Dynamic Programming

Divide and conquer:

- Divide the subproblem into independent subproblems.
- solve each subproblem independently
- combine the solutions.

Dynamic Programming

- Divide the subproblem into a series of overlapping subproblems.
- solve them ⟸ need extra care
- combine the solutions.

Dynamic Programming: It solves optimization problems.

Main idea

- compute the solutions to the subproblems once
- Store the solution of the subproblems in a table / dictionary
- They can be reused (repeatedly) in a later stage.

⟹ It trades space for time

Rod cutting problem

Input: A rod of length n unit.

A table of prices p_i for $i = 1, 2, \ldots, n$

where p_i is the price of a rod of length i

Goal: Find the maximum revenue

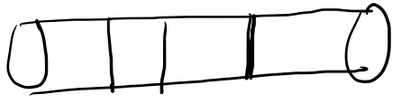
cut the rod into different pieces and sell it.

Ex^m

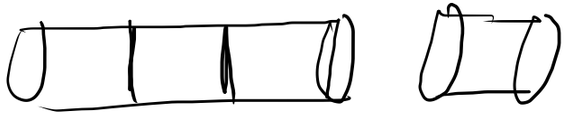
length i	1	2	3	4	5	6	. . . .
price p_i	1	5	8	9	10	17	

The shop owner has a rod of length 4

Solution



9

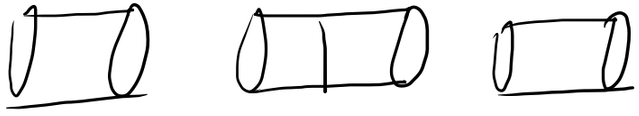


8 + 1

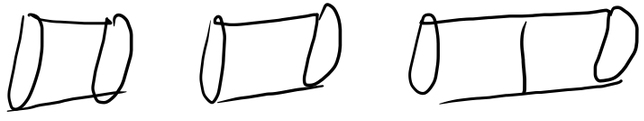
1 + 8

5 + 5

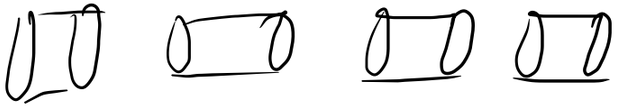
5 + 1 + 1



1 + 5 + 1

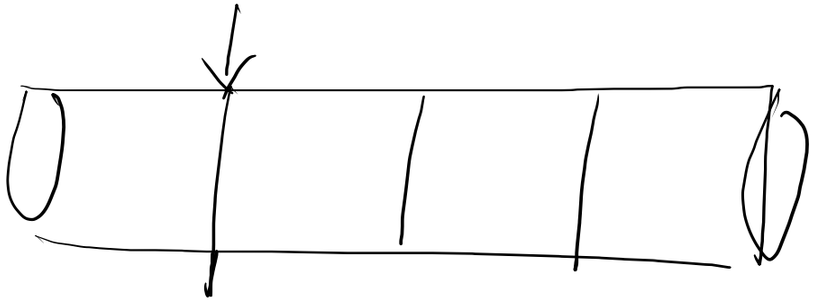


1 + 1 + 5



1 + 1 + 1 + 1

A simple algorithm



- Try all possible cuts of the rod
- Track the best solution.

running time :- $2^{n-1} \approx O(2^n)$

can we do better?

maximum revenue = r_n

$r_i \leftarrow$ maximum revenue for a rod of length i

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

In general:

- making an initial cut into two pieces of length i and $n-i$
- optimally cutting these two pieces
- we donot know ahead of time which initial cut gives the optimum revenue.
- we have to consider all possible value for i and pick that with the maximum revenue.

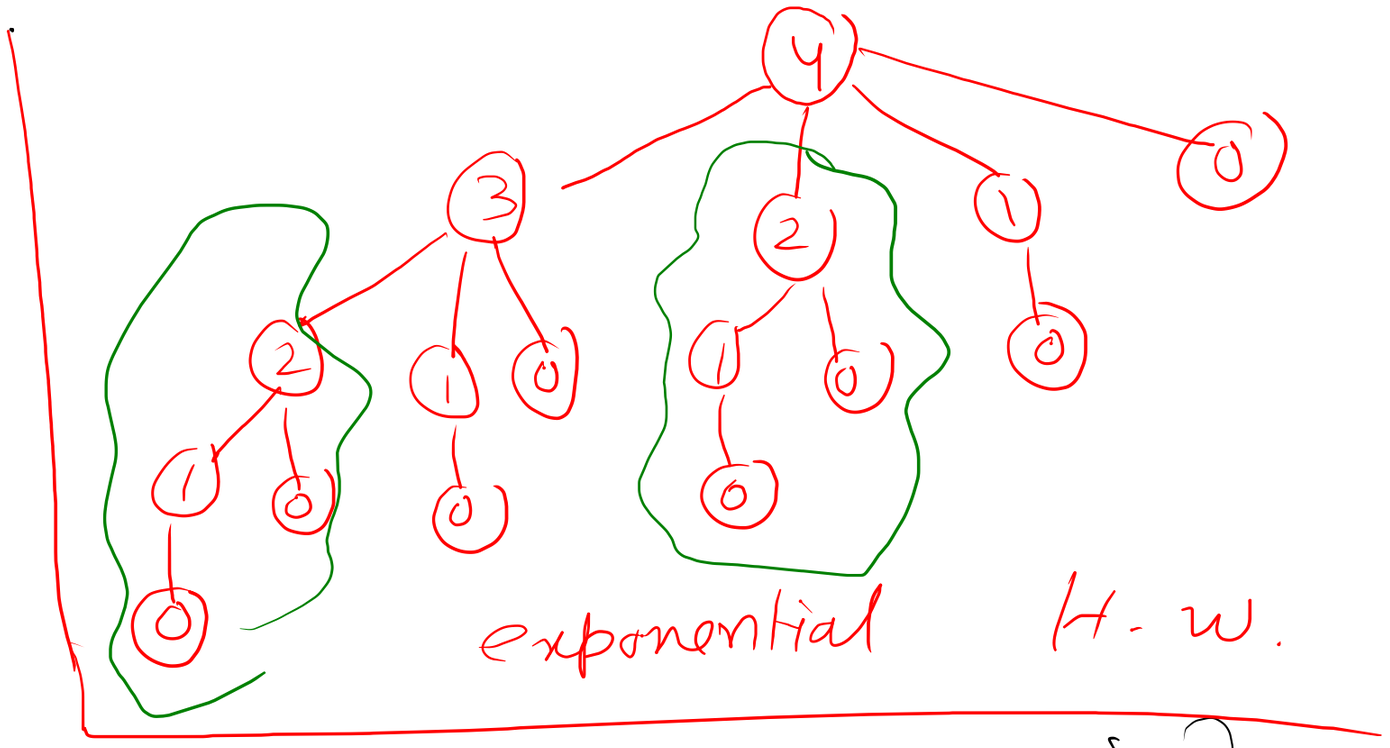
$$r_n = \max \{ p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1 \}$$

Question:
Do we really need two subproblems?

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

Algorith rod cutting

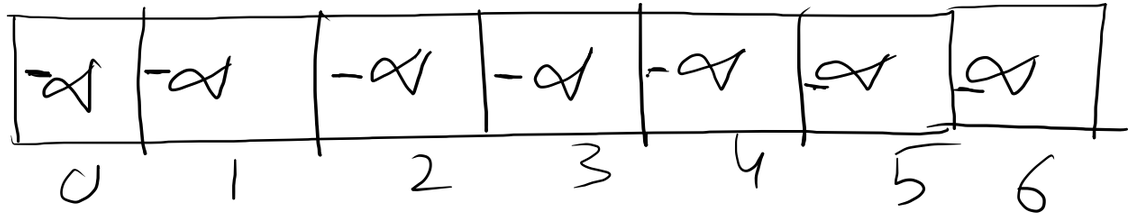
```
rod-cut (P, n)
  if  n == 0
      return 0

  q = -∞
  for i = 1 to n
      q = max { q, P[i] + rod-cut (P, n-i) }

  return q
```



Problem: several subproblems are called many times.

Top-down approach

r




```
memoized-rod-cut(P, n)
  let r[0,..,n] be a new array
  for i = 1 to n
      r[i] ← -∞
  return memoized-rod-cut-aux(P, n)
```

```
memoized-rod-cut-aux(P, n)
  if r[n] ≥ 0
      return r[n]
  if n == 0
      q = 0
  else
      q = -∞
      for i = 1 to n
          q = max{ q, P[i] + memoized-cut-rod-aux(P, n) }
  r[n] = q
  return q
```

running time

# of independent subproblems

× time taken without recursive call

Two ways to solve the problem

1. Top-down with memoization
2. Bottom up with tabulation