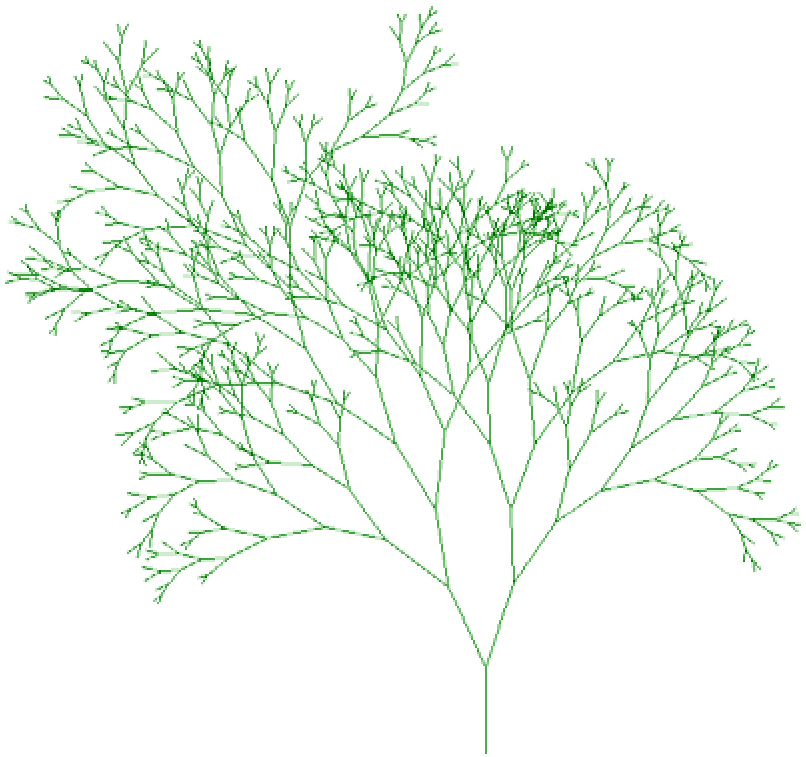
Data Structures in Python
Chapter 4

1. Recursion Concepts
2. Recursion Stack and Memoization
3. **Recursive Algorithms**
4. Recursive Graphics

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

Agenda

- Recursion and Stack
- More Examples and Algorithms
  - Radix Conversion
  - The Fibonacci Sequence
  - The Towers of Hanoi



Radix Conversion

- Radix is the base of number representation.
- Examples:
  - Decimal, 10
  - Binary, 2
  - Octal, 8
  - Hexadecimal, 16

Decimal	Binary	Octal	Hexadecimal
20	10100_2	24_8	14_{16}
7	111_2	7_8	7_{16}
32	100000_2	40_8	20_{16}

Radix Conversion

- Radix conversion by division from larger base to a smaller base.
- Example: Convert a decimal number into other bases
  - radix(99, 2) 1100011
  - radix(99, 3) 10200
  - radix(99, 4) 1203
  - radix(99, 5) 344
  - radix(99, 6) 243
  - radix(99, 7) 201
  - radix(99, 8) 143
  - radix(99, 9) 120

Radix Conversion

- Radix conversion from other bases to decimal
  - Digits are multiplied by powers of the base or10, 8, 2, or whatever.

  - Decimal numbers multiply digits by powers of 10

  $9507_{10} = 9 \times 10^3 + 5 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$

  - Octal numbers - power of 8

  $1567_8 = 1 \times 8^3 + 5 \times 8^2 + 6 \times 8^1 + 7 \times 8^0$
  $= 512 + 320 + 48 + 7 = 887_{10}$

  - Binary numbers - power of 2

  $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
  $= 8 + 4 + 0 + 1 = 13_{10}$

Radix Conversion Example:

- Convert 5 from base 10 to base 2.
  1. Divide 5 by new base 2, then quotient 2 and **remainder 1**
  2. Divide quotient 2 by 2, then quotient 1 and **remainder 0**
  3. Divide quotient 1 by 2, then quotient 0 and **remainder 1**
     Stop when the quotient is 0.

$$5_{10} = 101_2$$

- Convert 99 from base 10 to base 8.
  1. Divide 99 by new base 8, then quotient 12 and **remainder 3**
  2. Divide quotient 12 by 8, then quotient 1 and **remainder 4**
  3. Divide quotient 1 by 8, then quotient 0 and **remainder 1**
     Stop when the quotient is 0.

$$99_{10} = 143_8$$

Possible Solutions:

- We could either
  - store remainders in a list by appending.
    - must continue the output until we get the quotient = 0
  - reverse the list
  - return the result as a compact string from the list.

- Iterative Algorithm
  - while the decimal number > 0
    - Divide the decimal number by the new base.
    - Set the decimal number = decimal number divided by the base.
    - Store the remainder to the left of any preceding remainders.

Recursive Algorithm

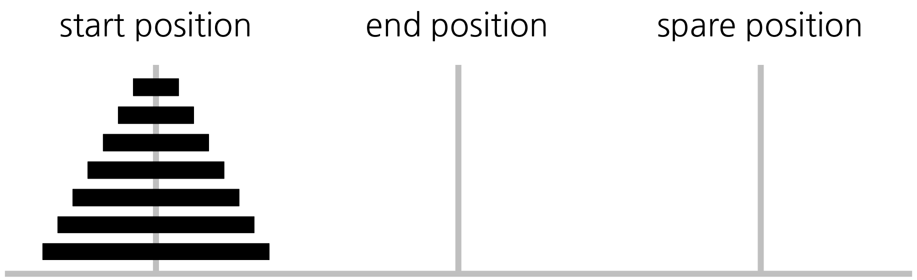
- Base case:
  - if decimal number == 0
    - do nothing (or return "")
- Recursive case
  - if decimal number > 0
    - solve a simpler version of the problem
      - use the quotient as the argument to the next call
    - store the current remainder (number % base) in the correct place

```
def radix(num, base):
    if num == 0:
        return ''
    return radix(num//base, base) + str(num % base)
```

Note: This code does not convert a decimal to a hexadecimal.
It is left as an exercise.

The Towers of Hanoi

- The famous towers of Hanoi consists of n discs and three poles.
  - The discs are of different size and have holes to fit themselves on the poles.
  - Initially all the discs are on one pole, e.g., pole A.
  - The task is to move all n discs to another pole, while obeying the following rules.
    - Move only one disc at a time.
    - Never place a larger disc on a smaller one.
  - One legend says that the world will end when a certain group of monks accomplishes this task in a temple with 64 golden discs on three diamond needles. But how can the monks accomplish the task at all, playing the rules?
  - To solve the problem, **our goal is to issue a sequence of instructions for moving the discs.**

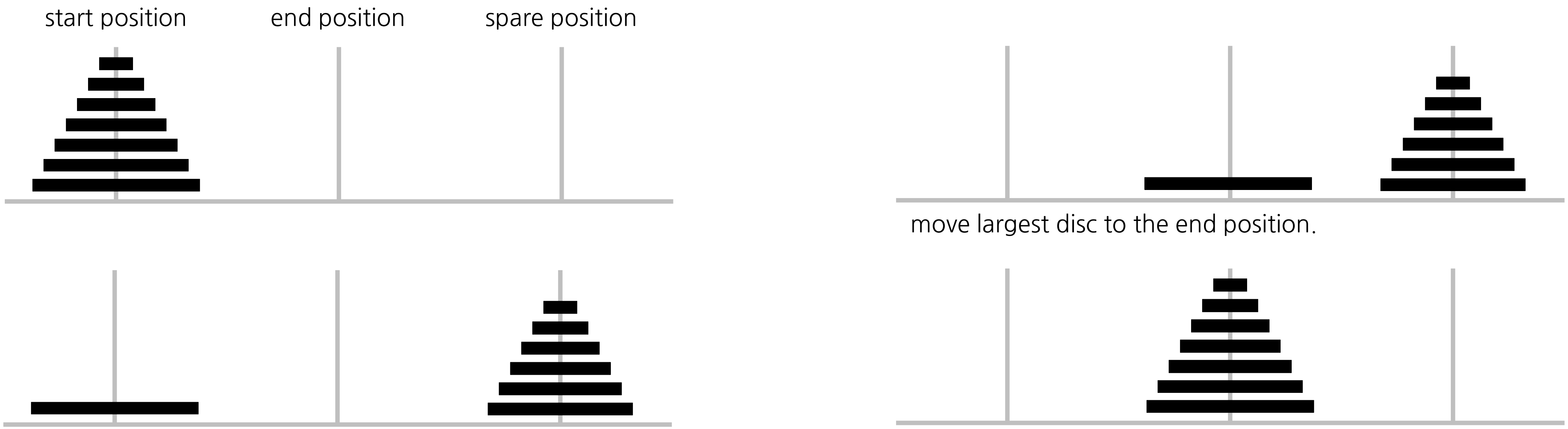


The Towers of Hanoi

- Examples:
  - https://www.youtube.com/watch?v=q6RicK1FCUs
  - https://sikaleo.tistory.com/29 (한국어)

The Towers of Hanoi

- Recursive algorithm:
  1. Move the top **n-1** discs from **start to spare (using end), recursively**.
  2. Move the **remaining (largest)** disc from **start to end**.
  3. Move the **n-1** discs from **spare to end (using start), recursively.**



move largest disc to the end position.

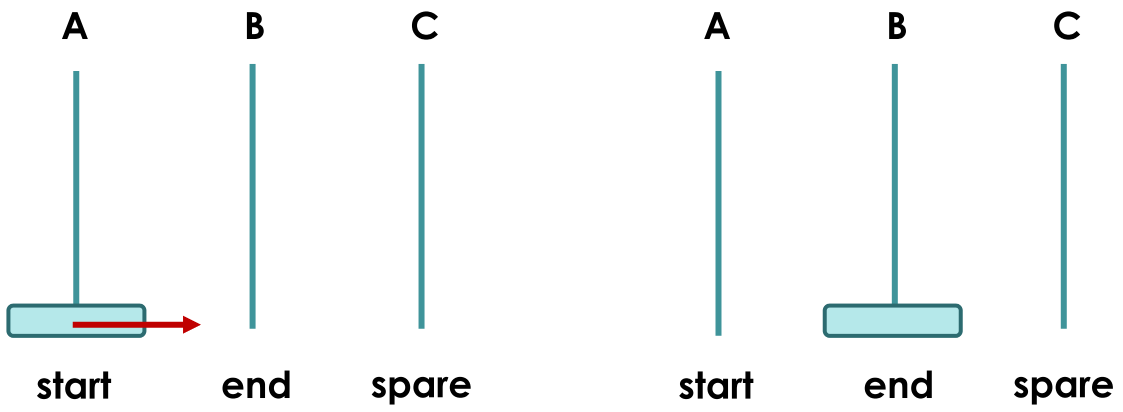
move n-1 discs to the spare position (using end) recursively

move n-1 discs to the end position (using start) recursively.

The Towers of Hanoi

- Recursive algorithm:
  1. Move the top **n-1** discs from **start to spare (using end), recursively**.
  2. Move the **remaining (largest)** disc from **start to end**.
  3. Move the **n-1** discs from **spare to end (using start), recursively.**

One disc case:
(1) move a disc from A to B.

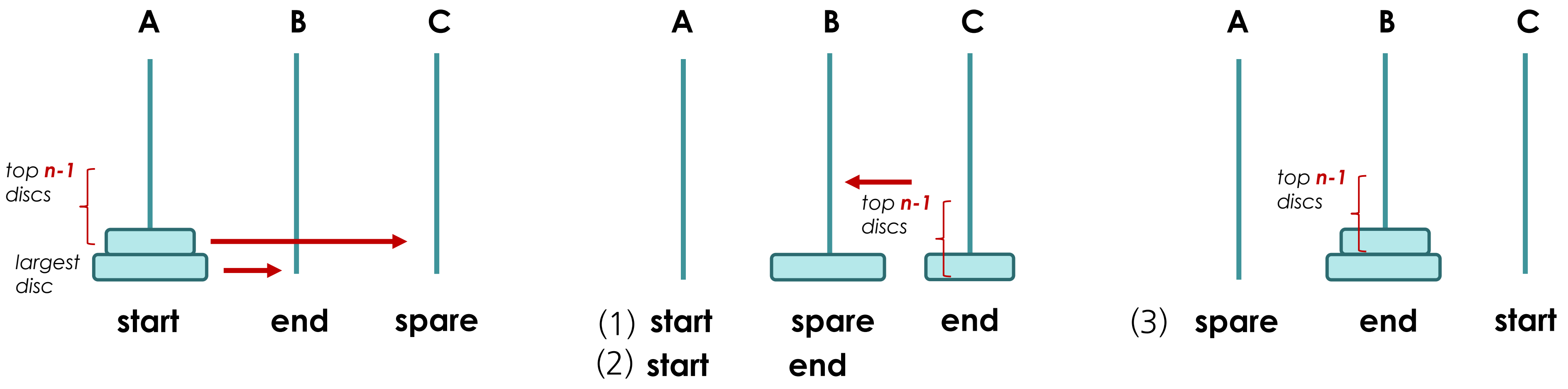


The Towers of Hanoi

- Recursive algorithm:
  1. Move the top **n-1** discs from **start to spare (using end), recursively**.
  2. Move the **remaining (largest)** disc from **start to end**.
  3. Move the **n-1** discs from **spare to end (using start), recursively.**

Two discs case:
(1) move a disc from A to C **using B.**
(2) move a disc from A to B.
(3) move a disc from C to B **using A.**

since it is not the end(or destination)



The Towers of Hanoi

Three discs case:
(1) move **two discs** from A to C **using B**.
(2) move a disc from A to B.
(3) move **two discs** from C to B **using A**

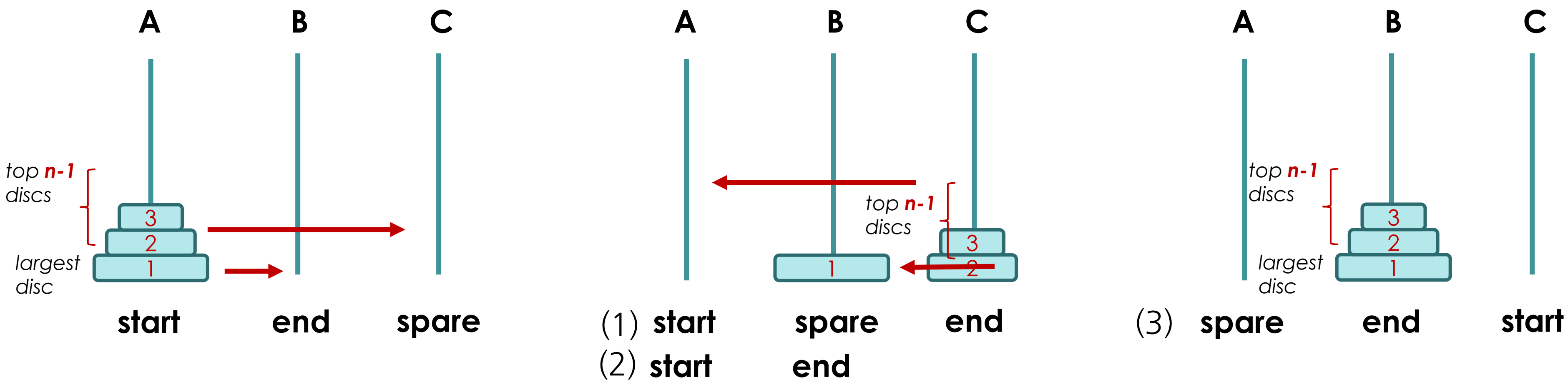
for n discs

for n discs:
(1) move **n – 1 discs** from A to C **using B**.
(2) move a disc from A to B.
(3) move **n – 1 discs** from C to B **using A**

since it is not the end(or destination)

This is a recursive step.
We already have done this two discs case before.

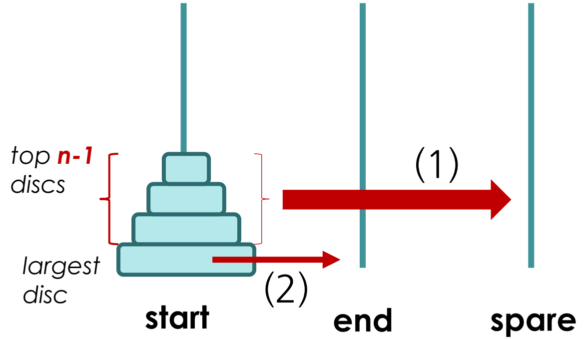
```
def hanoi(n, start, end, spare):
    if n >= 1:
        hanoi(n - 1, start, spare, end)
        print(f"move disc {n} from {start} to {end}")
        hanoi(n - 1, spare, end, start)
if __name__=='__main__':
    hanoi(3, 'A', 'B', 'C')
```

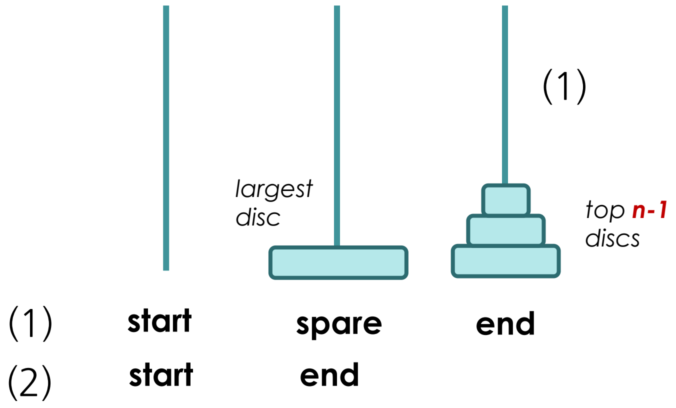


The Towers of Hanoi

- Recursive algorithm:
  1. Move the top **n-1** discs from **start to spare (using end), recursively**.
  2. Move the **remaining (largest)** disc from **start to end**.
  3. Move the **n-1** discs from **spare to end (using start), recursively.**

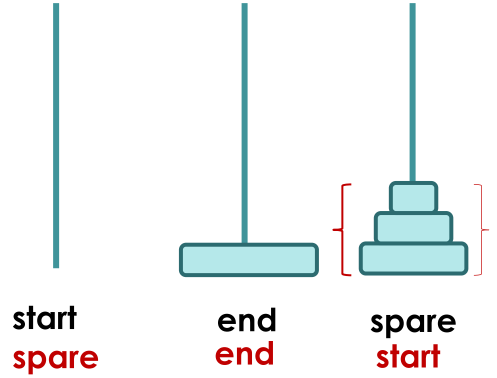
4 discs problem

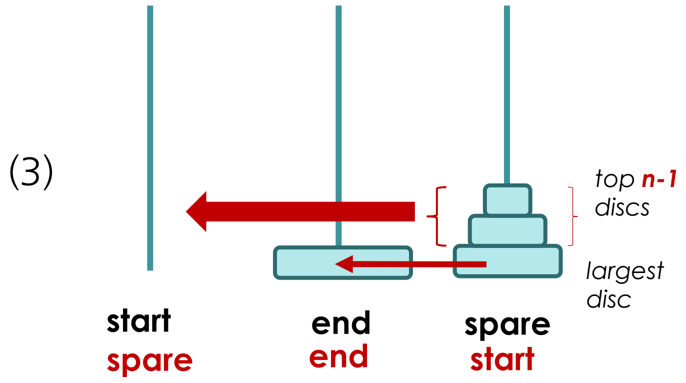




(3) It becomes a **3 discs problem.**
Go back to step 1.
Treat the **spare as start** and
the **start as spare**.

3 discs problem





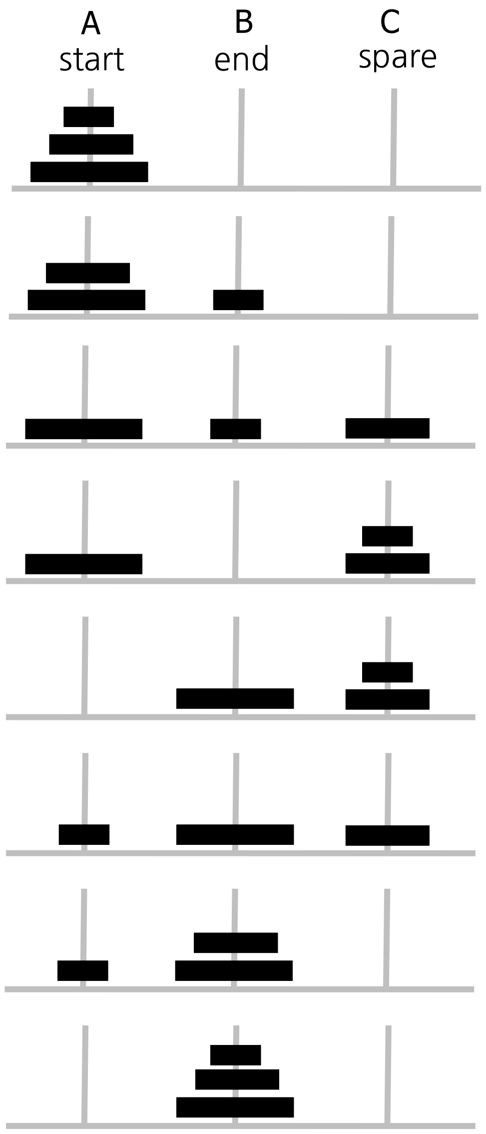
The Towers of Hanoi - Algorithm

- Question: How many moves and recursive calls made?

```
def hanoi(n, start, end, spare):
    if n >= 1:
        hanoi(n - 1, start, spare, end)
        print(f"move disc {n} from {start} to {end}")
        hanoi(n - 1, spare, end, start)

if __name__=='__main__':
    hanoi(3, 'A', 'B', 'C')
```

```
move disc 1 from A to B
move disc 2 from A to C
move disc 1 from B to C
move disc 3 from A to B
move disc 1 from C to A
move disc 2 from C to B
move disc 1 from A to B
```



The Towers of Hanoi - Coding Exercise

- Idea: It is hard to check the correctness of the previous **`hanoi()`**.
  - Let us use a list to present a disc in a pole and display the result as shown below. The number in a list represents the size of the disc. `tower()` prints the current status of the tower in a list format. Test the cases such as n = 1, 2, 3, 4, 5, 6.
-

```python
def hanoi(n, start, end, spare):
    if n >= 1:
        None

def tower(A, B, C):
    print(None)

if __name__=='__main__':
    n = 3
    A = [* range(1, n+1)]
    B = []
    C = []
    tower(A, B, C)
    hanoi(n, A, B, C)
```

```
start-[1, 2, 3]     end-[]          spare-[]
start-[2, 3]        end-[1]         spare-[]
start-[3]           end-[1]         spare-[2]
start-[3]           end-[]          spare-[1, 2]
start-[]            end-[3]         spare-[1, 2]
start-[1]           end-[3]         spare-[2]
start-[1]           end-[2, 3]      spare-[]
start-[]            end-[1, 2, 3]   spare-[]
```

The Towers of Hanoi - Time complexity

- Recursive algorithm:
  - Move the top n-1 discs from start to spare.
  - Move the remaining (largest) disc from start to end.
  - Move the n-1 discs from spare to end.

$$\mathrm{hanoi}(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \cdot \mathrm{hanoi}(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Exercise: How many years will take to move 64 discs?
(1) hanoi(1) = 1
(2) hanoi(2) = 3
(3) hanoi(3) = 7
(4) hanoi(4) = 15
(5) hanoi(5) = 31
(6) hanoi(32) = 4,294,967,295
(7) hanoi(64) = 18,446,744,073,709,600,000

hanoi(n = 4)
hanoi(4) = 2*hanoi(3) + 1 = 2*(2*hanoi(2) + 1) + 1 = 2*(2*(2*hanoi(1) + 1) + 1) + 1 = 2*(2*(2*1 + 1) + 1) + 1 = 2*(2*(3) + 1) + 1 = 2*(7) + 1 = 15

The Towers of Hanoi - Time complexity

- Solving the recurrence equation of the Hanoi Tower.
  - $T(n) = 2T(n-1) + 1$
    $T(n-1) = 2T(n-2) + 1$
    $T(n-2) = 2T(n-3) + 1$

    T(n) can be rewritten some substitutions
    $T(n) = 2(2(2T(n-3) + 1) + 1) + 1$
    $= 2^3 T(\mathbf{n-3}) + 2^2 + 2^1 + 1$
    ...
    Expand this T(n) until it has $T(\mathbf{n-k})$ term since we know $T(1) = 1$.

  After generalization
  - $T(n) = 2^k T(\mathbf{n-k}) + 2^{k-1} + 2^{k-2} + \dots 2^2 + 2^1 + 1$

  Since base condition $T(1) = 1$, and then $\mathbf{n - k} = 1$, $k = n - 1$
  - Replace k with $k = n - 1$.
  - $T(n) = 2^{n-1} T(0) + 2^{n-2} + 2^{n-3} + \dots 2^2 + 2^1 + 1 = \mathbf{2^n - 1}$
  - The time complexity is $O(2^n)$
  - For 5 discs, $n = 5$, it will take $2^5 - 1 = 31$ moves.

The Towers of Hanoi - Time complexity

- Write a recursive function to compute the number of disc's move first. Then compute the number of years to move 64 discs, while assuming that a group of monks really work diligently to move the disc fast like a computer clock speed or one disc per nano second (10^{-9} sec). Show your code and computation below:

Summary

- Recursion simplifies program structure at a cost of function calls (using the system stack).
- Understand and learn how to implement the recursive functions for different applications.