Algorithms

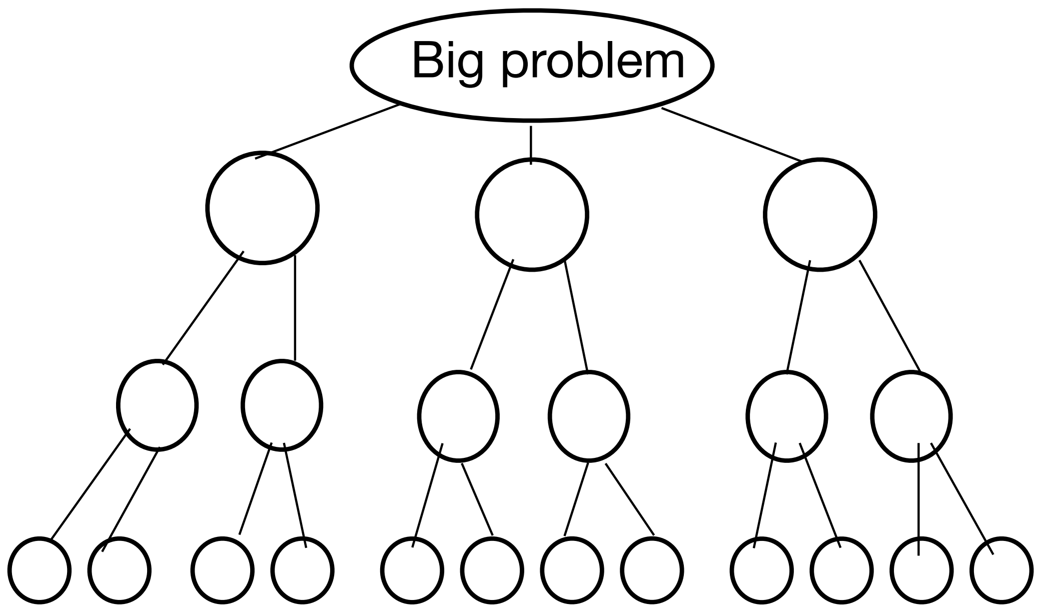
Lecture 1: Dynamic Programming

Anxiao (Andrew) Jiang

CH 15. Dynamic Programming

But first, let's recall "Divide and Conquer"

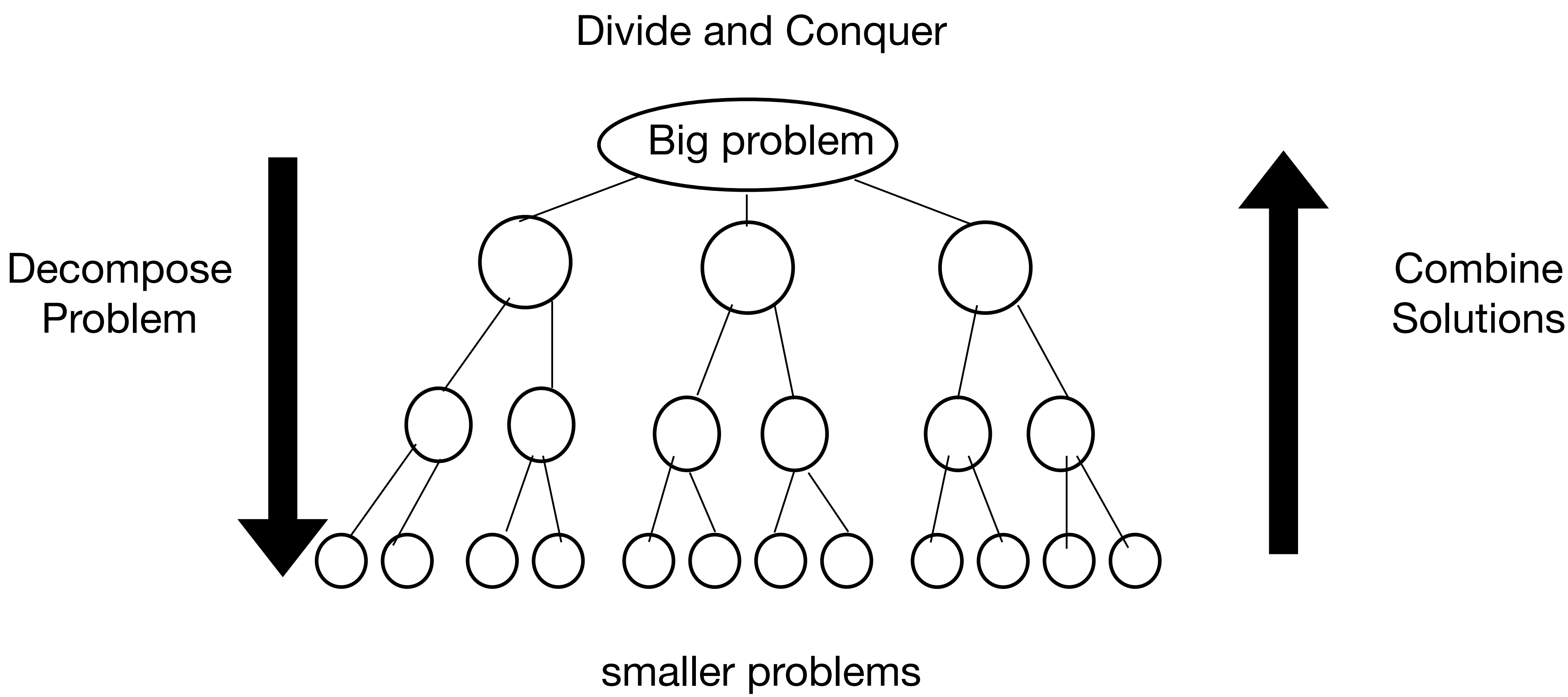
Divide and Conquer



smaller problems

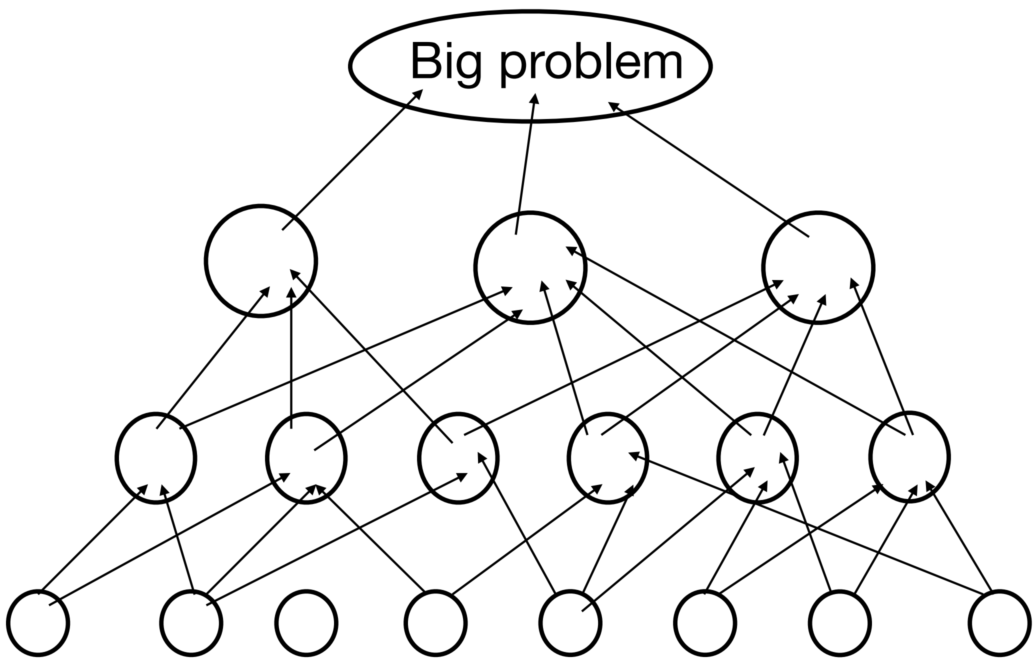
CH 15. Dynamic Programming

But first, let's recall "Divide and Conquer"



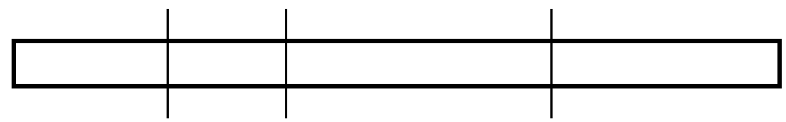
Dynamic Programming

Difference: the solution to a smaller problem can be used more than once by bigger problems.
As a result, dynamic programming can be more efficient than “divide and conquer”.



smaller problems

15.1 Rod Cutting Problem



Input: A rod of length n.

For i = 1,2,3,…,n, a rod of length i has price $p_i \geq 0$

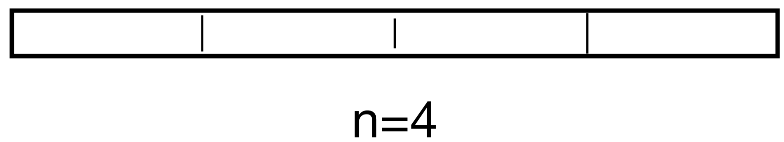
Output: How to cut the rod to maximize the total price?

15.1 Rod Cutting Problem (Example)

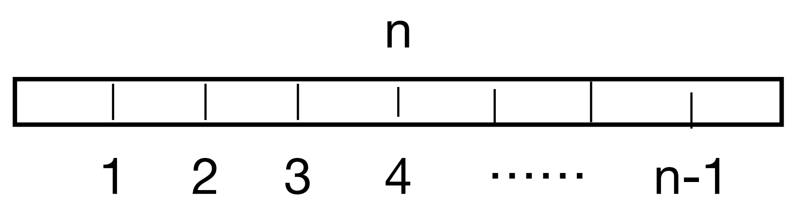
n=4

i	1	2	3	4
p_i	1	5	8	9

15.1 Rod Cutting Problem (Example)



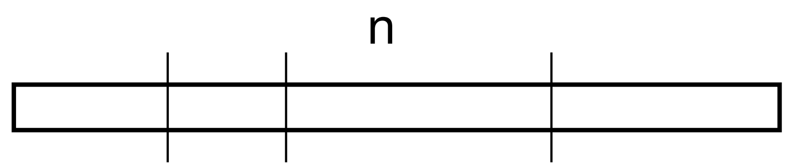
i	1	2	3	4
p_i	1	5	8	9



Should we use exhaustive search?

Time complexity is too high: 2^{n-1}

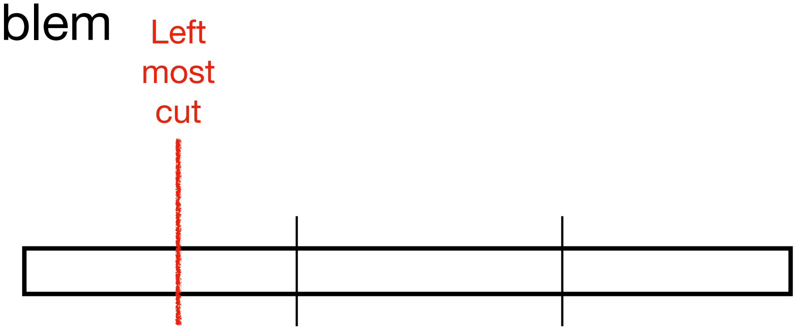
15.1 Rod Cutting Problem



For $i = 0, 1, 2, \cdots, n$

define r_i as the maximum price for cutting a rod of length i

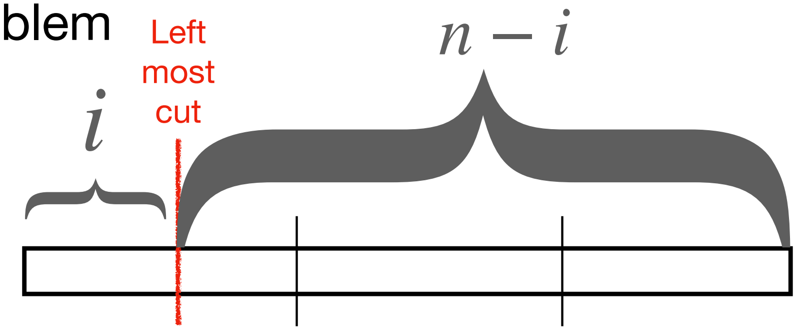
15.1 Rod Cutting Problem



For $i = 0, 1, 2, \cdots, n$

define r_i as the maximum price for cutting a rod of length i

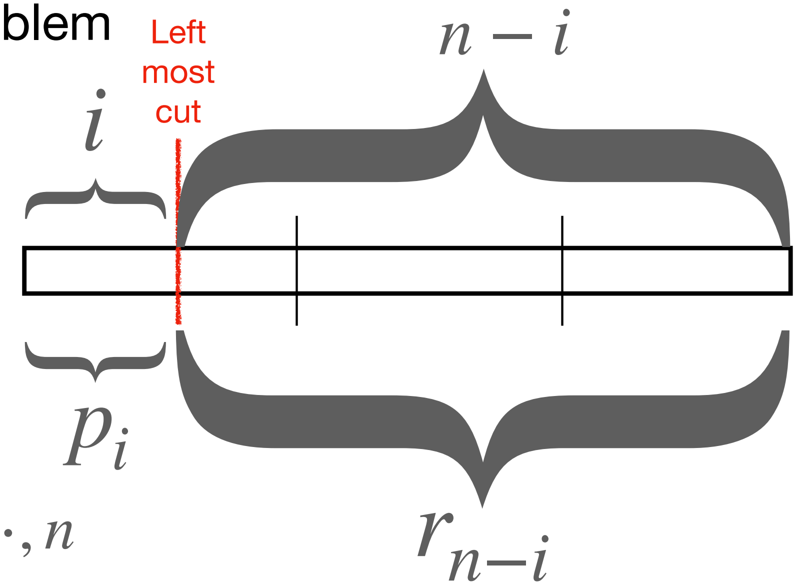
15.1 Rod Cutting Problem



For $i = 0,1,2,\cdots,n$

define r_i as the maximum price for cutting a rod of length i

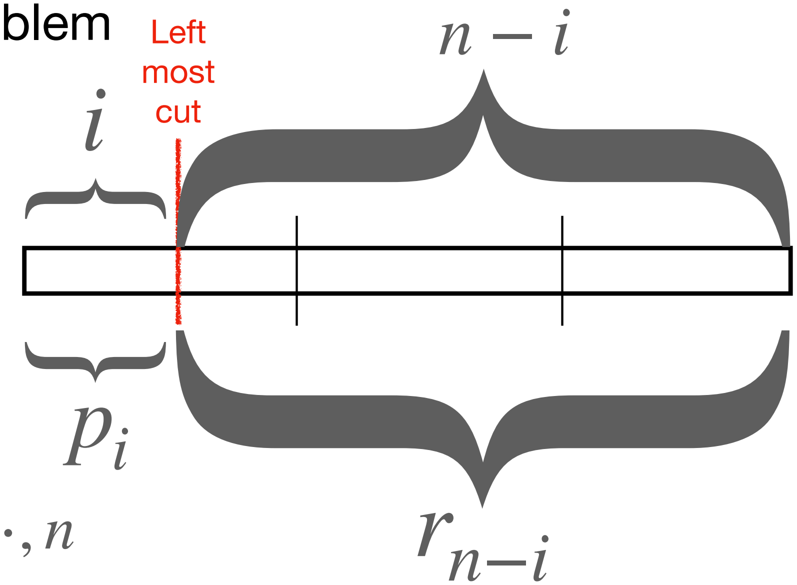
15.1 Rod Cutting Problem



For $i = 0,1,2,\cdots,n$

define r_i as the maximum price for cutting a rod of length i

15.1 Rod Cutting Problem

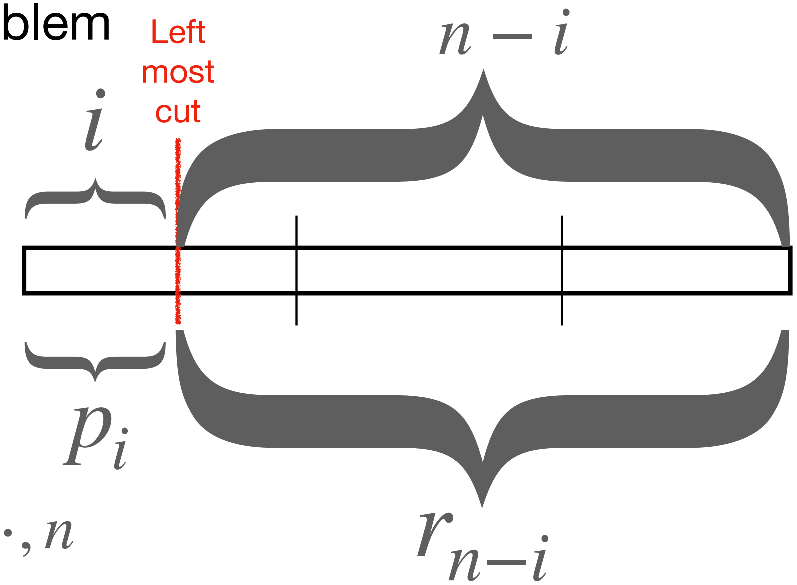


For $i = 0,1,2,\cdots,n$

define r_i as the maximum price for cutting a rod of length i

$$r_n = p_i + r_{n-i}$$

15.1 Rod Cutting Problem



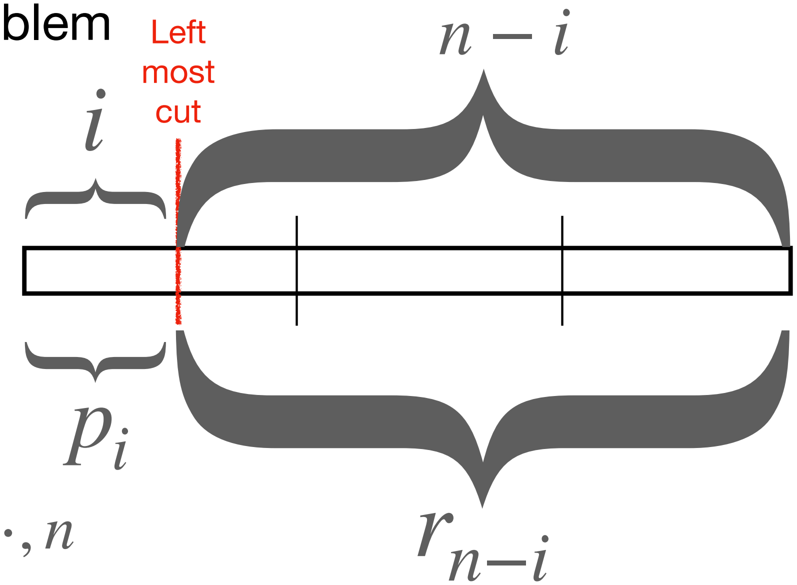
For $i = 0, 1, 2, \cdots, n$

define r_i as the maximum price for cutting a rod of length i

Recursive Function:

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

15.1 Rod Cutting Problem



For $i = 0,1,2,\cdots,n$

define r_i as the maximum price for cutting a rod of length i

Recursive Function:

Bigger problem $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$ Smaller problem

The “smaller solution” will be re-used multiple times

Rod Cutting Problem:

Length i	1	2	3	4	······	n
Price p_i	p_1	p_2	p_3	p_4	······	p_n

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

Rod Cutting Problem:

Length	i	1	2	3	4	……	n
Price	p_i	p_1	p_2	p_3	p_4	……	p_n

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

Rod Cutting Problem:

Length	i	1	2	3	4	······	n
Price	p_i	p_1	p_2	p_3	p_4	······	p_n

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

Memorize solutions

Re-use solutions

Rod Cutting Problem:

Length i	1	2	3	4	......	n
Price p_i	p_1	p_2	p_3	p_4	......	p_n

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

Memorize solutions

Re-use solutions

Rod Cutting Problem:

Length i	1	2	3	4	······	n
Price p_i	p_1	p_2	p_3	p_4	······	p_n

Recursive Function: $$r_n = \max_{1 \leq i \leq n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

Memorize solutions

Re-use solutions

Rod Cutting Problem:

Length	i	1	2	3	4	······	n
Price	p_i	p_1	p_2	p_3	p_4	······	p_n

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$ Find out where to make optimal cuts

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

Rod Cutting Problem:

Length i	1	2	3	4	······	n
Price p_i	p_1	p_2	p_3	p_4	······	p_n

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\} = p_{i*} + r_{n-i*}$$

First optimal cut is at length $i*$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

For example, if n=10, and

$$r_{10} = p_3 + r_7$$

then the left-most piece has length 3

Time Complexity of an Algorithm

Time Complexity:

How long it takes to run the algorithm……

Time Complexity:

How long it takes to run the algorithm as a function of the input size n ……

Time Complexity:

How long it takes to run the algorithm as a function of the input size n in the worst case.

Time Complexity:

How long it takes to run the algorithm as a function of the input size n in the worst case.

Simplification:

1) Every basic operation has complexity 1.

2) We consider only the order of the total number of basic operations.

Time Complexity:

How long it takes to run the algorithm as a function of the input size n in the worst case.

Simplification:

1) Every basic operation has complexity 1.

2) We consider only the order of the total number of basic operations.

$$n + 3n^2 \Rightarrow O(n^2)$$

$$n \lg n + n^3 \Rightarrow O(n^3)$$

$$2^n + 100^{99} \cdot n^2 \Rightarrow O(2^n)$$

Time Complexity of the Dynamic-Programming Algorithm for Rod Cutting Problem:

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$r_0 = 0$$
$$r_1 = p_1$$
$$r_2 = \max\{p_1 + r_1,\ p_2 + r_0\}$$
$$r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\}$$
$$r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\}$$
$$\vdots$$
$$r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\}$$

Time Complexity of the Dynamic-Programming Algorithm for Rod Cutting Problem:

Recursive Function: $$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}$$

Compute solutions bottom-up (from smaller to bigger) using the recursive function:

$$
\begin{array}{ll}
r_0 = 0 & \le O(n) \\
r_1 = p_1 & \le O(n) \\
r_2 = \max\{p_1 + r_1,\ p_2 + r_0\} & \le O(n) \\
r_3 = \max\{p_1 + r_2,\ p_2 + r_1,\ p_3 + r_0\} & \le O(n) \\
r_4 = \max\{p_1 + r_3,\ p_2 + r_2,\ p_3 + r_1,\ p_4 + r_0\} & \le O(n) \\
\vdots & \vdots \\
r_n = \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ p_3 + r_{n-3},\ \cdots,\ p_n + r_0\} & \le O(n)
\end{array}
$$

Time Complexity: $O(n^2)$

4 Essential Steps for Presenting an Algorithm
(Required for all homework and tests)

4 Essential Steps for Presenting an Algorithm
(Required for all homework and tests)

1. Explain the idea of your algorithm **(help us understand the main idea)**

2. Pseudo-code of algorithm **(show us exactly how computing is done)**

3. Prove correctness of your algorithm **(prove rigorously it always finds the right solution)**

4. Analyze time complexity of algorithm **(show us it is efficient)**