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

- **Hill-climbing**: Start with a solution, improve it
Dynamic Programming (DP): A Simple Example

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: $O(c^n)$

Running Time:

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

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

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: $O(n)$

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 $fib[i]$

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

3. Find the right order for solving the subtasks ($i = 1,\ldots,n$)
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:
$x = \text{anynymousarrayofletters}$ : $\text{True}$
$x = \text{anhuymousarrayofhetters}$ : $\text{False}$
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:
\( x = \text{anonymousarrayofletters} \) : True
\( x = \text{anhuymousarrayofhetters} \) : 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 = \( S(n) \)

| 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 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**

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

**STEP 3: Order of Subtasks**

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

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMIOUS</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>True</td>
<td>False</td>
<td>True</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**

$S(k) = \text{True}$ if $\exists$ $j < k$ 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>$x$</th>
<th>ANONYMousy</th>
<th>ARRAY</th>
<th>OF</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>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T</td>
<td></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
$\text{False}$ 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!]

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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
$\text{False}$ 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!]

| $x$ | 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 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
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 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
**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**

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

**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>ANONYMYSARARRAYOFFLETTERRS</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</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
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>$x$</th>
<th>ANONYMYSARAYOFLETTERSS</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 F 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**

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

**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 | F |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
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 $\exists j < k$ 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>$x$</th>
<th>ANONYM</th>
<th>OUS</th>
<th>AR</th>
<th>ARR</th>
<th>AY</th>
<th>OF</th>
<th>LET</th>
<th>TERS</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>T T T T T F F F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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

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

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

**STEP 3: Order of Subtasks**
\[
S(1), S(2), S(3), ..., S(n) \quad [\text{Do not solve recursively!}]
\]
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**

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

**STEP 2: Express Recursively**

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

**STEP 3: Order of Subtasks**

\[ S(1), S(2), S(3), ..., S(n) \quad [ \text{Do not solve recursively!} \]
**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

\( \text{False} \) 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! ]

---

| \( k \) | \( 0 \) | \( 1 \) | \( 2 \) | \( 3 \) | \( 4 \) | \( 5 \) | \( 6 \) | \( 7 \) | \( 8 \) | \( 9 \) | \( 10 \) | \( 11 \) | \( 12 \) | \( 13 \) | \( 14 \) | \( 15 \) | \( 16 \) | \( 17 \) | \( 18 \) | \( 19 \) | \( 20 \) | \( 21 \) | \( 22 \) | \( 23 \) |
| \( S \) | \( T \) | \( T \) | \( T \) | \( T \) | \( T \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( T \) | \( F \) | \( T \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) | \( F \) |
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**

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

**STEP 2: Express Recursively**

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

**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 | 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
$\text{False}$ 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!]

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

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

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

| $x$ | 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 | T | F | T | F | F | F | F | T | T | F | F | F | F | T | 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**

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

<table>
<thead>
<tr>
<th>$x$</th>
<th>ANONYMous</th>
<th>ARRAYY</th>
<th>O F L E T T E R S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>0  1  2   3  4  5</td>
<td>6  7   8</td>
<td>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  F</td>
<td>F  F  F  F  T  T  F  F  F  T  F  T  F</td>
<td></td>
</tr>
</tbody>
</table>

x
### 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

$S(k) = \text{True}$ if $\exists \ j < k$ 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 | F | F | F | F | F | T | T | F | F | F | F | T | F | T | F | T | F | T | F |

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

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

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

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

**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**

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

| $x$ | 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 | T | F | F | F | T | F | F | F | T | F | F | F | T | F | F | F |
String reconstruction

**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! ]

<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>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
## String reconstruction

**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! ]

---

| 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 | F | F | F | F | 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

$S(k) = \text{False}$ 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!]

| $x$   | 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 | T | T | F | F | F | F | T | F | 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) = 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)

Algorithm:
S[0] = true
for k = 1 to n:
    S[k] = false
    for j = 1 to k:
        if S[j-1] and dict(x[j..k])
            S[k] = true

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.
## 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

$$S(k) = \text{True} \iff \text{there is } j < k \text{ s.t. } S(j) \text{ is True,}$$

$$\text{and } x[j+1..k] \text{ is a valid word}$$

### STEP 3: Order of Subtasks

$S(1), S(2), S(3), \ldots, S(n)$

### 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.

### Reconstructing the Document:

<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>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>T</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here, $x$ is the given document, $k$ is the index, $S(k)$ indicates whether the sequence up to $k$ is valid, and $D(k)$ is the starting position of the word that ends at $x[k]$. The table represents the truth values of $S(k)$ and the corresponding $D(k)$ values.
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.

| x  | A | N | O | N | Y | M | O | U | S | A | 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 | F | F | 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

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

### STEP 2: Express Recursively

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

### 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 \( D(k) = \text{starting position} \)  
of the word that ends at \( x[k] \)

Reconstruct text by following these pointers.

### Table:

<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(k) )</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>F</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>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>( D(k) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</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**
\[
S(k) = \begin{cases} 
\text{True} & \text{if } x[1..k] \text{ is a valid} \\
\text{False} & \text{otherwise}
\end{cases}
\]

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

**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 \( D(k) = \) starting position of the word that ends at \( x[k] \)
Reconstruct text by following these pointers.

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\hline
\hline
S & T & T & T & T & F & F & F & F & T & T & F & F & F & T & F & F & F & T & F & F & F & F & T & T \\
\hline
\hline
\end{array}
\]
## 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**

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

---

<table>
<thead>
<tr>
<th>$S$</th>
<th>ANONYMOUS</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>$D$</td>
<td>1 1 2 3 - - - - - - - - - - - - - - - - -</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**
\[
S(k) = \text{True} \quad \text{if } x[1..k] \text{ is a valid sequence of words}
= \text{False} \quad \text{otherwise}
\]

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

**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 \( D(k) = \) 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 | T | T | F | F | F | T | F | F | T | F | F | T | F | F | T | T |
| \( D \) | 1 | 1 | 2 | 3 | - | - | - | - | 1 | 10 | - | - | 10 | - | 15 | - | - | 17 | - | - | 17 | 17 |
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 there is $j < k$ s.t. $S(j)$ is True, and $x[j+1..k]$ is a valid word

**STEP 3: Order of Subtasks**

$S(1), S(2), S(3), \ldots, S(n)$

**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.

Reconstructing Document:

```
<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>S</td>
<td>T T T T F F F F T T F F F F T F T F F F F T T F F F F T T T T</td>
</tr>
<tr>
<td>D</td>
<td>1 1 2 - - - - - 1 10 - - - 10 - - 15 - - 17 - - 17 17</td>
</tr>
</tbody>
</table>
```
How to Write a DP Solution

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

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

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)

See Sample HW Solutions for more examples!
Dynamic Programming

• String Reconstruction

• Longest Common Subsequence
Longest Common Subsequence (LCS)

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

**Example:**

- $x = A,C,G,T,A,G$
- $y = G,T,C,C,A,C$
- $\text{LCS}(x, y) = G,T,A$
**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 $x[i] = y[j]$

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

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

**STEP 3: Order of subtasks**

Row by row, top to bottom
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**

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

**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**

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

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

**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:**

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

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

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

<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>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></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) = $ 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**

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

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

**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></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)), 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**

$S(i, j) =$ 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**

\( S(i,j) = S(i-1,j-1) + 1, \) 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:**

\(x = A,C,G,T,A,G\)  \hspace{1cm} \text{LCS}(x, y) = G,T,A

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

**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**

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

**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:**

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

**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**

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

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

**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) = Length of LCS of x[1..i] and y[1..j]

Output of algorithm = S(n,m)

**STEP 2: Express recursively**

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

**STEP 3: Order of subtasks**

Row by row, top to bottom
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
**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**

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

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

**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.

**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

**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**

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

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 \(L(m,n)\)

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

## Divide-and-conquer

A problem of size $n$ is decomposed into a few subproblems which are significantly smaller (e.g. $n/2$, $3n/4$, ...)

Therefore, size of subproblems decreases geometrically.

eg. $n$, $n/2$, $n/4$, $n/8$, etc

Use a recursive algorithm.

## Dynamic programming

A problem of size $n$ is expressed in terms of subproblems that are not much smaller (e.g. $n-1$, $n-2$, ...)

A recursive algorithm would take exp. time.

Saving grace: in total, there are only polynomially many subproblems.

Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.
DP: Common Subtasks

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

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

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

Dynamic Programming

- String Reconstruction
- Longest Common Subsequence
- Subset Sum
**Subset Sum**

**Problem:** Given a list of positive integers $a[1..n]$ and an integer $t$, is there some subset of $a$ that sums to exactly $t$?

**Example:** $a = [12, 1, 3, 8, 20, 50]$  

**STEP 1: Define subtasks**

For $i=1..n$, $s=1..t$,  

$S(i,s) = \text{True, if some subset of } S[1..i] \text{ adds to } s$  

$= \text{False, otherwise}$  

Output = $S(n, t)$

**STEP 2: Express recursively**

If $a[i] \leq s$,  

$S(i,s) = S(i-1, s-a[i]) \text{ OR } S(i-1, s)$

Else: $S(i, s) = S(i-1, s)$

**STEP 3: Order of subtasks**

Row by row, increasing column

Running Time = $O(nt)$

How to reconstruct the subset?
Subset Sum

**Problem:** Given a list of positive integers $a[1..n]$ and an integer $t$, is there some subset of $a$ that sums to exactly $t$?

**Example:** $a = [12, 1, 3, 8, 20, 50 ]$

**STEP 1: Define subtasks**
For $i=1..n$, $s=1..t$,
$S(i,s) = \text{True, if some subset of } S[1..i] \text{ adds to } s$
$= \text{False, ow}$

Output = $S(n, t)$

**STEP 2: Express recursively**
If $a[i] \leq s$,
$S(i,s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s)$
Else: $S(i, s) = S(i - 1, s)$

**STEP 3: Order of subtasks**
Row by row, increasing column

<table>
<thead>
<tr>
<th></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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Running Time = $O(nt)$

How to reconstruct the subset?
**Subset Sum**

**Problem:** Given a list of positive integers $a[1..n]$ and an integer $t$, is there some subset of $a$ that sums to exactly $t$?

**Example:** $a = [12, 1, 3, 8, 20, 50]$  

**STEP 1: Define subtasks**
For $i=1..n$, $s=1..t$,  
$S(i,s) = True$, if some subset of $S[1..i]$ adds to $s$  
$= False$, ow  
Output = $S(n, t)$

**STEP 2: Express recursively**
If $a[i] \leq s$,  
$S(i,s) = S(i - 1, s - a[i])$ OR $S(i - 1, s)$
Else: $S(i, s) = S(i - 1, s)$

**STEP 3: Order of subtasks**
Row by row, increasing column

<table>
<thead>
<tr>
<th></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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Running Time = $O(nt)$

How to reconstruct the subset?
**Problem:** Given a list of positive integers \(a[1..n]\) and an integer \(t\), is there some subset of \(a\) that sums to exactly \(t\)?

**Example:** \(a = [12, 1, 3, 8, 20, 50]\)

**STEP 1: Define subtasks**
For \(i=1..n\), \(s=1..t\),
\(S(i,s) = \text{True, if some subset of } S[1..i] \text{ adds to } s\)
\(= \text{False, ow}\)
Output = \(S(n, t)\)

**STEP 2: Express recursively**
If \(a[i] \leq s\),
\(S(i,s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s)\)
Else: \(S(i, s) = S(i - 1, s)\)

**STEP 3: Order of subtasks**
Row by row, increasing column

Running Time = \(O(nt)\)

How to reconstruct the subset?
Subset Sum

**Problem:** Given a list of positive integers \(a[1..n]\) and an integer \(t\), is there some subset of \(a\) that sums to exactly \(t\)?

**Example:** \(a = [12, 1, 3, 8, 20, 50]\)

**STEP 1: Define subtasks**
For \(i=1..n, s=1..t\),
\[S(i,s) = \text{True, if some subset of } S[1..i] \text{ adds to } s\]
\[= \text{False, ow}\]
Output = \(S(n, t)\)

**STEP 2: Express recursively**
If \(a[i] \leq s\),
\[S(i,s) = S(i-1, s-a[i]) \text{ OR } S(i-1, s)\]
Else: \(S(i,s) = S(i-1, s)\)

**STEP 3: Order of subtasks**
Row by row, increasing column

Running Time = \(O(nt)\)

How to reconstruct the subset?
Problem: Given a list of positive integers \( a[1..n] \) and an integer \( t \), is there some subset of \( a \) that sums to exactly \( t \)?

Example: \( a = [12, 1, 3, 8, 20, 50] \)

\[
\begin{align*}
\text{STEP 1: Define subtasks} & \\
\text{For } i = 1..n, \ s = 1..t, & \\
S(i, s) = \text{True, if some subset of } S[1..i] \text{ adds to } s & \\
& = \text{False, ow} \\
\text{Output } = S(n, t)
\end{align*}
\]

\[
\begin{align*}
\text{STEP 2: Express recursively} & \\
\text{If } a[i] \leq s, & \\
S(i, s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s) & \\
\text{Else: } S(i, s) = S(i - 1, s)
\end{align*}
\]

\[
\begin{align*}
\text{STEP 3: Order of subtasks} & \\
\text{Row by row, increasing column}
\end{align*}
\]

\[
\begin{align*}
t = 44 & \quad \text{True} & t = 14 & \quad \text{False}
\end{align*}
\]

Running Time = \( O(nt) \)

How to reconstruct the subset?
**Problem:** Given a list of positive integers \( a[1..n] \) and an integer \( t \), is there some subset of \( a \) that sums to exactly \( t \)?

**Example:** \( a = [12, 1, 3, 8, 20, 50] \)

**STEP 1: Define subtasks**
For \( i=1..n, s=1..t \),
\( S(i,s) = \) True, if some subset of \( S[1..i] \) adds to \( s \)
= False, ow
Output = \( S(n, t) \)

**STEP 2: Express recursively**
\( \text{If } a[i] \leq s, \)
\( S(i,s) = S(i - 1, s - a[i]) \) OR \( S(i - 1, s) \)
Else: \( S(i, s) = S(i - 1, s) \)

**STEP 3: Order of subtasks**
Row by row, increasing column

Running Time = \( O(nt) \)
How to reconstruct the subset?
Subset Sum

**Problem:** Given a list of positive integers $a[1..n]$ and an integer $t$, is there some subset of $a$ that sums to exactly $t$?

**Example:** $a = [12, 1, 3, 8, 20, 50]$

**STEP 1: Define subtasks**
For $i=1..n$, $s=1..t$,
$S(i,s) = \text{True, if some subset of } S[1..i] \text{ adds to } s$
$= \text{False, ow}$
Output = $S(n, t)$

**STEP 2: Express recursively**
If $a[i] \leq s$,
$S(i,s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s)$
Else: $S(i, s) = S(i - 1, s)$

**STEP 3: Order of subtasks**
Row by row, increasing column

Running Time = $O(nt)$

How to reconstruct the subset?
**Problem:** Given a list of positive integers \( a[1..n] \) and an integer \( t \), is there some subset of \( a \) that sums to exactly \( t \)?

**Example:** \( a = [ 12, 1, 3, 8, 20, 50 ] \)

**STEP 1: Define subtasks**
For \( i=1..n \), \( s=1..t \),
\[
S(i, s) = \begin{cases} 
  \text{True, if some subset of } S[1..i] \\
  \text{adds to } s \\
  \text{False, otherwise}
\end{cases}
\]
Output = \( S(n, t) \)

**STEP 2: Express recursively**
If \( a[i] \leq s \),
\[
S(i, s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s)
\]
Else: \( S(i, s) = S(i - 1, s) \)

**STEP 3: Order of subtasks**
Row by row, increasing column

<table>
<thead>
<tr>
<th></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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</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>
</tr>
<tr>
<td>2</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>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Running Time = \( O(nt) \)

How to reconstruct the subset?
**Subset Sum**

**Problem:** Given a list of positive integers \( a[1..n] \) and an integer \( t \), is there some subset of \( a \) that sums to exactly \( t \)?

**Example:** \( a = [ 12, 1, 3, 8, 20, 50 ] \)

**STEP 1: Define subtasks**
For \( i=1..n \), \( s=1..t \),
\( S(i,s) = \) True, if some subset of \( S[1..i] \) adds to \( s \)
\( = \) False, ow
Output = \( S(n, t) \)

**STEP 2: Express recursively**
If \( a[i] \leq s \),
\( S(i,s) = S(i - 1, s - a[i]) \) OR \( S(i - 1, s) \)
Else: \( S(i, s) = S(i - 1, s) \)

**STEP 3: Order of subtasks**
Row by row, increasing column

Running Time = \( O(nt) \)

How to reconstruct the subset?
**Subset Sum**

**Problem:** Given a list of positive integers $a[1..n]$ and an integer $t$, is there some subset of $a$ that sums to exactly $t$?

**Example:** $a = [12, 1, 3, 8, 20, 50]$

**STEP 1: Define subtasks**
For $i=1..n$, $s=1..t$,
$S(i,s) = \text{True, if some subset of } S[1..i] \text{ adds to } s$  
$= \text{False, ow}$

Output = $S(n, t)$

**STEP 2: Express recursively**
If $a[i] \leq s$,
$S(i,s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s)$
Else: $S(i, s) = S(i - 1, s)$

**STEP 3: Order of subtasks**
Row by row, increasing column

<table>
<thead>
<tr>
<th>$s$</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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</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>
</tr>
<tr>
<td>2</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>
</tr>
<tr>
<td>3</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>
</tr>
<tr>
<td>4</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>
</tr>
<tr>
<td>5</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>
</tr>
<tr>
<td>6</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>
</tr>
</tbody>
</table>

Running Time = $O(nt)$

How to reconstruct the subset?
**Subset Sum**

**Problem:** Given a list of positive integers $a[1..n]$ and an integer $t$, is there some subset of $a$ that sums to exactly $t$?

**STEP 1: Define subtasks**
For $i=1..n$, $s=1..t$,

$S(i, s) = \text{True, if some subset of } S[1..i] \text{ adds to } s$

$= \text{False, ow}$

Output = $S(n, t)$

**STEP 2: Express recursively**
If $a[i] \leq s$,

$S(i, s) = S(i - 1, s - a[i]) \text{ OR } S(i - 1, s)$

Else: $S(i, s) = S(i - 1, s)$

**STEP 3: Order of subtasks**
Row by row, increasing column

**Reconstructing the subset:**
Define an array $D(i, s)$.

If $S(i, s) = \text{True, and } S(i - 1, s - a[i]) = \text{True}$

$D(i, s) = (i - 1, s - a[i])$

Else: $D(i, s) = (i - 1, s)$

Reconstruct the subset by following the pointers from $D(n, t)$

**Running Time** = $O(nt)$
Dynamic Programming

• String Reconstruction
• Longest Common Subsequence
• Subset Sum
• Independent Set in a Tree
Independent Set

**Independent Set:** Given a graph $G = (V, E)$, a subset of vertices $S$ is an independent set if there are no edges between them.

**Max Independent Set Problem:** Given a graph $G = (V, E)$, find the largest independent set in $G$.

**Max Independent Set** is a notoriously hard problem! We will look at a restricted case, when $G$ is a tree.
Max. Independent Set in a Tree

A set of nodes is an **independent set** if there are no edges between the nodes.

**Two Cases at node u:**
1. Don’t include u
2. Include u, and don’t include its children
Max. Independent Set in a Tree

A set of nodes is an independent set if there are no edges between the nodes.

**STEP 1: Define subtask**

\[ l(u) = \text{size of largest independent set in subtree rooted at } u \]

We want \( l(r) \), where \( r = \text{root} \).

**STEP 2: Express recursively**

\[
l(u) = \max \left\{ \sum_{\text{children w of } u} I(w), 1 + \sum_{\text{grandchildren w of } u} I(w) \right\}
\]

Base case: for leaf nodes, \( l(u) = 1 \).

**STEP 3: Order of subtasks**

Reverse order of distance from root; use BFS!

Two Cases at node \( u \):
1. Don’t include \( u \)
2. Include \( u \), and don’t include its children
Max. Independent Set in a Tree

A set of nodes is an **independent set** if there are no edges between the nodes.

**STEP 1: Define subtask**

\[ I(u) = \text{size of largest independent set in subtree rooted at } u \]

We want \( I(r) \), where \( r = \text{root} \)

**STEP 2: Express recursively**

\[
I(u) = \max \left\{ \sum_{\text{children } w \text{ of } u} I(w), 1 + \sum_{\text{grandchildren } w \text{ of } u} I(w) \right\}
\]

Base case: for leaf nodes, \( I(u) = 1 \)

**STEP 3: Order of subtasks**

Reverse order of distance from root; use BFS!

**Running Time: \( O(n) \)**

Edge \((u, v)\) is examined in Step 2 at most twice:

(1) \( v \) is a child of \( u \)

(2) \( v \) is a grandchild of \( u \)'s parent

There are \( n-1 \) edges in a tree on \( n \) nodes.
Dynamic Programming

• String Reconstruction
• Longest Common Subsequence
• Subset Sum
• Independent Set in a Tree
• All Pairs Shortest Paths
Problem: Given n nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

Does Dijkstra’s algorithm work?
Ans: No! Example: s-v Shortest Paths
All Pairs Shortest Paths (APSP)

**Problem:** Given $n$ nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**Structure:**
For all $x, y$:
- either $SP(x, y) = d_{xy}$
- Or there exists some $z$ s.t $SP(x, y) = SP(x, z) + SP(y, z)$

**Property:** If there is no negative weight cycle, then for all $x, y$, $SP(x, y)$ is simple (that is, includes no cycles)
**All Pairs Shortest Paths**

**Problem:** Given $n$ nodes and distances $d_{ij}$ (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

**STEP 1: Define Subtasks**

$D(i,j,k) =$ length of shortest path from $i$ to $j$ with intermediate nodes in $\{1,2,...k\}$

Shortest Path lengths $= D(i,j,n)$

**STEP 2: Express Recursively**

$D(i,j,k) =$ min$\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\}$

Base case: $D(i,j,0) = d_{ij}$

**STEP 3: Order of Subtasks**

By increasing order of $k$

**Running Time** $= O(n^3)$

**Exercise:**
Reconstruct the shortest paths
Summary: 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!)
# Summary: Dynamic Programming vs Divide and Conquer

## Divide-and-conquer

A problem of size $n$ is decomposed into a few subproblems which are significantly smaller (e.g. $n/2$, $3n/4$,...)

Therefore, size of subproblems decreases geometrically.

eg. $n$, $n/2$, $n/4$, $n/8$, etc

Use a recursive algorithm.

## Dynamic programming

A problem of size $n$ is expressed in terms of subproblems that are not much smaller (e.g. $n-1$, $n-2$,...)

A recursive algorithm would take exp. time.

Saving grace: in total, there are only polynomially many subproblems.

Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.
Summary: Common Subtasks in DP

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 $y_1, y_2, ..., y_j$

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

Case 4: Input: a rooted tree. Subproblem: a subtree
Summary: How to Write a Dynamic Programming Solution

1. Define the subproblem (in words)

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

2. Write down recurrence relation

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

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)

See Sample HW Solutions for more examples!