# CMPT307: Dynamic Programming

Week 7-1

**Xian Qiu**

Simon Fraser University

xianq@sfu.ca

# Rod Cutting

- to cut a rod of length $n$ inches into small pieces
- price is $p_i$, if the piece has length $i$
- maximize the total price

| length $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| price $p_i$ | 1 | 3 | 5 | 11 | 13 | 17 | 21 | 22 | 23 | 29 |

- divide-and-conquer?
- greedy?
- brute force?

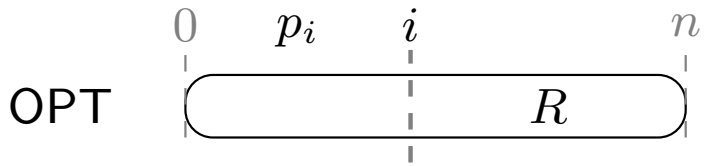
# Dynamic Programming

1. characterize the structure of an optimal solution
   cut-and-paste argument
2. define the optimal value in a recursive way
   the running time is estimated accordingly
3. compute the optimal value for the original problem
   some indices often need to be searched by brute force
4. construct an optimal solution from computed information
   save the indices found in step 3

# Optimal Substructure

## Optimal substructure

OPT incorporates optimal solutions to independent subproblems.




## Lemma

The cutting for $R$ in OPT is optimal for subproblem $R$ (with rod size $n - i$).

cut-and-paste argument

- ▷ assume there is a better cutting for $R$
- ▷ using this cutting for $R$ in OPT yields a better solution

# Recursive Formula

- let $r_n$ be the optimal solution value for rod size $n$ and $i$ be some cut point in an optimal solution

$$r_n = p_i + r_{n-i}, \quad 1 \leq i \leq n$$

- what is the value of $i$?

$$r_n = \max_{1 \leq i \leq n} \{p_i + r_{n-i}\}$$ recursive formula

- how to compute $r_n$? straightforward implementation?

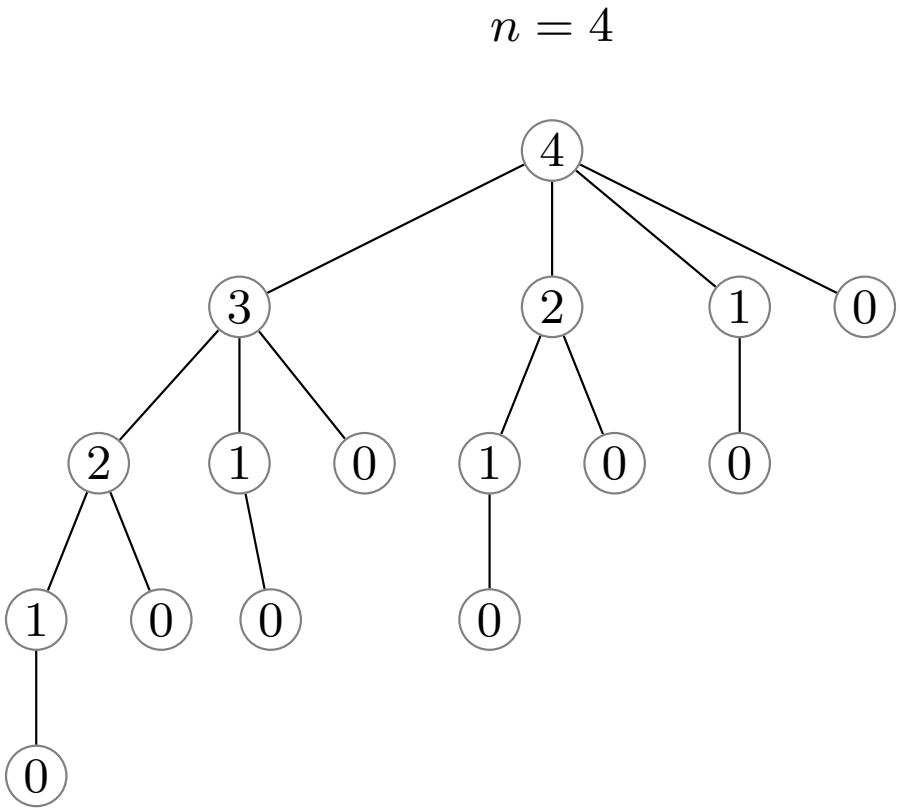
# Straightforward Implementation

CUT-ROD$(p, n)$

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

- $T(n) = 1 + \sum_{i=0}^{n-1} T(i) \Rightarrow T(n) = 2^n$ assume $T(0) = 1$
- it solves the same subproblems repeatedly!

# Recursion Tree




node label = size of subproblem

# Top Down with Memoization

idea: use memory to save computation

top-down with memoization: implement recursively and save the result for each subproblem

MEMOIZED-CUT-ROD$(p, n)$

```
1 r[0..n] = −∞;                                  // use r to save results
2 return MEMOIZED-CUT-ROD-AUX(p, n, r);
```

# Top Down with Memoization

MEMOIZED-CUT-ROD-AUX$(p, n, r)$

```
1  if r[n] ≥ 0 then
2      return r[n];                          // r_n has already been solved
3  if n == 0 then
4      q = 0;
5  else
6      q = −∞;
7      for i = 1 to n do
8          q = max{q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r)};
9      r[n] = q;                             // save result
10 return q;
```

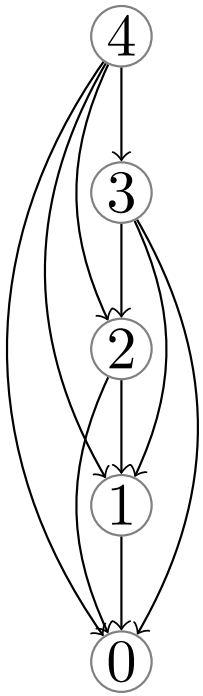
# Bottom Up Method

bottom-up: directly solve subproblems and save the results

BOTTOM-UP-CUT-ROD($p, n$)

```
1  r[0..n] = 0;
   // subproblem has rod size j
2  for j = 1 to n do
3      q = -∞;
4      for i = 1 to j do
5          q = max{q, p[i] + r[j - i]};
6      r[j] = q ;                                  // save result
7  return r[n];
```

# Subproblem Graph




$\#$ nodes $= n + 1$

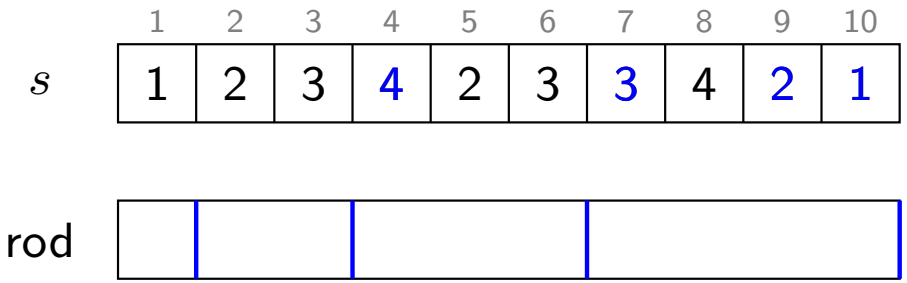
$\#$ edges $= \frac{n(n+1)}{2}$

$T(n) = \Theta(n^2)$

# Reconstructing a Solution

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

- idea: save "index $i$" for each subproblem
- define $s[j]$ = optimal cut off size, for rod length $j$
- cut at $s[j]$ and yield new subproblem $j - s[j]$



# Pseudocode

EXTENDED-BOTTOM-UP-CUT-ROD$(p, n)$

```
1 r[0..n] = 0 and s[0..n] = 0;
2 for j = 1 to n do
3     q = -∞;
4     for i = 1 to j do
5         if q < p[i] + r[j - i] then
6             q = p[i] + r[j - i];
7             s[j] = i;
8     r[j] = q;
9 return r and s;
```