets define C[i] as the price of the optimal cut of a rod up until length *i*
- Let V_k be the price of a cut at length *k*
- How to develop a solution:
- We define the smallest sub problems first, and store their solution
- Remember the recursive solution

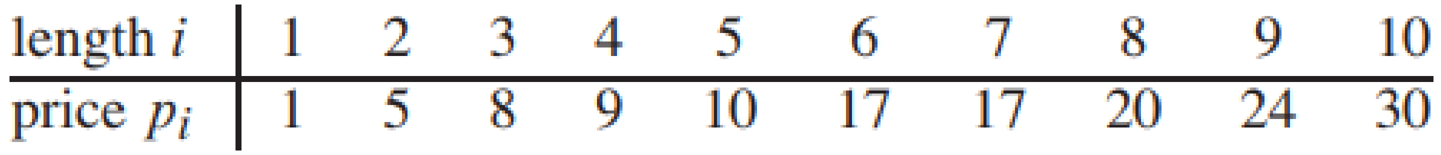
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i}) .$$

Just change the recursive call to an array index

$$\text{C}[i] = max_{1 \le k \le i} \{ V_k + C[i-k]\}$$

Memoization

Rod Cutting DP



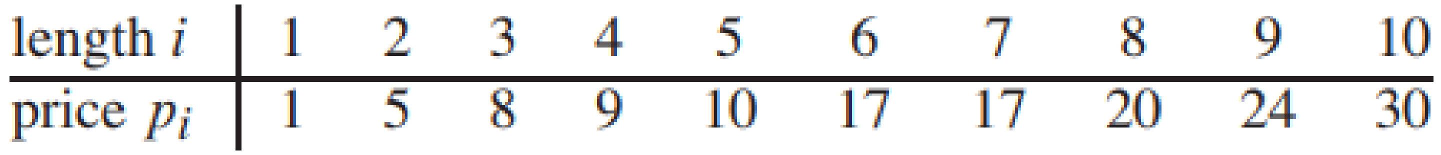
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$C[i] = max_{1\leq\, k\, \leq i}\,\{\, V_k + C[i-k]\}$$

Len (i)	1	2	3	4	5	6	7	8
C[i]	1							

$$C[2] = \max \begin{cases} V_1 + C[1] = 1 + 1 = 2 \\ V_2 = 5 \end{cases}$$

Rod Cutting DP



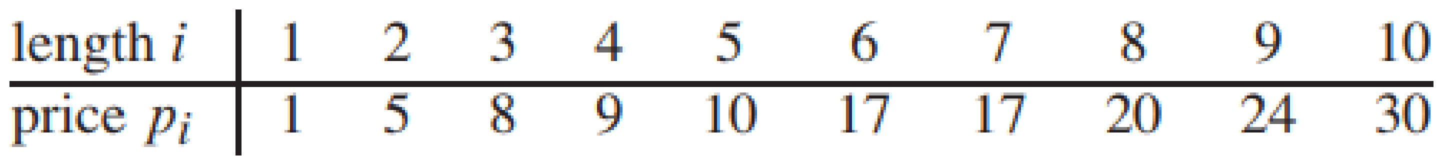
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$C[i] = max_{1\leq\, k\, \leq i}\,\{\,V_k + C[i-k]\}$$

Len (i)	1	2	3	4	5	6	7	8
C[i]	1	5						

$$C[3] = \max \begin{cases} V_1 + C[2] = 1 + 5 = 6 \\ V_2 + C[1] = 5 + 1 = 6 \\ V_3 = 8 \end{cases}$$

Rod Cutting DP



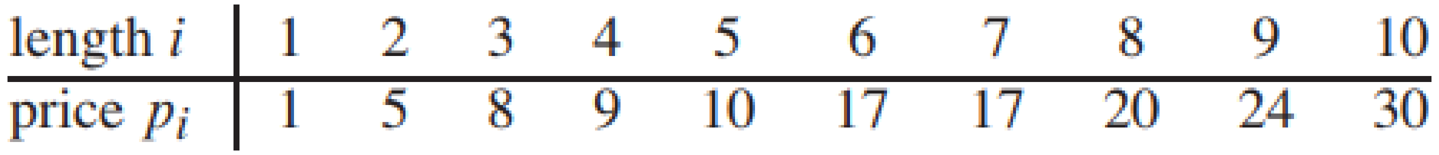
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$C[i] = max_{1 \leq k \leq i} \{ V_k + C[i-k] \}$$

Len (i)	1	2	3	4	5	6	7	8
C[i]	1	5	8					

$$C[4] = \max \begin{cases} V_1 + C[3] = 1 + 8 = 9 \\ V_2 + C[2] = 5 + 5 = 10 \\ V_3 + C[1] = 8 + 1 = 9 \\ V_4 = 9 \end{cases}$$

Rod Cutting DP



length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$C[i] = max_{1\leq\, k\, \leq i}\, \{\, V_k + C[i-k]\}$$

Len (i)	1	2	3	4	5	6	7	8
C[i]	1	5	8	10				

$$C[5] = \max \begin{cases} V_1 + C[4] = 1 + 10 = 11 \\ V_2 + C[3] = 5 + 8 = 13 \\ V_3 + C[2] = 8 + 5 = 13 \\ V_4 + C[1] = 9 + 1 = 10 \\ V_5 = 10 \end{cases}$$

DP Algorithm

- Usually you just need to put for loops above the recursive definition to make the DP algorithm

$$C[i] = max_{1\leq\ k\ \leq i}\ \{\ V_k + C[i-k]\}$$

DP Algorithm

- *Rod-Cutting-DP(V, n)*

```
        C[0] = 0
        for i= 1 to n
            max = -∝
            for k = 1 to i
                if $(max < V_k + C[i-k])$
                    $max = V_k + C[i-k]$
            C[i] = max
```

- Usually you just need to put for loops above the recursive definition to make the DP algorithm

DP Algorithm Time Complexity

- *Rod-Cutting-DP(V, n)*

```
        C[0] = 0
        for i= 1 to n
            max = - ∝
            for k = 1 to i
                if $(max < V_k + C[i-k])$
                    $max = V_k + C[i-k]$
            C[i] = max
```

Time complexity = O (n^2)

Compare it to exponential 2^n time of brute force recursive solution