Rod Cutting Problem

(with Dynamic Programming)

201800000 OOO

Reference : 주니온TV 아무거나연구소 @YouTube (배준현 경북대학교 컴퓨터학부 교수)

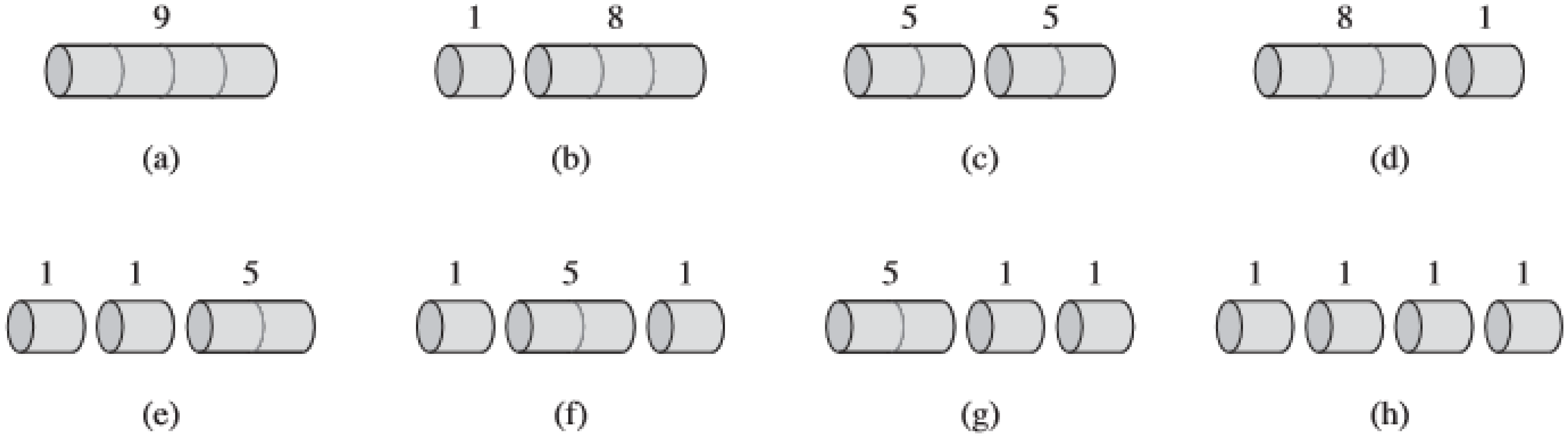
Rod Cutting Problem

- Given a rod of length n
- Pieces of rod of length i = 1, 2, 3, …, n
- Prices of each piece of rod $p_i = p_1, p_2, p_3, \ldots, p_n$
- Determine the maximum value obtainable(r_n) by cutting up the rod and selling the pieces.

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

Rod Cutting Problem

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



Recurrence Relation

(Divide and Conquer)

- Ending Condition : $n = 1$
  - $r_1 = p_1$
- Recurrence Condition : $n > 1$
  - $r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$

    $= \max_{1 \le i \le n}(p_i + r_{n-i})$
- An optimal solution to an instance of a problem contains optimal solutions to all substance, therefore, the principle of optimality is applied in this problem.

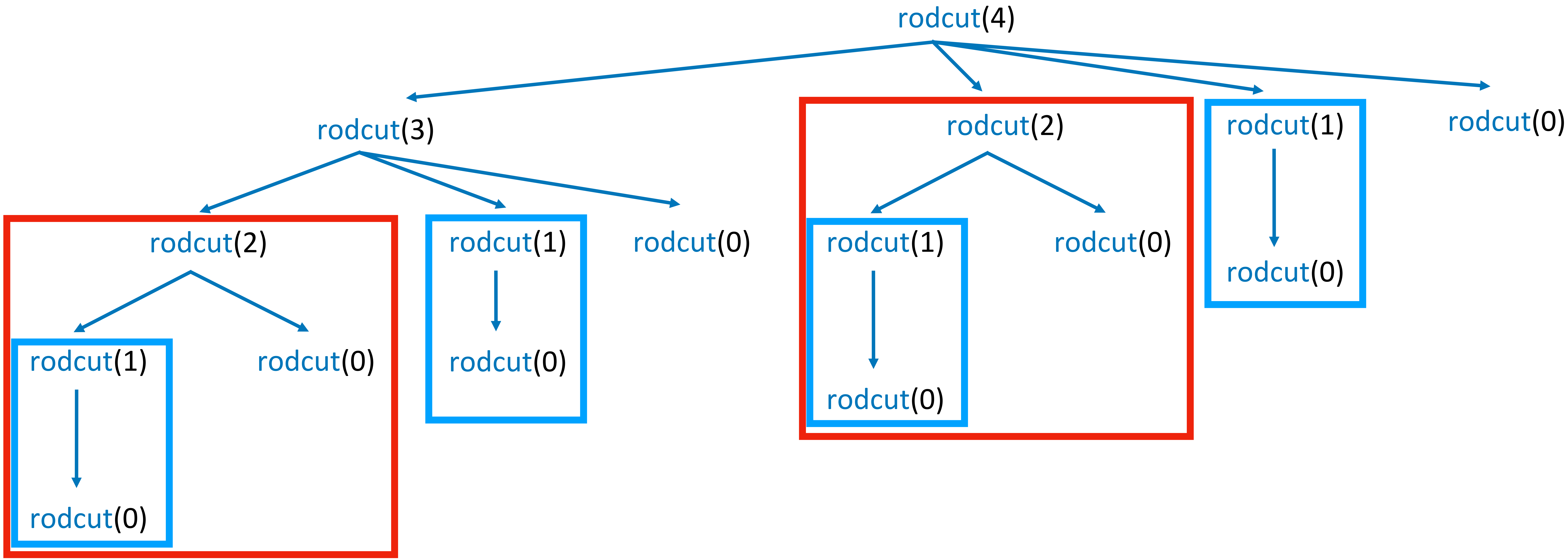
Recurrence Relation

(Divide and Conquer)

```python
# Divide and Conquer : Recurrence Relation
def rodcut(n, p):
    if n == 0:
        return 0
    else:
        r = -1
        for i in range(1, n + 1):
            r = max(r, p[i] + rodcut(n - i, p))
        return r
```

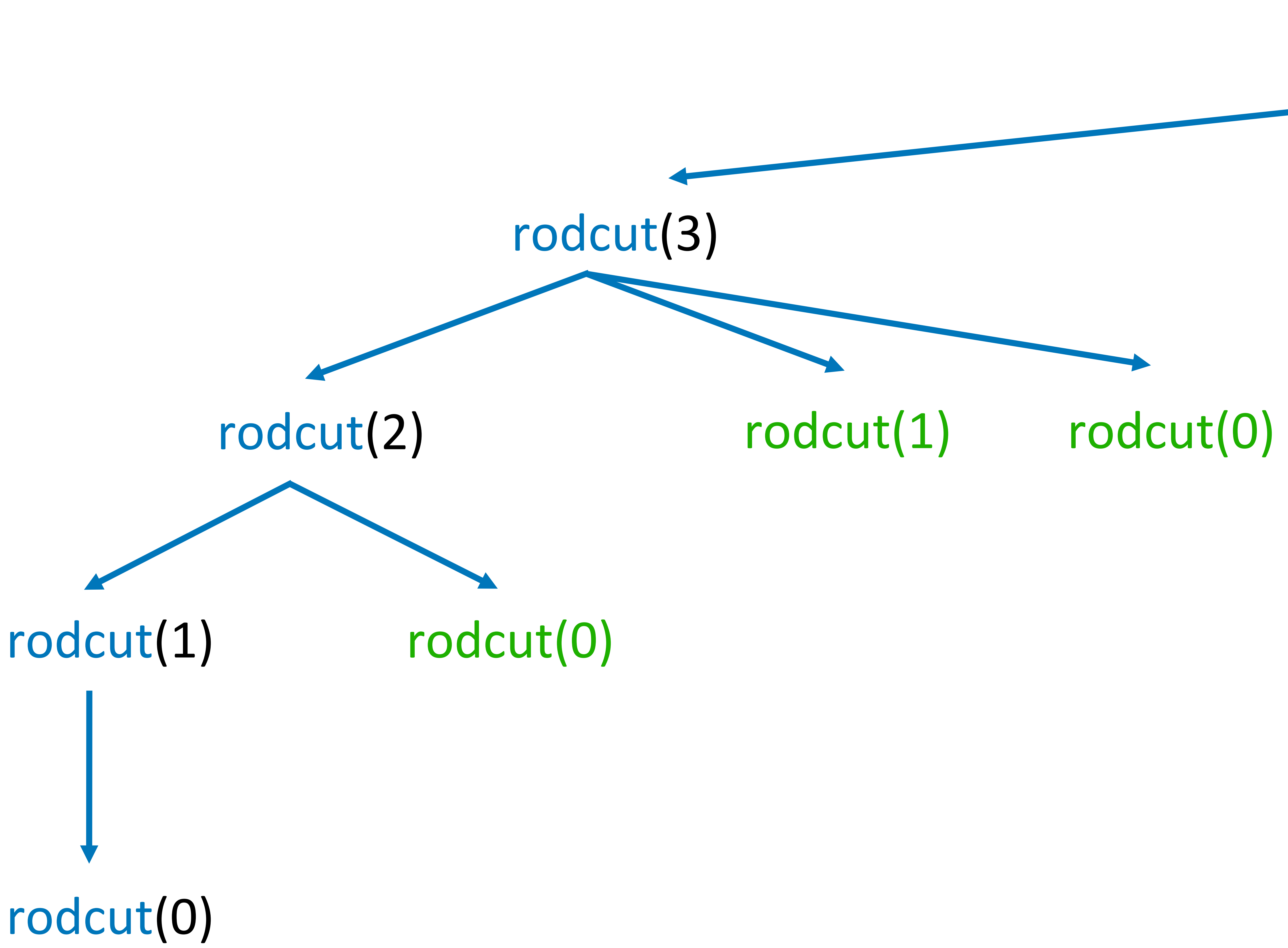
Recurrence Relation

(Divide and Conquer)



Top-down : Memoization

(Store result of each subproblem in array or table)



length	1	2	3	4
profit	1	5	8	9

r	0	1	2	3	4
profit	0	1	5	8	10

Top-down : Memoization

(Store result of each subproblem in array or table)

```python
# Dynamic Programming : Memoization
def rodcut(n, p, r):
    if r[n] < 0:
        if n == 0:
            r[n] = 0
        else :
            r[n] = -1
            for i in range(1, n + 1):
                r[n] = max(r[n], p[i] + rodcut(n - i, p, r))
    return r[n]
```

Bottom-up : Tabulization

(Solving by not recurrence but loop)

```python
# Dynamic Programming : Tabulization
def rodcut(n, p):
    r = [0] * (n + 1)
    for i in range(1, n + 1):
        r[i] = -1
        for j in range(1, i + 1):
            r[i] = max(r[i], r[i - j] + p[j])
    return r[n]
```

Bottom-up : Tabulization

(Solving by not recurrence but loop)

p[1]

r[1] + p[1]
p[2]

r[2] + p[1]
r[1] + p[2]
p[3]

r[3] + p[1]
r[2] + p[2]
r[1] + p[3]
p[4]

length	1	2	3	4
profit	1	5	8	9

r	0	1	2	3	4
profit	0	1	5	8	10