Dynamic Programming (DP)

DP = subproblems + reuse

DP = recursion + memorization

time = no. of subproblems * time per subproblem (treating recursive calls as O(1))

Fabonacci numbers: $F_1 = F_2 = 1, F_n = F_{n-1} + F_{n-2}$ — O(n)

Naive algorithm:

```
fib(n):
    if n ≤ 2:
        f = 1
    else:
        f = fib(n-1) + fib(n-2)
    return f
```

Memorized DP algorithm:

```
memo = {}
fib(n):
    if n in memo:
        return memo(n)
    if n ≤ 2:
        f = 1
    else:
        f = fib(n-1) + fib(n-2)
    memo[n] = f
    return f
```

Runtime: $O(n)$

Bottom-up DP algorithm:

```
fib = []
for k = 1 to n:
    if k ≤ 2:
        f = 1
    else:
        f = fib[k-1] + fib[k-2]
```

return fib[n]

Here we use O(n) memory, we can reduce it to O(1).

Rod Cutting — O(n^2)

$length - i$	1	2	3	4	5	6	7	8	9
$price - p_i$	1	5	8	9	10	17	17	20	24
$revenue - r_i$	1	5	8	10	13	17	18	22	25

There are 2^{n-1} ways of cutting the rod, since we can each cut or not cut at each of the $n-1$ postitions. (Considering permutations 1+1+2 and 1+2+1 are the same, $2^{\Theta(\sqrt{n})}$)

optimal revenue $r_n = p_i + r_{n-i}$

$r_n = max(p_1 + r_{n-1}, p_2 + r_{n-2}, ..., p_n + r_0)$ where we define $r_0 = 0$

CUTROD(n)

if n = 0:

  return 0

q = - ∞

for i = 1 to n:

  q = max(q, p_i + CUTROD(n-i))

return q

Runtime: $T(n) = 1 + \Sigma_{j=0}^{n-1} T(j) = 2^n$

CUTROD(n)

if n in memo:

  return memo[n]

if n = 0:

  return 0

q = - ∞

for i = 1 to n:

  q = max(q, p_i + CUTROD(n-i))

memo[n] = q

return q

Runtime: no. of subproblems * time per subproblem = $n \cdot \Theta(n) = \Theta(n^2)$

Bottom-up version

```
r[0] = 0
for j = 1 to n:
    q = - ∞
    for i = 1 to j:
        if q < p[i] + r[j-1]:
            q = p[i] + r[j-1]
            s[j] = i
    r[j] = q
m = n
while m > 0:
    print(s[m])
    m = m - s[m]
return r[n]
```

Remark: It is easy to also output the actual partition.

Another possible solution $r_n = \max_{i \leq j}(r_{i-1} + P_{j-i+1} + r_{n-j})$, $O(n^3)$

Longest Common Subsequence — O(nm)

Given strings $X = x_{1\ldots m}$, $Y = y_{1\ldots n}$, find the (length of) the longest common subsequenc LCS

HYPERLINKING

DOLPHINSPEAK

Subproblems: for each prefix (i=0, 1, ..., m) of X and each prefix (i=0, 1, ..., m) of Y, what is LCS?

Optimal substructure: $X = x_{1\ldots m}, Y = y_{1\ldots n}$, $z = z_{1\ldots k}$

1. If $x_m = y_n$, then $z_k = x_m = y_n$, then z_{j-1} is the LCS of x_{m-1}, y_{j-1}
2. If $x_m \neq y_n$, then
    a. If $z_k \neq x_m$, then z_{j-1} is the LCS of x_{m-1}, y_j
    b. If $z_k \neq y_n$, then z_{j-1} is the LCS of x_m, y_{n-1}

Proof 1. If $z_k \neq x_m$, then we could append $x_m = y_n$ to end of Z, in contradiction.
Therefore $z_k = x_m = y_n$. ~~[illegible]~~ Z_{k-1} is a common subsequence of X_{m-1} and Y_{n-1}.
If it is not the longest, then there is another common subseq of length $\geq k$, and by appending $z_k = x_m = y_n$, we get a common subseq of X and Y of len $\geq k+1$, in contradiction
2a. Z is a common subseq of X_{m-1} and Y. If there is another common subseq of X_{m-1} & Y of length $> k$, it's also of X & Y, in contradiction.
2b. Similarly. ∎

Rucurrence:

$$C[i,j] = \begin{cases} 0 & i = j = 0 \\ C[i-1, j-1] + 1 & i,j > 0, x_i = y_j \\ max(C[i-1,j], C[j-1,i]) & i,j > 0, x_i \neq y_j \end{cases}$$

Runtime: O(nm) subproblems → O(nm)

Knapsack — O(nS)

- list of n items each of size s_i and value v_i
- knapsack of size S
- find maximum value

1. Subproblems: items $1, ..., i$ and remaining capacity $X \leq S$
   $\theta(n \cdot S)$ subproblems
2. Guess: is item i in or not?
3. Recurrence: $DP(X, i) = max(DP(X, i-1), DP(X - S_i, i-1) + v_i)$
4. Runtime: $\theta(n \cdot S)$