DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

DYNAMIC PROGRAMMING (ROD-CUTTING)

AGENDA

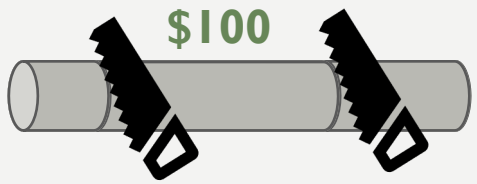
- Review of design techniques
- Case study: rod-cutting problem

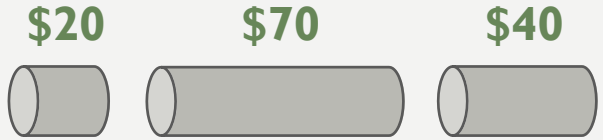
DESIGN TECHNIQUES

- Divide and conquer (divide, conquer, and combine)
  - Signature algorithms: mergesort, quicksort, maximum-subarray problem
  - Signature time complexity: $O(n \lg n)$
- Prune and search (Divide, prune, and conquer)
  - Signature algorithms: binary search, max-min algorithms, randomized select, select
  - Signature time complexity: $O(\lg n)$, $O(n)$
- Design data structure
  - Signature algorithm: Heapsort $O(n \lg n)$
- More to come…

THE ROD-CUTTING STORY

- Sterling Enterprises buys long steel rods and cuts them into shorter rods, which it then sells.
  - Each cut is free.
  - The selling price of the shorter rods are determined based off market demands.
    - The price is not proportional to the length.





- The management of Serling Enterprises wants to know ***the best way to cut up the rods***.

THE ROD-CUTTING STORY
SOLVE IT BY SOFTWARE PROGRAM

- As a software engineer, you have been consulted by Sterling Enterprises with finding **the best way** to cut up the rods.
- To understand the needs of your client. You would then ask the following questions.
  - What is the original length of the rod?
  - Describe the "best way" or describe the goal.
  - If the goal is to profit as much as possible from selling the shorter (not necessarily) rods, what are the market prices of rods of different lengths?

THE ROD-CUTTING STORY
STARTING WITH THE ALGORITHM

- The Sterling Enterprises management returns the following information.
  - The goal is to get maximum profit from selling the rods.
  - The table below shows the market prices of rods with different lengths.
    - The table reads as follows.
      - Sterling Enterprises can sell a 1-inch rod for \$1, 2-inch for \$5, 3-inch for \$8, 4-inch for \$9, …

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

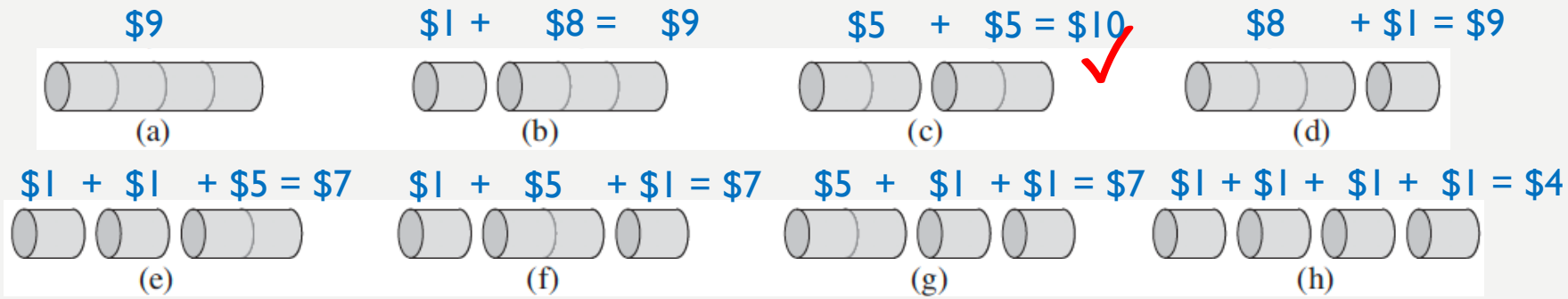
- If Sterling Enterprises originally bought 100 4-inch rods, what would be the best way to cut them?

AN INTUITIVE ROD-CUTTING SOLUTION

- Assume that we always cut at an inch mark **from the left end** of the rod.

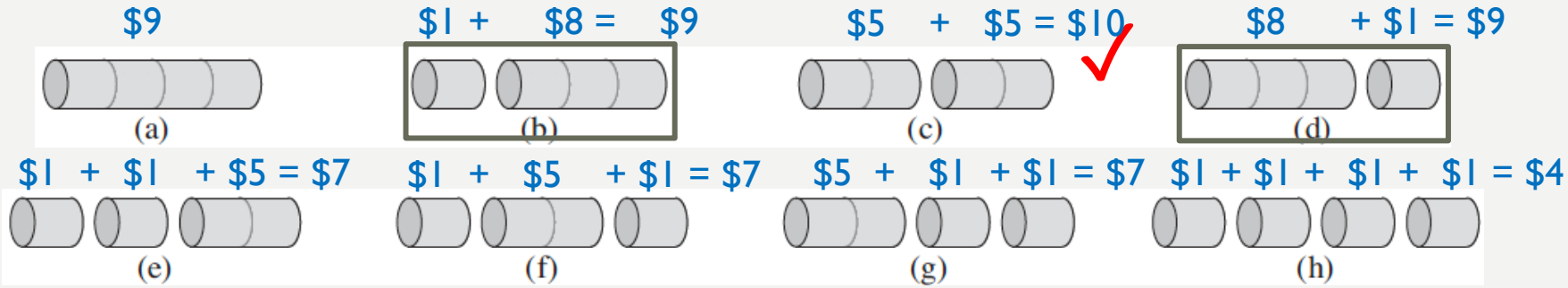
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- An intuitive solution
  - Find out all possible ways to cut up the 4-inch rod. Compute the profit and choose the biggest one.



SECOND THINKING

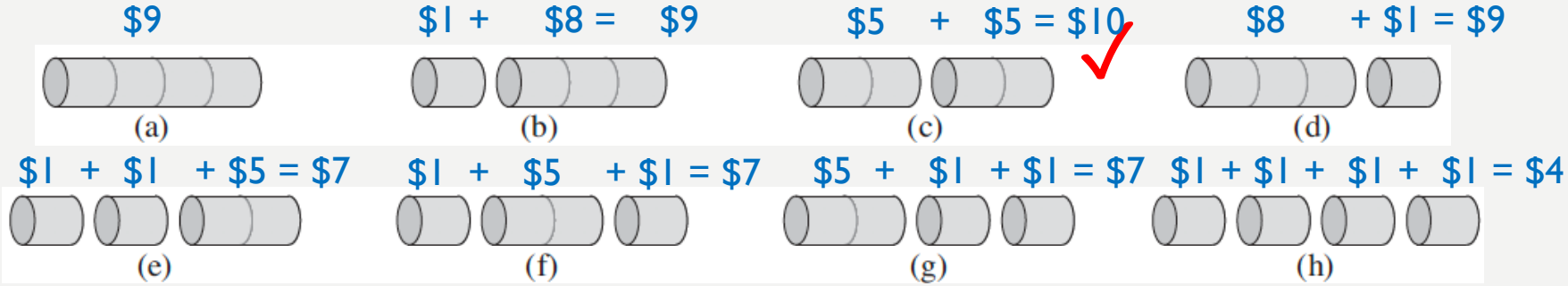
- Here is the work we did.



- Is there **overlapping cases** across two or multiple ways of cutting up the rods?

SECOND THINKING CONT'D

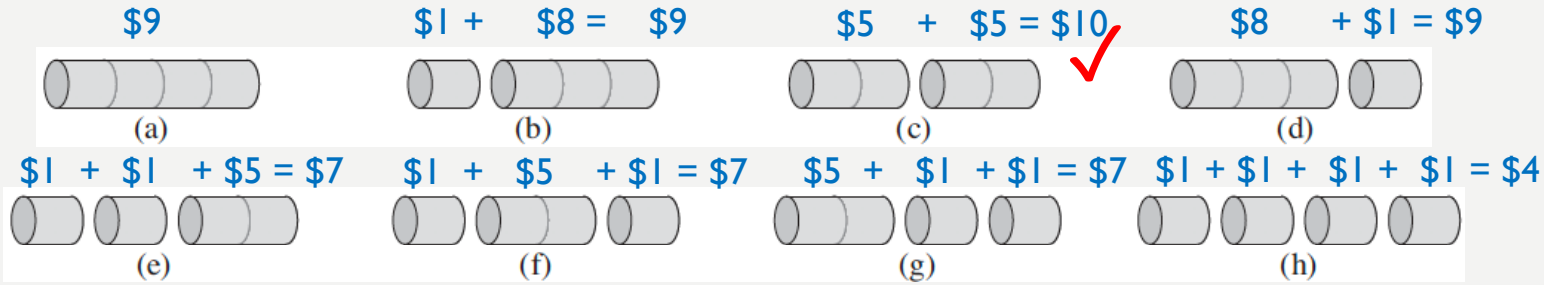
- Here is the work we did.



- Would you say that (e), (f), and (h) are the results of subsequent cuts following the 1st cut at the 1-inch mark shown in (b)?
- Would you say that (g) is the result of one subsequent cut following (c)?
- Subsequent cuts can be made as if we are initiating the cut on a shorter rod.

CONSTRUCT A MODEL BY APPLYING ABSTRACTION

- Let us use r_i denote **the maximum revenue** Sterling Enterprises can obtain from cutting up an i-inch rod.
  - Then r_4 can be solved by finding the maximum of all possible revenues generated by the ways shown below.



  - The goal is to calculate

$$r_4 = \boldsymbol{max}\,(\,p_4,\ r_1 + r_3,\ r_2 + r_2,\ r_3 + r_1,\ r_1 + r_1 + r_2,\ r_1 + r_2 + r_1,\ r_2 + r_1 + r_1,\ r_1 + r_1 + r_1 + r_1\,)$$

CONSTRUCT A MODEL BY APPLYING ABSTRACTION

- In our goal:

$$r_4 = \boldsymbol{max}\,(\,p_4,\ r_1 + r_3,\ r_2 + r_2,\ r_3 + r_1,\ r_1 + r_1 + r_2,\ r_1 + r_2 + r_1,\ r_2 + r_1 + r_1,\ r_1 + r_1 + r_1 + r_1\,)$$

  - $r_1 + r_2, r_2 + r_1,$ and $r_1 + r_1 + r_1$ can be viewed as two possible solutions to obtaining r_3.
  - $r_1 + r_1$ can be viewed as a possible solution to obtaining r_2.

- We can imagine that if we were to cut a longer rod for the maximum profit, say a 10-inch rod, the maximum profit of a 4-inch rod **could be** part of the solution.
  - $r_{10} = r_6 + r_4$

CONSTRUCT A MODEL BY APPLYING ABSTRACTION

- Based off our previous observations

$$r_4 = \boldsymbol{max}\,(\, p_4,\ r_1 + r_3,\ r_2 + r_2,\ r_3 + r_1,\ r_1 + \cancel{r_1 + r_2},\ r_1 + \cancel{r_2 + r_1},\ r_2 + \cancel{r_1 + r_1},\ \cancel{r_1 + r_1 + r_1 + r_1}\,)$$

obtainable by solving r_3 — obtainable by solving r_2

$$= \boldsymbol{max}\,(\, p_4,\ \boldsymbol{p_1} + r_3,\ \boldsymbol{p_2} + r_2,\ \boldsymbol{p_3} + r_1\,)$$

- Pay attention to the subscripts of the p and r notations.
  - For each term of the max expression, the subscripts add up to 4.

CONSTRUCT A MODEL
GENERALIZATION

- For the sake of argument, let us use n to denote the original length of the rod.
  - Our goal is the maximum revenue r_n,

$$r_n = \boldsymbol{max}\left(p_n,\ \ p_1 + r_{n-1},\ \ \ p_2 + r_{n-2},\ \ p_3 + r_{n-3},\ \ \ldots,\ \ p_{n-1} + r_1 \right)$$

    - The subscript of p and r notations add up to n.

- An even simpler version, $r_n = \boldsymbol{\max\limits_{1 \le i \le n}}\left(\ \ p_i + r_{n-i} \ \ \right)$

- Is the formula correct?

THE OPTIMAL SUBSTRUCTURE

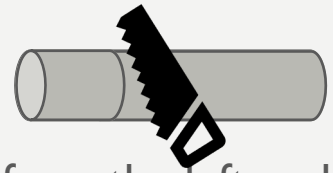
- What is the meaning of the formula: $r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$
  - Recall that r_i is defined to be the maximum revenue obtainable from cutting up an i-inch rod.

- The formula means the maximum revenue obtainable from cutting up a(n) _____-inch rod is composed of the selling price of a(n) _____-inch rod and the maximum revenue obtainable from cutting up a(n) _____-inch rod.
  - In other words, the **optimal** solution to a problem incorporates the **optimal** solution to the related subproblem(s).
  - Is it always the case?

THE OPTIMAL SUBSTRUCTURE

- We define this characteristic: the **optimal** solution to a problem incorporates the **optimal** solution to the related subproblem(s) as the ***optimal substructure*** of the problem.
  - Is it always the case?

- Suppose that we want to solve the case where the original rod is 15-inch.
  - In other words, the goal is to calculate r_{15}.
  - If we know that the optimal solution is to make the first cut at the 5-inch mark from the left end.
  - Does that mean the subproblem arising from the first cut must also be optimally solved.
  - $r_{15} = p_5 +$?



THE PROOF OF THE OPTIMAL SUBSTRUCTURE

- We want to make sure that the rod-cutting problem has an optimal substructure before we continue to design an algorithm using $r_n = \max(p_i + r_{n-i})$.
  - Otherwise, our algorithm won't be correct.
- Proof by contradiction
  - If we know before hand that cutting a 15-inch rod at 5-inch mark will generate the maximum profit of $100.
    - $20 made from selling the 5-inch rod and $80 made from cutting up and selling the remaining 10-inch rod.
  - Then $80 must be the maximum profit we can make from cutting up a 10-inch rod.
  - Otherwise, …

BREAKOUT CHALLENGE (10 MINUTES)

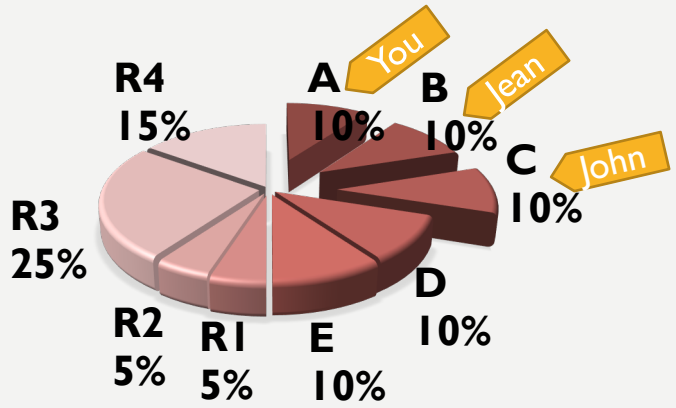
- Complete the proof of the optimal substructure of the rod-cutting problem by filling out the blanks.
  - Hint: use the **assumption** to derive a contradiction against the definition of r_i.

BREAKOUT CHALLENGE (10 MINUTES)

- Proof
  - Based off $r_n = \max(p_i + r_{n-i})$, the optimal solution r_n must incorporate the optimal solution to cutting up an $(n - i)$-inch rod.
  - The goal is to prove that _____ is the optimal solution to cutting up an $(n - i)$-inch rod.
  - **Assume** r^*_{n-i} is the maximum revenue obtainable from cutting up an $(n - i)$-inch rod and that $r^*_{n-i} \neq r_{n-i}$. Obviously, r^*_{n-i} _____ ($<$ or $>$) r_{n-i}.
  - We can construct a new revenue r^*_n by pasting r^*_{n-i} to _____, yielding r^*_n = _____ + _____ and r^*_n _____ ($<$ or $>$) r_n, which contradicts the definition of r_n.
  - Therefore, _____ is the optimal solution to cutting up an $(n - i)$-inch rod.

PIE CUTTING

- You are having dinner with two of your friends Jean and John. The last course is a pie that has been cut up in the following way.
  - Five 10% slices, A, B, C, D, and E
  - Two 5% slices, r1 and r2
  - One 25% slice, r3
  - One 15% slice, r4



- Each of you have been served one 10% slice, and you can have up to two slices.
- Which of the remain of the remaining slices would you pick such that you have most of the pie? How to prove that you have the most between you and your friend?

DESIGN AN ALGORITHM
AN INTUITIVE DESIGN

- After having proven the ***optimal substructure*** of the rod-cutting problem, we can design an algorithm based off the recursive function $r_n = \max(p_i + r_{n-i})$ to compute r_n.

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

- Input
  - p is an array of prices, $p[i]$ represents the price of an i-inch rod.
  - n is the length of the original array.
- The **shaded the statement** is computing the profit recursively.

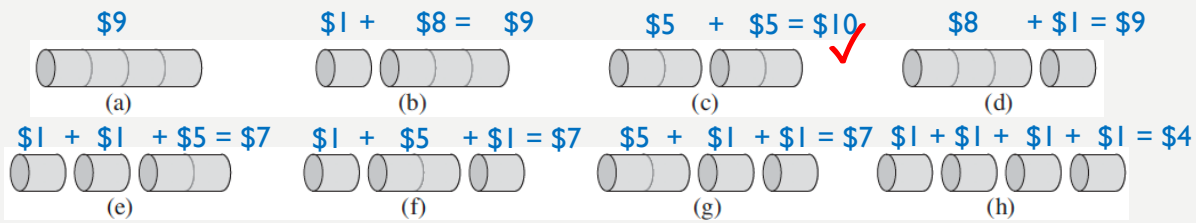
INTUITIVE DESIGN
WALK-THROUGH A BIT

- Call CUT-ROD $(p, 4)$
  - p is the array of prices shown in the table

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

  - 4 is the original length
- The optimal solution is cutting the rod into 2-inch and 2-inch.

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



INTUITIVE DESIGN WALK-THROUGH

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

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

- Call CUT-ROD $(p, 4)$
  - $\boldsymbol{i} = 1, q = \max(q, p[1] + \text{CUT-ROD}\ (p, 3))$
    - Recurse CUT-ROD $(p, \mathbf{3})$
      - $i = 1 \leq \mathbf{3}, q = \max(q, p[1] + \text{CUT-ROD}\ (p, 2))$
        - Recurse CUT-ROD $(p, \mathbf{2})$
          - $i = 1 \leq \mathbf{2}, q = \max(q, p[1] + \sout{\text{CUT-ROD}\ (p, 1)}) = \max(-\infty, 1 + \mathbf{1}) = 2$
            - Recurse CUT-ROD $(p, \mathbf{1}) \Longrightarrow$ Return $\max(-\infty, 1 + \mathbf{0})$
              - $i = 1 \leq \mathbf{1}, q = \max(q, p[1] + \sout{\text{CUT-ROD}\ (p, 0)}) = 1$
                - Recurse CUT-ROD $(p, 0)$
                  - **Return** 0

1

0

ROD-CUT $(p, 0)$
ROD-CUT $(p, 1)$
ROD-CUT $(p, 2)$
ROD-CUT $(p, 3)$
ROD-CUT $(p, 4)$

INTUITIVE DESIGN WALK-THROUGH

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

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

- Call CUT-ROD $(p, 4)$
  - $i = 1, q = \max(q, p[1] + \text{CUT-ROD}\ (p, 3))$
    - Recurse CUT-ROD $(p, \mathbf{3})$
      - $i = 1 \leq \mathbf{3}, q = \max(q, p[1] + \text{~~CUT-ROD}\ (p, 2)\text{~~}) = \max(-\infty, 1 + \mathbf{5}) = \mathbf{6}$ (5)
        - Recurse CUT-ROD $(p, \mathbf{2})$
          - $i = 1 \leq \mathbf{2}, q = \max(q, p[1] + \text{~~CUT-ROD}\ (p, 1)\text{~~}) = \max(-\infty, 1 + \mathbf{1}) = 2$ (1)
          - $i = 2 \leq \mathbf{2}, q = \max(2, p[2] + \text{~~CUT-ROD}\ (p, 0)\text{~~}) = \max(2, 5 + \mathbf{0}) = 5$ (0)
            - Recurse CUT-ROD $(p, \mathbf{0})$
              - **Return** 0

ROD-CUT $(p, 0)$
ROD-CUT $(p, 2)$
ROD-CUT $(p, 3)$
ROD-CUT $(p, 4)$

INTUITIVE DESIGN WALK-THROUGH

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

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

- Call CUT-ROD $(p, 4)$
  - $i = 1, q = \max(q, p[1] + \text{CUT-ROD}\ (p, 3))$
    - Recurse CUT-ROD $(p, \mathbf{3})$
      - $i = 1 \leq \mathbf{3}, q = \max(q, p[1] + \text{~~CUT-ROD}\ (p, 2)\text{~~}) = \max(-\infty, 1 + \mathbf{5}) = \mathbf{6}$ (5)
      - $i = 2 \leq \mathbf{3}, q = \max(\mathbf{6}, p[2] + \text{~~CUT-ROD}\ (p, 1)\text{~~}) = \max(6, 5 + \mathbf{1}) = \mathbf{6}$ (1)
        - Recurse CUT-ROD $(p, \mathbf{1})$
          - $i = 1 \leq \mathbf{1}, q = \max(q, p[1] + \text{~~CUT-ROD}\ (p, 0)\text{~~}) = \max(-\infty, 1 + \mathbf{0}) = 1$ (0)
            - Recurse CUT-ROD $(p, \mathbf{0})$
              - **Return** 0

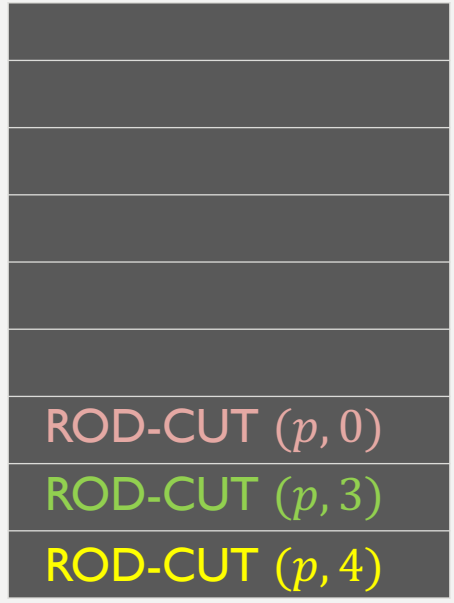
ROD-CUT $(p, 0)$
ROD-CUT $(p, 1)$
ROD-CUT $(p, 3)$
ROD-CUT $(p, 4)$

INTUITIVE DESIGN WALK-THROUGH

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

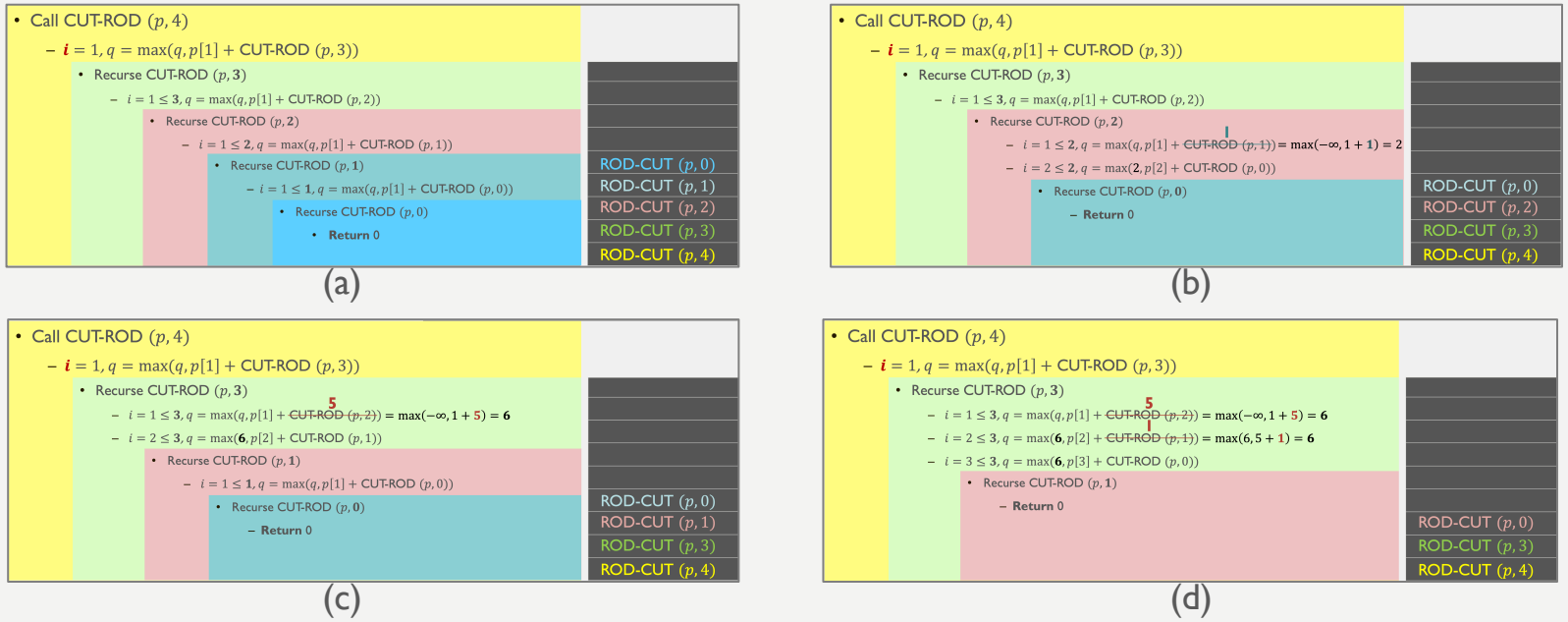
```
CUT-ROD (p, n)
1  if n == 0
2        return 0
3  q = −∞
4  for i = 1 to n
5        q = max(q, p[i] + CUT-ROD (p, n − i))
6  return q
```

- Call CUT-ROD $(p, 4)$
  - $i = 1, q = \max(q, p[1] + \text{CUT-ROD}\ (p, 3))$
    - Recurse CUT-ROD $(p, \mathbf{3})$
      - $i = 1 \leq \mathbf{3}, q = \max(q, p[1] + \overset{\mathbf{5}}{\text{CUT-ROD}\ (p, 2)}) = \max(-\infty, 1 + \mathbf{5}) = \mathbf{6}$
      - $i = 2 \leq \mathbf{3}, q = \max(\mathbf{6}, p[2] + \overset{\mathbf{1}}{\text{CUT-ROD}\ (p, 1)}) = \max(6, 5 + \mathbf{1}) = \mathbf{6}$
      - $i = 3 \leq \mathbf{3}, q = \max(\mathbf{6}, p[3] + \overset{\mathbf{0}}{\text{CUT-ROD}\ (p, 0)}) = \max(6, 8 + \mathbf{0}) = \mathbf{8}$
        - Recurse CUT-ROD $(p, \mathbf{1})$
          - **Return** 0



EFFICIENCY OF THE INTUITIVE ALGO.

- From the previous steps of walking through the intuitive algorithm we notice that some recursions are computed more than one time.



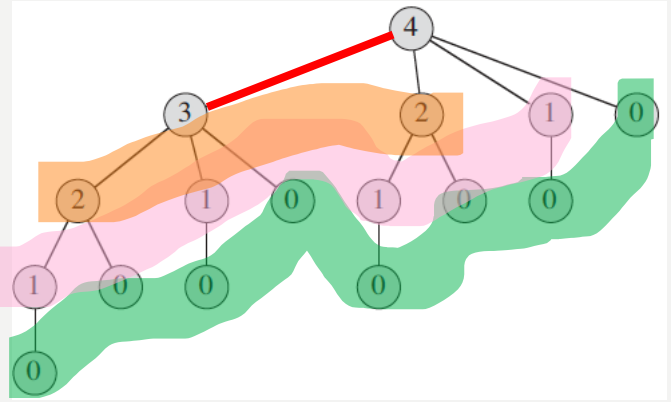
EFFICIENCY OF THE INTUITIVE ALGO. THE RUNNING TIME

- Complete the **cost** and **time** columns.
- The running time
  $T(n) = \Theta(1) + \Theta(n) +$
  $$\Theta(1) \cdot \sum_{i=1}^{n} T(n-i)$$
- How to solve such a recurrence?

CUT-ROD (p, n)		Cost	Time
1	**if** $n == 0$	$\Theta(1)$	1
2	**return** 0	$\Theta(1)$	base
3	$q = -\infty$	$\Theta(1)$	1
4	**for** $i = 1$ **to** n	$\Theta(1)$	$n + 1$
5	$q = \max(q, p[i] +$ CUT-ROD $(p, n - i))$	$\Theta(1)$	?
6	**return** q	$\Theta(1)$	1

ANOTHER VIEW OF THE INTUITIVE ALGORITHM

- The recursion tree of the top-down naïve implementation.
  - Each nodes corresponds to a subproblem.
  - An edge from a parent with label s to a child with label t corresponds to cutting off an initial piece of size $s - t$ and leaving a remaining subproblem of size t.
  - A path from the root to a leaf corresponds to one way of cutting up a rod of length n.
    - How many different ways to cut up a rod of length n?
  - In general, this tree has 2^n nodes and 2^{n-1} leaves.
- The running time is $O(2^n)$.



THE IMPROVED TOP-DOWN ALGORITHM WITH MEMOIZATION

- The word "memorized" or "memoizaiton" comes from the word ***memo***.
- This technique uses additional memory to save computation time.
  - A ***time-memory trade-off***.
  - The new $r[0..n]$ array keeps a record of that subproblems that have been solved.
- Complexity
  - $T(n) = \Theta(n) + f(n)$, where $f(n)$ is the cost of MEMOIZED-CUT-ROD-AUX.
  - $S(n) = \Theta(n)$

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 MEMO**IZED-CUT-ROD-AUX (p, n, r)

MEMOIZED-CUT-ROD-AUX VS CUT-ROD

- The **MEMO**IZED-CUT-ROD-AUX

```
MEMOIZED-CUT-ROD-AUX (p, n, r)
1  if r[n] ≥ 0
2        return r[n]
3  if n == 0
4        q = 0
5  else q = −∞
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
```

The handling of the ***memo***.

- The intuitive CUT-ROD

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

Although the two algorithms have the same structure, a recursive call to MEMOIZED-CUT-ROD-AUX algorithm that solves a previously solved subproblem returns **immediately**.

THE TOP-DOWN ALGORITHM WITH MEMOIZATION RUNNING TIME

- The running time of the ***memo***ized top-down algorithm.
  - Each subproblem is solved exactly _____ time(s).
    - Cutting at 1 inch, then solve subproblem for _____.
    - Cutting at 2 inch, then solve subproblem for _____.
    - Cutting at 3 inch, then solve subproblem for _____.
    - …
    - Cutting at n-1 inch, then solve problem for _____.
- $T(n) = \Theta($ ___n^2___ $)$

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

```
MEMOIZED-CUT-ROD-AUX (p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = −∞
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
```

THE ITERATIVE COUNTERPART BOTTOM-UP-CUT-ROD

- Use the natural ordering of the subproblems: a subproblem of size i is "smaller" than a subproblem of size j if $i < j$.
- The procedure solves subproblems of sizes $j = 0, 1, \ldots, n$, in that order.
- It also keeps a record of the subproblems that have been solved.
- Running time $T(n) = \Theta(\_\_\_\_\_)$.
  - Related work is done in the *Application of Asymptotic* assignment.

```
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 = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

RECONSTRUCTING THE SOLUTION
MODIFIED BOTTOM-UP VERSION

- Both top-down and bottom-up algorithms calculates the maximum revenue.
- However, sometimes we need to know how to achieve the maximum/optimal solution.

EXTENDED-BOTTOM-UP-CUT-ROD (p, n)	
1	let $r[0..n]$ and $s[0..n]$ be new arrays
2	$r[0] = 0$
3	**for** $j = 1$ **to** n
4	$\quad q = -\infty$
5	$\quad$ **for** $i = 1$ **to** j
6	$\quad\quad$ **if** $q < p[i] + r[j - i]$
7	$\quad\quad\quad q = p[i] + r[j - i]$
8	$\quad\quad\quad s[j] = i$
9	$\quad r[j] = q$
10	**return** r and s

PRINT-CUT-ROD-SOLUTION USING THE MODIFIED BOTTOM-UP VERSION

- The PRINT-CU-ROD-SOLUTION algorithm prints the optimal solution from the r and s table.

```
PRINT-CUT-ROD-SOLUTION (p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD (p, n)
2  while n > 0
3      print s[n]
4      n = n − s[n]
```

- Modify the top-down version with memoization such that it also maintains a similar $s[0..n]$ table.
- Then modify the PRINT-CUT-ROD-SOLUTION to use the modified top-down version with memoization to print the optimal solution.

NEXT UP
DYNAMIC PROGRAMMING

- Elements of DP

REFERENCE

- Screenshots are taken from the textbook.