# Dynamic Programming

# Dynamic Programming

- A divide-and- conquer method
  - Partition the problem into subproblems
  - Solve the subproblems recursively
  - Combine their solutions to solve the original problem

- Optimization problems
  - 수 많은 solution 중 optimal solution (minimize or maximize) 를 찾는 문제

  (ex) navigation, scheduling, ……

  - DP
    - Global optimum 을 찾는 방법
    - Exponential time ⇒ polynomial time

# Dynamic Programming

- Basic steps
  1. Characterize the structure of an optimal solution
  2. Recursively define the value of an optimal solution
  3. Compute the value of an optimal solution in a bottom-up fashion
  4. Construct an optimal solution from computed information

# Assembly Line Scheduling

- Assembly Line
  - A serial manufacturing system

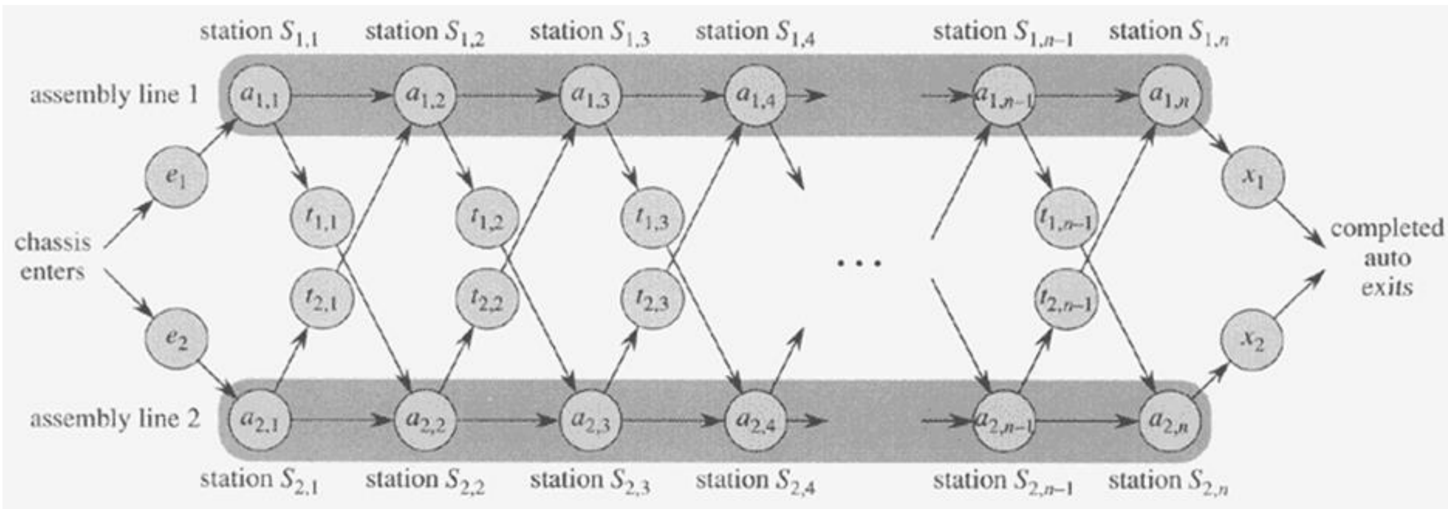
  (ex) Car assembly line

(기아자동차 assembly line)


(대우자동차 assembly line)

# Assembly Line Scheduling

- Problem




**$S_{i,j}$:** i-th line의 j-th station (i=1,2, j=1,…,n), $S_{1,j}$ 와 $S_{2,j}$ 는 동일한 기능

**$a_{i,j}$:** station $S_{i,j}$ 에서 소요되는 작업시간 (i=1,2, j=1,…,n)

**$e_i$**: entry time, line i 에 chassis 를 입고하는데 소요되는 시간 (i=1,2)

**$x_i$**: exit time, line i 에서 완성차를 출고하는데 소요되는 시간 (i=1,2)

**$t_{i,j}$:** station Si,j 이후 chassis 를 다른 line 으로 transfer 하는 데 소요되는 시간 (i=1,2, j=1,…,n-1)

(같은 line 에서 station 간의 이동 소요 시간=0)

# Assembly Line Scheduling

- Problem
  - Find the fastest way (minimize the assembly time)
  - 한 대의 차를 가장 빨리 조립하는 route
- Brute force approach
  - Number of possible ways : $2^n$ (exponential time)
  - Infeasible when $n$ is large

# Assembly Line Scheduling

**(Step 1) The structure of the fastest way**

$S_{1,j}$ 를 통과하는 fastest way 는 다음 둘 중 하나

- $S_{1,j-1}$를 통과한 fastest way 가 $S_{1,j}$를 직접 통과하는 경우
- $S_{2,j-1}$를 통과한 fastest way 가 $S_{1,j}$로 transfer 하여 통과하는 경우

$S_{2,j}$ 를 통과하는 fastest way 는 다음 둘 중 하나

- $S_{2,j-1}$를 통과한 fastest way 가 $S_{2,j}$를 직접 통과하는 경우
- $S_{1,j-1}$를 통과한 fastest way 가 $S_{2,j}$로 transfer 하여 통과하는 경우

⇒ j-th station 을 통과하는 fastest way 는, 각 line 의 (j-1) station 을 통과하는 fastest way 로부터 구한다.

⇒ recursive structure

# Assembly Line Scheduling

## (Step 2) A Recursive Solution

$f_i[j]$: starting point에서 station $S_{i,j}$ 까지 통과하는 데 소요되는 최소 시간
$f^*$ : starting point에서 final point 까지 통과하는 데 소요되는 최소 시간(fastest time)

$$f_1[1] = e_1 + a_{1,1}$$
$$f_2[1] = e_2 + a_{2,1}$$

$$f_1[2] = \min(f_1[1] + a_{1,2},\ f_2[1] + t_{2,1} + a_{1,2})$$
$$f_2[2] = \min(f_2[1] + a_{2,2},\ f_1[1] + t_{1,2} + a_{2,2})$$

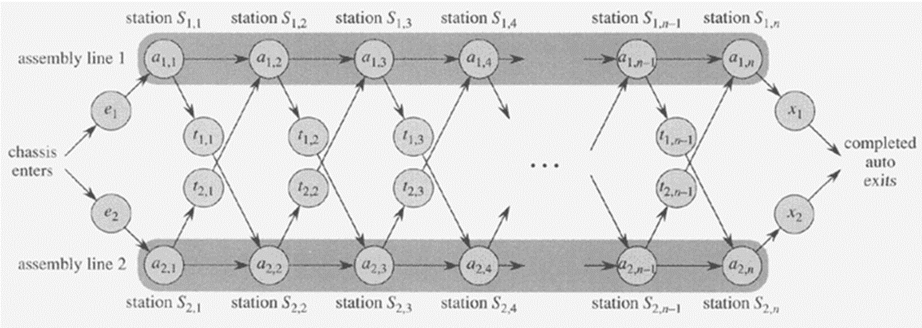
$$\vdots$$

$$f_1[j] = \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j})$$
$$f_2[j] = \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j})$$

*Principle of optimality*

$$\vdots$$

$$f^* = \min(f_1[n] + x_1,\ f_2[n] + x_2)$$



# Assembly Line Scheduling

## (step 3) Computing the fastest times

FASTEST-WAY(*a,t,e,x,n*)

- $f_i[j], f^*, l_i[j], l^*$ 계산 및 저장
- Running time: Θ(*n*)

```
FASTEST-WAY(a, t, e, x, n)
f1[1] ← e1 + a1,1
f2[1] ← e2 + a2,1
for j ← 2 to n
     do if f1[j − 1] + a1,j ≤ f2[j − 1] + t2,j−1 + a1,j
           then f1[j] ← f1[j − 1] + a1,j
                l1[j] ← 1
           else f1[j] ← f2[j − 1] + t2,j−1 + a1,j
                l1[j] ← 2
        if f2[j − 1] + a2,j ≤ f1[j − 1] + t1,j−1 + a2,j
           then f2[j] ← f2[j − 1] + a2,j
                l2[j] ← 2
           else f2[j] ← f1[j − 1] + t1,j−1 + a2,j
                l2[j] ← 1
if f1[n] + x1 ≤ f2[n] + x2
   then f* = f1[n] + x1
        l* = 1
   else f* = f2[n] + x2
        l* = 2
```

$$l_i[j] = \begin{cases} 1 & \text{,if } S_{1,j-1} \rightarrow S_{i,j} \\ 2 & \text{,if } S_{2,j-1} \rightarrow S_{i,j} \end{cases}$$

$$l^* = \begin{cases} 1 & \text{,if } S_{1,n} \rightarrow final \\ 2 & \text{,if } S_{2,n} \rightarrow final \end{cases}$$

# Assembly Line Scheduling

## (step 4) Constructing the fastest way

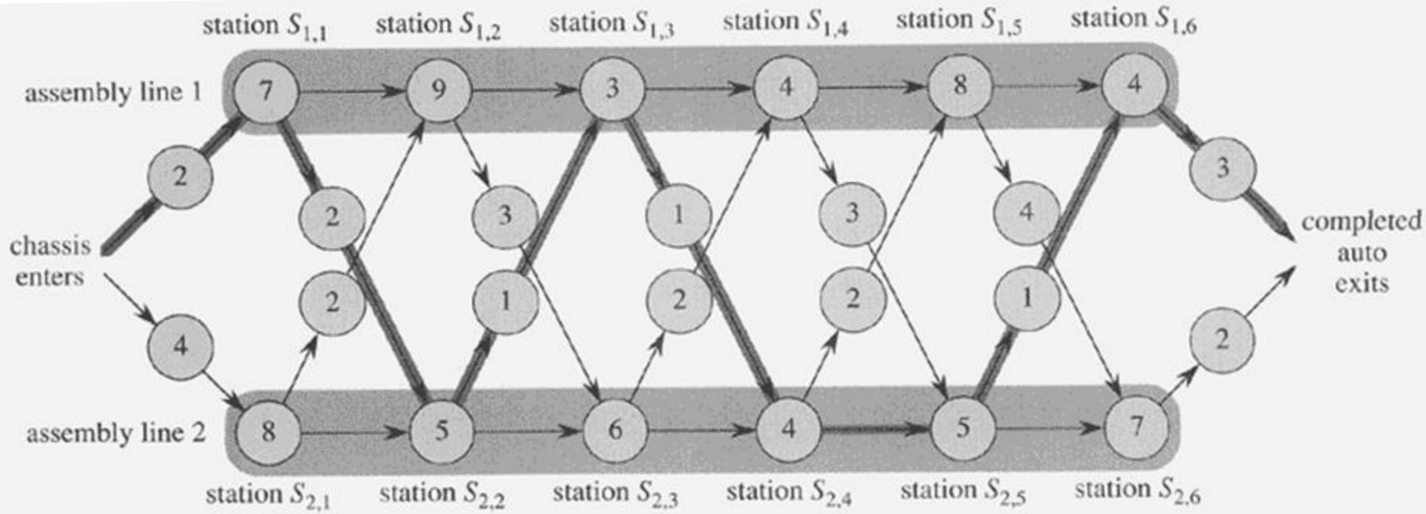
PRINT-STATIONS(*l*,*n*)

- fastest way 인쇄
- $l^*$ 부터 역방향으로 $l_i[j]$ trace

```
PRINT-STATIONS(l, n)
i ← l*
print "line " i ", station " n
for j ← n downto 2
    do i ← l_i[j]
        print "line " i ", station " j − 1
```

# Assembly Line Scheduling

(Example)




(a)

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $f_1[j]$ | 9 | 18 | 20 | 24 | 32 | 35 |
| $f_2[j]$ | 12 | 16 | 22 | 25 | 30 | 37 |

$f^* = 38$

| $j$ | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| $l_1[j]$ | 1 | 2 | 1 | 1 | 2 |
| $l_2[j]$ | 1 | 2 | 1 | 2 | 2 |

$l^* = 1$

(b)

**Figure 15.2** **(a)** An instance of the assembly-line problem with costs $e_i$, $a_{i,j}$, $t_{i,j}$, and $x_i$ indicated. The heavily shaded path indicates the fastest way through the factory. **(b)** The values of $f_i[j]$, $f^*$, $l_i[j]$, and $l^*$ for the instance in part (a).

# Assembly Line Scheduling

(Q)

# Longest Common Subsequence

- Subsequence

  Sequence X = <A,B,C,B,D,A,B>

  Subsequence of X : <B,C,D,B>, <A,B,D> … *(원소 순서 유지)*

- Common subsequence

  Sequences: X = <A,B,C,B,D,A,B> , Y=<B,D,C,A,B,A>

  Common subsequences: <A,B> , <B,C,A> , <B,C,A,B>, <B,D,A,B> …

  Longest common subsequence: <B,C,A,B>, <B,D,A,B> (length = 4)

# Longest Common Subsequence

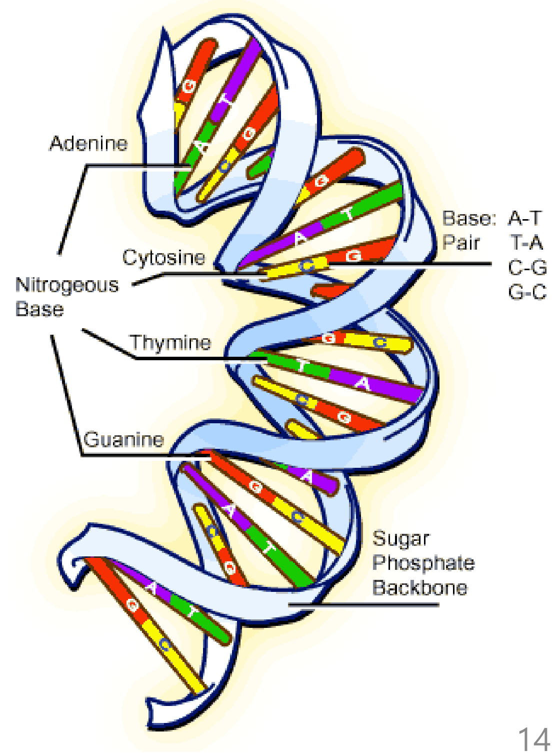
- LCS problem
  - Given: X = < x1, x2, … , $x_m$>, Y = <y1, y2, … , $y_n$>
  - Find: a longest common subsequence Z = <z1, z2, … , zk >

- Application: 유전자의 유사성 판별 문제
  - DNA : {A, C, G, T} 로 구성된 sequence

    A: adenine, C:cytosine, G:guanine, T:thymine

    S1 = <ACCGGTCGAGTGCGCGAGTTCAGTC>

    S2 = <GTCGTAGTCAAGTCGTAGCTCAGTT>




- Brute-force approach
  - 발생 가능한 모든 subsequence 를 생성하여 비교
  - $_mC_1 + {_mC_2} + \cdots + {_mC_m} = 2^m$ (exponential time)

# Longest Common Subsequence

**(Step 1) Characterizing**

$X = < x_1, x_2, \ldots , x_m>$

$X_i = < x_1, x_2, \ldots , x_i >$ ; i-th ***prefix*** of X

Theorem 15.1 (Optimal substructure of LCS)

$X = < x_1, x_2, \ldots , x_m>$, $Y= <y_1, y_2, \ldots , y_n>$

LCS $Z = < z_1, z_2, \ldots , z_k >$

1. If $x_m = y_n$ , then $z_k=x_m=y_n$ and $Z_{k-1}$ is an LCS of $X_{m-1}$ and $Y_{n-1}$
2. If $x_m \neq y_n$ , then $z_k \neq x_m$ implies that Z is an LCS of $X_{m-1}$ and Y
3. If $x_m \neq y_n$ , then $z_k \neq y_n$ implies that Z is an LCS of X and $Y_{n-1}$

# Longest Common Subsequence

## (Step 2) Recursive solution

*c*[*i, j*] : length of an LCS of the sequences $X_i$ and $Y_j$

$$c[i, j] = \begin{cases} 0 & i = 0 \; or \; j = 0 \\ c[i-1, j-1] + 1 & i, j > 0 \; and \; x_i = y_j \\ \max(c[i, j-1], c[i-1, j] & i, j > o \; and \; x_i \neq y_j \end{cases}$$

| | *j* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| *i* | $y_j$ | | B | D | C | A | B | A |
| 0 | $x_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↖ 1 | ←1 | ↖ 1 |
| 2 | B | 0 | ↖ 1 | ←1 | ←1 | ↑ 1 | ↖ 2 | ←2 |
| 3 | C | 0 | ↑ 1 | ↑ 1 | ↖ 2 | ←2 | ↑ 2 | ↑ 2 |
| 4 | B | 0 | ↖ 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ 3 | ←3 |
| 5 | D | 0 | ↑ 1 | ↖ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 |
| 6 | A | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ 3 | ↑ 3 | ↖ 4 |
| 7 | B | 0 | ↖ 1 | ↑ 2 | ↑ 2 | ↑ 3 | ↖ 4 | ↑ 4 |

# Longest Common Subsequence

## (Step 3) Computing

- Data member

  c[i,j]: i=0,…,m , j=0,…,n : length

  b[i,j]: i=1,…,m, j=1,…n : optimal pointer (↖, ↑, ← )

- 알고리즘: LCS-LENGTH(X,Y)
  - running time: O(mn)

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
      do c[i, 0] ← 0
for j ← 0 to n
      do c[0, j] ← 0
for i ← 1 to m
      do for j ← 1 to n
               do if x_i = y_j
                     then c[i, j] ← c[i − 1, j − 1] + 1
                          b[i, j] ← "↖"
                     else if c[i − 1, j] ≥ c[i, j − 1]
                             then c[i, j] ← c[i − 1, j]
                                  b[i, j] ← "↑"
                             else c[i, j] ← c[i, j − 1]
                                  b[i, j] ← "←"
return c and b
```

# Longest Common Subsequence

**(Step 4) Constructing**

PRINT-LCS(b, X, i, j)

- running time: O(m+n)

```
PRINT-LCS(b, X, i, j)
if i = 0 or j = 0
    then return
if b[i, j] = "↖"
    then PRINT-LCS(b, X, i − 1, j − 1)
         print x_i
elseif b[i, j] = "↑"
    then PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

# Longest Common Subsequence

(Q)

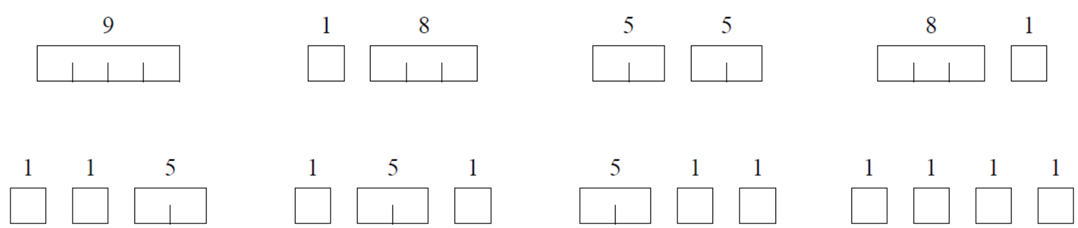
# Rod Cutting

- Problem
  - Input: length $n$ , table of price $p_i$ $(i = 1, \cdots, n)$
  - Output: maximum revenue $r_n$
    - revenue = sum of the prices for the individual rods
  - Complexity
    - $2^{n-1}$ cases for a rod of length $n$

<table of price>

| length $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| price $p_i$ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

(ex) n = 4




: 8 cases

: maximum revenue = $p_2 + p_2$ = 5 + 5 = 10

# Rod Cutting

**(Step 1) Characterizing**

$r_n$ : maximum revenue for a rod of length $n$

→ maximum of

- $p_n$: the price we get by not making a cut,
- $r_1 + r_{n-1}$: the maximum revenue from a rod of 1 inch and a rod of $n-1$ inches,
- $r_2 + r_{n-2}$: the maximum revenue from a rod of 2 inches and a rod of $n-2$ inches, ...
- $r_{n-1} + r_1$.

| length $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| price $p_i$ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

| $i$ | $r_i$ | optimal solution |
|---|---|---|
| 1 | 1 | 1 (no cuts) |
| 2 | 5 | 2 (no cuts) |
| 3 | 8 | 3 (no cuts) |
| 4 | 10 | $2+2$ |
| 5 | 13 | $2+3$ |
| 6 | 17 | 6 (no cuts) |
| 7 | 18 | $1+6$ or $2+2+3$ |
| 8 | 22 | $2+6$ |

# Rod Cutting

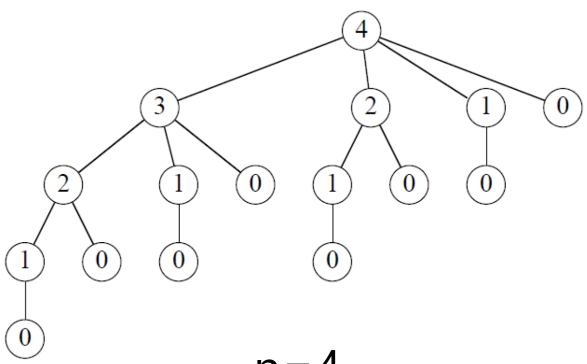
## (Step 2) Recursive solution

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$$

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

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

Running time
: T(n) = $\Theta(2^n)$



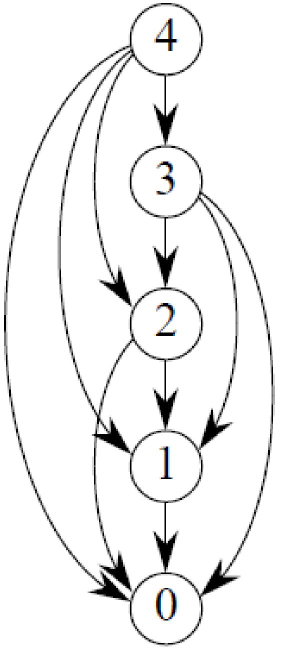

n=4

# Rod Cutting

## (Step 3) Computing

```
BOTTOM-UP-CUT-ROD(p, n)
  let r[0..n] be a new array
  r[0] = 0
  for j = 1 to n
      q = -∞
      for i = 1 to j
          q = max(q, p[i] + r[j - i])
      r[j] = q
  return r[n]
```

Running time
: $T(n) = \Theta(n^2)$




$r[1] = p[1] + r[0]$
$r[2] = \max(p[1] + r[1], p[2] + r[0])$
$r[3] = \max(p[1] + r[2], p[2] + r[1], p[3] + r[0])$
……

# Rod Cutting

## (Step 4) Constructing

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
  let r[0..n] and s[0..n] be new arrays
  r[0] = 0
  for j = 1 to n
      q = −∞
      for i = 1 to j
          if q < p[i] + r[j − i]
              q = p[i] + r[j − i]
              s[j] = i
      r[j] = q
  return r and s
```

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

# Rod Cutting

(Ex) n=8

| length $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| price $p_i$ | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $r[i]$ | | | | | | | | | |
| $s[i]$ | | | | | | | | | |