



THE UNIVERSITY OF
MELBOURNE

COMP90038

Algorithms and Complexity

Lecture 6: Recursion
(with thanks to Harald Søndergaard)

Toby Murray



toby.murray@unimelb.edu.au



DMD 8.17 (Level 8, Doug McDonell Bldg)



<http://people.eng.unimelb.edu.au/tobym>



@tobycmurray

Tutorials



- We have some very empty tutorials on Monday:
 - Monday 12pm, David Caro 205
 - Monday 3:15pm (David Caro 201 and 205)
- We also have some tutorials that are over attended
 - Note: you are not permitted to exceed the capacity of the tutorial rooms

Recursion

- We've already seen some examples
- A very natural approach when the data structure is recursive (e.g. lists, trees)
- But also examples of naturally recursive array processing algorithms
- Next week we'll express **depth first graph traversal** recursively (the natural way); later we'll meet other examples of recursion too

Example: Factorial



- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial



- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$F(5)$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$F(5) = F(4) \cdot 5$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial



- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial



- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```


Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5) \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$   
   $n: 5$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$   
  
   $n: 5$   
  
   $result: 1$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

n : 5

result: 5

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$   
  
   $n: 4$   
  
   $result: 5$ 
```


Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

n : 4

result: 20

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

n : 3

result: 20

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$   
  
   $n: 3$   
  
   $result: 60$ 
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

n : 2

result: 60

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

n : 2

result: 120

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$   
  
n: 1  
  
result: 120
```

Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$   
  
   $n: 0$   
  
   $result: 120$ 
```


Example: Factorial

- $n!$: we can use recursion (left) or iteration (right)

```
function FAC( $n$ )  
  if  $n = 0$  then  
    return 1  
  return FAC( $n - 1$ ) *  $n$ 
```

$$\begin{aligned} F(5) &= F(4) \cdot 5 \\ &= (F(3) \cdot 4) \cdot 5 \\ &= ((F(2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((F(1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 \\ &= (((((F(0) \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5) \\ &= (((((1 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5) \\ &= 120 \end{aligned}$$

```
function FAC( $n$ )  
   $result \leftarrow 1$   
  while  $n > 0$  do  
     $result \leftarrow result * n$   
     $n \leftarrow n - 1$   
  return  $result$ 
```

n : 0

result: 120

*iterative version
normally
preferred since it is
constant space*

Example: Fibonacci Numbers



THE UNIVERSITY OF
MELBOURNE

- To generate the n th number of sequence: 1 1 2 3 5
8 13 21 34 55 ...

```
function FIB( $n$ )  
  if  $n = 0$  then  
    return 1  
  if  $n = 1$  then  
    return 1  
  return FIB( $n - 1$ ) + FIB( $n - 2$ )
```

Follows the mathematical
definition of Fibonacci
numbers very closely.

Easy to understand

But performs lots of
redundant computation

Basic operation: addition

Complexity is **exponential** in n

Fibonacci Again

- Of course we only need to remember the latest two items. Recursive version: left; iterative version: right

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

```
function FIB( $n$ )  
   $a \leftarrow 1$   
   $b \leftarrow 0$   
  while  $n > 0$  do  
     $t \leftarrow a$   
     $a \leftarrow a + b$   
     $b \leftarrow t$   
     $n \leftarrow n - 1$   
  return  $a$ 
```

Time complexity of both
solutions is linear in n

(There is a cleverer, still recursive, way which is $O(\log n)$.)

Fibonacci Again



- Of course we only need to remember the latest two items. Recursive version: left; iterative version: right

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib($n, 1, 0$)

Time complexity of both
solutions is linear in n

```
function FIB( $n$ )  
   $a \leftarrow 1$   
   $b \leftarrow 0$   
  while  $n > 0$  do  
     $t \leftarrow a$   
     $a \leftarrow a + b$   
     $b \leftarrow t$   
     $n \leftarrow n - 1$   
  return  $a$ 
```

(There is a cleverer, still recursive, way which is $O(\log n)$.)

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Fibonacci
Sequence:
1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

*Fibonacci
Sequence:
1 1 2 3 5 8 ...*

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

Fib(5,1,0)

*Fibonacci
Sequence:*

1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

Fib(5, 1, 0)
= Fib(4, 1, 1)

Fibonacci
Sequence:
1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

Fib(5,1,0)
= Fib(4,1,1)
= Fib(3,2,1)

Fibonacci
Sequence:
1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

Fib(5,1,0)
= Fib(4,1,1)
= Fib(3,2,1)
= Fib(2,3,2)

Fibonacci
Sequence:
1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

Fib(5,1,0)
= Fib(4,1,1)
= Fib(3,2,1)
= Fib(2,3,2)
= Fib(1,5,3)

Fibonacci
Sequence:
1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

Fib(5,1,0)
= Fib(4,1,1)
= Fib(3,2,1)
= Fib(2,3,2)
= Fib(1,5,3)
= Fib(0,8,5)

Fibonacci
Sequence:
1 1 2 3 5 8 ...

Tracing Recursive Fibonacci



THE UNIVERSITY OF
MELBOURNE

```
function FIB( $n, a, b$ )  
  if  $n = 0$  then  
    return  $a$   
  return FIB( $n - 1, a + b, a$ )
```

Initial call: Fib(5, 1, 0)

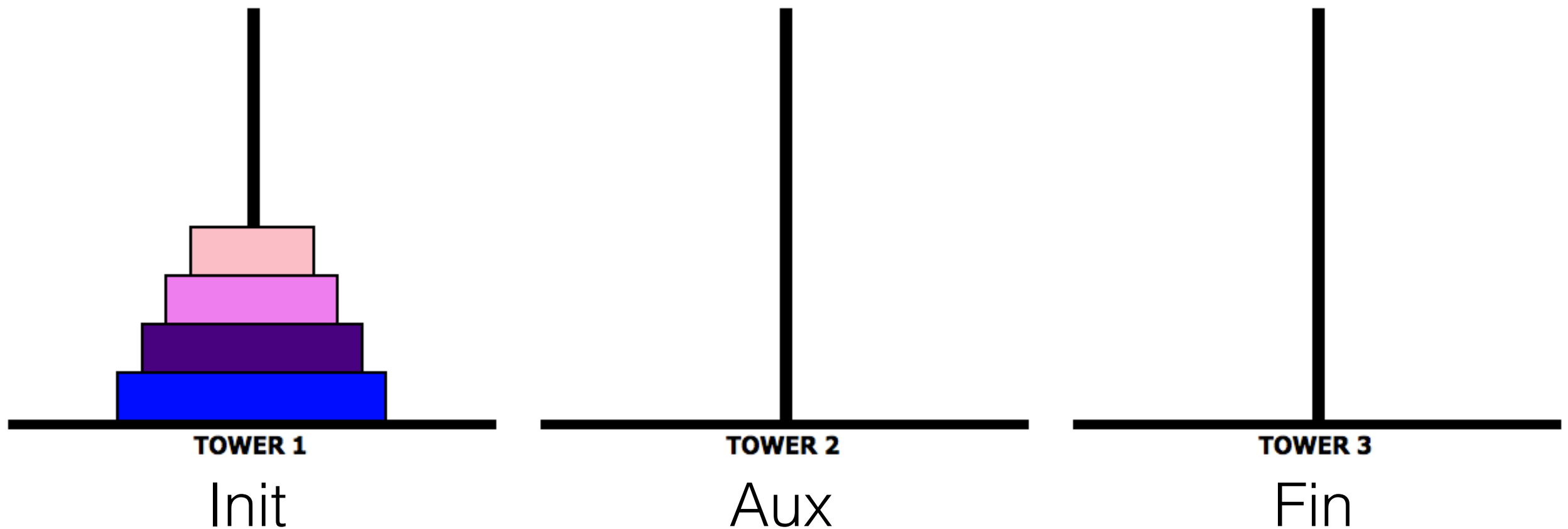
Fib(5,1,0)
= Fib(4,1,1)
= Fib(3,2,1)
= Fib(2,3,2)
= Fib(1,5,3)
= Fib(0,8,5) = 8

*Fibonacci
Sequence:
1 1 2 3 5 8 ...*

Tower of Hanoi



THE UNIVERSITY OF
MELBOURNE

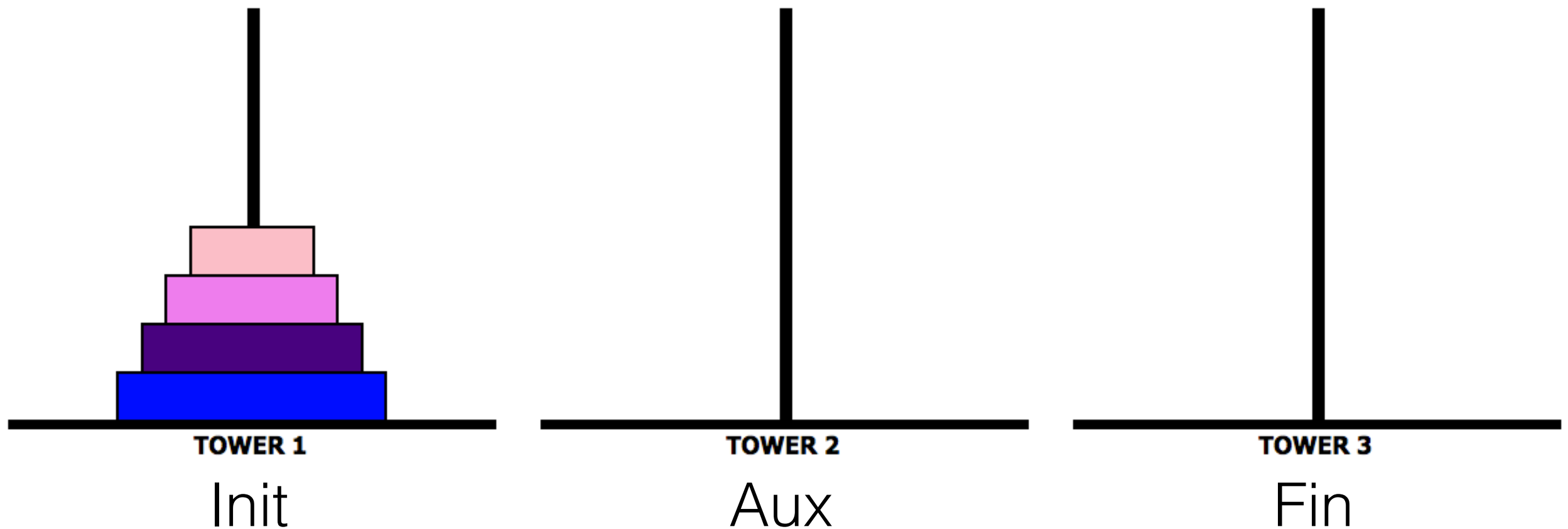


Move n disks from *Init* to *Fin*. A larger disk can never be placed on top of a smaller one.

Tower of Hanoi: Recursive Solution



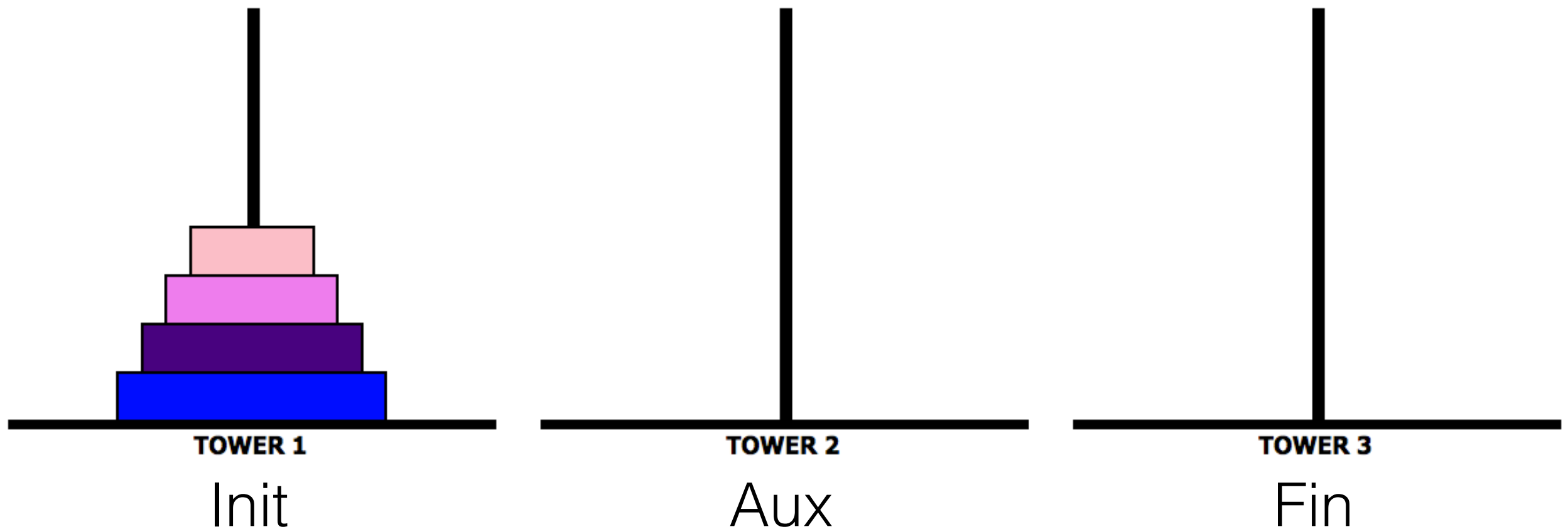
THE UNIVERSITY OF
MELBOURNE



Tower of Hanoi: Recursive Solution



THE UNIVERSITY OF
MELBOURNE

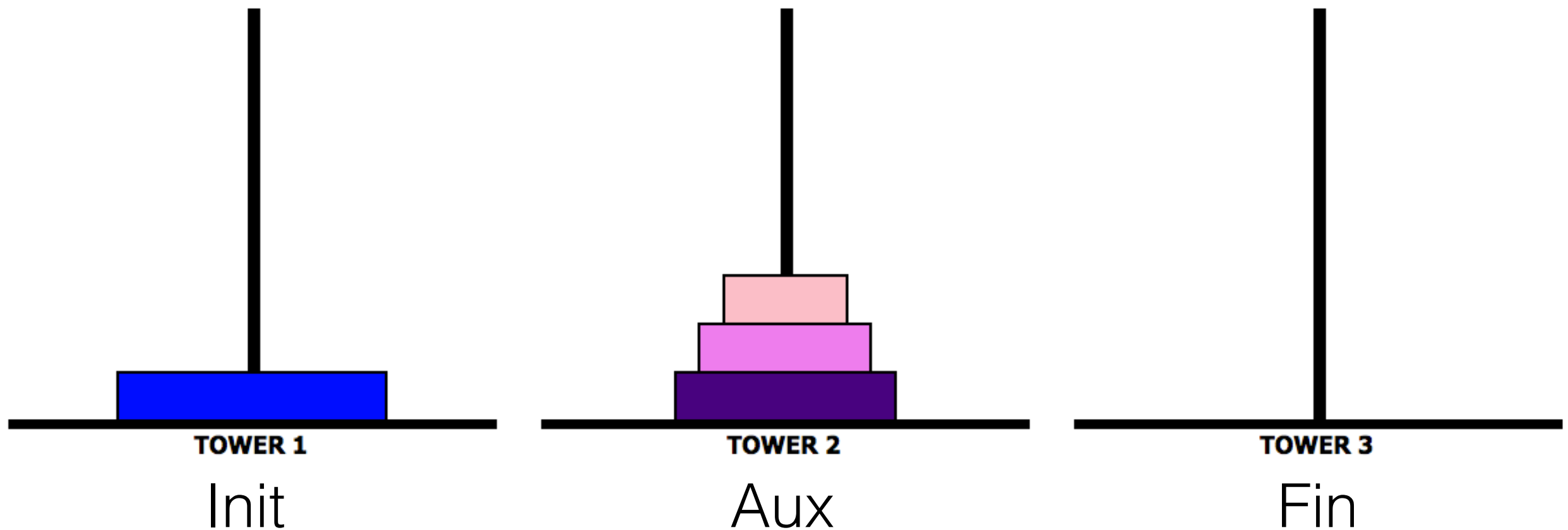


Move $n-1$ disks from Init to Aux.

Tower of Hanoi: Recursive Solution



THE UNIVERSITY OF
MELBOURNE

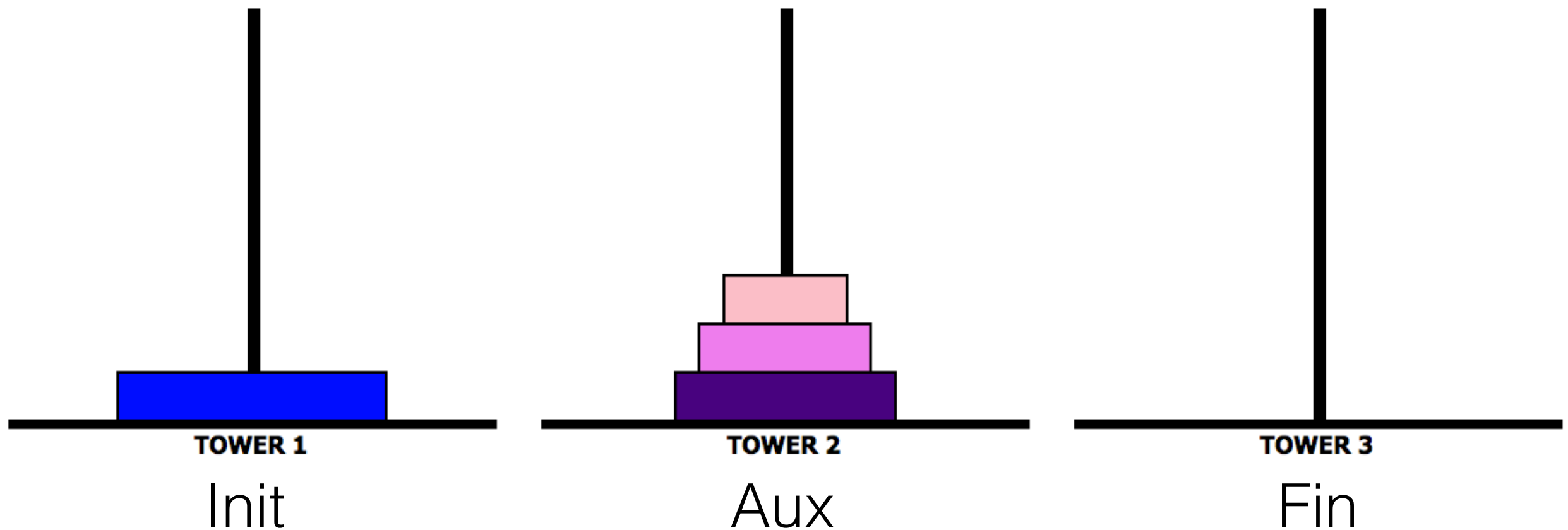


Move $n-1$ disks from Init to Aux.

Tower of Hanoi: Recursive Solution



THE UNIVERSITY OF
MELBOURNE

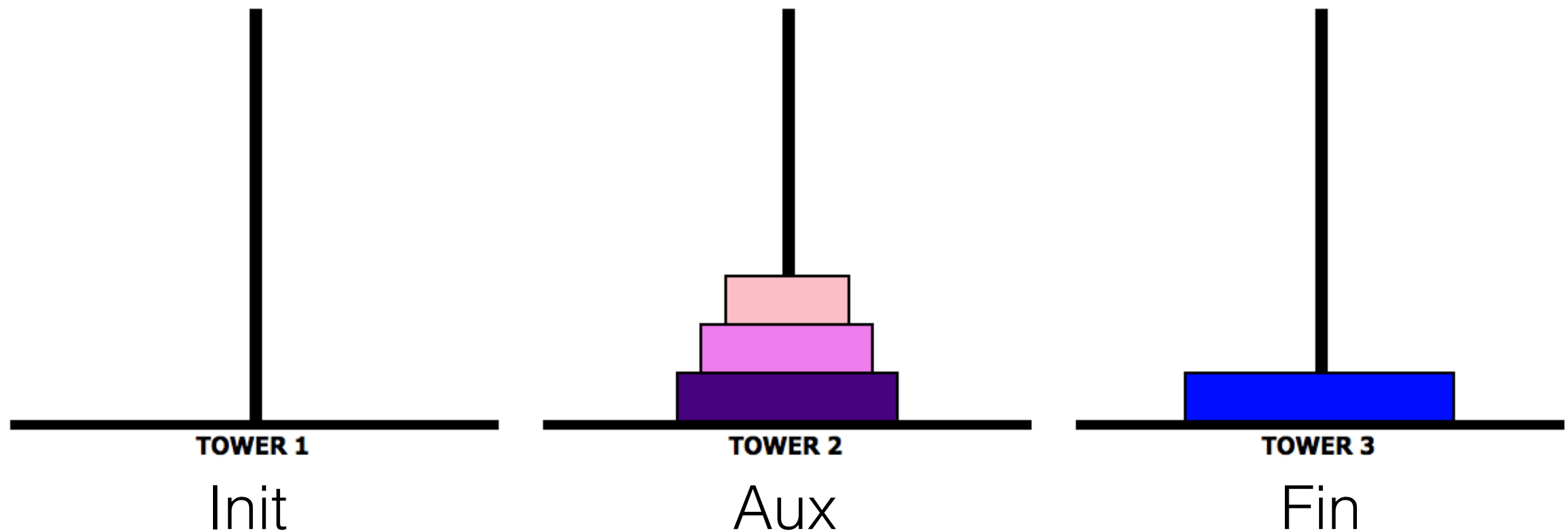


Move $n-1$ disks from Init to Aux.
Then move the n th disk to Fin.

Tower of Hanoi: Recursive Solution



THE UNIVERSITY OF
MELBOURNE

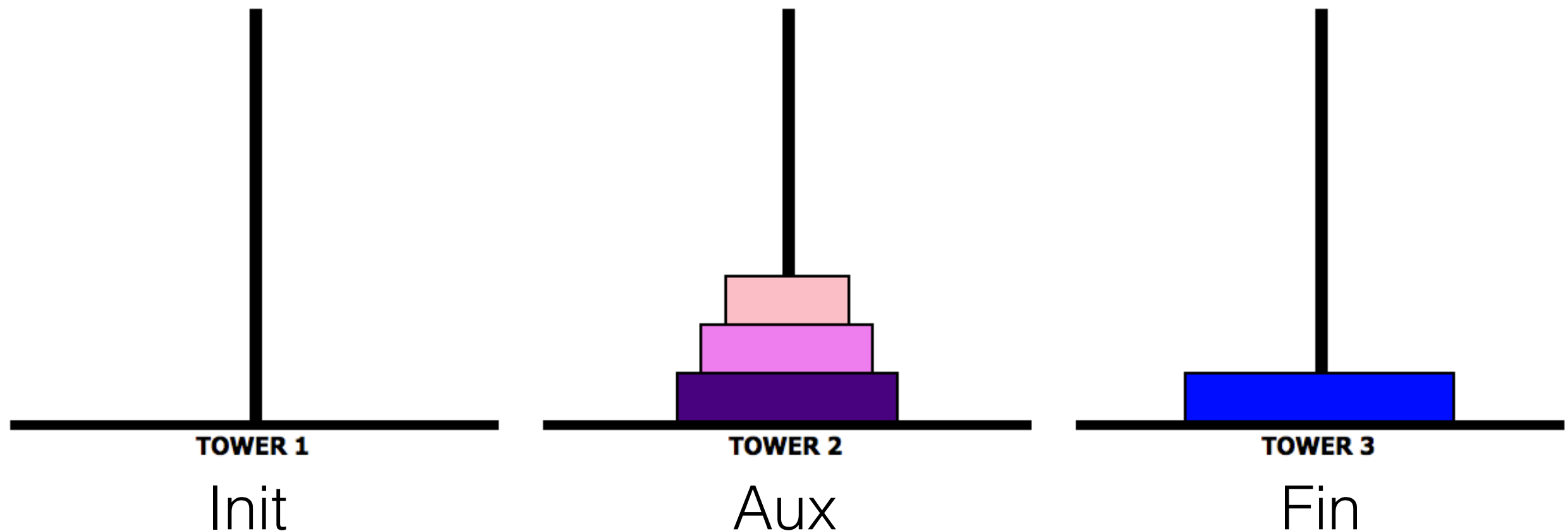


Move $n-1$ disks from Init to Aux.
Then move the n th disk to Fin.

Tower of Hanoi: Recursive Solution



THE UNIVERSITY OF
MELBOURNE

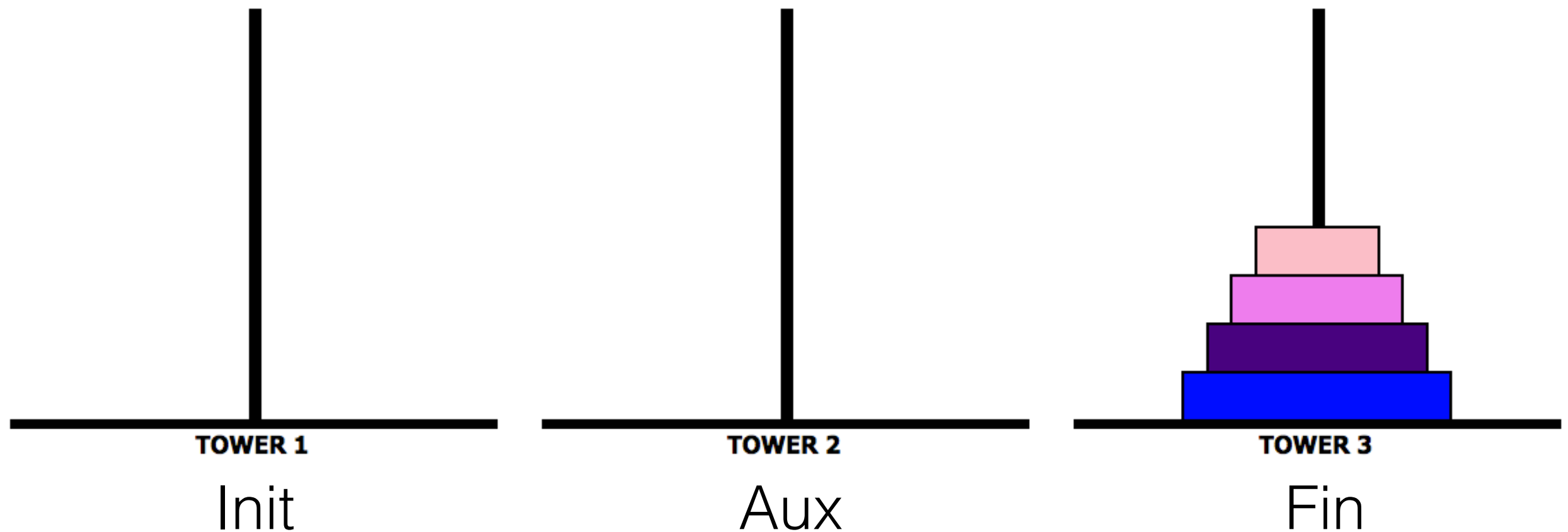


Move $n-1$ disks from Init to Aux.
Then move the n th disk to Fin.
Then move the $n-1$ disks from Aux to Fin.

Tower of Hanoi: Recursive Solution



THE UNIVERSITY OF
MELBOURNE

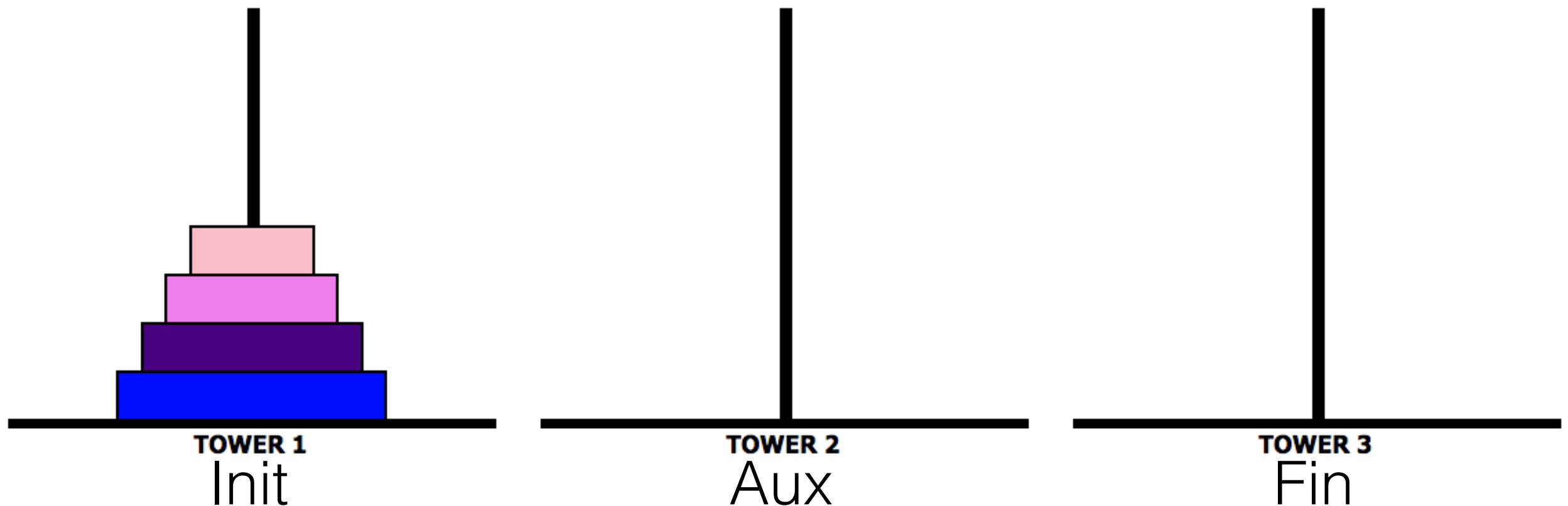


Move $n-1$ disks from Init to Aux.
Then move the n th disk to Fin.
Then move the $n-1$ disks from Aux to Fin.

Tower Of Hanoi: Recursive Algorithm



```
function HANOI( $n$ , init, aux, fin)  
  if  $n > 0$  then  
    HANOI( $n - 1$ , init, fin, aux)  
    Move one disk from init to fin  
    HANOI( $n - 1$ , aux, init, fin)
```

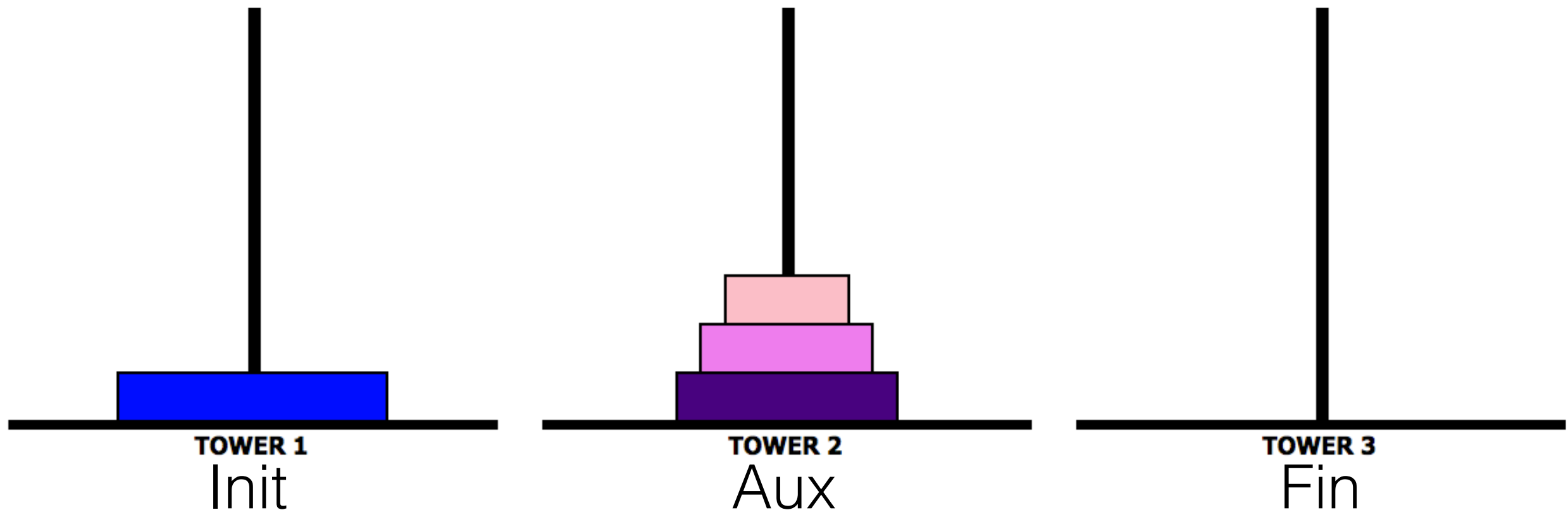


Tower Of Hanoi: Recursive Algorithm



THE UNIVERSITY OF
MELBOURNE

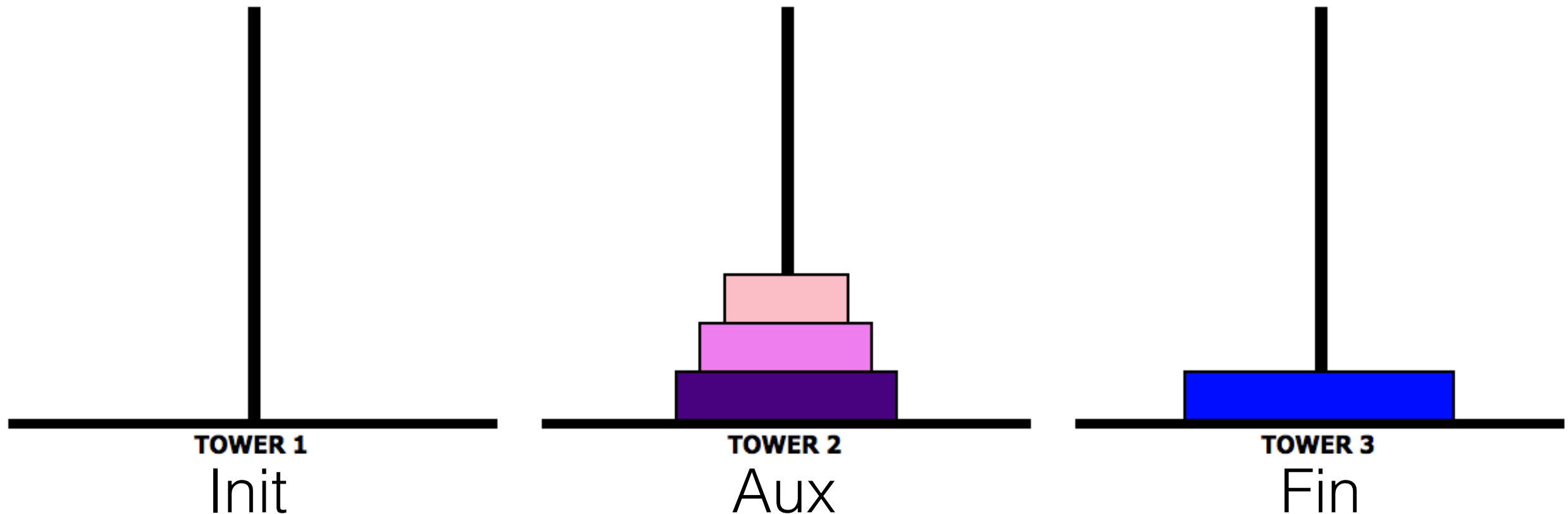
```
function HANOI( $n$ , init, aux, fin)  
  if  $n > 0$  then  
    HANOI( $n - 1$ , init, fin, aux)  
    Move one disk from init to fin  
    HANOI( $n - 1$ , aux, init, fin)
```



Tower Of Hanoi: Recursive Algorithm



```
function HANOI( $n$ , init, aux, fin)  
  if  $n > 0$  then  
    HANOI( $n - 1$ , init, fin, aux)  
    Move one disk from init to fin  
    HANOI( $n - 1$ , aux, init, fin)
```

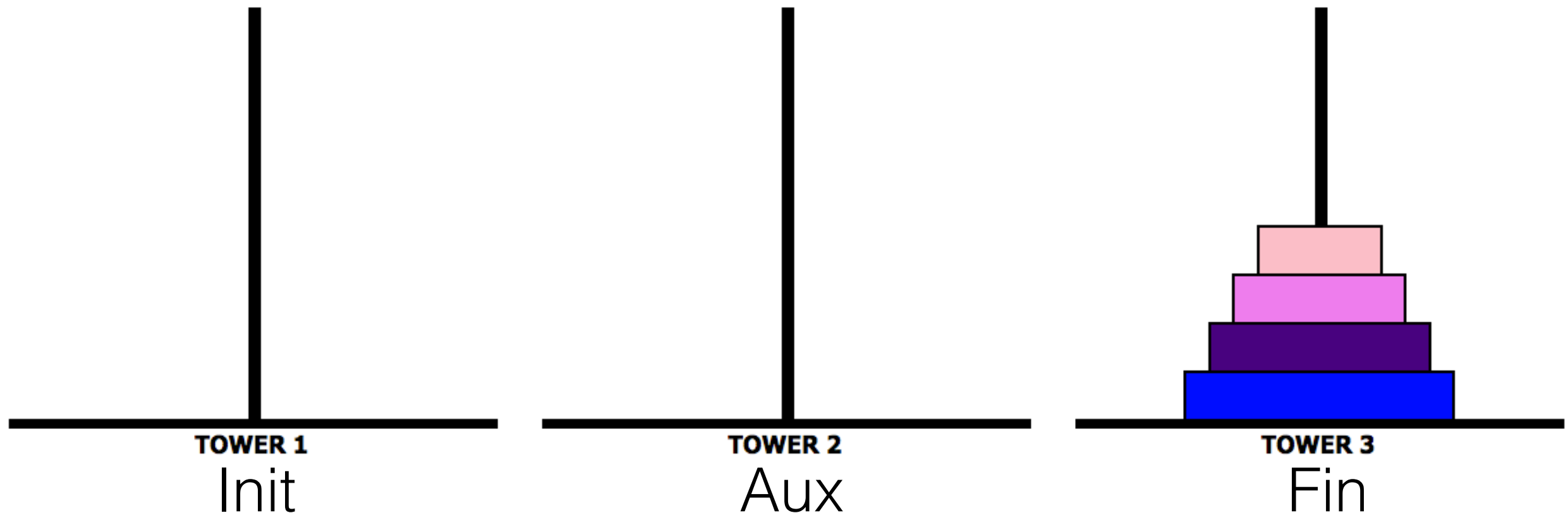


Tower Of Hanoi: Recursive Algorithm



THE UNIVERSITY OF
MELBOURNE

```
function HANOI( $n$ , init, aux, fin)  
  if  $n > 0$  then  
    HANOI( $n - 1$ , init, fin, aux)  
    Move one disk from init to fin  
    HANOI( $n - 1$ , aux, init, fin)
```



Tracing Tower of Hanoi Recursive Algorithm



THE UNIVERSITY OF
MELBOURNE

<http://vornlocher.de/tower.html>

A Challenge: Coin Change Problem

- There are 6 different kinds of Australian coin
- In cents, their values are: 5, 10, 20, 50, 100, 200
- In how many different ways can I produce a handful of coins adding up to \$4?
- This is not an easy problem!
- Key to solving it is to find a way to break it down into simpler sub-problems

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE



Coin Change Problem: Decomposition

\$4



THE UNIVERSITY OF
MELBOURNE



Coin Change Problem: Decomposition

\$4

made from



THE UNIVERSITY OF
MELBOURNE

Coin Change Problem: Decomposition

\$4

made from



THE UNIVERSITY OF
MELBOURNE

Coin Change Problem: Decomposition

\$4



THE UNIVERSITY OF
MELBOURNE

made from



Does the bag contain at least one \$2 coin?

Coin Change Problem: Decomposition

\$4



THE UNIVERSITY OF
MELBOURNE

made from



Does the bag contain at least one \$2 coin?



Coin Change Problem: Decomposition

\$4

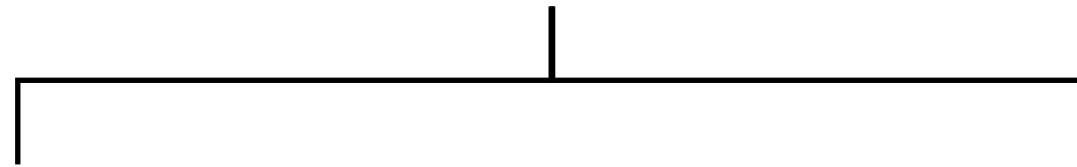


THE UNIVERSITY OF
MELBOURNE

made from



Does the bag contain at least one \$2 coin?



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



made from

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



made from



Coin Change Problem: Decomposition



\$4

made from



Does the bag contain at least one \$2 coin?



Yes

No



+ \$2



made from



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



made from



No



Coin Change Problem: Decomposition



\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



made from



No

\$4



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



made from



No

\$4



made from

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

\$4

made from



Does the bag contain at least one \$2 coin?



Yes



+ \$2



made from



No

\$4



made from



Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

+

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

The number of ways of making \$4 is therefore:

+

Coin Change Problem: Decomposition

The number of ways of making \$4 is therefore:

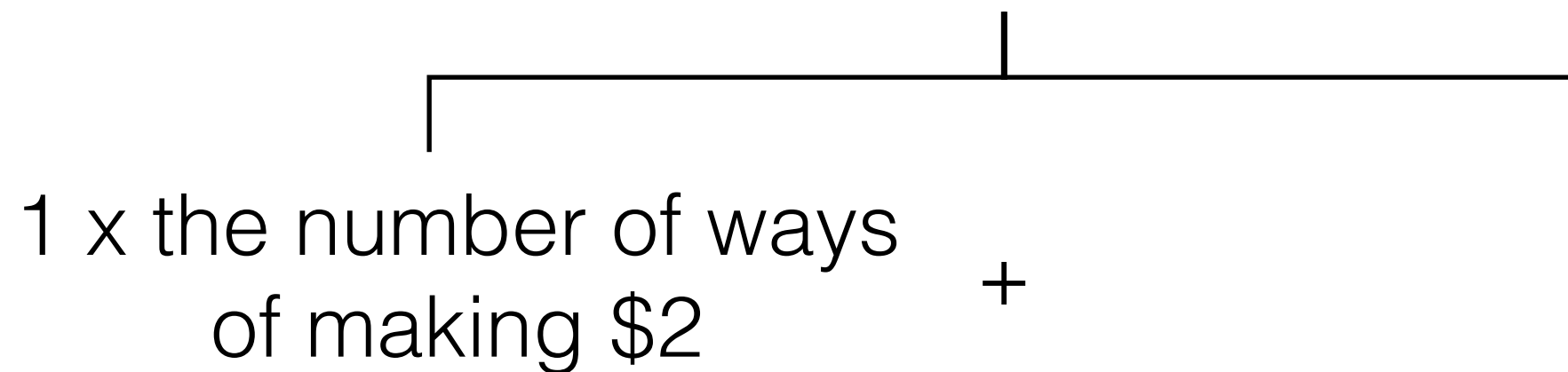
1 x the number of ways
of making \$2 +

Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

The number of ways of making \$4 is therefore:

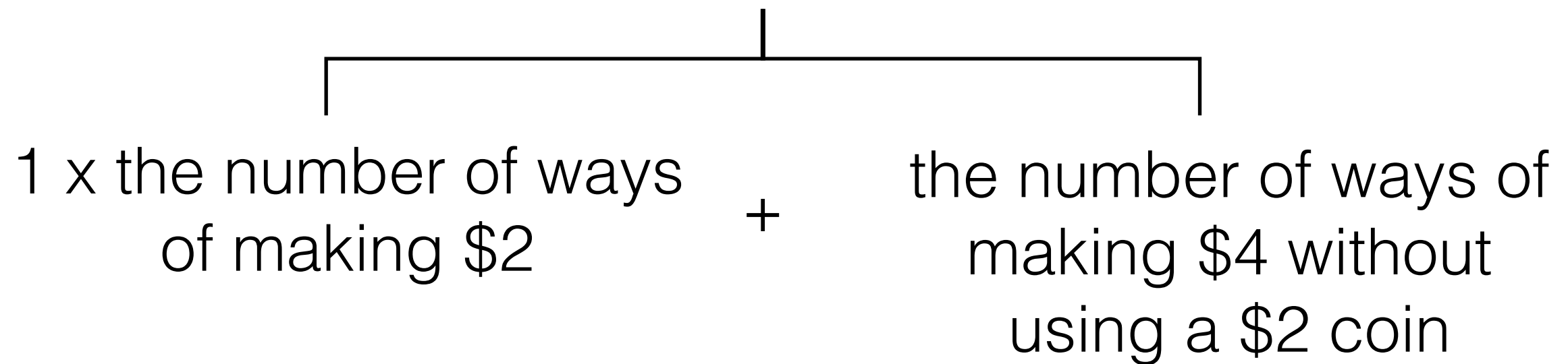


Coin Change Problem: Decomposition



THE UNIVERSITY OF
MELBOURNE

The number of ways of making \$4 is therefore:



Coin Change Problem: Partial Algorithm



THE UNIVERSITY OF
MELBOURNE

```
function WAYS(amount, denominations)
    // ... base cases ....
    d ← selectLargest(denominations)
    return WAYS(amount – d, denominations) +
           WAYS(amount, denominations \ {d})
```

For example:

$$\begin{aligned} \text{Ways}(400, \{5, 10, 20, 50, 100, 200\}) = \\ \text{Ways}(200, \{5, 10, 20, 50, 100, 200\}) + \\ \text{Ways}(400, \{5, 10, 20, 50, 100\}) \end{aligned}$$

Coin Change Problem: Base Cases



THE UNIVERSITY OF
MELBOURNE

Coin Change Problem: Base Cases



THE UNIVERSITY OF
MELBOURNE

- Each time we recurse, we decrease either:

Coin Change Problem: Base Cases



- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or

Coin Change Problem: Base Cases



- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)

Coin Change Problem: Base Cases



- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:
 - amount = 0:

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:
 - amount = 0: there is **one** way (using no coins)

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:
 - amount = 0: there is **one** way (using no coins)
 - amount < 0:

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:
 - amount = 0: there is **one** way (using no coins)
 - amount < 0: there are **no ways** to make this

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - denominations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:
 - amount = 0: there is **one** way (using no coins)
 - amount < 0: there are **no ways** to make this
 - denominations = \emptyset (and amount > 0):

Coin Change Problem: Base Cases

- Each time we recurse, we decrease either:
 - amount (by subtracting some quantity from it), or
 - demonisations (by removing an item from the set)
- Consider each of these separately.
 - amount base cases:
 - amount = 0: there is **one** way (using no coins)
 - amount < 0: there are **no ways** to make this
 - **denominations** = \emptyset (and amount > 0):
there are **no ways** to make this amount

Coin Change Problem: Full Recursive Algorithm



THE UNIVERSITY OF
MELBOURNE

```
function WAYS(amount, denominations)
  if amount = 0 then
    return 1
  if amount < 0 then
    return 0
  if denominations =  $\emptyset$  then
    return 0
  d  $\leftarrow$  selectLargest(denominations)
  return WAYS(amount – d, denominations) +
    WAYS(amount, denominations \ {d})
```

Initial call: WAYS(amount, {5, 10, 20, 50, 100, 200}).

Recursive Solution and its Complexity

- Although our recursive algorithm is short and elegant, it is not the most efficient way of solving the problem.
- Its running time grows **exponentially** as you grow the input amount.
- More efficient solutions can be developed using **memoing** or **dynamic programming**—more about that later (around Week 10).

Next Time...

- Graphs, trees, graph traversal and allied algorithms.