# Algorithms for Programming Contests - Week 08

Prof. Dr. Javier Esparza,
Vincent Fischer,
Jakob Schulz,
conpra@model.cit.tum.de

9th December 2025

# Fibonacci Numbers

## Definition

The *Fibonacci Numbers* are the numbers $(F_n)_{n\in\mathbb{N}}$ recursively defined by

$$F_n := \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-2} + F_{n-1} & \text{if } n \geq 2 \end{cases}$$

# Fibonacci Numbers

## Definition

The *Fibonacci Numbers* are the numbers $(F_n)_{n\in\mathbb{N}}$ recursively defined by

$$F_n := \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-2} + F_{n-1} & \text{if } n \geq 2 \end{cases}$$

**procedure** $\textsc{Fib}(n)$
    **if** $n < 2$ **then**
        **return** $n$
    **else**
        **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$

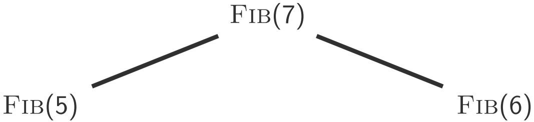
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$

$\textsc{Fib}(7)$

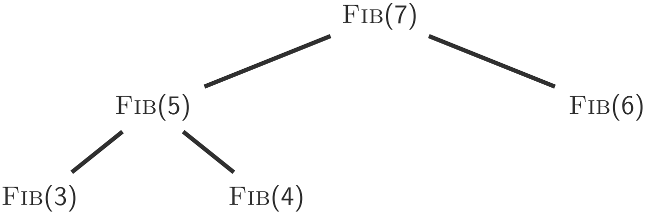
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



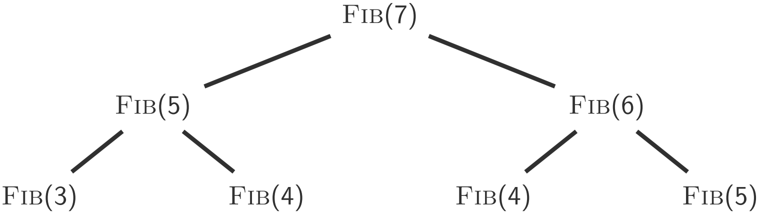
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



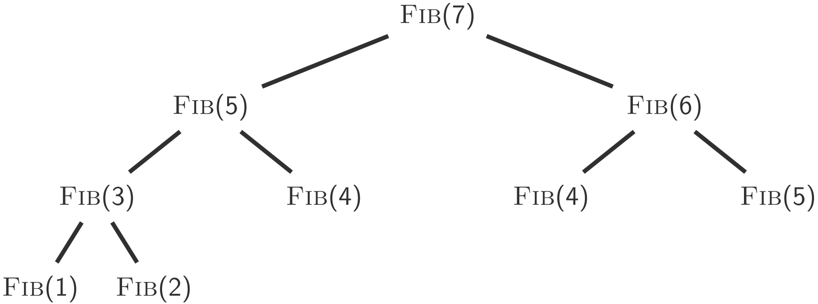
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



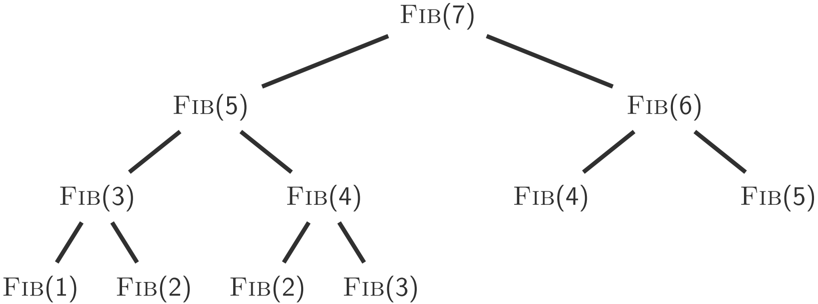
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



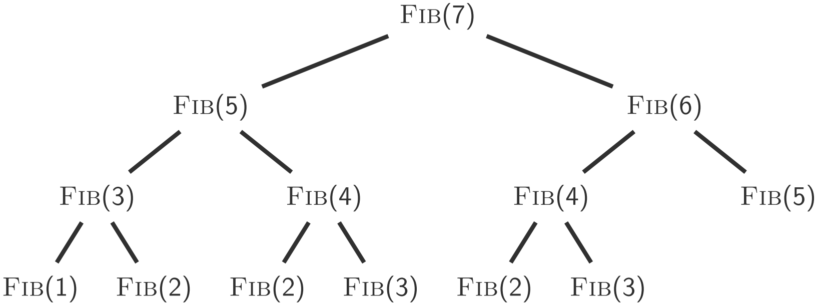
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



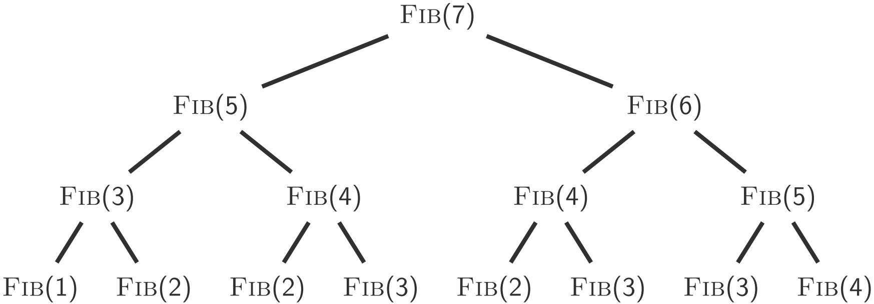
# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



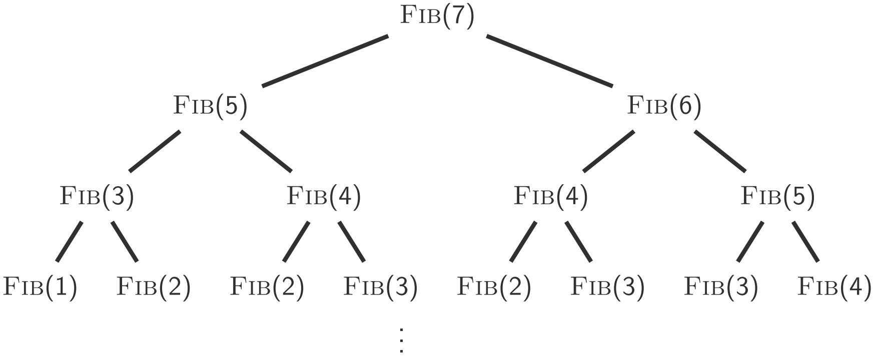
# Fibonacci Numbers

$\text{FIB}(n)$: **return** $\text{FIB}(n-2) + \text{FIB}(n-1)$



# Fibonacci Numbers

$\textsc{Fib}(n)$: **return** $\textsc{Fib}(n-2) + \textsc{Fib}(n-1)$



# Execution Times

| $n$ | C | OCaml | Rust | Julia | Python |
|---|---|---|---|---|---|
| 5 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.30 | 0:00.01 |
| 10 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.31 | 0:00.01 |
| 15 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.31 | 0:00.01 |
| 20 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.31 | 0:00.02 |
| 25 | 0:00.00 | 0:00.01 | 0:00.00 | 0:00.32 | 0:00.05 |
| 30 | 0:00.01 | 0:00.05 | 0:00.02 | 0:00.31 | 0:00.38 |
| 40 | 0:01.86 | 0:06.53 | 0:01.86 | 0:00.30 | TIMEOUT |
| 50 | TIMEOUT | TIMEOUT | TIMEOUT | 0:00.30 | TIMEOUT |

# Two approaches

**Algorithm** Bottom-Up

**procedure** $\textsc{Fib}(n)$
  Initialize array $f$ of size $n+1$
  $f[0] \leftarrow 0$
  $f[1] \leftarrow 1$
  **for** $i = 2, 3, \ldots, n$ **do**
    $f[i] \leftarrow f[i-2] + f[i-1]$
  **return** $f[n]$

# Two approaches

**Algorithm** Bottom-Up

```
procedure FIB(n)
    Initialize array f of size n + 1
    f[0] ← 0
    f[1] ← 1
    for i = 2, 3, ..., n do
        f[i] ← f[i − 2] + f[i − 1]
    return f[n]
```

**Algorithm** Top-Down

```
Statically initialize CACHE
procedure FIB(n)
    if n < 2 then
        return n
    if CACHE(n) empty then
        CACHE(n) ← FIB(n−2) + FIB(n−1)
    return CACHE(n)
```

# Execution Times

E.g. Bottom-Up:

| $n$ | C | OCaml | Rust | Julia | Python |
|---|---|---|---|---|---|
| 5 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.32 | 0:00.05 |
| 10 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.31 | 0:00.05 |
| 15 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.30 | 0:00.05 |
| 20 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.32 | 0:00.05 |
| 25 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.30 | 0:00.05 |
| 30 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.30 | 0:00.05 |
| 40 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.32 | 0:00.05 |
| 50 | 0:00.00 | 0:00.00 | 0:00.00 | 0:00.32 | 0:00.05 |

# Dynamic Programming

Whenever a problem

# Dynamic Programming

Whenever a problem

1. can be solved by recursively solving subproblems and

# Dynamic Programming

Whenever a problem

1. can be solved by recursively solving subproblems and
2. subproblems are generated repeatedly

# Dynamic Programming

Whenever a problem

1. can be solved by recursively solving subproblems and
2. subproblems are generated repeatedly

one can use Dynamic Programming (DP).

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

Straight-forward to implement

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

  Straight-forward to implement
  Only computing relevant subproblems

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

    Straight-forward to implement
    Only computing relevant subproblems
    Good for sparse statespace

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

  Straight-forward to implement
  Only computing relevant subproblems
  Good for sparse statespace

## Bottom-Up

- Fill table for all smaller subproblems first
- Save results in array

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

  Straight-forward to implement
  Only computing relevant subproblems
  Good for sparse statespace

## Bottom-Up

- Fill table for all smaller subproblems first
- Save results in array

  Better cache locality

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

  Straight-forward to implement
  Only computing relevant subproblems
  Good for sparse statespace

## Bottom-Up

- Fill table for all smaller subproblems first
- Save results in array

  Better cache locality
  No Hash collisions possible

# DP: Two approaches

## Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

  Straight-forward to implement
  Only computing relevant subproblems
  Good for sparse statespace

## Bottom-Up

- Fill table for all smaller subproblems first
- Save results in array

  Better cache locality
  No Hash collisions possible
  Good for dense statespace

# Two approaches

Assume we are given a Problem where Dynamic Programming is applicable:

# Two approaches

Assume we are given a Problem where Dynamic Programming is applicable:

- Let $I$ be the set of all inputs, and write $g(i)$ for the desired output on input $i \in I$.

# Two approaches

Assume we are given a Problem where Dynamic Programming is applicable:

- Let $I$ be the set of all inputs, and write $g(i)$ for the desired output on input $i \in I$.
- For each $i \in I$, assume there are subproblems $i_1, \ldots, i_k \in I$ (with $k \in \mathbb{N}$) s.t. $g(i)$ can be solved by solving $g(i_1), \ldots, g(i_k)$ and combining the results with some function $f_i$, i.e.

$$g(i) = f_i(g(i_1), \ldots, g(i_k))$$

Write $s(i) := (i_1, \ldots, i_k)$ and assume there is some procedure $\textsc{Subproblems}$ computing $s$.

# Two approaches

Assume we are given a Problem where Dynamic Programming is applicable:

- Let $I$ be the set of all inputs, and write $g(i)$ for the desired output on input $i \in I$.
- For each $i \in I$, assume there are subproblems $i_1, \ldots, i_k \in I$ (with $k \in \mathbb{N}$) s.t. $g(i)$ can be solved by solving $g(i_1), \ldots, g(i_k)$ and combining the results with some function $f_i$, i.e.

  $$g(i) = f_i(g(i_1), \ldots, g(i_k))$$

  Write $s(i) := (i_1, \ldots, i_k)$ and assume there is some procedure $\textsc{Subproblems}$ computing $s$.
- Usually, we assume there is some well-founded order $<$ on $I$ s.t. for each $i \in I$, we have $i_1, \ldots, i_k < i$, where $(i_1, \ldots, i_k) = s(i)$.

# Two approaches

Assume we are given a Problem where Dynamic Programming is applicable:

- Let $I$ be the set of all inputs, and write $g(i)$ for the desired output on input $i \in I$.
- For each $i \in I$, assume there are subproblems $i_1, \ldots, i_k \in I$ (with $k \in \mathbb{N}$) s.t. $g(i)$ can be solved by solving $g(i_1), \ldots, g(i_k)$ and combining the results with some function $f_i$, i.e.

  $$g(i) = f_i(g(i_1), \ldots, g(i_k))$$

  Write $s(i) := (i_1, \ldots, i_k)$ and assume there is some procedure $\textsc{Subproblems}$ computing $s$.
- Usually, we assume there is some well-founded order $<$ on $I$ s.t. for each $i \in I$, we have $i_1, \ldots, i_k < i$, where $(i_1, \ldots, i_k) = s(i)$.

Then, for some $i_0 \in I$, we can find $g(i_0)$...

# Top-Down approach

…Top-Down:

- Start with $i_0$
- Look up if the result is already cached
- If not, recursively call subproblems and compute result
- Store result in cache
- Return result

**Algorithm** DP (Top-Down)

```
Statically initialize CACHE
procedure G(i)
    if CACHE(i) empty then
        (i_1, ..., i_k) ← SUBPROBLEMS(i)
        CACHE(i) ← f_i(G(i_1), ..., G(i_k))
    return CACHE(i)
```

# Top-Down: Remark

- For Rust, the cached crate can easily convert any function to a cached version

# Top-Down: Remark

- For Rust, the cached crate can easily convert any function to a cached version
- However, crates cannot be used in our course!

# Bottom-Up approach

...Bottom-Up:

- Iteratively fill table with results of needed subproblems, in "increasing" order

# Bottom-Up approach

...Bottom-Up:

- Let $S := \{ i \in I \mid i \leq i_0 \}$

# Bottom-Up approach

...Bottom-Up:

- Let $S := \{ i \in I \mid i \leq i_0 \}$
- Assume we have ranking/unranking functions for $S$, i.e. bijections

$$\begin{aligned} r :& \{0, 1, \ldots, |S| - 1\} \to S, \\ u :& S \to \{0, 1, \ldots, |S| - 1\} \end{aligned}$$

where $u = r^{-1}$.

# Bottom-Up approach

...Bottom-Up:

- Let $S := \{\, i \in I \mid i \leq i_0 \,\}$
- Assume we have ranking/unranking functions for $S$, i.e. bijections

$$\begin{aligned} r :& \{0, 1, \ldots, |S| - 1\} \to S, \\ u :& S \to \{0, 1, \ldots, |S| - 1\} \end{aligned}$$

  where $u = r^{-1}$.
- Further assume that the ranking function is order-preserving (or, equivalently, that $u$ is a topological order).

# Bottom-Up approach

...Bottom-Up:

- Let $S := \{\, i \in I \mid i \leq i_0 \,\}$
- Assume we have ranking/unranking functions for $S$, i.e. bijections

$$\begin{aligned} r :&\{0, 1, \ldots, |S| - 1\} \to S, \\ u :&S \to \{0, 1, \ldots, |S| - 1\} \end{aligned}$$

  where $u = r^{-1}$.
- Further assume that the ranking function is order-preserving (or, equivalently, that $u$ is a topological order).
- Create an array $G$ of size $|S|$.

# Bottom-Up approach

...Bottom-Up:

- Let $S := \{\, i \in I \mid i \leq i_0 \,\}$
- Assume we have ranking/unranking functions for $S$, i.e. bijections

  $$\begin{aligned} r :&\{0, 1, \ldots, |S| - 1\} \to S, \\ u :&S \to \{0, 1, \ldots, |S| - 1\} \end{aligned}$$

  where $u = r^{-1}$.
- Further assume that the ranking function is order-preserving (or, equivalently, that $u$ is a topological order).
- Create an array $G$ of size $|S|$.
- Fill array in increasing order, s.t. $G[j] = g(r(j))$.

# Bottom-Up approach

...Bottom-Up:

- Let $S := \{ i \in I \mid i \leq i_0 \}$
- Assume we have ranking/unranking functions for $S$, i.e. bijections

$$\begin{aligned} r :& \{0, 1, \ldots, |S| - 1\} \to S, \\ u :& S \to \{0, 1, \ldots, |S| - 1\} \end{aligned}$$

  where $u = r^{-1}$.
- Further assume that the ranking function is order-preserving (or, equivalently, that $u$ is a topological order).
- Create an array $G$ of size $|S|$.
- Fill array in increasing order, s.t. $G[j] = g(r(j))$.
- Return $G[u(i_0)]$.

**Algorithm** DP (Bottom-Up)

```
procedure G(i)
    Initialize array G of size |S|
    for j = 0, 1, ..., |S| − 1 do
        i ← r(j)
        (i_1, ..., i_k) ← SUBPROBLEMS(i)
        (j_1, ..., j_k) ← (u(i_1), ..., u(i_k))
        G[j] ← f_i(G[j_1], ..., G[j_k])
    return G[u(i_0)]
```

# Bottom-Up: Remarks

- $S$ need not contain all of $\{ i \in I \mid i \leq i_0 \}$. Only requirements:

# Bottom-Up: Remarks

- $S$ need not contain all of $\{ i \in I \mid i \leq i_0 \}$. Only requirements:
  1. $i_0 \in S$

# Bottom-Up: Remarks

- $S$ need not contain all of $\{\, i \in I \mid i \leq i_0 \,\}$. Only requirements:
  1. $i_0 \in S$
  2. For all $i \in S$ and $j \in s(i)$, $j \in S$

# Bottom-Up: Remarks

- $S$ need not contain all of $\{ i \in I \mid i \leq i_0 \}$. Only requirements:
  1. $i_0 \in S$
  2. For all $i \in S$ and $j \in s(i)$, $j \in S$
- The smallest set $S$ satisfying these requirements is exactly the set of instances generated by the top-down approach

# Bottom-Up: Remarks

- $S$ need not contain all of $\{ i \in I \mid i \leq i_0 \}$. Only requirements:
  1. $i_0 \in S$
  2. For all $i \in S$ and $j \in s(i)$, $j \in S$
- The smallest set $S$ satisfying these requirements is exactly the set of instances generated by the top-down approach
- However, coming up with good ranking functions is usually very hard!

# Bottom-Up: Remarks

- $S$ need not contain all of $\{ i \in I \mid i \leq i_0 \}$. Only requirements:
  1. $i_0 \in S$
  2. For all $i \in S$ and $j \in s(i)$, $j \in S$
- The smallest set $S$ satisfying these requirements is exactly the set of instances generated by the top-down approach
- However, coming up with good ranking functions is usually very hard!
- In practice, often
  - multi-dimensional array used, i.e. instances are ranked by tuples

# Bottom-Up: Remarks

- $S$ need not contain all of $\{ i \in I \mid i \leq i_0 \}$. Only requirements:
  1. $i_0 \in S$
  2. For all $i \in S$ and $j \in s(i)$, $j \in S$
- The smallest set $S$ satisfying these requirements is exactly the set of instances generated by the top-down approach
- However, coming up with good ranking functions is usually very hard!
- In practice, often
  - multi-dimensional array used, i.e. instances are ranked by tuples
  - need not implement "real" instances $S$ and ranking/unranking functions, instead directly work on indices (index tuples)

# Change Making

What is the minimum number of coins to make 40 cents?


1$


25¢


20¢


1¢

What are the subproblems?

# Change Making

What is the minimum number of coins to make 40 cents?


1\$ 25¢ 20¢ 1¢

What are the subproblems?

Add 1 coin to the solutions for 40¢ − 25¢, 40¢ − 20¢, 40¢ − 1¢

# Change Making

What is the minimum number of coins to make 40 cents?


1$


25¢


20¢


1¢

What are the subproblems?
Add 1 coin to the solutions for 40¢ − 25¢, 40¢ − 20¢, 40¢ − 1¢

Let $g(i)$ := min. number of coins needed to make $i$¢

# Change Making

What is the minimum number of coins to make 40 cents?


1\$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for 40¢ − 25¢, 40¢ − 20¢, 40¢ − 1¢

Let $g(i)$ := min. number of coins needed to make $i$¢

$$s(i) = \{\, i - j \mid j \in \{100, 25, 20, 1\} \wedge i - j \geq 0 \,\}$$

# Change Making

What is the minimum number of coins to make 40 cents?


1$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for 40¢ − 25¢, 40¢ − 20¢, 40¢ − 1¢

Let $g(i)$ := min. number of coins needed to make $i$¢

$$s(i) = \{\, i - j \mid j \in \{100, 25, 20, 1\} \wedge i - j \geq 0 \,\}$$

$$g(i) = \begin{cases} \min \{\, g(j) \mid j \in s(i) \,\} + 1 & \text{if } s(i) \neq () \\ & \end{cases}$$

# Change Making

What is the minimum number of coins to make 40 cents?

1\$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for 40¢ − 25¢, 40¢ − 20¢, 40¢ − 1¢

Let $g(i) :=$ min. number of coins needed to make $i$¢

$$s(i) = \{\, i - j \mid j \in \{100, 25, 20, 1\} \wedge i - j \geq 0 \,\}$$

$$g(i) = \begin{cases} \min\{\, g(j) \mid j \in s(i) \,\} + 1 & \text{if } s(i) \neq () \\ 0 & \text{if } s(i) = () \text{ and } i = 0 \\ & \end{cases}$$

# Change Making

What is the minimum number of coins to make 40 cents?


1\$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for 40¢ − 25¢, 40¢ − 20¢, 40¢ − 1¢

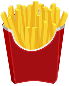
Let $g(i) :=$ min. number of coins needed to make $i$¢

$$s(i) = \{\, i - j \mid j \in \{100, 25, 20, 1\} \wedge i - j \geq 0 \,\}$$

$$g(i) = \begin{cases} \min\{\, g(j) \mid j \in s(i) \,\} + 1 & \text{if } s(i) \neq () \\ 0 & \text{if } s(i) = () \text{ and } i = 0 \\ \texttt{IMPOSSIBLE} & \text{if } s(i) = () \text{ and } i > 0 \end{cases}$$

# 0/1 Knapsack

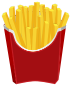
Maximum capacity $W = 10$




| | | | | |
|---|---|---|---|---|
| Values $v_i$: | 10 | 40 | 30 | 50 |
| Weights $w_i$: | 5 | 4 | 6 | 3 |

# 0/1 Knapsack

Maximum capacity $W = 10$

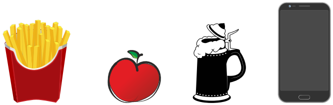
   

| | | | | |
|---|---|---|---|---|
| Values $v_i$: | 10 | 40 | 30 | 50 |
| Weights $w_i$: | 5 | 4 | 6 | 3 |

Possible approach: Build 2-dimensional table $G[n+1][W+1]$ where $G[i, w]$ considers a backpack of size $w \leq W$ and items $1, \ldots, i$ only.
Recurse as follows:

# 0/1 Knapsack

Maximum capacity $W = 10$

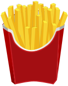


| | | | | |
|---|---|---|---|---|
| Values $v_i$: | 10 | 40 | 30 | 50 |
| Weights $w_i$: | 5 | 4 | 6 | 3 |

Possible approach: Build 2-dimensional table $G[n+1][W+1]$ where $G[i,w]$ considers a backpack of size $w \leq W$ and items $1, \ldots, i$ only.
Recurse as follows:

$$G[i,w] \leftarrow \begin{cases} 0 & \text{if } i = 0 \\ G[i-1,w] & \text{if } i > 0 \wedge w_i > w \\ \max(G[i-1,w], G[i-1,w-w_i] + v_i) & \text{if } i > 0 \wedge w_i \leq w \end{cases}$$

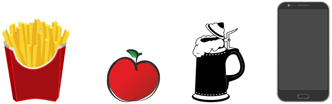
# 0/1 Knapsack

| | | | | | |
|---|---|---|---|---|---|
| | |  |  |  |  |
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |

Max. capacity $W = 10$

$$\mathtt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

# 0/1 Knapsack

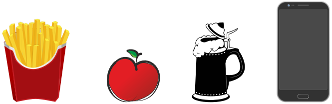


| | | | | | |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

# 0/1 Knapsack

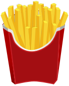


| | | | | | |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

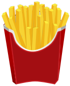
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

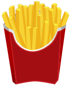
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | | | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

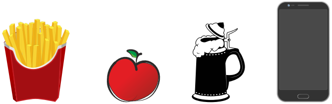
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | | | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

# 0/1 Knapsack

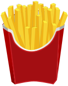


| | | | | | |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

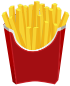
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

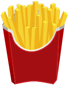
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | |

Max. capacity $W = 10$

$$\max \begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

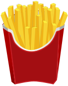
# 0/1 Knapsack

| | | | | | |
|---|---|---|---|---|---|
| | |  |  |  |  |
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | |

Max. capacity $W = 10$

$$\mathtt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

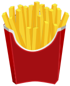
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

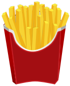
# 0/1 Knapsack

| | | | | | |
|---|---|---|---|---|---|
| | |  |  |  |  |
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

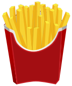
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |
| 7 | 0 | 10 | 40 | 40 | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

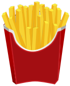
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |
| 7 | 0 | 10 | 40 | 40 | |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

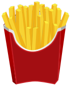
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |
| 7 | 0 | 10 | 40 | 40 | 90 |

Max. capacity $W = 10$

$$\mathtt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

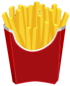
# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |
| 7 | 0 | 10 | 40 | 40 | 90 |
| 8 | 0 | 10 | 40 | 40 | 90 |
| 9 | 0 | 10 | 50 | 50 | 90 |
| 10 | 0 | 10 | 50 | 70 | **90** |

Max. capacity $W = 10$

$$\mathtt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

# 0/1 Knapsack

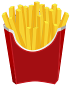
| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |
| 7 | 0 | 10 | 40 | 40 | 90 |
| 8 | 0 | 10 | 40 | 40 | 90 |
| 9 | 0 | 10 | 50 | 50 | 90 |
| 10 | 0 | 10 | 50 | 70 | **90** |

Max. capacity $W = 10$

$$\texttt{max}\begin{pmatrix} G[i-1, w], \\ G[i-1, w - w_i] + v_i \end{pmatrix}$$

How to compute the solution?

# 0/1 Knapsack

| | |  |  |  |  |
|---|---|---|---|---|---|
| $v_i$: | | 10 | 40 | 30 | 50 |
| $w_i$: | | 5 | 4 | 6 | 3 |
| $w$ | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 50 |
| 4 | 0 | 0 | 40 | 40 | 50 |
| 5 | 0 | 10 | 40 | 40 | 50 |
| 6 | 0 | 10 | 40 | 40 | 50 |
| 7 | 0 | 10 | 40 | 40 | 90 |
| 8 | 0 | 10 | 40 | 40 | 90 |
| 9 | 0 | 10 | 50 | 50 | 90 |
| 10 | 0 | 10 | 50 | 70 | **90** |

Max. capacity $W = 10$

$$\max \begin{pmatrix} G[i-1, w], \\ G[i-1, w-w_i] + v_i \end{pmatrix}$$

How to compute the solution?
`predecessor` array storing incoming edge

# Longest Increasing Subsequence

## Problem

Given a sequence of numbers

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

What is the length of the longest *increasing subsequence* (LIS)?

# Longest Increasing Subsequence

## Problem

Given a sequence of numbers

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

What is the length of the longest *increasing subsequence* (LIS)?

## Answer

6. But how? What are the subproblems?

# Longest Increasing Subsequence

## Problem

Given a sequence of numbers

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

What is the length of the longest *increasing subsequence* (LIS)?

## Answer

6. Compute the solution for shorter sequences and build up.

# Longest Increasing Subsequence

s[i] := “What is the length of the longest increasing subsequence that ends at the position i.”

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | | | | | | | | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | | | | | | | | |

# Longest Increasing Subsequence

s[i] := “What is the length of the longest increasing subsequence that ends at the position i.”

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | | | | | | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | | | | | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | | | | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | | | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | | |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | 2 | |

# Longest Increasing Subsequence

s[i] := “What is the length of the longest increasing subsequence that ends at the position i.”

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | 2 | 4 |

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | 2 | 4 |

## Observation

$s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | 2 | 4 |

## Observation

$s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

## Easy algorithm

Compute $s[i] = \max \{ s[j] \mid j < i \wedge v[j] < v[i] \} + 1$.

# Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence that ends at the position i."

| v[i] | 0, | 8, | 4, | 12, | 2, | 10, | 6, | 14, | 1, | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | 2 | 4 |

## Observation

$s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

## Easy algorithm

Compute $s[i] = \max \{ s[j] \mid j < i \wedge v[j] < v[i] \} + 1$. $\mathcal{O}(n^2)$

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | | | | | | | | | | |

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | | | | | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := “What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$.”

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | | | | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | | | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 1 | | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 2 | | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 2 | 3 | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 4 | 3 | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 4 | 5 | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 4 | 6 | | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 4 | 6 | 7 | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$."

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 8 | 6 | 7 | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: improved algorithm

m[$l$] := “What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length $l$ ending at $v[j]$.”

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| v[i] | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 |
| m[i] | 0 | 0 | 8 | 6 | 9 | | | | | |

```
if v.length() == 0:
  return 0

parent[0] = None
m[0] = 0
maxlength = 1

for i in {1, 2, ..., v.length() - 1}:
  if v[i] <= v[m[0]]:
    parent[i] = None
    m[0] = i
    continue
  # Binary Search for smallest j, s.t.
  # v[i] <= v[m[j]] and j < i
  j = search()
  parent[i] = m[j - 1]
  m[j] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

# LIS: Remarks

- Second version runs in $\mathcal{O}(n \log n)$

# LIS: Remarks

- Second version runs in $\mathcal{O}(n \log n)$
- Reference: Fredman (1975)

# LIS: Remarks

- Second version runs in $\mathcal{O}(n \log n)$
- Reference: Fredman (1975)
- Fredman also showed: worst-case complexity of any LIS algorithm in $\Omega(n \log n)$