CSE 202
Dynamic Programming

An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.
Chapter 6
Dynamic Programming
Algorithm Design Paradigms

- **Exhaustive Search**
- **Greedy Algorithms**: Build a solution incrementally piece by piece
- **Divide and Conquer**: Divide into parts, solve each part, combine results
- **Dynamic Programming**: Divide into subtasks, perform subtask by size. Combine smaller subtasks to larger ones
Dynamic Programming History

Bellman. Pioneered the systematic study of dynamic programming in the 1950s.

Etymology.

- Dynamic programming = planning over time.
- Secretary of Defense was hostile to mathematical research.
- Bellman sought an impressive name to avoid confrontation.
  - "it's impossible to use dynamic in a pejorative sense"
  - "something not even a Congressman could object to"

Richard E. Bellman

From Wikipedia, the free encyclopedia

Richard Ernest Bellman (August 26, 1920 – March 19, 1984) was an applied mathematician, celebrated for his invention of dynamic programming in 1953, and important contributions in other fields of mathematics.

Contents [show]

Biography [edit]

Bellman was born in 1920 in New York City, where his father John James Bellman ran a small grocery store on Bergen Street near Prospect Park in Brooklyn. Bellman completed his studies at Abraham Lincoln High School in 1937,[1] and studied mathematics at Brooklyn College where he received a BA in 1941. He later earned an MA from the University of Wisconsin–Madison. During World War II he worked for a Theoretical Physics Division group in Los Alamos. In 1946 he received his Ph.D. at Princeton under the supervision of Solomon Lefschetz.[2]

He was a professor at the University of Southern California, a Fellow in the American Academy of Arts and Sciences (1975), and a member of the National Academy of Engineering (1977).

He was awarded the IEEE Medal of Honor in 1979, "for contributions to decision processes and control system theory, particularly the creation and application of dynamic programming". His key work is the Bellman equation.
Dynamic Programming Applications

Areas.
- Bioinformatics.
- Control theory.
- Information theory.
- Operations research.
- Computer science: theory, graphics, AI, systems, ...

Some famous dynamic programming algorithms.
- Viterbi for hidden Markov models.
- Unix diff for comparing two files.
- Smith-Waterman for sequence alignment.
- Bellman-Ford for shortest path routing in networks.
- Cocke-Kasami-Younger for parsing context free grammars.
Dynamic Programming (DP): A Simple Example

**Problem:** Compute the n-th Fibonacci number 1, 1, 2, 3, 5, 8, 13, 21, ...

**Recursive Solution**

```
function Fib1(n)
    if n = 1 return 1
    if n = 2 return 1
    return Fib1(n-1) + Fib1(n-2)
```

**Running Time:**

\[ T(n) = T(n-1) + T(n-2) + 1 \]
Dynamic Programming (DP): A Simple Example

**Problem:** Compute the n-th Fibonacci number 1, 1, 2, 3, 5, 8, 13, 21, ...

**Recursive Solution**

```python
function Fib1(n)
    if n = 1 return 1
    if n = 2 return 1
    return Fib1(n-1) + Fib1(n-2)
```

**Running time:** $O(c^n)$

**Running Time:**

$T(n) = T(n-1) + T(n-2) + 1$

$T(n) = O(c^n)$

$c^2 - c - 1 = 0$  
$c = 1.618...$
Dynamic Programming (DP): A Simple Example

**Problem:** Compute the n-th Fibonacci number 1, 1, 2, 3, 5, 8, 13, 21, ...

**Recursive Solution**

```python
function Fib1(n)
    if n = 1 return 1
    if n = 2 return 1
    return Fib1(n-1) + Fib1(n-2)
```

**Running time:** $O(c^n)$

**Dynamic Programming Solution**

```python
function Fib2(n)
    Create an array fib[1..n]
    fib[1] = 1
    fib[2] = 1
    for i = 3 to n:
        fib[i] = fib[i-1] + fib[i-2]
    return fib[n]
```

**Running Time:**

$$T(n) = T(n-1) + T(n-2) + 1$$

$$T(n) = O(c^n)$$

$c^2 - c - 1 = 0$  \quad $c = 1.618...$
Dynamic Programming (DP): A Simple Example

Problem: Compute the n-th Fibonacci number

Recursive Solution

function Fib1(n)
if n = 1 return 1
if n = 2 return 1
return Fib1(n-1) + Fib1(n-2)

Running time: $O(c^n)$

Running Time:

$T(n) = T(n-1) + T(n-2) + 1$
$T(n) = O(c^n)$

Dynamic Programming Solution

function Fib2(n)
Create an array fib[1..n]
fib[1] = 1
fib[2] = 1
for i = 3 to n:
    fib[i] = fib[i-1] + fib[i-2]
return fib[n]

Running time: $O(n)$

Running Time:

$T(n) = O(n)$
Why does DP do better?

**Problem:** Compute the n-th Fibonacci number

**Recursive Solution**

```python
function Fib1(n)
    if n = 1 return 1
    if n = 2 return 1
    return Fib1(n-1) + Fib1(n-2)
```

**Running time:** $O(c^n)$

**Dynamic Programming Solution**

```python
function Fib2(n)
    Create an array fib[1..n]
    fib[1] = 1
    fib[2] = 1
    for i = 3 to n:
        fib[i] = fib[i-1] + fib[i-2]
    return fib[n]
```

**Running time:** $O(n)$
Dynamic Programming

Main Steps:

1. Divide the problem into subtasks

2. Define the subtasks *recursively* (express larger subtasks in terms of smaller ones)

3. Find the right order for solving the subtasks (but do not solve them recursively!)
Dynamic Programming

Dynamic Programming Solution

```python
function Fib2(n)
    Create an array fib[1..n]
    fib[1] = 1
    fib[2] = 1
    for i = 3 to n:
        fib[i] = fib[i-1] + fib[i-2]
    return fib[n]
```

Running time: \( O(n) \)

Main Steps:

1. Divide the problem into subtasks: compute \( \text{fib}[i] \)

2. Define the subtasks recursively (express larger subtasks in terms of smaller ones)

3. Find the right order for solving the subtasks \((i = 1,..,n)\)
Dynamic Programming

- String Reconstruction
- ...
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

Example:
$x = \text{anonymousarrayofletters} : \text{True}$
$x = \text{anhuymousarrayofhetters} : \text{False}$
String reconstruction

Given: document $x[1..n]$ : an array of characters
   dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word

Is $x$ a sequence of valid words?

Example:
$x = \text{anonymousarrayofletters} : \text{True}$
$x = \text{anhuymousarrayofhetters} : \text{False}$

STEP 1: Define subtask
$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
   $\text{False}$ otherwise
Output of algorithm = $S(n)$
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

Example:
$x = \text{anonymousarrayofletters} : \text{True}$
$x = \text{anhuymousarrayofhetters} : \text{False}$

STEP 1: Define subtask
$S(k) = \text{True} \quad \text{if} \quad x[1..k] \text{ is a valid sequence of words}$
$\quad \text{False} \quad \text{otherwise}$
Output of algorithm = $S(n)$

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
Given: document $x[1..n]$ : an array of characters
   dictionary function $dict(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \begin{cases} 
\text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
\text{False} & \text{otherwise}
\end{cases}$

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $S$  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
Given: document \( x[1..n] \) : an array of characters

dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[ S(k) = \text{True} \quad \text{if} \quad x[1..k] \quad \text{is a valid sequence of words} \]

\[ S(k) = \text{False} \quad \text{otherwise} \]

**STEP 2: Express Recursively**

\[ S(k) = \text{True iff } \exists \ j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word} \]
Given: document $x[1..n]$ : an array of characters
dictionary function $dict(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = True$ if $x[1..k]$ is a valid sequence of words
False otherwise

**STEP 2: Express Recursively**

$S(k) = True$ iff $\exists j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]

<table>
<thead>
<tr>
<th>k</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A</td>
<td>N</td>
<td>O</td>
<td>N</td>
<td>Y</td>
<td>M</td>
<td>O</td>
<td>U</td>
<td>S</td>
<td>A</td>
<td>R</td>
<td>A</td>
<td>Y</td>
<td>O</td>
<td>F</td>
<td>L</td>
<td>E</td>
<td>T</td>
<td>E</td>
<td>R</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Wednesday, April 23, 14
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$\text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists$ $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$\text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True} \iff \exists j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}$

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
| $S$ | T | T |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
**String reconstruction**

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**
$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
False otherwise

**STEP 2: Express Recursively**
$S(k) = \text{True}$ iff $\exists$ $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**
$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]
String reconstruction

Given: document x[1..n] : an array of characters
dictionary function dict(w): returns true if w is a valid word
Is x a sequence of valid words?

STEP 1: Define Subtask
S(k) = True if x[1..k] is a valid sequence of words
False otherwise

STEP 2: Express Recursively
S(k) = True iff ∃ j < k s.t. S(j) is True, and x[j+1..k] is a valid word

STEP 3: Order of Subtasks
S(1), S(2), S(3), ..., S(n) [ Do not solve recursively! ]
String reconstruction

Given: document $x[1..n]$ : an array of characters
    dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
    False otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists$ $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|11|12|13|14|15|16|17|18|19|20|21|22|23|
| $S$ | T | T | T | T | T | F |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
String reconstruction

Given: document \( x[1..n] \) : an array of characters

dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[
S(k) = \begin{cases} 
  \text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
  \text{False} & \text{otherwise}
\end{cases}
\]

**STEP 2: Express Recursively**

\[
S(k) = \text{True iff } \exists \ j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}
\]

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), ..., S(n) \) [ Do not solve recursively! ]

| \( x \) | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| \( k \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| \( S \) | T | T | T | T | F | F |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words

$S(k) = \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists$ $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMOUS</th>
<th>ARRAY</th>
<th>LETTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T T T T T T F F F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**String reconstruction**

Given: document $x[1..n]$ : an array of characters

dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word

Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words

$S(k) = \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists$ $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]
Given: document \( x[1..n] \): an array of characters

dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\( S(k) = \text{True} \quad \text{if} \quad x[1..k] \text{ is a valid sequence of words} \)

\( \text{False} \quad \text{otherwise} \)

**STEP 2: Express Recursively**

\( S(k) = \text{True} \quad \text{iff} \quad \exists \ j < k \quad s.t. \quad S(j) \text{ is True}, \quad \text{and} \quad x[j+1..k] \text{ is a valid word} \)

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), \ldots, S(n) \) \[ \text{Do not solve recursively!} \]
String reconstruction

Given: document $x[1..n]$ : an array of characters
    dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
    False otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists$ j < k s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]

---

<table>
<thead>
<tr>
<th>k</th>
<th>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>ANONYMOSARAYOFFLETTERS</td>
</tr>
<tr>
<td>S</td>
<td>TTTTFFFTTT</td>
</tr>
</tbody>
</table>

---

Wednesday, April 23, 14
String reconstruction

Given: document $x[1..n]$ : an array of characters

dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word

Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words

False otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists \ j < k \text{ s.t. } S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $S$ | T | T | T | T | T | F | F | F | F | F | F | F | F | F | T | T | F | T | T | F | T | T | F |
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$\text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists \ j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
| $S$ | T | T | T | T | F | F | F | F | F | T | T | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word

Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words

$S(k) = \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMOUS ARRAY OF LETTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23</td>
</tr>
<tr>
<td>$S$</td>
<td>T   T   T   F   F   F   F   T   T   F   F   F   F   T</td>
</tr>
</tbody>
</table>
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \begin{cases} 
\text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
\text{False} & \text{otherwise}
\end{cases}$

**STEP 2: Express Recursively**

$S(k) = \text{True} \iff \exists j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}$

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]

<table>
<thead>
<tr>
<th>$k$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$\text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists \ j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]
String reconstruction

Given: document \( x[1..n] \): an array of characters

dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[ S(k) = \text{True} \quad \text{if} \quad x[1..k] \text{ is a valid sequence of words} \]

\[ \text{False} \quad \text{otherwise} \]

**STEP 2: Express Recursively**

\[ S(k) = \text{True} \iff \exists \ j < k \ \text{s.t.} \ S(j) \text{ is True, and} \ x[j+1..k] \text{ is a valid word} \]

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), \ldots, S(n) \) [ Do not solve recursively! ]

<table>
<thead>
<tr>
<th>( x )</th>
<th>A N O N Y M O U S A R R A Y O F L E T T E R S</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23</td>
</tr>
<tr>
<td>( S )</td>
<td>T T T T T F F F F F T T F F F F F T F F F F</td>
</tr>
</tbody>
</table>
Given: document \( x[1..n] \): an array of characters

dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word

Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[ S(k) = \text{True} \quad \text{if} \quad x[1..k] \text{ is a valid sequence of words} \]
\[ S(k) = \text{False} \quad \text{otherwise} \]

**STEP 2: Express Recursively**

\[ S(k) = \text{True} \iff \exists \ j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word} \]

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), ..., S(n) \) [Do not solve recursively!]

<table>
<thead>
<tr>
<th>( x )</th>
<th>ANONYMOUS</th>
<th>ARRAY</th>
<th>LETTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( S )</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True} \quad \text{if } x[1..k] \text{ is a valid sequence of words}$
$\quad \text{False} \quad \text{otherwise}$

**STEP 2: Express Recursively**

$S(k) = \text{True} \iff \exists \ j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}$

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n) \ [\text{Do not solve recursively!}]$

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
| $S$ | T | T | T | T | T | F | F | F | F | F | F | T | F | F | F | T | F | F | T | F | F | F | F | T |
### String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word

Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words

False otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists \ j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words

$S(k) = \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$ [ Do not solve recursively! ]
String reconstruction

Given: document \( x[1..n] \): an array of characters
dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word
Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[ S(k) = \text{True} \quad \text{if} \quad x[1..k] \text{ is a valid sequence of words} \]
\[ \quad \quad \text{False} \quad \text{otherwise} \]

**STEP 2: Express Recursively**

\[ S(k) = \text{True} \quad \text{iff} \quad \exists \ j < k \quad \text{s.t.} \quad S(j) \quad \text{is True, and} \quad x[j+1..k] \quad \text{is a valid word} \]

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), ..., S(n) \) [ Do not solve recursively! ]

<table>
<thead>
<tr>
<th>( k )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
String reconstruction

Given: document $x[1..n]$ : an array of characters
   dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
   $\text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff $\exists \ j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$ [ Do not solve recursively! ]

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $S$ | T | T | T | T | F | F | F | F | F | T | F | F | F | T | F | T | F | F | T | F | F | T | F | F | T | T |
Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

STEP 1: Define Subtask
$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$\quad \text{False}$ otherwise

STEP 2: Express Recursively
$S(k) = \text{True}$ iff $\exists \ j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

STEP 3: Order of Subtasks
$S(1), S(2), S(3), \ldots, S(n)$

Algorithm:
$S[0] = \text{true}$
for $k = 1$ to $n$:
$\quad S[k] = \text{false}$
for $j = 1$ to $k$:
$\quad \quad \text{if } S[j-1] \text{ and } \text{dict}(x[j..k])$
$\quad \quad \quad S[k] = \text{true}$

Reconstructing Document:
Define array $D(1,\ldots,n)$:
If $S(k) = \text{true}$, then $D(k) = \text{starting position}$
of the word that ends at $x[k]$
Reconstruct text by following these pointers.
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$= \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff there is $j < k$ s.t. $S(j)$ is True,
and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$

**Reconstructing Document:**
Define array $D(1,..n)$:
If $S(k) = \text{True}$, then $D(k) = \text{starting position}$
of the word that ends at $x[k]$
Reconstruct text by following these pointers.

**Example:**

$\ \\
\begin{array}{cccccccccccccccccccccccccccc}
\hline
\hline
S & T & T & T & T & F & F & F & F & F & T & T & F & F & F & T & F & F & F & F & T & F & F & T & T \\
\end{array}
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words ?

**STEP 1: Define Subtask**

$S(k) = True$ if $x[1..k]$ is a valid sequence of words
$= False$ otherwise

**STEP 2: Express Recursively**

$S(k) = True$ iff there is $j < k$ s.t. $S(j)$ is True,
and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$

Reconstructing Document:
Define array $D(1,..n)$:
If $S(k) = True$, then $D(k) =$ starting position of the word that ends at $x[k]$
Reconstruct text by following these pointers.

| x | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| k | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| S | T | T | T | T | F | F | F | F | F | T | T | F | F | F | T | F | F | F | F | F | T | T |
| D | I | I | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
String reconstruction

Given: document \( x[1..n] \) : an array of characters
dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word
Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\( S(k) = \text{True} \quad \text{if} \ x[1..k] \ \text{is a valid} \)
\( \quad \quad \quad \text{sequence of words} \)
\( = \text{False} \quad \text{otherwise} \)

**STEP 2: Express Recursively**

\( S(k) = \text{True} \quad \text{iff} \ \text{there is} \ j < k \ \text{s.t.} \ S(j) \ \text{is True,} \)
\( \quad \quad \quad \text{and} \ x[j+1..k] \ \text{is a valid word} \)

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), ..., S(n) \)

Reconstructing Document:

Define array \( D(1,..n) \):
If \( S(k) = \text{True} \), then \( D(k) = \text{starting position} \)
of the word that ends at \( x[k] \)

Reconstruct text by following these pointers.

| \( x \) | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| \( k \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| \( D \) | 1 | 2 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| \( S \) | T | T | T | T | F | F | F | F | F | T | T | F | F | F | T | F | T | F | F | F | T | T |
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $dict(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$= \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ iff there is $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$

**Reconstructing Document:**

Define array $D(1,..n)$:
If $S(k) = \text{True}$, then $D(k) = \text{starting position of the word that ends at } x[k]$

Reconstruct text by following these pointers.

| x   | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| k   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
| S   | T | T | T | T | F | F | F | F | F | F | F | T | T | F | F | F | F | T | F | F | F | F | F | T |
| D   | 1 | 1 | 2 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
String reconstruction

Given: document \( x[1..n] \): an array of characters
dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word
Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**

\[
S(k) = \begin{cases} 
\text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
\text{False} & \text{otherwise} 
\end{cases}
\]

**STEP 2: Express Recursively**

\[
S(k) = \begin{cases} 
\text{True} & \text{iff there is } j < k \text{ s.t. } S(j) \text{ is True,} \\
& \text{and } x[j+1..k] \text{ is a valid word} 
\end{cases}
\]

**Reconstructing Document:**

Define array \( D(1,..n) \):
If \( S(k) = \text{True} \), then \( D(k) = \) starting position of the word that ends at \( x[k] \)
Reconstruct text by following these pointers.

**STEP 3: Order of Subtasks**

\( S(1), S(2), S(3), ..., S(n) \)

---

| \( x \) | A | N | O | N | Y | M | O | U | S | A | R | A | R | Y | O | F | L | E | T | T | E | R | S |
| \( k \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| \( S \) | T | T | T | T | F | F | F | F | F | F | T | T | F | F | T | F | F | F | F | T | F | F | F | F | T |
| \( D \) | 1 | 1 | 2 | 3 | - | - | - | - | I | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
String reconstruction

Given: document $x[1..n]$ : an array of characters
dictionary function $\text{dict}(w)$: returns true if $w$ is a valid word
Is $x$ a sequence of valid words?

**STEP 1: Define Subtask**

$S(k) = \text{True}$ if $x[1..k]$ is a valid sequence of words
$= \text{False}$ otherwise

**STEP 2: Express Recursively**

$S(k) = \text{True}$ if there is $j < k$ s.t. $S(j)$ is True,
and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), ..., S(n)$

**Reconstructing Document:**

Define array $D(1,..n)$:
If $S(k) = \text{True}$, then $D(k) = \text{starting position}$
of the word that ends at $x[k]$

Reconstruct text by following these pointers.

---

| $x$ | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $S$ | T | T | T | T | F | F | F | F | F | T | T | F | F | T | F | F | T | F | F | F | F | T | T |
| $D$ | 1 | 1 | 2 | 3 | - | - | - | - | 1 | 10 | - | - | - | 10 | - | 15 | - | - | 17 | - | - | 17 | 17 |

---

Wednesday, April 23, 14
String reconstruction

Given: document \( x[1..n] \): an array of characters
dictionary function \( \text{dict}(w) \): returns true if \( w \) is a valid word
Is \( x \) a sequence of valid words?

**STEP 1: Define Subtask**
\[
S(k) = \begin{cases} 
  \text{True} & \text{if } x[1..k] \text{ is a valid sequence of words} \\
  \text{False} & \text{otherwise} 
\end{cases}
\]

**STEP 2: Express Recursively**
\[
S(k) = \text{True} \text{ iff there is } j < k \text{ s.t. } S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word}
\]

**STEP 3: Order of Subtasks**
\( S(1), S(2), S(3), ..., S(n) \)

**Reconstructing Document:**
Define array \( D(1,..n) \):
If \( S(k) = \text{True} \), then \( D(k) = \) starting position of the word that ends at \( x[k] \)
Reconstruct text by following these pointers.

| \( x \) | A | N | O | N | Y | M | O | U | S | A | R | R | A | Y | O | F | L | E | T | T | E | R | S |
| \( k \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| \( S \) | T | T | T | F | F | F | F | F | T | T | F | F | F | T | F | F | F | T | F | F | F | F | T | T |
| \( D \) | 1 | 1 | 2 | 3 | - | - | - | - | 1 | 10 | - | - | - | 10 | - | 15 | - | - | 17 | - | - | 17 | 17 |
How to Write a Dynamic Programming Solution

1. Define the subproblem (in words)
   
   \[ S(k) = \text{True} \iff x[1..k] \text{ is a valid sequence of words} \]
   
   \[ S(k) = \text{False} \iff \text{otherwise} \]

2. Write down recurrence relation
   
   \[ S(k) = \text{True} \iff \exists j < k \ s.t. \ S(j) \text{ is True, and } x[j+1..k] \text{ is a valid word} \]

3. Base case, Final solution, Order
   
   Solution: \( S(n) \)
   Base Case: \( S(0) = 0 \)
   Evaluation Order: \( S(1), \ldots, S(n) \)

4. Correctness Proof (by induction)

5. Running time analysis (usually easy, but not always)
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Example:**

$x = A, C, G, T, A, G$

$y = G, T, C, C, A, C$

$LCS(x, y) = G, T, A$
**Longest Common Subsequence (LCS)**

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence.

**Example:**

$x = A, C, G, T, A, G$
$y = G, T, C, C, A, C$

$LCS(x, y) = G, T, A$

**STEP 1: Define subtasks**

$S(i,j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm $= S(n,m)$
Longest Common Subsequence (LCS)

**Problem:** Given two sequences \(x[1..m]\) and \(y[1..n]\), find their longest common subsequence

**Example:**


**STEP 1: Define subtasks**

\(S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]\)

Output of algorithm = \(S(n,m)\)

**STEP 2: Express recursively**

\(S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]\)

\(= \max(S(i-1,j), S(i,j-1)), \text{ ow}\)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Example:**

$x = A,C,G,T,A,G$
$y = G,T,C,C,A,C$

LCS($x, y$) = G,T,A

**STEP 1: Define subtasks**

$S(i,j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm $= S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, otherwise

**STEP 3: Order of subtasks**

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences \(x[1..m]\) and \(y[1..n]\), find their longest common subsequence

**Example:**

\[x = A,C,G,T,A,G\]
\[y = G,T,C,C,A,C\]
\[LCS(x, y) = G,T,A\]

**STEP 1: Define subtasks**

\(S(i,j) =\) Length of LCS of \(x[1..i]\) and \(y[1..j]\)

Output of algorithm = \(S(n,m)\)

**STEP 2: Express recursively**

\[S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]\]
\[= \max(S(i-1,j), S(i,j-1)), \text{ otherwise}\]

**STEP 3: Order of subtasks**

Row by row, top to bottom
Problem: Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

Example:

$x = A, C, G, T, A, G$
$y = G, T, C, C, A, C$
$LCS(x, y) = G, T, A$

STEP 1: Define subtasks
$S(i,j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm = $S(n,m)$

STEP 2: Express recursively
$S(i,j) =$ $S(i-1,j-1) + 1$, if $x[i] = y[j]$
$= \max(S(i-1,j), S(i,j-1))$, ow

STEP 3: Order of subtasks
Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence

**Example:**

\( x = A,C,G,T,A,G \)
\( y = G,T,C,C,A,C \)

LCS\((x, y) = G,T,A\)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**STEP 1: Define subtasks**

\( S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j] \)

Output of algorithm = \( S(n,m) \)

**STEP 2: Express recursively**

\( S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j] \)
\( = \max(S(i-1,j), S(i,j-1)), \text{ otherwise} \)

**STEP 3: Order of subtasks**

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence.

**Example:**

$x = A, C, G, T, A, G$

$y = G, T, C, C, A, C$

$LCS(x, y) = G, T, A$

**STEP 1: Define subtasks**

$S(i, j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm = $S(n, m)$

**STEP 2: Express recursively**

$S(i, j) = S(i-1, j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1, j), S(i, j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
**Longest Common Subsequence (LCS)**

**Problem:** Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence.

**Example:**

\( x = \text{A, C, G, T, A, G} \)  \quad \text{LCS}(x, y) = \text{G, T, A} \)

\( y = \text{G, T, C, C, A, C} \)

**STEP 1: Define subtasks**

\( S(i,j) = \) Length of LCS of \( x[1..i] \) and \( y[1..j] \)

Output of algorithm = \( S(n,m) \)

**STEP 2: Express recursively**

\( S(i,j) = S(i-1,j-1) + 1 \), if \( x[i] = y[j] \)

\( = \max(S(i-1,j), S(i,j-1)) \), ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
Problem: Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence.

Example:

$x = A, C, G, T, A, G$

$y = G, T, C, C, A, C$

$LCS(x, y) = G, T, A$

**STEP 1: Define subtasks**

$S(i, j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm = $S(n, m)$

**STEP 2: Express recursively**

$S(i, j) =$ $S(i-1, j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1, j), S(i, j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
Problem: Given two sequences \(x[1..m]\) and \(y[1..n]\), find their longest common subsequence

Example:

\[x = A,C,G,T,A,G\]
\[y = G,T,C,C,A,C\]

LCS\((x, y) = G,T,A\)

STEP 1: Define subtasks

\[S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]\]

Output of algorithm = \(S(n,m)\)

STEP 2: Express recursively

\[S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]\]
\[= \text{max}(S(i-1,j), S(i,j-1)), \text{ ow}\]

STEP 3: Order of subtasks

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Example:**

$x = A, C, G, T, A, G$
$y = G, T, C, C, A, C$
$LCS(x, y) = G, T, A$

**STEP 1: Define subtasks**

$S(i,j) = \text{Length of LCS of } x[1..i]$
and $y[1..j]$

Output of algorithm = $S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1, j-1) + 1$, if $x[i] = y[j]$
$= \max(S(i-1, j), S(i, j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence.

**Example:**

\[ x = A, C, G, T, A, G \]
\[ y = G, T, C, C, A, C \]

LCS\((x, y)\) = G, T, A

**STEP 1: Define subtasks**

\[ S(i,j) = \text{Length of LCS of } x[1..i] \]
\[ \text{and } y[1..j] \]

Output of algorithm = \( S(n,m) \)

**STEP 2: Express recursively**

\[ S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j] \]
\[ = \max(S(i-1,j), S(i,j-1)), \text{ otherwise} \]

**STEP 3: Order of subtasks**

Row by row, top to bottom

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>T</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>T</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
### Longest Common Subsequence (LCS)

**Problem:** Given two sequences \(x[1..m]\) and \(y[1..n]\), find their longest common subsequence

**Example:**

\[x = A,C,G,T,A,G\]
\[y = G,T,C,C,A,C\]

**LCS** \((x, y) = G,T,A\)

#### STEP 1: Define subtasks

\(S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]\)

Output of algorithm = \(S(n,m)\)

#### STEP 2: Express recursively

\[S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]\]
\[= \max(S(i-1,j), S(i,j-1)), \text{ ow}\]

#### STEP 3: Order of subtasks

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Example:**

$x = A,C,G,T,A,G$
$y = G,T,C,C,A,C$

$LCS(x, y) = G,T,A$

**STEP 1: Define subtasks**

$S(i,j) = \text{Length of LCS of } x[1..i]$
and $y[1..j]$

Output of algorithm = $S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$
$= \max(S(i-1,j), S(i,j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences \(x[1..m]\) and \(y[1..n]\), find their longest common subsequence

**Example:**

\(x = A, C, G, T, A, G\)
\(y = G, T, C, C, A, C\)

\(\text{LCS}(x, y) = G, T, A\)

**STEP 1: Define subtasks**

\(S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]\)

Output of algorithm = \(S(n,m)\)

**STEP 2: Express recursively**

\(S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j]\)
\(\quad = \max(S(i-1,j), S(i,j-1)), \text{ otherwise}\)

**STEP 3: Order of subtasks**

Row by row, top to bottom
Problem: Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence

Example:

\( x = A, C, G, T, A, G \)
\( y = G, T, C, C, A, C \)
LCS(\( x, y \)) = \( G, T, A \)

STEP 1: Define subtasks

\( S(i,j) = \text{Length of LCS of } x[1..i] \)
and \( y[1..j] \)

Output of algorithm = \( S(n,m) \)

STEP 2: Express recursively

\( S(i,j) = S(i-1,j-1) + 1, \) if \( x[i] = y[j] \)
\( = \max(S(i-1,j), S(i,j-1)), \) ow

STEP 3: Order of subtasks

Row by row, top to bottom
Problem: Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence.

Example:

$x = A, C, G, T, A, G$

$y = G, T, C, C, A, C$

$LCS(x, y) = G, T, A$

---

STEP 1: Define subtasks

$S(i, j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm = $S(n, m)$

STEP 2: Express recursively

$S(i, j) = S(i-1, j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1, j), S(i, j-1))$, ow

---

STEP 3: Order of subtasks

Row by row, top to bottom
Problem: Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence.

Example:

\[ x = \text{A,C,G,T,A,G} \]
\[ y = \text{G,T,C,C,A,C} \]

LCS\((x, y) = \text{G,T,A}\)

STEP 1: Define subtasks

\( S(i,j) = \) Length of LCS of \( x[1..i] \) and \( y[1..j] \)

Output of algorithm \( = S(n,m) \)

STEP 2: Express recursively

\[ S(i,j) = S(i-1,j-1) + 1, \text{ if } x[i] = y[j] \]
\[ = \max(S(i-1,j), S(i,j-1)), \text{ ow} \]

STEP 3: Order of subtasks

Row by row, top to bottom
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Example:**

$x = A, C, G, T, A, G$
$y = G, T, C, C, A, C$

$LCS(x, y) = G, T, A$

**STEP 1: Define subtasks**

$S(i,j) = $ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm = $S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
### Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence.

**Example:**

$x = A,C,G,T,A,G$

$y = G,T,C,C,A,C$

$LCS(x, y) = G,T,A$

### STEP 1: Define subtasks

$S(i,j) =$ Length of LCS of $x[1..i]$ and $y[1..j]$

Output of algorithm = $S(n,m)$

### STEP 2: Express recursively

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, otherwise

### STEP 3: Order of subtasks

Row by row, top to bottom

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Longest Common Subsequence (LCS)

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Example:**

$x = A,C,G,T,A,G$

$y = G,T,C,C,A,C$

$LCS(x, y) = G,T,A$

**STEP 1: Define subtasks**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

Output of algorithm = $S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom
**Longest Common Subsequence (LCS)**

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**Algorithm:**

```plaintext
STEP 1: Define subtasks
S(i,j) = Length of LCS of x[1..i] and y[1..j]
Output of algorithm = S(n,m)

STEP 2: Express recursively
S(i,j) = S(i-1,j-1) + 1, if x[i] = y[j]
= max(S(i-1,j), S(i,j-1)), ow

STEP 3: Order of subtasks
Row by row, top to bottom
```

```plaintext
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
    for j = 1 to m:
        if x[i] = y[j]:
            S[i,j] = S[i-1,j-1] + 1
        else:
            S[i,j] = max(S[i-1,j], S[i,j-1])
return S[n,m]
```
Longest Common Subsequence (LCS)

**Problem:** Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence.

**Algorithm:**

```
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
    for j = 1 to m:
        if x[i] = y[j]:
            S[i,j] = S[i-1,j-1] + 1
        else:
            S[i,j] = max{ S[i-1,j], S[i,j-1] }
return S[n,m]
```

**STEP 1: Define subtasks**

\( S(i,j) = \) Length of LCS of \( x[1..i] \)

and \( y[1..j] \)

Output of algorithm = \( S(n,m) \)

**STEP 2: Express recursively**

\( S(i,j) = \)

\( = S(i-1,j-1) + 1 \), if \( x[i] = y[j] \)

\( = \max( S(i-1,j), S(i,j-1) ) \), ow

**STEP 3: Order of subtasks**

Row by row, top to bottom

**Running Time:** \( O(mn) \)
**Longest Common Subsequence (LCS)**

**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**STEP 1: Define subtasks**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

Output of algorithm = $S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom

**Algorithm:**

```
for i = 0 to n: S[i,0] = 0
for j = 0 to m: S[0,j] = 0
for i = 1 to n:
    for j = 1 to m:
        if x[i] = y[j]:
            S[i,j] =
                S[i-1,j-1] + 1
        else:
            S[i,j] = \max{
                S[i-1,j], S[i,j-1]}
return S[n,m]
```

**Running Time:** $O(mn)$

How to reconstruct the actual subsequence?
**Longest Common Subsequence (LCS)**

**Problem:** Given two sequences \( x[1..m] \) and \( y[1..n] \), find their longest common subsequence

**STEP 1: Define subtasks**

\[
S(i, j) = \text{Length of LCS of } x[1..i] \\
\quad \text{and } y[1..j]
\]

Output of algorithm = \( S(n,m) \)

**STEP 2: Express recursively**

\[
S(i, j) = \begin{cases} 
S(i-1, j-1) + 1, & \text{if } x[i] = y[j] \\
\max(S(i-1, j), S(i, j-1)), & \text{ow}
\end{cases}
\]

**STEP 3: Order of subtasks**

Row by row, top to bottom

**To reconstruct LCS:**

Define \( L(i, j) \):

\[
L(i, j) = \begin{cases} 
(i - 1, j - 1), & \text{if } x[i] = y[j] \\
(i - 1, j), & \text{ow if } S(i-1,j) > S(i, j-1) \\
(i, j - 1), & \text{ow}
\end{cases}
\]

Reconstruct LCS by following the \( L(i,j) \) pointers, starting with \( L(m,n) \)
**Problem:** Given two sequences $x[1..m]$ and $y[1..n]$, find their longest common subsequence

**STEP 1: Define subtasks**

$S(i,j) = \text{Length of LCS of } x[1..i] \text{ and } y[1..j]$

Output of algorithm = $S(n,m)$

**STEP 2: Express recursively**

$S(i,j) = S(i-1,j-1) + 1$, if $x[i] = y[j]$

$= \max(S(i-1,j), S(i,j-1))$, ow

**STEP 3: Order of subtasks**

Row by row, top to bottom

**To reconstruct LCS:**

Define $L(i, j)$:

$L(i, j) = (i - 1, j - 1)$, if $x[i] = y[j]$

$= (i - 1, j)$, ow if $S(i-1,j) > S(i, j-1)$

$= (i, j - 1)$, ow

Reconstruct LCS by following the $L(i,j)$ pointers, starting with $L(m,n)$

**Running Time:** $O(mn)$
# Dynamic Programming vs Divide and Conquer

<table>
<thead>
<tr>
<th>Divide-and-conquer</th>
<th>Dynamic programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>A problem of size $n$ is decomposed into a few subproblems which are significantly smaller (e.g. $n/2, 3n/4,...$)</td>
<td>A problem of size $n$ is expressed in terms of subproblems that are not much smaller (e.g. $n-1, n-2,...$)</td>
</tr>
<tr>
<td>Therefore, size of subproblems decreases geometrically. eg. $n$, $n/2$, $n/4$, $n/8$, etc</td>
<td>A recursive algorithm would take exp. time.</td>
</tr>
<tr>
<td>Use a recursive algorithm.</td>
<td>Saving grace: in total, there are only polynomially many subproblems.</td>
</tr>
<tr>
<td></td>
<td>Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.</td>
</tr>
</tbody>
</table>
Case 1: Input: $x_1, x_2, \ldots, x_n$ Subproblem: $x_1, \ldots, x_i$. 

\[
\begin{array}{cccccccc}
X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10}
\end{array}
\]
DP: Common Subtasks

Case 1: Input: $x_1, x_2, ..., x_n$ Subproblem: $x_1, ..., x_i$.

Case 2: Input: $x_1, x_2, ..., x_n$ and $y_1, y_2, ..., y_m$ Subproblem: $x_1, ..., x_i$ and $y_1, y_2, ..., y_j$.
DP: Common Subtasks

**Case 1:** Input: \( x_1, x_2, ..., x_n \) Subproblem: \( x_1, ..., x_i \)

\[
\begin{array}{cccccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\
\end{array}
\]

**Case 2:** Input: \( x_1, x_2, ..., x_n \) and \( y_1, y_2, ..., y_m \) Subproblem: \( x_1, ..., x_i \) and \( y_1, y_2, ..., y_j \)

\[
\begin{array}{cccccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\
  y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \\
\end{array}
\]

**Case 3:** Input: \( x_1, x_2, ..., x_n \) Subproblem: \( x_i, ..., x_j \)

\[
\begin{array}{cccccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\
\end{array}
\]
**DP: Common Subtasks**

**Case 1:** Input: $x_1, x_2, .., x_n$ Subproblem: $x_1, .., x_i$.

```
x_1  x_2  x_3  x_4  x_5  x_6  x_7  x_8  x_9  x_10
```

**Case 2:** Input: $x_1, x_2, .., x_n$ and $y_1, y_2, .., y_m$ Subproblem: $x_1, .., x_i$ and $y_1, y_2, .., y_j$.

```
x_1  x_2  x_3  x_4  x_5  x_6  x_7  x_8  x_9  x_10
y_1  y_2  y_3  y_4  y_5  y_6  y_7  y_8
```

**Case 3:** Input: $x_1, x_2, .., x_n$. Subproblem: $x_i, .., x_j$.

```
x_1  x_2  x_3  x_4  x_5  x_6  x_7  x_8  x_9  x_10
```

**Case 4:** Input: a rooted tree. Subproblem: a subtree.

![Tree Diagram]
6.1 Weighted Interval Scheduling
Weighted Interval Scheduling

Weighted interval scheduling problem.

- Job $j$ starts at $s_j$, finishes at $f_j$, and has weight or value $v_j$.
- Two jobs compatible if they don't overlap.
- Goal: find maximum weight subset of mutually compatible jobs.
Unweighted Interval Scheduling Review

Recall. Greedy algorithm works if all weights are 1.
- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

Observation. Greedy algorithm can fail spectacularly if arbitrary weights are allowed.
Weighted Interval Scheduling

**Notation.** Label jobs by finishing time: \( f_1 \leq f_2 \leq \ldots \leq f_n \).

**Def.** \( p(j) = \text{largest index } i < j \text{ such that job } i \text{ is compatible with } j \). 

**Ex:** \( p(8) = 5, p(7) = 3, p(2) = 0 \).
Dynamic Programming: Binary Choice

Notation. $OPT(j) =$ value of optimal solution to the problem consisting of job requests $1, 2, ..., j$.

- **Case 1:** $OPT$ selects job $j$.
  - can't use incompatible jobs $\{ p(j) + 1, p(j) + 2, ..., j - 1 \}$
  - must include optimal solution to problem consisting of remaining compatible jobs $1, 2, ..., p(j)$

- **Case 2:** $OPT$ does not select job $j$.
  - must include optimal solution to problem consisting of remaining compatible jobs $1, 2, ..., j-1$

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max \{ v_j + OPT(p(j)), \ OPT(j-1) \} & \text{otherwise} \end{cases}$$
Weighted Interval Scheduling: Brute Force

Brute force algorithm.

Input: \( n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n \)

Sort jobs by finish times so that \( f_1 \leq f_2 \leq \ldots \leq f_n \).

Compute \( p(1), p(2), \ldots, p(n) \)

Compute-Opt\( (j) \) {
    \textbf{if} (j = 0)
    return 0
    \textbf{else}
    return max(v_j + Compute-Opt(p(j)), Compute-Opt(j-1))
}
Weighted Interval Scheduling: Brute Force

**Observation.** Recursive algorithm fails spectacularly because of redundant sub-problems $\Rightarrow$ exponential algorithms.

**Ex.** Number of recursive calls for family of "layered" instances grows like Fibonacci sequence.

\[ p(1) = 0, \quad p(j) = j-2 \]
Weighted Interval Scheduling: Memoization

**Memoization.** Store results of each sub-problem in a cache; lookup as needed.

**Input:** $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p(1), p(2), \ldots, p(n)$

```plaintext
for j = 1 to n
    M[j] = empty ← global array
    M[j] = 0

M-Compute-Opt(j) {
    if (M[j] is empty)
        M[j] = max($w_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1)$)
    return M[j]
}
```
Weighted Interval Scheduling: Running Time

Claim. Memoized version of algorithm takes $O(n \log n)$ time.
- Sort by finish time: $O(n \log n)$.
- Computing $p(\cdot)$: $O(n)$ after sorting by start time.

- $M$-Compute-Opt$(j)$: each invocation takes $O(1)$ time and either
  - (i) returns an existing value $M[j]$
  - (ii) fills in one new entry $M[j]$ and makes two recursive calls

- Progress measure $\Phi = \# \text{ nonempty entries of } M[\cdot]$.
  - initially $\Phi = 0$, throughout $\Phi \leq n$.
  - (ii) increases $\Phi$ by 1 $\Rightarrow$ at most $2n$ recursive calls.

- Overall running time of $M$-Compute-Opt$(n)$ is $O(n)$.  

Remark. $O(n)$ if jobs are pre-sorted by start and finish times.
Weighted Interval Scheduling: Finding a Solution

Q. Dynamic programming algorithms computes optimal value. What if we want the solution itself?
A. Do some post-processing.

```
Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j) {
  if (j = 0)
    output nothing
  else if (v_j + M[p(j)] > M[j-1])
    print j
    Find-Solution(p(j))
  else
    Find-Solution(j-1)
}
```

- \# of recursive calls $\leq n \Rightarrow O(n)$.
**Weighted Interval Scheduling: Bottom-Up**

Bottom-up dynamic programming. Unwind recursion.

**Input:** \( n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n \)

Sort jobs by finish times so that \( f_1 \leq f_2 \leq \ldots \leq f_n \).

Compute \( p(1), p(2), \ldots, p(n) \)

Iterative-Compute-Opt {
  \( M[0] = 0 \)
  for \( j = 1 \) to \( n \)
    \( M[j] = \max(v_j + M[p(j)], M[j-1]) \)
}

Wednesday, April 23, 14
6.3 Segmented Least Squares
Segmented Least Squares

Least squares.

- Foundational problem in statistic and numerical analysis.
- Given n points in the plane: \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\).
- Find a line \(y = ax + b\) that minimizes the sum of the squared error:

\[
SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2
\]

Solution. Calculus \(\Rightarrow\) min error is achieved when

\[
a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}
\]
Segmented Least Squares

Segmented least squares.

- Points lie roughly on a sequence of several line segments.
- Given $n$ points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of lines that minimizes $f(x)$.

Q. What's a reasonable choice for $f(x)$ to balance accuracy and parsimony?

$\uparrow$

Goodness of fit

$\uparrow$

Number of lines

\[ y \]

\[ x \]
Segmented least squares.
- Points lie roughly on a sequence of several line segments.
- Given n points in the plane \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) with \(x_1 < x_2 < \ldots < x_n\), find a sequence of lines that minimizes:
  - the sum of the sums of the squared errors \(E\) in each segment
  - the number of lines \(L\)
- Tradeoff function: \(E + cL\), for some constant \(c > 0\). 
Dynamic Programming: Multiway Choice

Notation.
- \( \text{OPT}(j) = \) minimum cost for points \( p_1, p_{i+1}, \ldots, p_j \).
- \( e(i, j) = \) minimum sum of squares for points \( p_i, p_{i+1}, \ldots, p_j \).

To compute \( \text{OPT}(j) \):
- Last segment uses points \( p_i, p_{i+1}, \ldots, p_j \) for some \( i \).
- \( \text{Cost} = e(i, j) + c + \text{OPT}(i-1) \).

\[
\text{OPT}(j) = \begin{cases} 
0 & \text{if } j = 0 \\
\min_{1 \leq i \leq j} \{ e(i, j) + c + \text{OPT}(i-1) \} & \text{otherwise}
\end{cases}
\]
Segmented Least Squares: Algorithm

INPUT: n, p₁,…,pₙ, c

Segmented-Least-Squares() {
M[0] = 0
for j = 1 to n
    for i = 1 to j
        compute the least square error eᵢⱼ for the segment pᵢ,…, pⱼ

for j = 1 to n
    M[j] = \min_{1 \leq i \leq j} (eᵢⱼ + c + M[i-1])

return M[n]
}

Running time. \(O(n^3)\). can be improved to \(O(n^2)\) by pre-computing various statistics

- Bottleneck = computing e(i, j) for \(O(n^2)\) pairs, \(O(n)\) per pair using previous formula.