Institute of Pervasive Computing

RECURSION

Markus Jäger (Computer Science)
Florian Beck (Artificial Intelligence)
Bernhard Anzengruber (Artificial Intelligence)

Institute of Pervasive Computing
Johannes Kepler University Linz

markus.jaeger@jku.at
florian.beck@jku.at
bernhard.anzengruber-tanase@jku.at

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Algorithms and Data Structures 1
Exercise – 2023S

Institute of Pervasive Computing

RECURSION :: MOTIVATION AND DEFINITION

Many **complex real-world** problems can be solved very elegantly by compact recursive algorithms

- mathematical problems formulated recursively (e.g., Fibonacci numbers, Factorial, etc.)
- sorting algorithms
- traversal of binary tree

Definition

- method that calls itself (directly or indirectly)
- termination condition (anchor of the recursive function, used for escaping)
- recursive function calls (altered parameter)

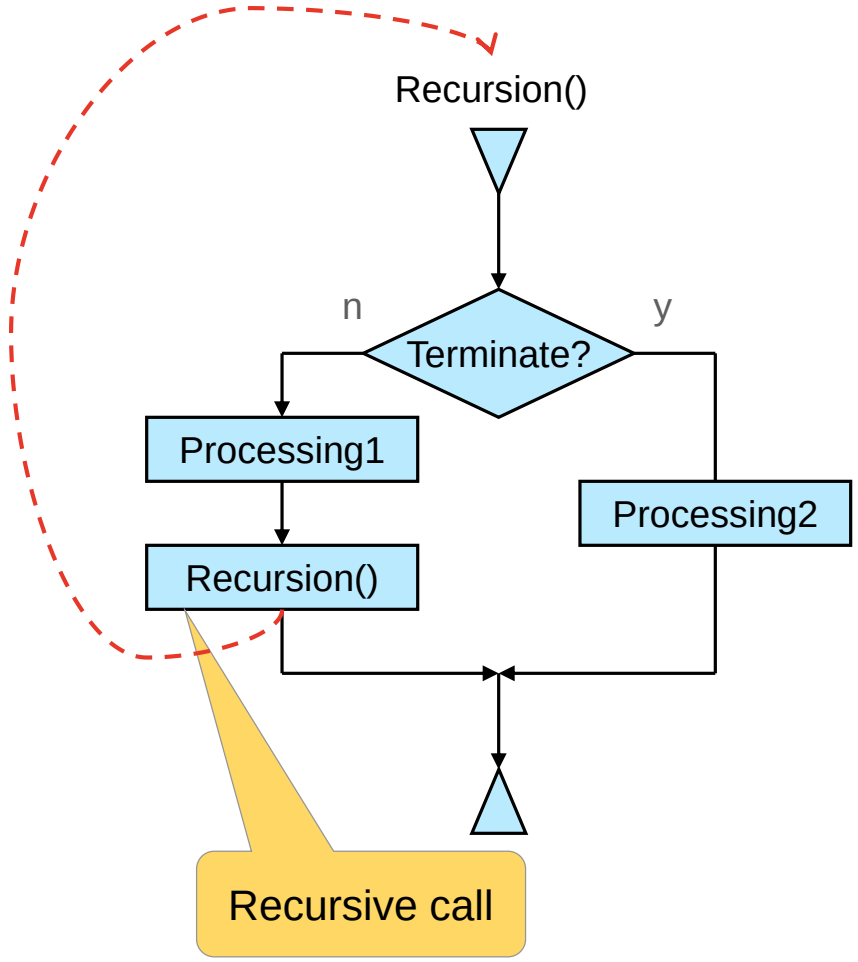
Indirect recursion

- two or more functions call each other repeatedly
- at least one of these functions is declared as recursive function (termination condition/anchor)

Recursive functions

- usually less efficient than iterative methods
- easier to read and understand than iterative methods

RECURSION :: BASIC STRUCTURE – DIRECT

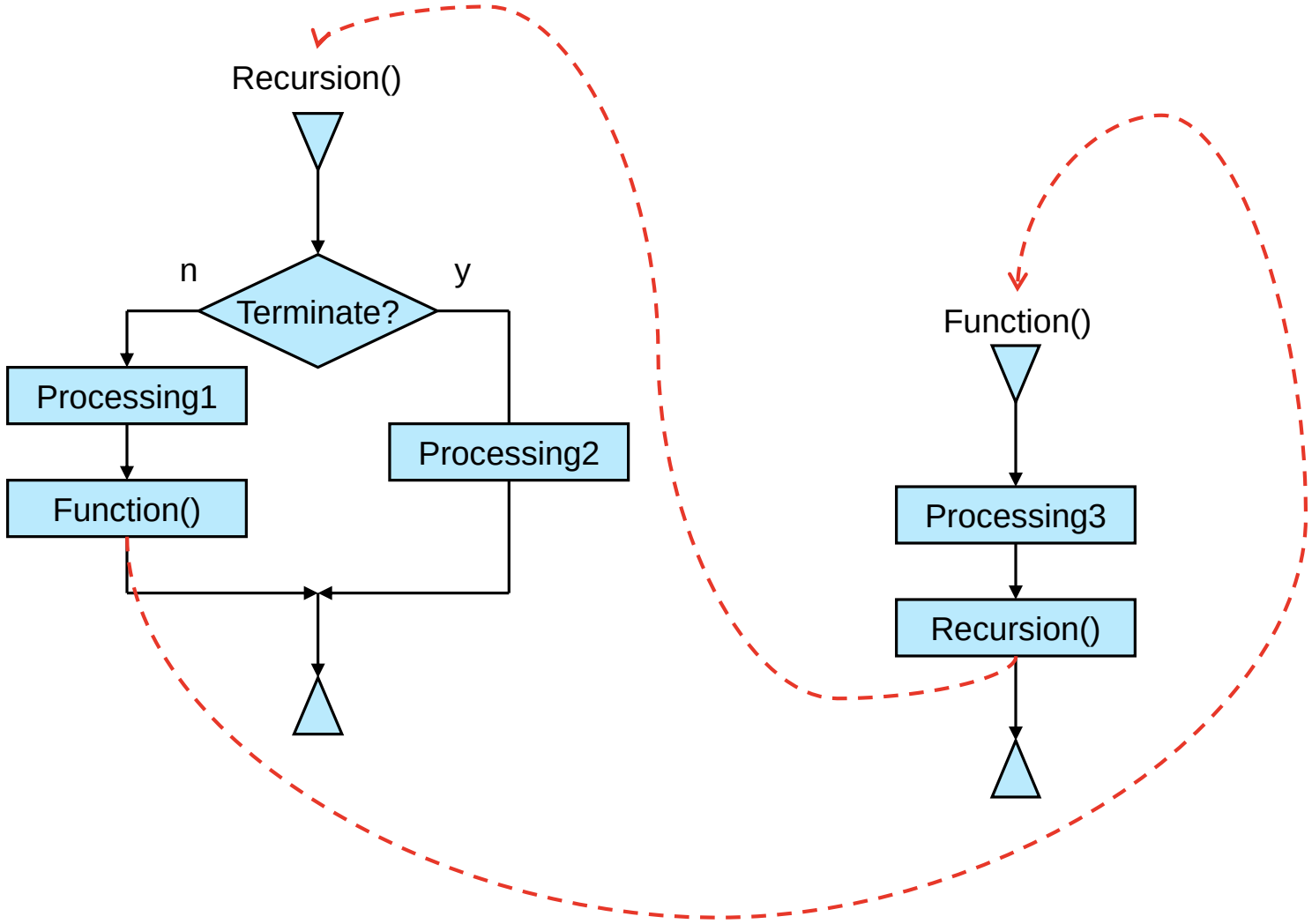



```
Recursion():

    if (TerminationCondition):
        // Recursion anchor
        Processing2

    else:
        Processing1
        Recursion()
```

RECURSION :: BASIC STRUCTURE (2) – INDIRECT



EXAMPLE :: ITERATION - RECURSION

Example: Sum of numbers from 0 (zero) to max. (including)

```
class RecursionTest:

    def sumIterative(max):
        k = 0
        for i in range(max, 0, -1)
            k += i
        return k


    def main():
        sumIterative(5)
        print(sumIterative(5))
```

```
class RecursionTest:

    def sumRecursive(i):
        if (i <= 0):
            return 0
        return i + sumRecursive(i - 1)


    def main():
        sumRecursive(5)
        print(sumRecursive(5))
```

note the altered parameter

run time check:

9000: Recursive algorithm (val=9000) took longer by (ms): 2

9999: Exception in thread "main" StackOverflowError
at RecursionTest.sumRecursive

EXAMPLE 1 :: SEQUENCE OF NUMBERS

The following sequence of numbers is given:

F = {5, 8, 11, 14, ...}

We are looking for different solutions to retrieve the value of the sequence by providing the index of the number to the function, i.e.:

```
f(<0) = -1, // error condition
f(0) = 5,
f(1) = 8,
f(2) = 11,
etc.
```

Find an algorithm in Python to solve this problem using

a) an iterative technique,

b) a recursion,

c) a direct calculation.

EXAMPLE 1 :: SOLUTION

a) Iterative

```
def f(i):

   y = 5

   if (i < 0):
      return -1

   while (i > 0):
      y += 3
      i -= 1

   return y
```

b) Recursive

```
def f(i):

   if (i < 0):
      return -1

   if (i == 0):
      return 5

   y = f(i - 1) + 3

   return y
```

Termination condition (recursion anchor)

Recursive call

c) Direct

```
def f(i):

   if (i < 0):
      return -1

   return 5 + 3 * i
```

EXAMPLE 2 :: MERGESORT

Input sequence S with length n

Divide: reduce problem

- sequence S has length 0 or 1 (already sorted)
- split S into two parts (S_1,S_2) each of ca. n/2 elements

Conquer: recursively sort S_1 and S_2

Combine: (merge)

- merge the sorted S_1 and S_2 to get a full sorted sequence

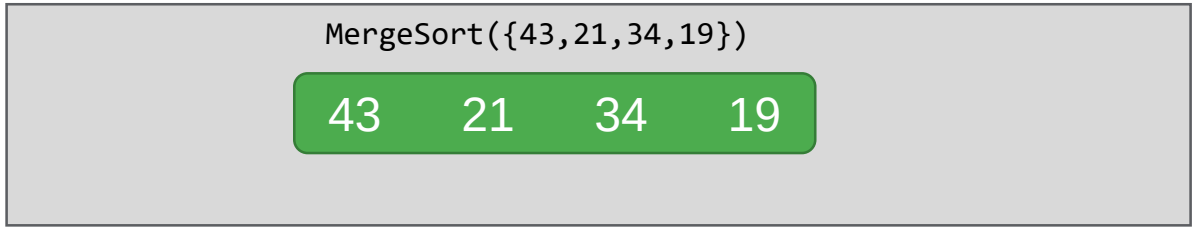
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S has zero or only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



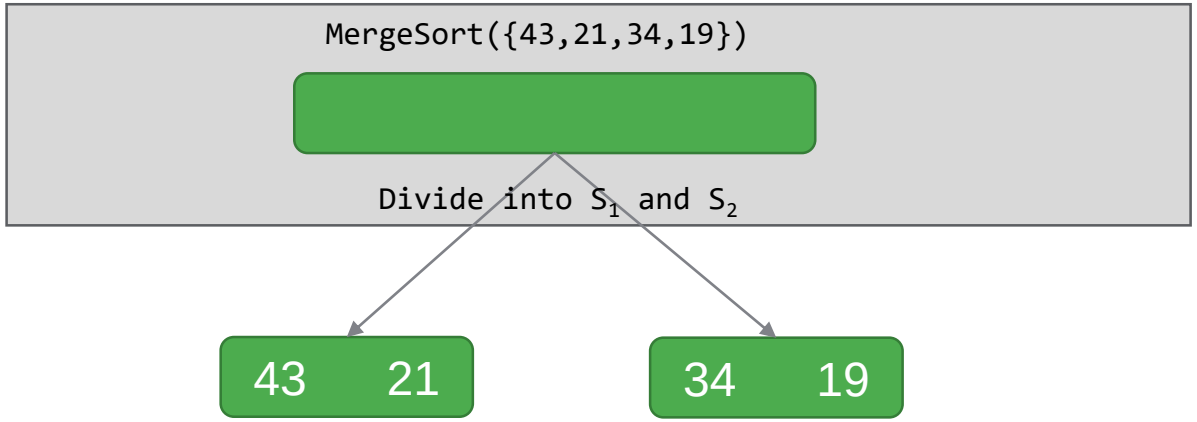
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



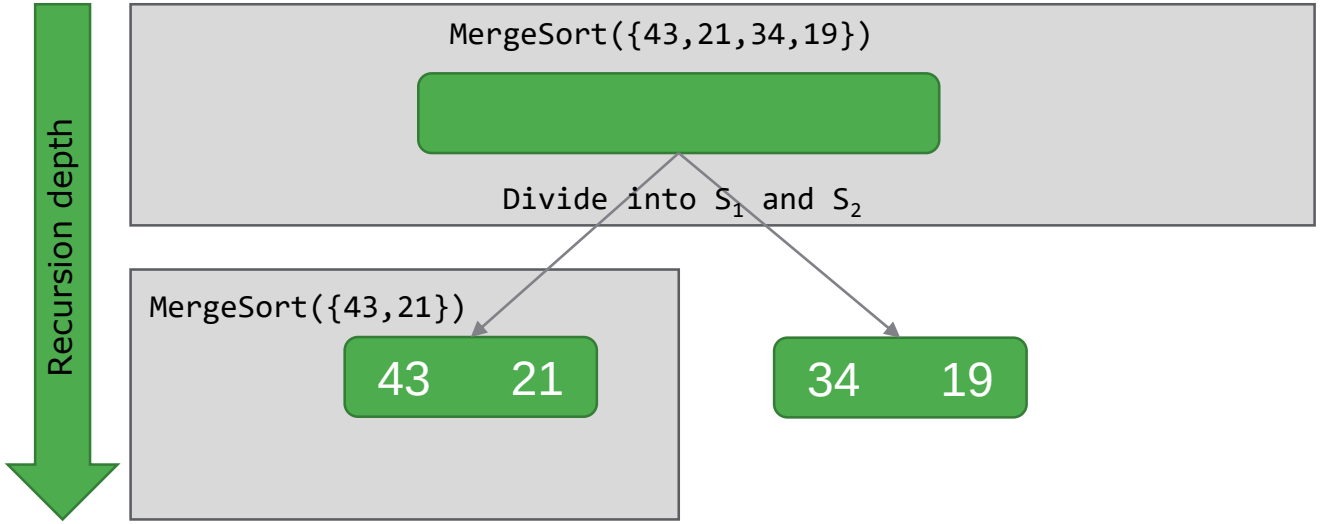
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



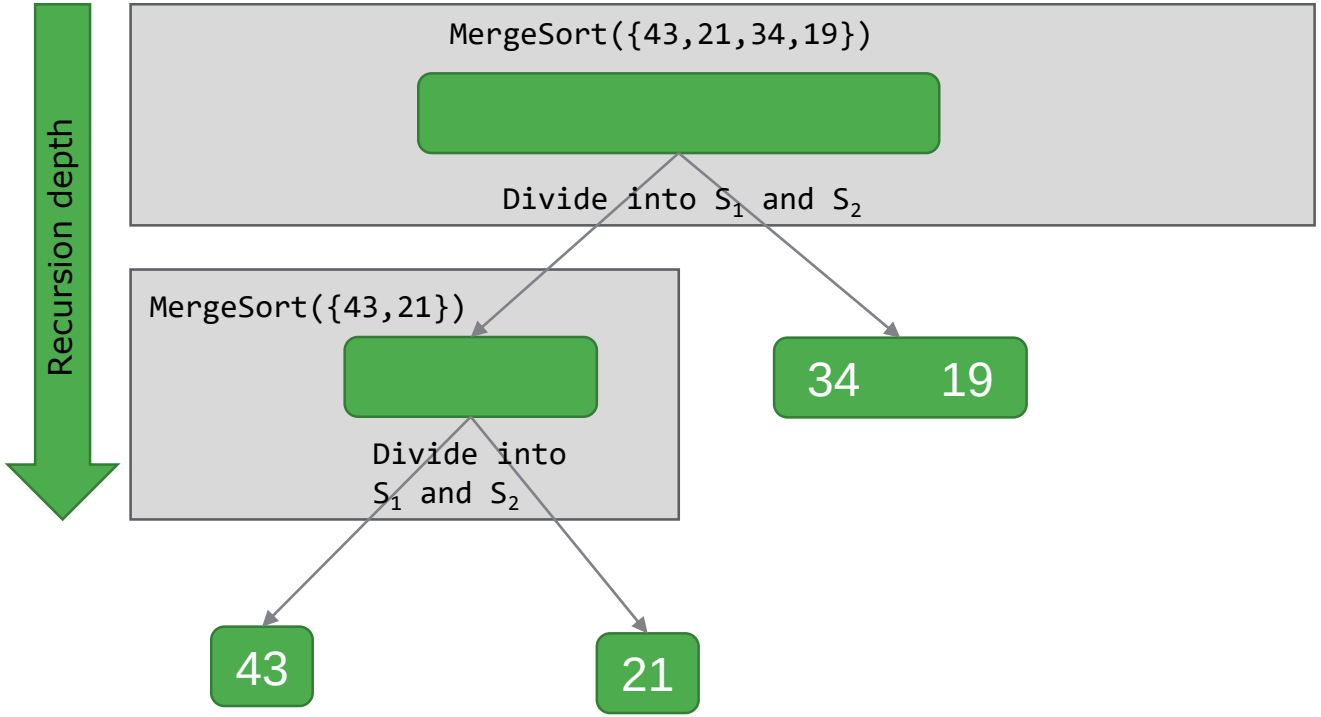
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



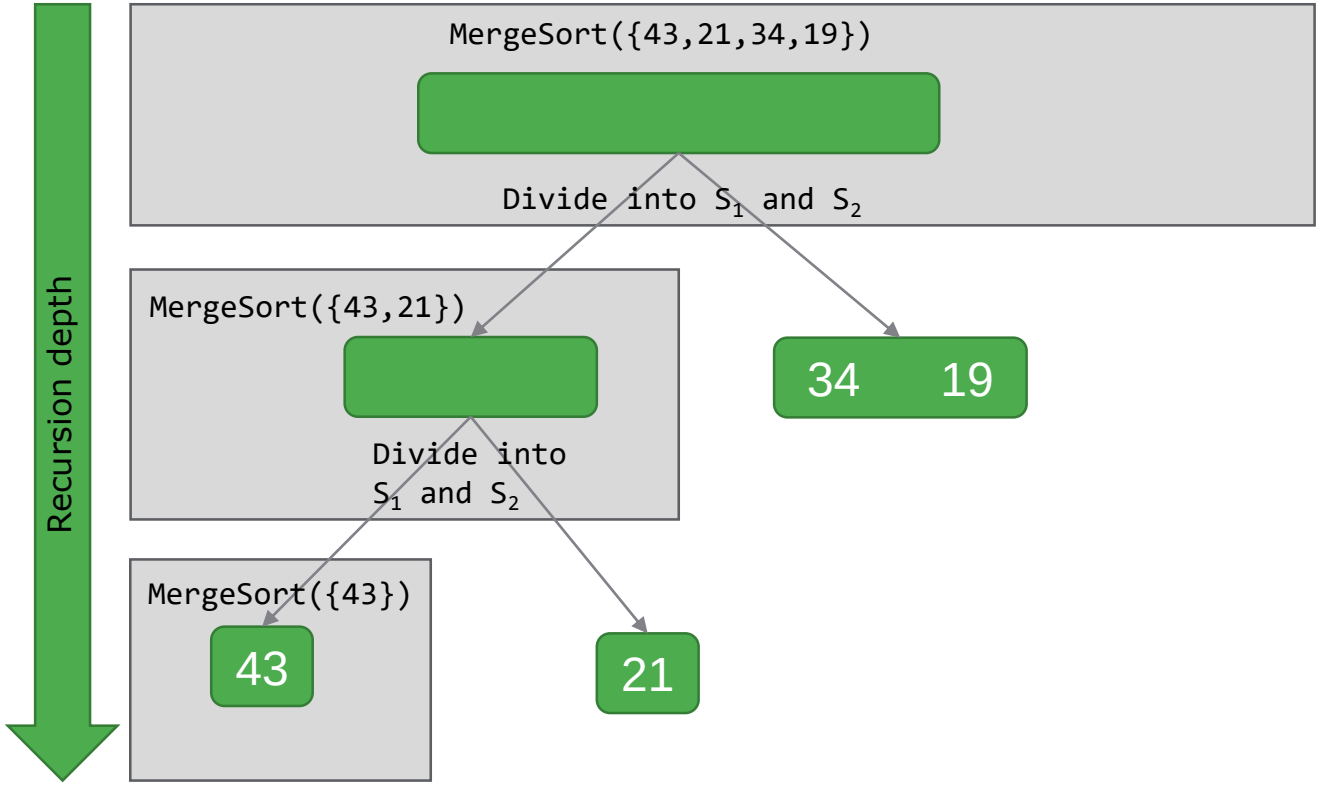
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



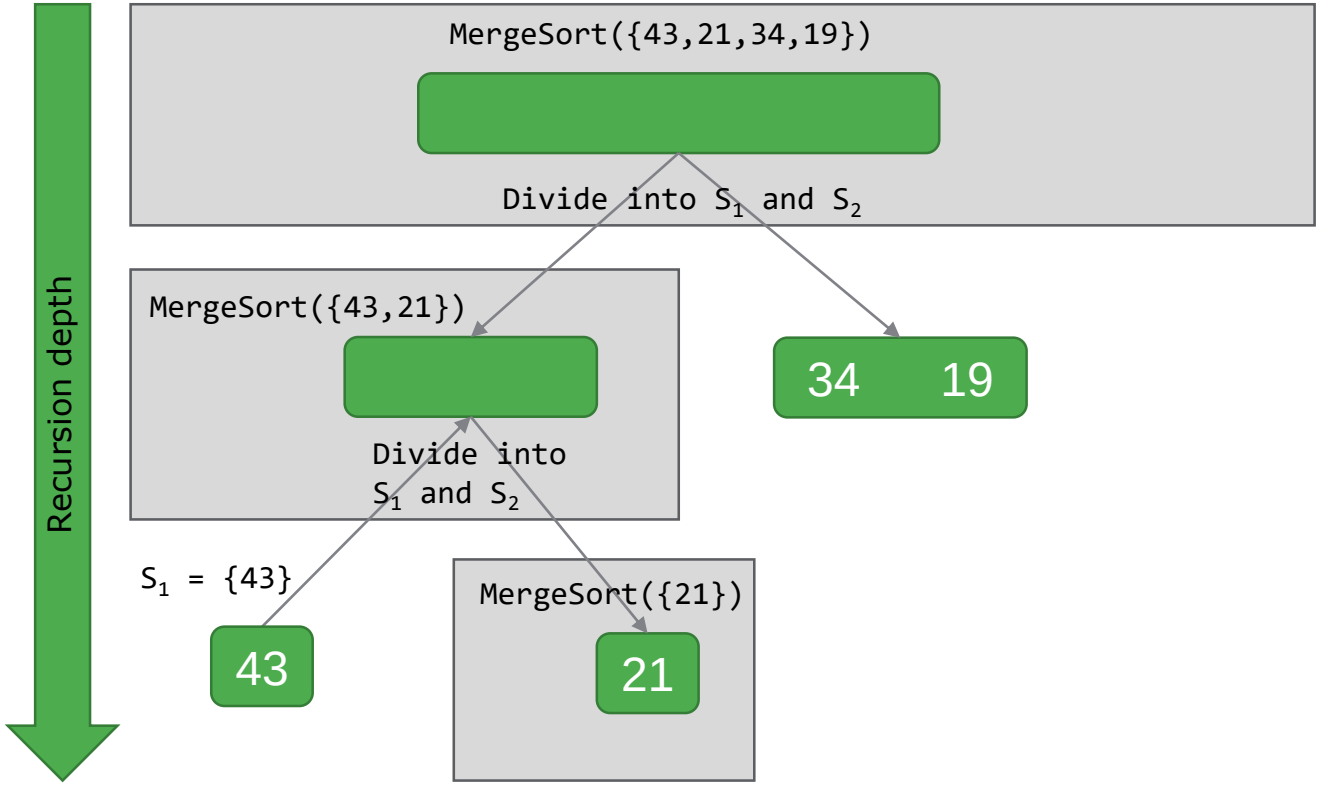
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



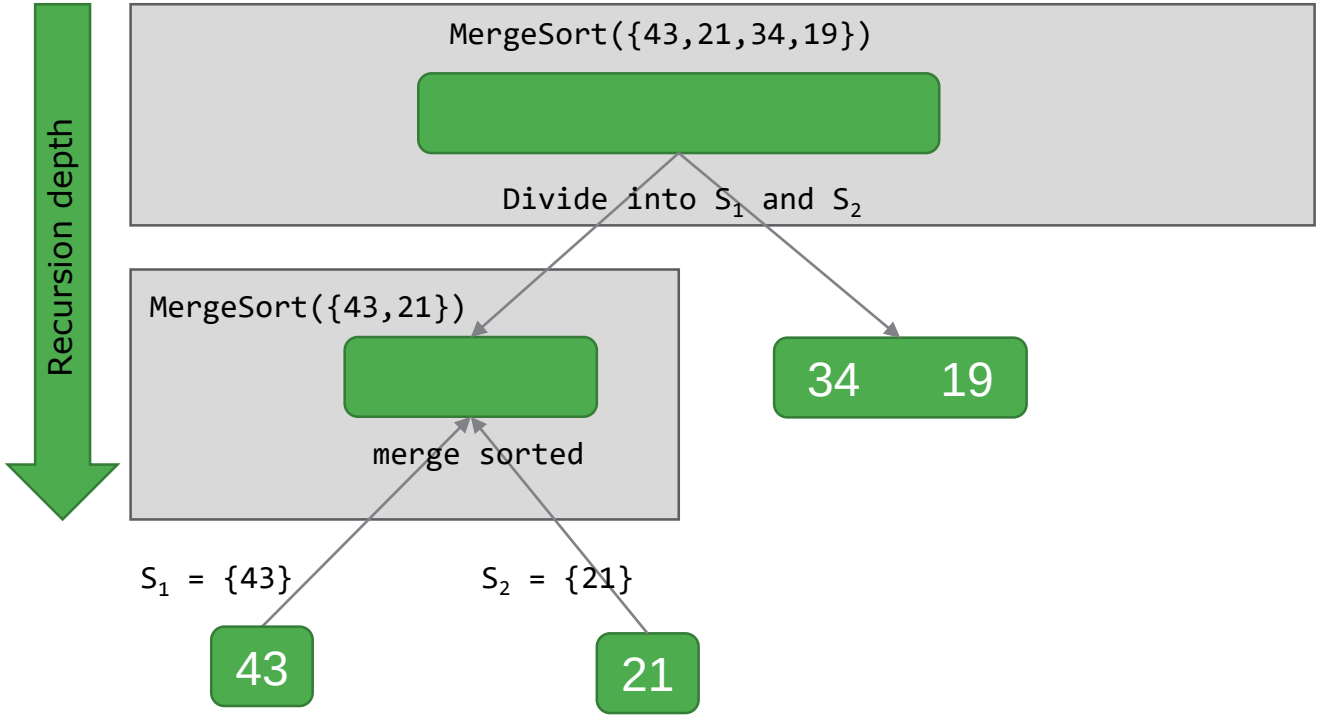
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



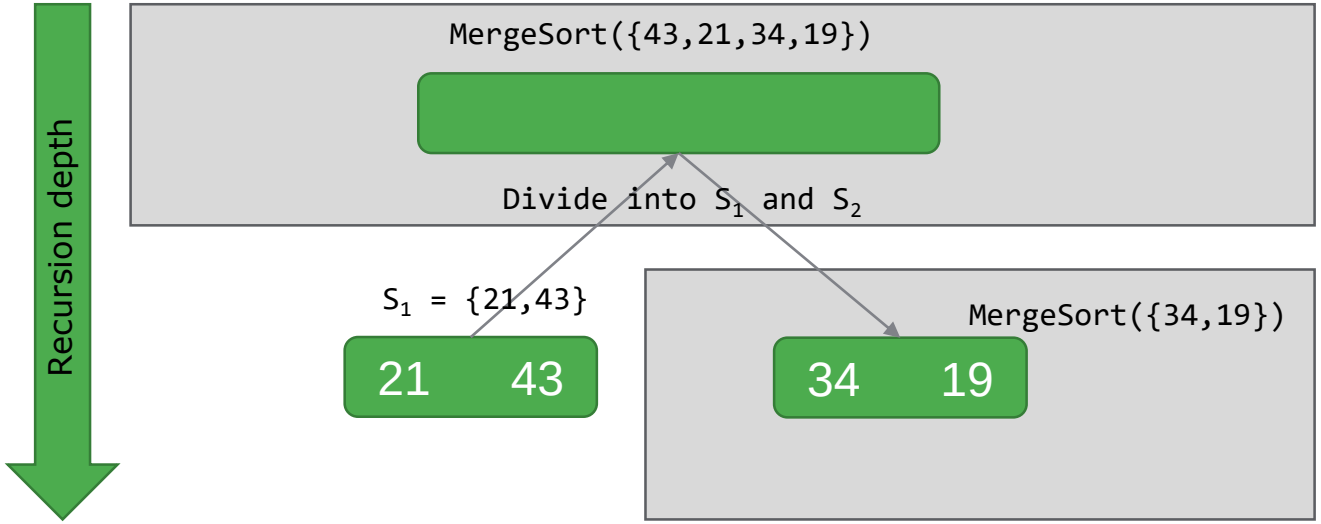
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



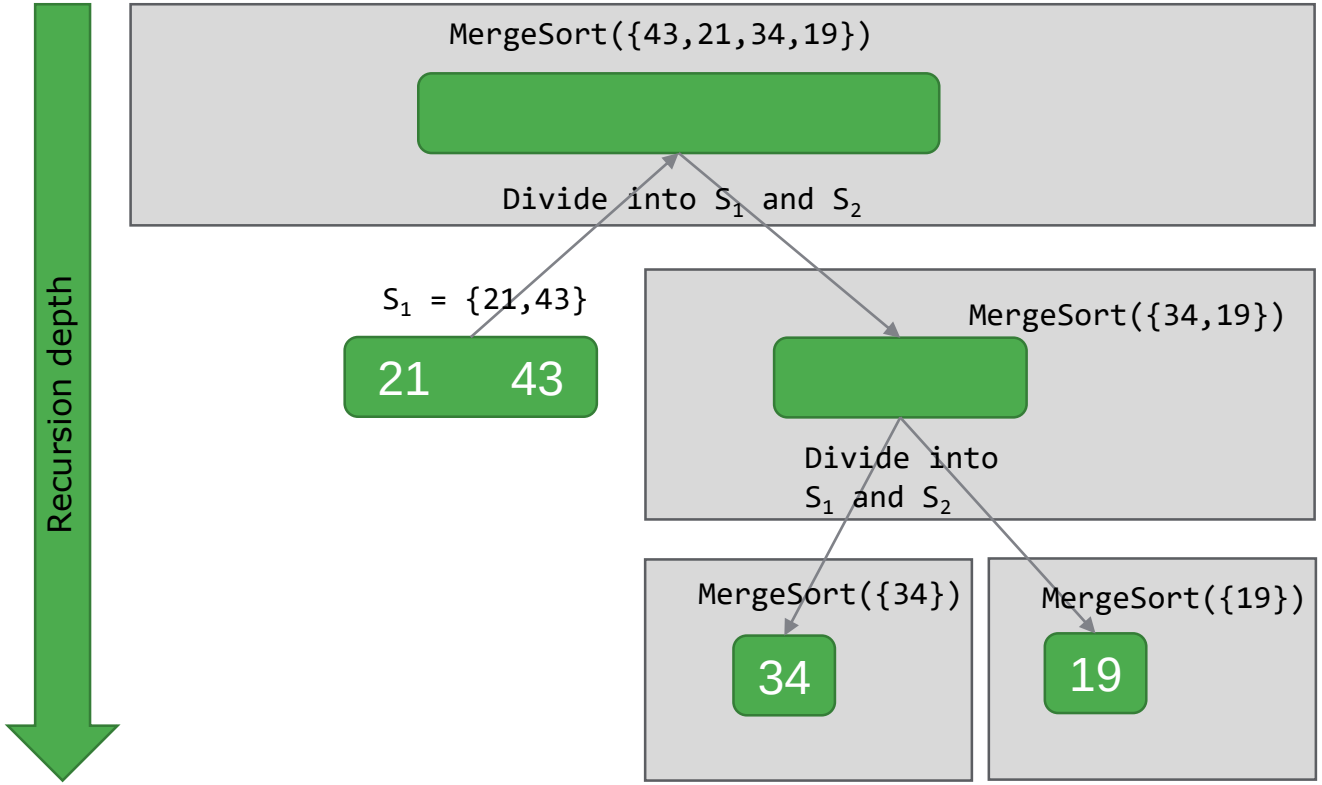
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



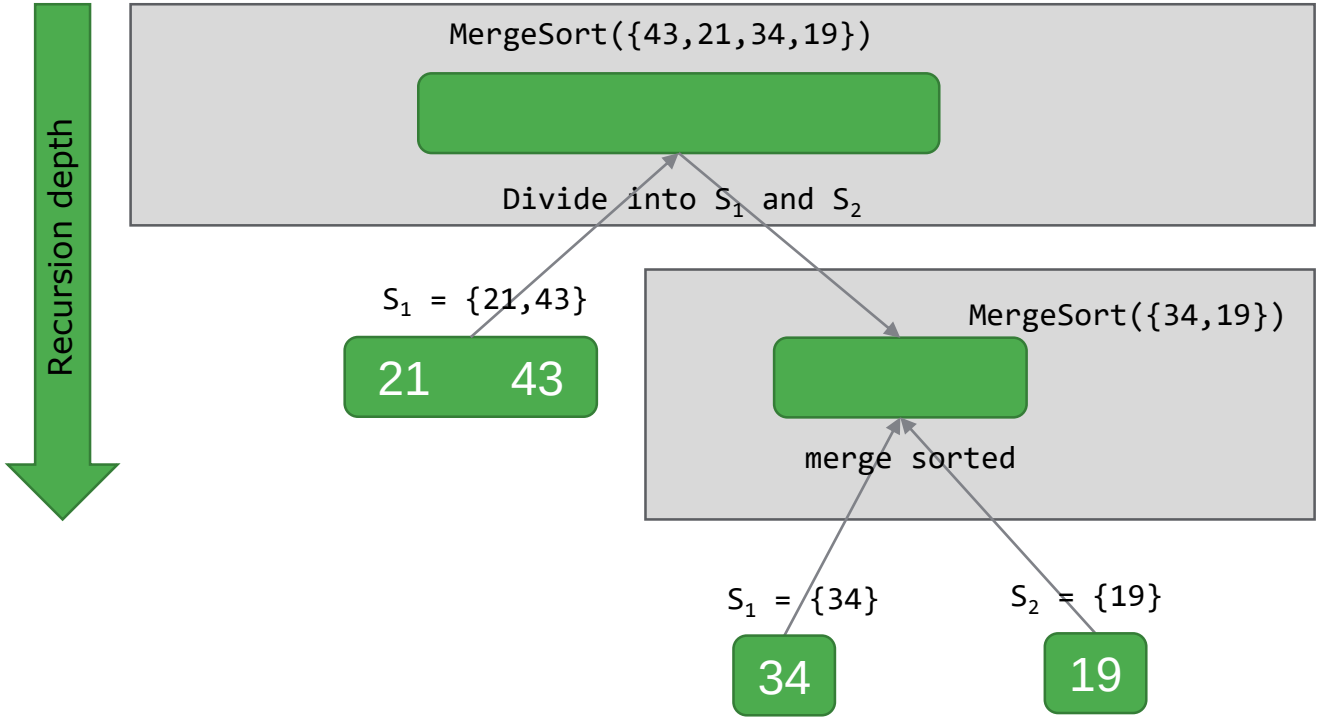
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



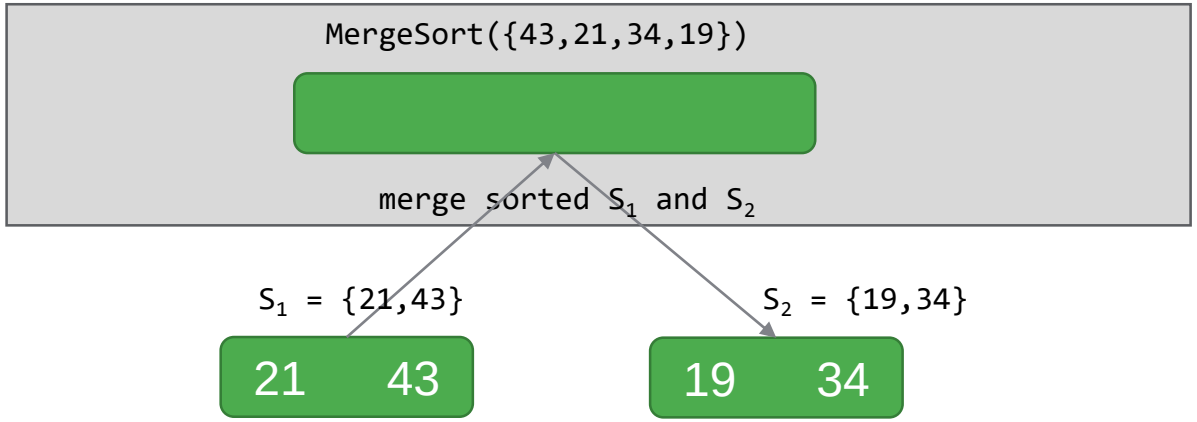
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S1 and S2; //DIVIDE
  S1 := MergeSort(S1);            //CONQUER
  S2 := MergeSort(S2);            //CONQUER
return Merge (S1, S2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



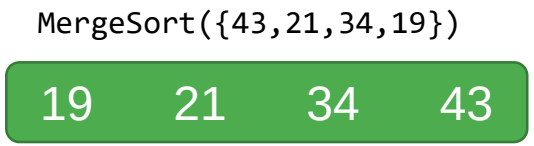
Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

EXAMPLE 2 :: MERGESORT ILLUSTRATION

Input sequence S={43,21,34,19}, n=4



Pseudocode

```
algorithm MergeSort(S) → S_S
 Input:  Sequence S
 Output: sorted sequence S_S

if S is only one element
   return S
else
  divide S in 2 halves S_1 and S_2; //DIVIDE
  S_1 := MergeSort(S_1);            //CONQUER
  S_2 := MergeSort(S_2);            //CONQUER
return Merge (S_1, S_2);            //COMBINE
```

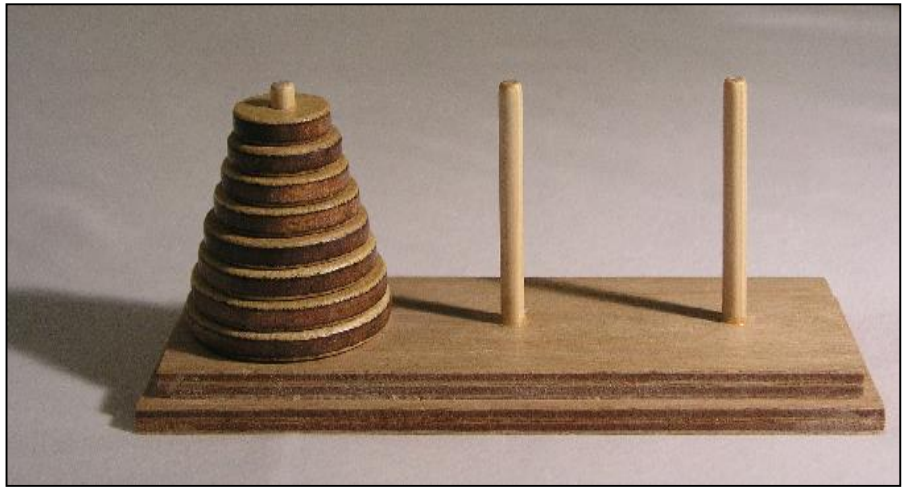
RECURSION :: TOWERS OF HANOI

Problem

- monks in Vietnam were asked to carry 64 gold disks from one tower (stack) to another
- each disk is of a different size
- there are 3 stacks, a source stack, a destination stack and an intermediate stack
- a disk is placed on one of three stacks but no disk can be placed on top of a smaller disk
- the source tower holds 64 disks

How will the monks solve this problem?

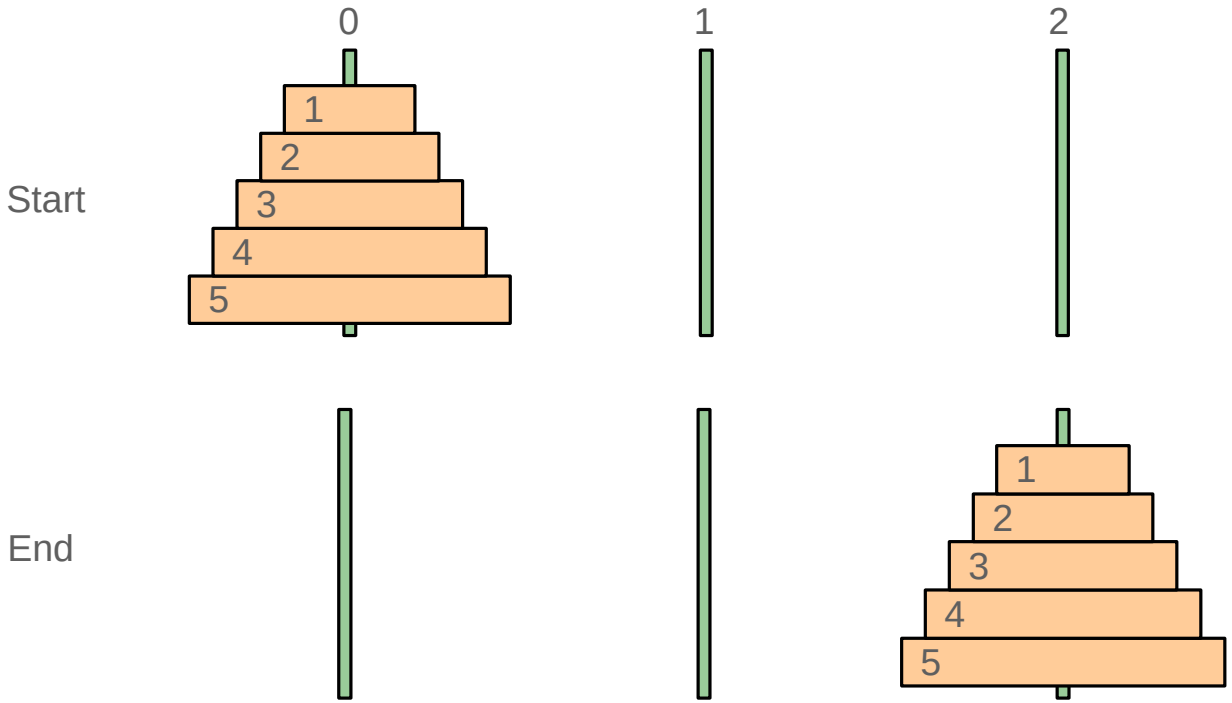
How long will it take them?



RECURSION :: TOWERS OF HANOI

Method

- to move any disk, first move the smaller disks off it
- if we had a method to move the top three disks to the middle position, we could put the biggest disk in its place
- assume we have this method and call it recursively



RECURSION :: TOWERS OF HANOI

Start with 1 disk (base case)

- move 1 disk from start tower to destination tower

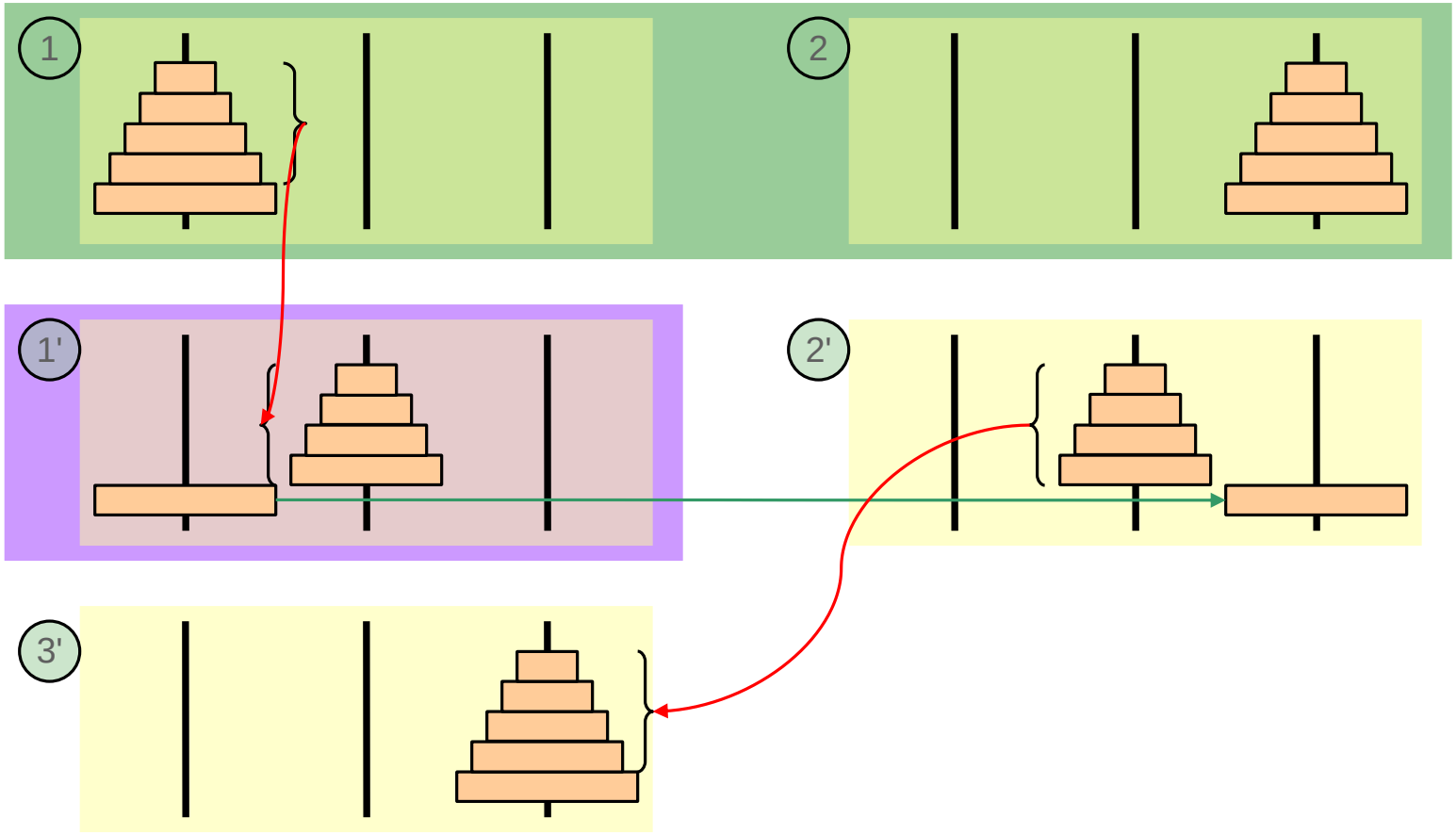
Move 2 disks

- move smaller disk from start tower to intermediate tower
- move larger disk from start tower to final tower
- move smaller disk from intermediate tower to final tower

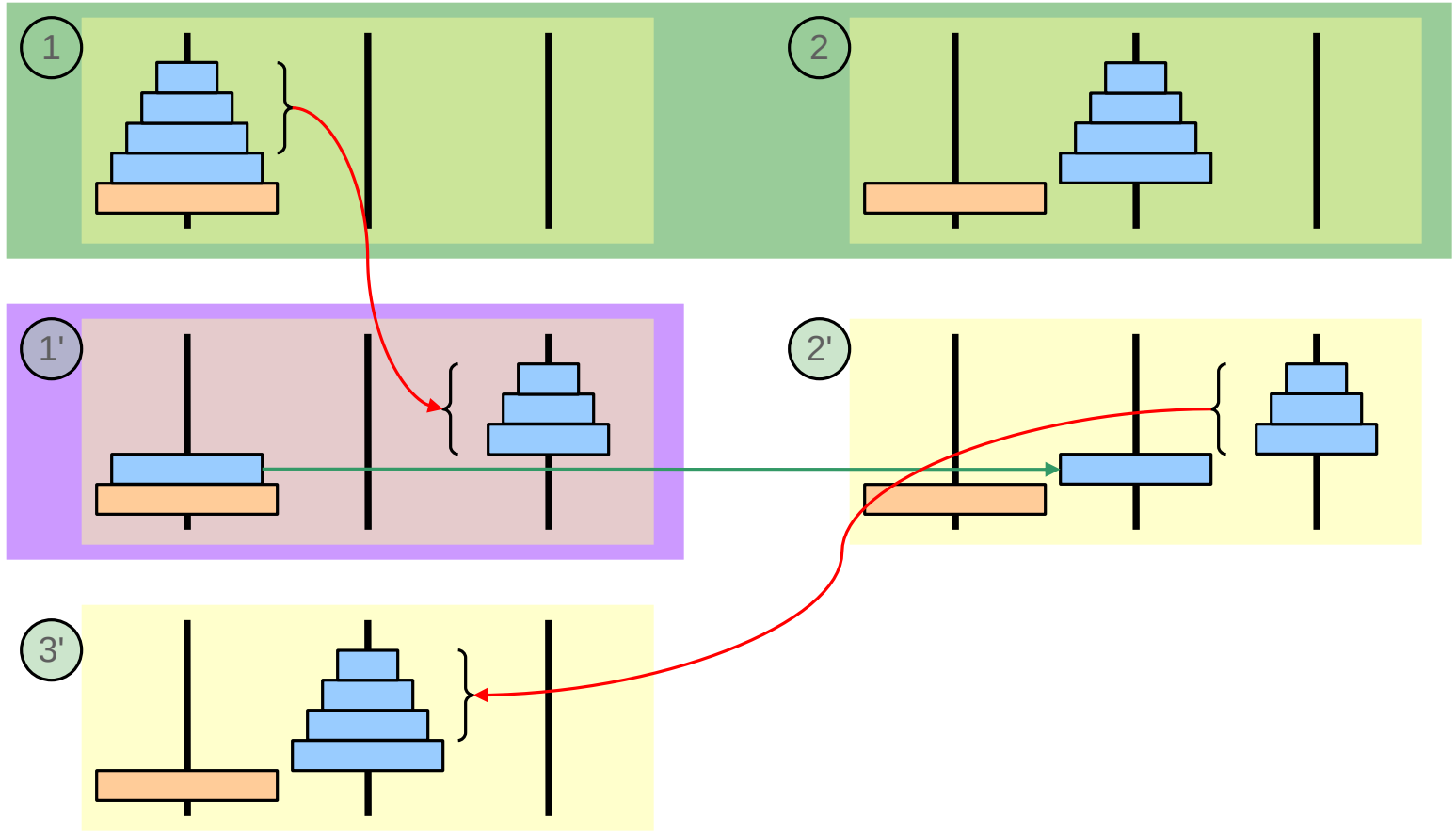
Move n disks

- solve the problem for n - 1 disks using the intermediate tower instead of the final tower
- move the biggest disk from start tower to final tower
- solve the problem for n - 1 disks using the intermediate tower instead of the start tower

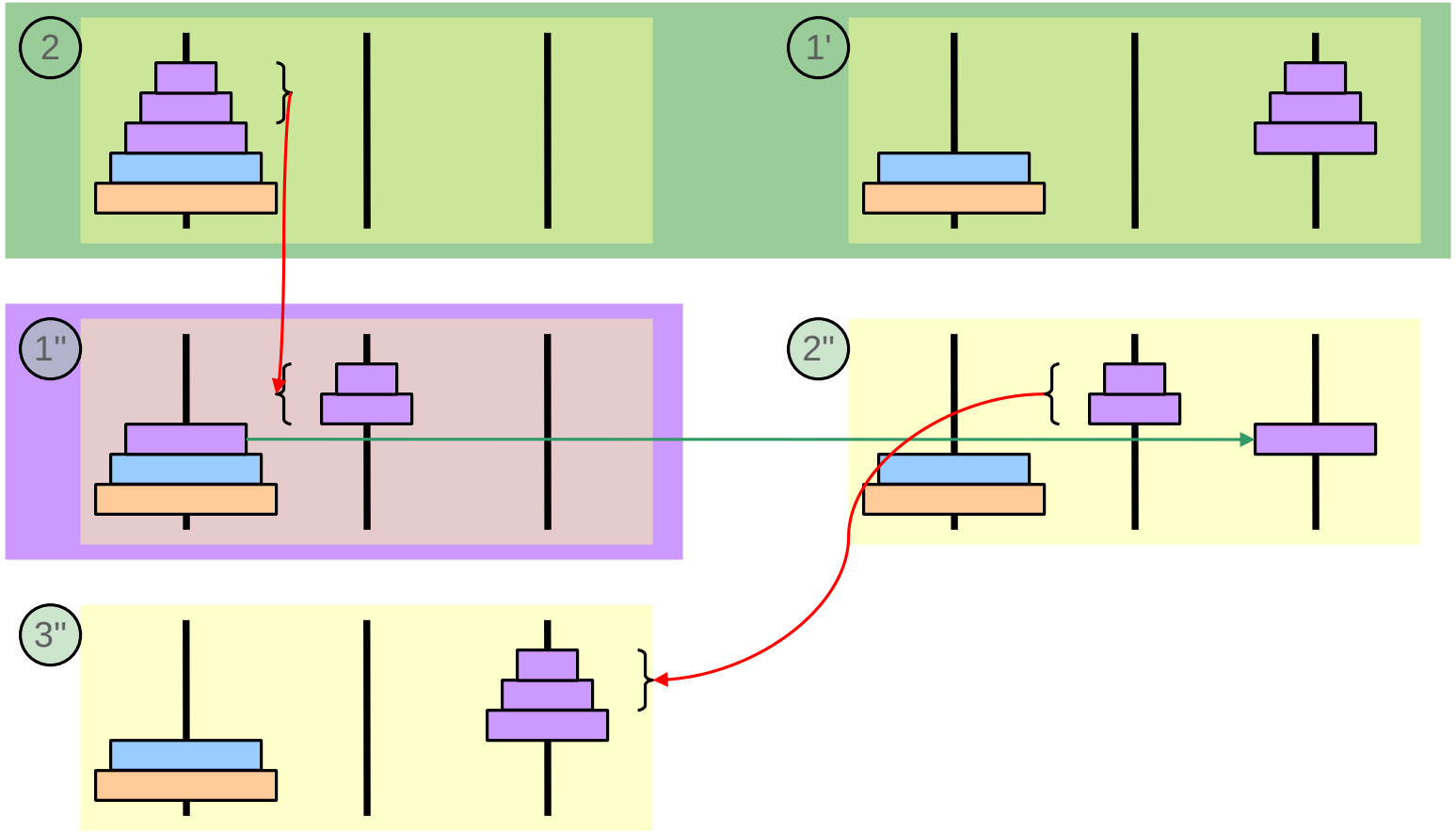
RECURSION :: TOWERS OF HANOI



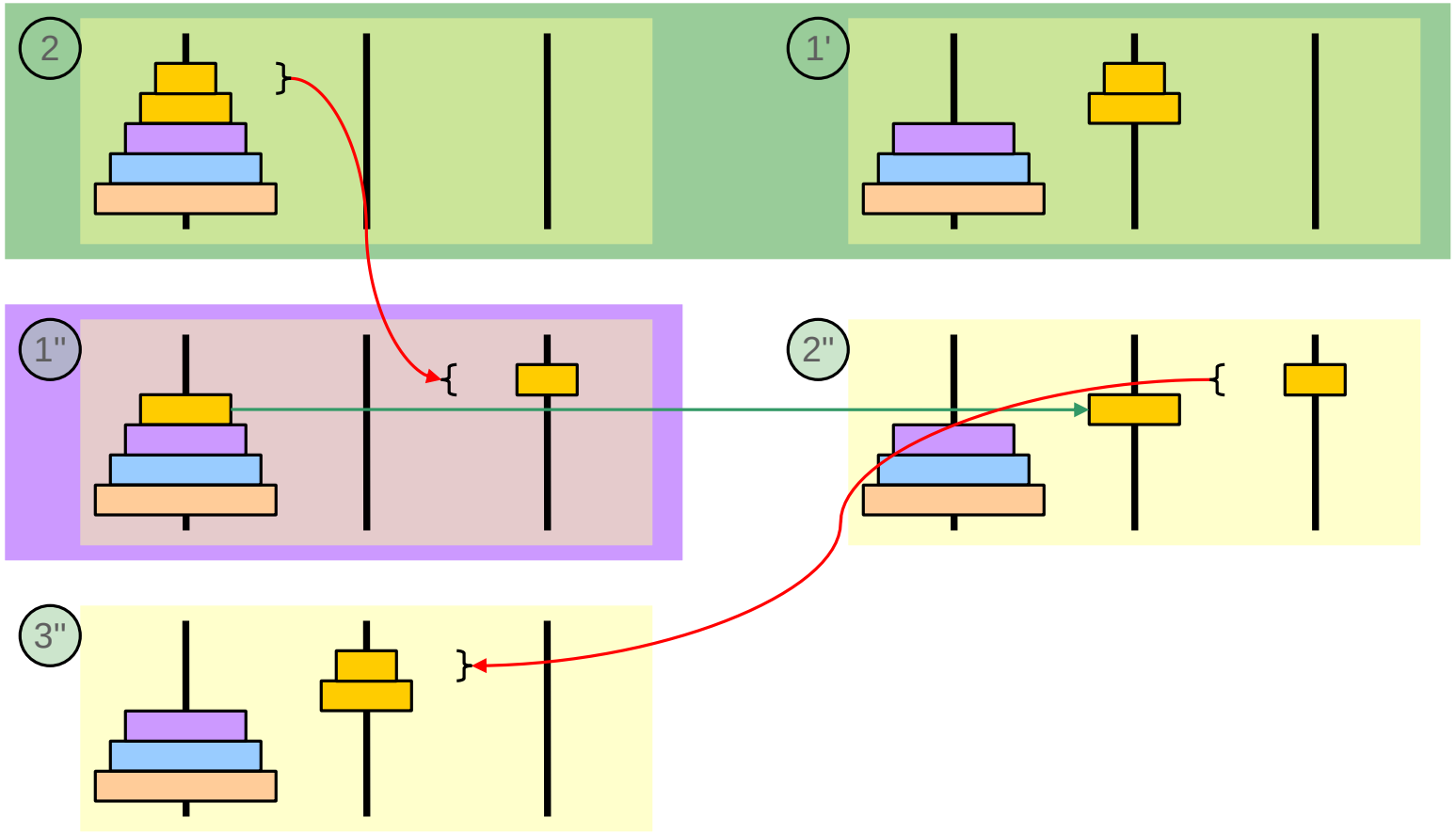
RECURSION :: TOWERS OF HANOI



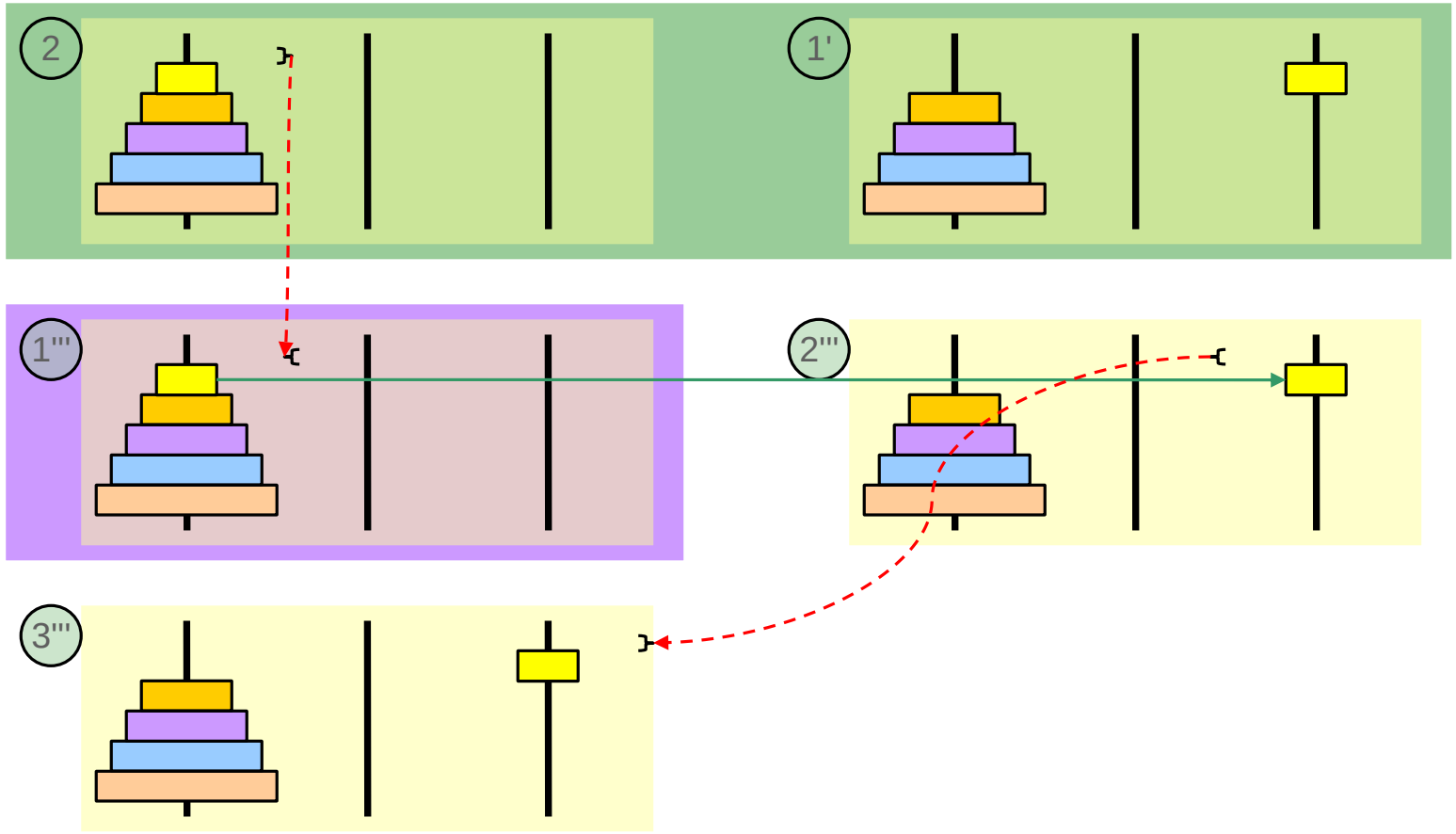
RECURSION :: TOWERS OF HANOI



RECURSION :: TOWERS OF HANOI



RECURSION :: TOWERS OF HANOI



RECURSION :: TOWERS OF HANOI

General solution

- **T (n, a, b)**
  - n … height of the tower
  - a … initial stack
  - b … destination stack
- **D (a, b)** **// Disk Operation**
  - a … initial stack
  - b … destination stack

- move a tower of height 5 from 0 to 2
  - T (5, 0, 2) = T (4, 0, 1) + D (0, 2) + T (4, 1, 2)
- move a tower of height 4 from 0 to 1
  - T (4, 0, 1) = T (3, 0, 2) + D (0, 1) + T (3, 2, 1)
- move a tower of height n…
  - T (n, a, b) = T (n-1, a, 3-(a+b)) + D (a, b) + T (n-1, 3-(a+b), b)

Disks	Moves
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

RECURSION :: TOWERS OF HANOI

Python code

```
class TowerOfHanoi:

   def D(a, b):
      print("Disc from " + a + " to " + b)


   def T(h, a, b):
      if (h > 0):
         T(h - 1, a, 3 - (a+b))
         D(a, b)
         T(h - 1, 3 - (a+b), b)


   def main():
      T(3, 0, 2)
```

15/01/2014

Execution times (with output of disc movement)

Tower of Hanoi (height=10) took (sec): 0.029
Tower of Hanoi (height=15) took (sec): 0.341
Tower of Hanoi (height=20) took (sec): 11
Tower of Hanoi (height=25) took (sec): 425
Tower of Hanoi (height=30) took (sec): 58722 (>16h)

Institute of Pervasive Computing

RECURSION

Markus Jäger (Computer Science)
Florian Beck (Artificial Intelligence)
Bernhard Anzengruber (Artificial Intelligence)

Institute of Pervasive Computing
Johannes Kepler University Linz

markus.jaeger@jku.at
florian.beck@jku.at
bernhard.anzengruber-tanase@jku.at

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Algorithms and Data Structures 1
Exercise – 2023S

Institute of Pervasive Computing