Dynamic Programming

Recursion:
To solve a problem for one input,
solve on other inputs
and combine results.

Benefits: often easy to find
Problem: usually exponential time.

How can we make it faster?

Memoization: whenever you compute
$f(x)$ in some recursive call,
store the answer. Next time
you call $f(x)$ just return it.

Memoization: $\text{time} = \text{possible inputs} \times \text{time per call}$

Recursion: $\text{time} = \text{# paths from input to base case}$
Can think of inputs as a DAG. 
A \rightarrow B if B's recursion uses A.

Fibonacci:

Recursion time = # paths = \phi^n \approx 1.6^n

Memoization time = (# inputs) \times (time per input)
= n \times n = n^2 \text{ bit operations}

Bottom-up DP: fill from left to right, looking at each edge.

Can either pull or push along edges.

Pull: compute a node's value when you visit it, based on previous values

Push: when you visit a computed node, update future nodes that use it.
(In Fibonacci: add your value to theirs)

Generally equivalent. Sometimes only have easy access to out- or in-edges.
Interval Scheduling:

Want to compute $\text{Sched}(I)$
where $I = \text{set of } (s_i, f_i, w_i)$ pairs
maximize $\sum_{i \in S} w_i$
for $S \subseteq I$ non-overlapping.

Naive recursion:

$\text{Sched}(I)$

Let $i = \text{first elt of I}$

Return $\min$ of

not chosen: $\text{Sched}(I \setminus i)$,

chosen: $\text{Sched}(I \setminus i \setminus \text{or anything conflicting with } i \setminus S) + w_i$.

Problem: $2^n$ possible inputs.

Solution: If I sorted by $f_i$, then

only $n+1$ inputs ever happen:
(suffix or I sorted by $s_i$)

$\Rightarrow$ memoized time is $n \cdot (\text{time per input})$

$n^2$ naively

$n$ more carefully
$\text{Sched}(\text{index in } I, f \text{ or last chosen})$
DAG: sort by $f_i$.

1 $\to$ j if $f_i \leq S_j$, or weight $w_i$.

$S \to$ everything weight $0$

everything $\to$ $+$ weight $w_j$

Answer = max weight $S \to +$ path.

Path $S \to i_1 \to i_2 \to \ldots \to i_k \to +$

- weight = $0 + w_1 + w_2 + \ldots + w_k$
  = total value of scheduling then

- $f_{i_1} \leq S_1 < f_{i_2} \leq S_2 < \ldots$

$\Rightarrow$ is valid schedule.

Many DP problems have this form:

- Convert to a DAG
- Find max-weight (or min-weight) $S \to +$ path.
- Time = # edges in DAG.
Stamps
values \( s_1, \ldots, s_n \)
want collection of value \( c \)

w/ freshest stamps

What is the DAG?

nodes = value
\[ x \rightarrow x + s_i \quad \forall i, x \text{ of weight } 1 \]

Answer = shortest \( 0 \rightarrow c \) path.
Time = \( \# \) edges = \( c \cdot n \).