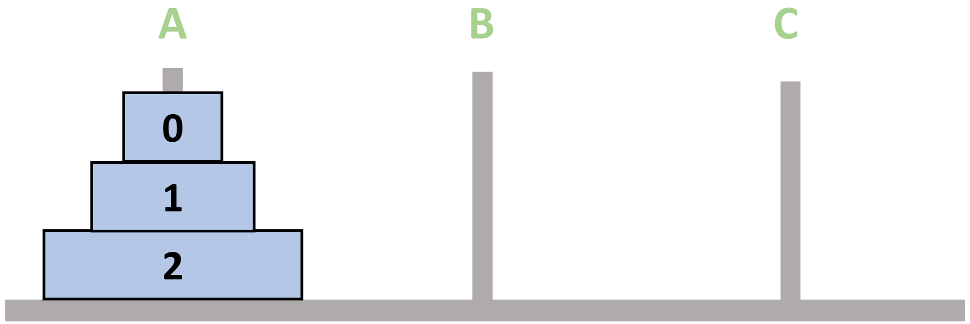
Towers of Hanoi

(Algorithmic Problems)

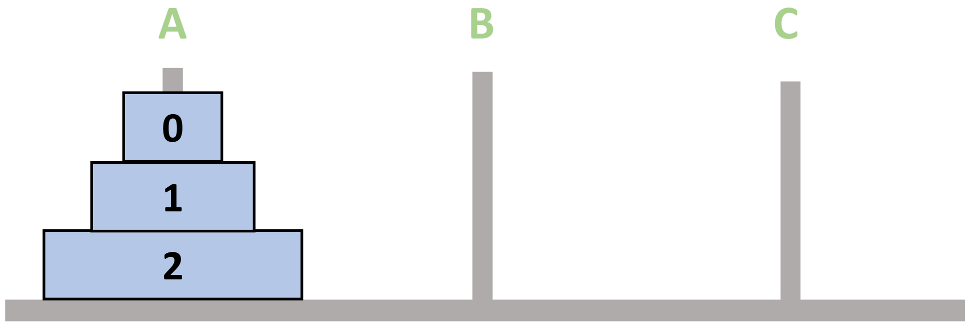
Towers of Hanoi



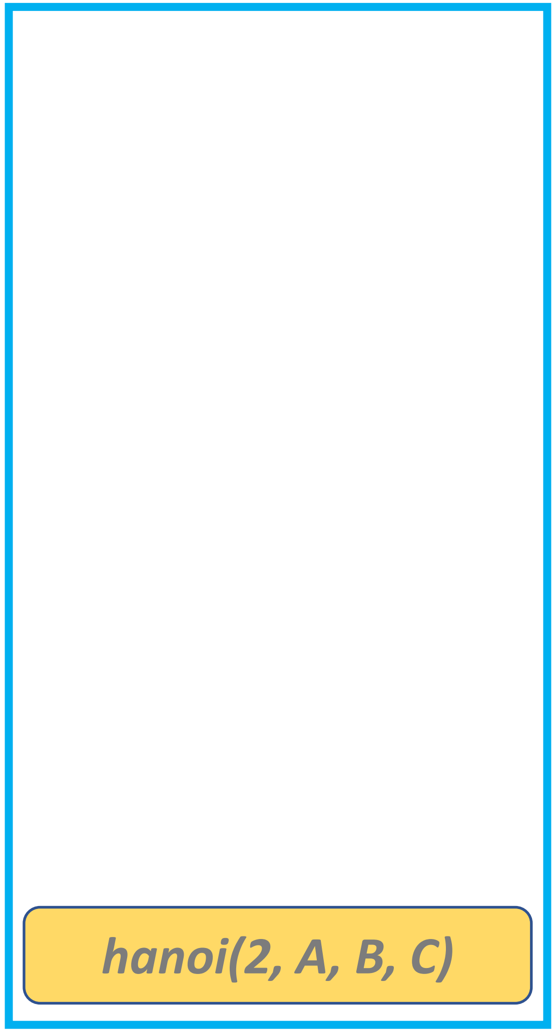
```
def hanoi(disk, source, middle, dest):

      if n==0:
         move disk from source to dest
         return

      hanoi(disk-1, source, dest, middle)
      move disk from source to dest
      hanoi(disk-1, middle, source, dest)
```

STACK

Towers of Hanoi



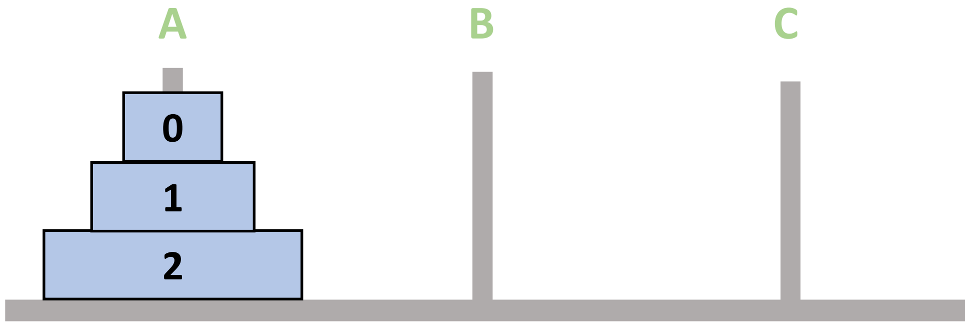
```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```



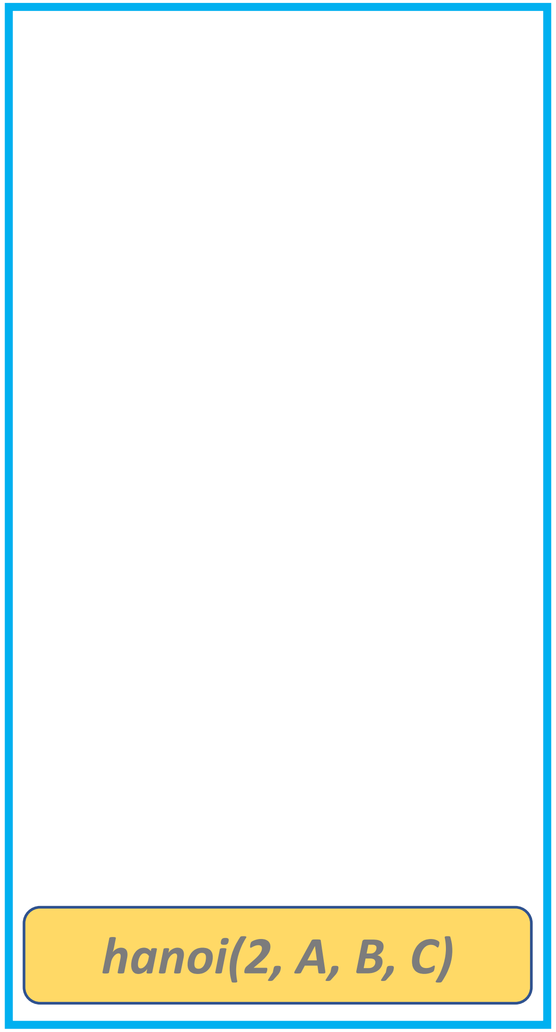
hanoi(2, A, B, C)

STACK

Towers of Hanoi

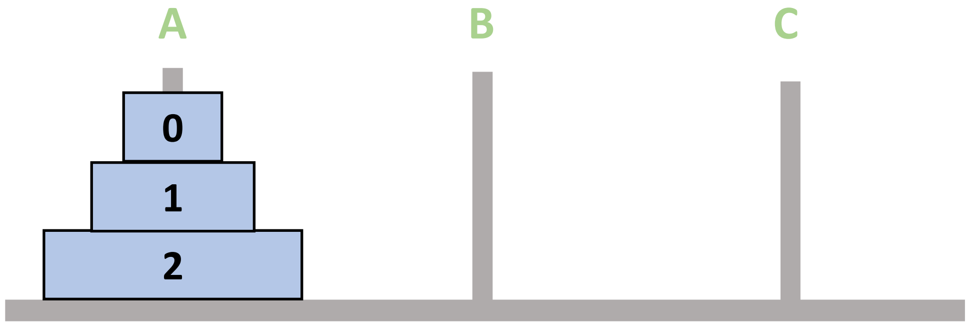


```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

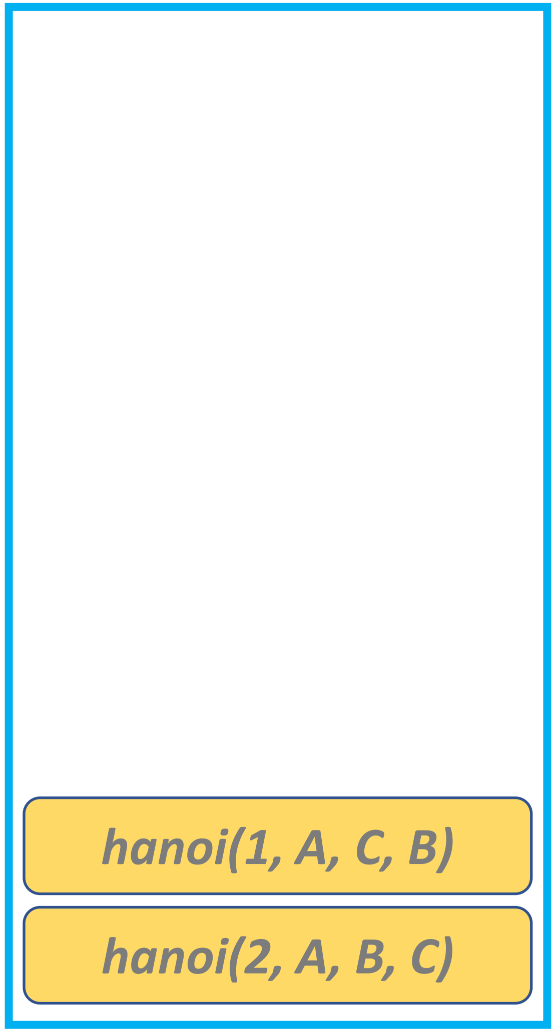


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

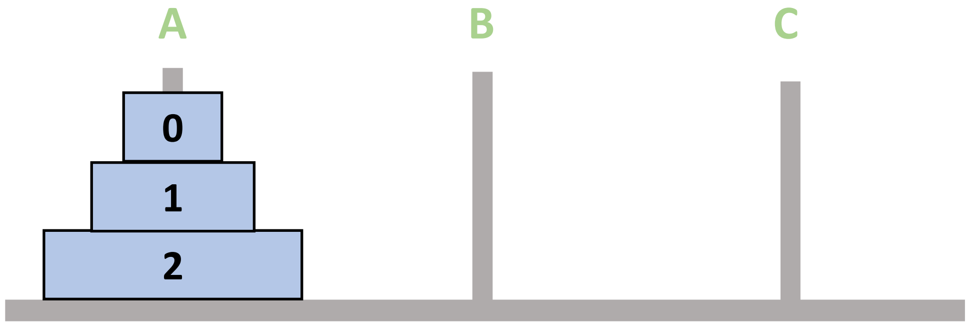


hanoi(1, A, C, B)

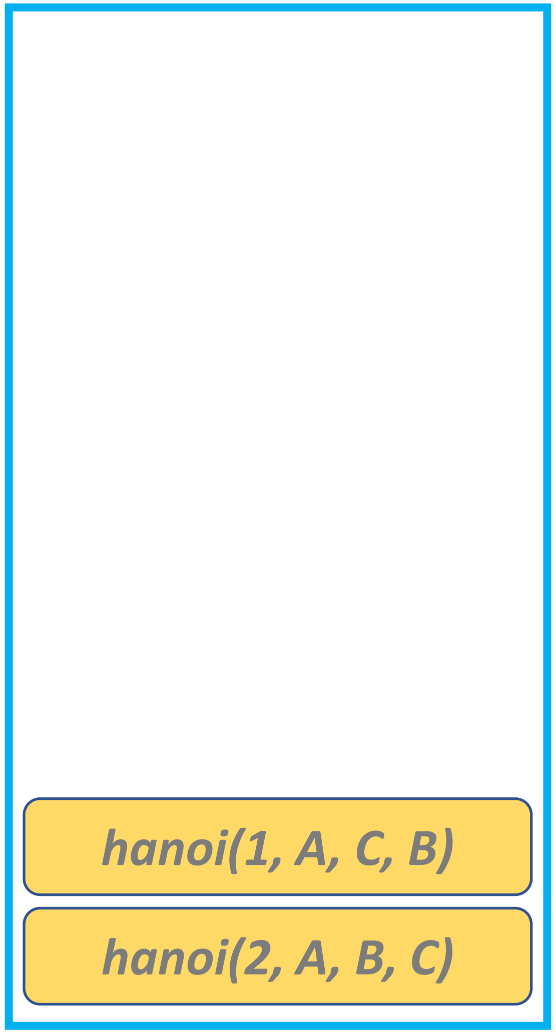
hanoi(2, A, B, C)

STACK

Towers of Hanoi

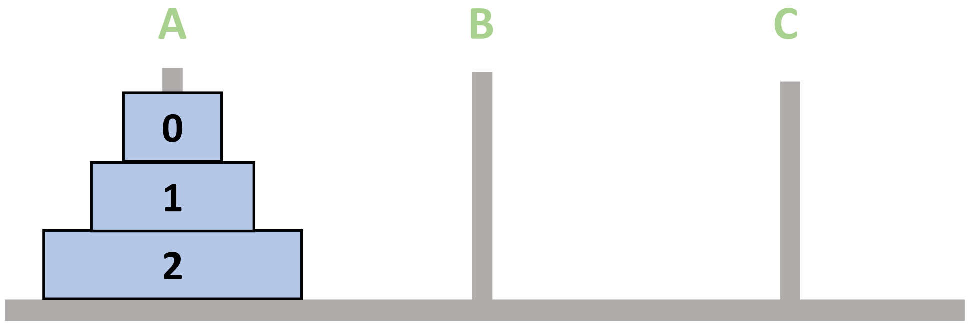


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

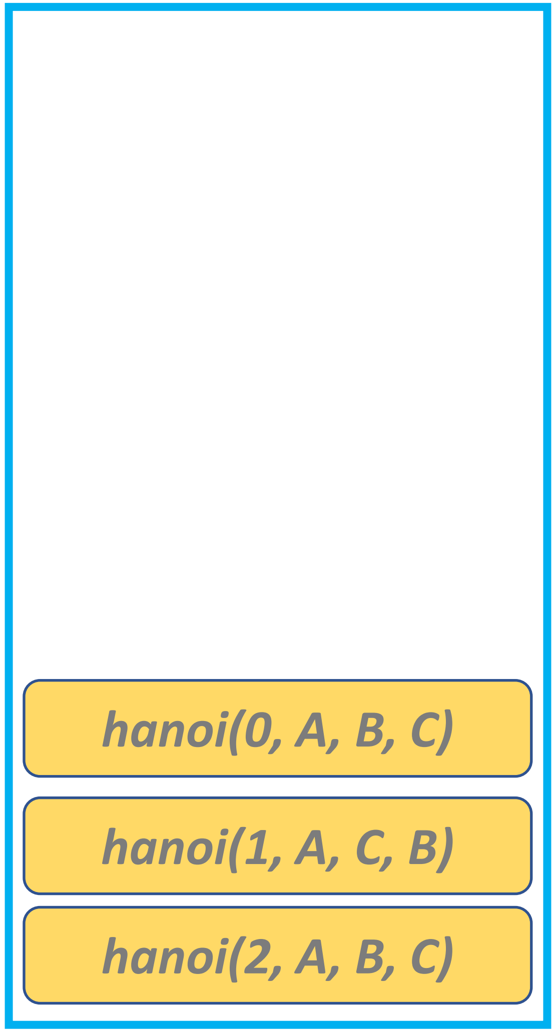


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



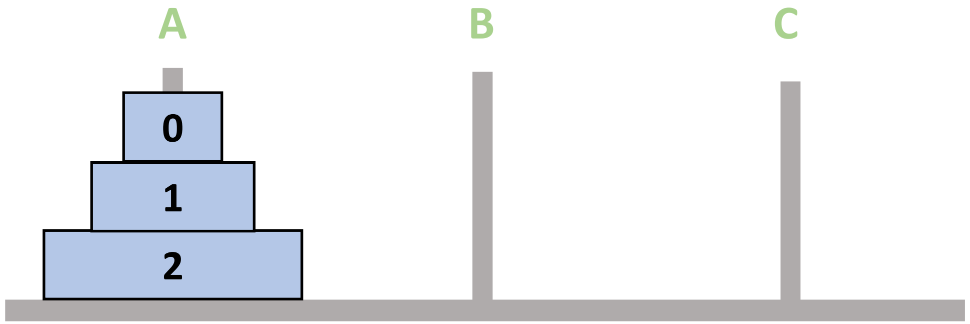
hanoi(0, A, B, C)

hanoi(1, A, C, B)

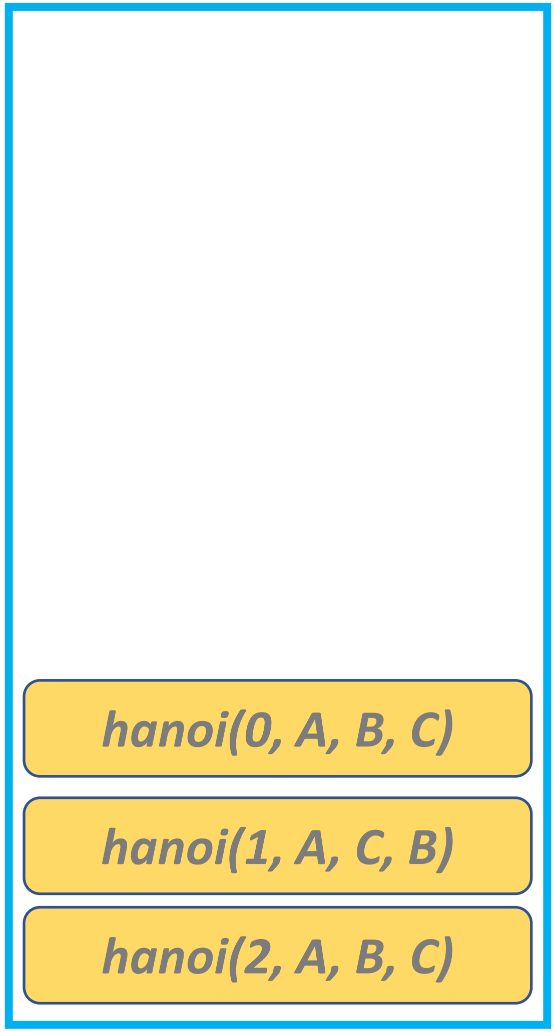
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

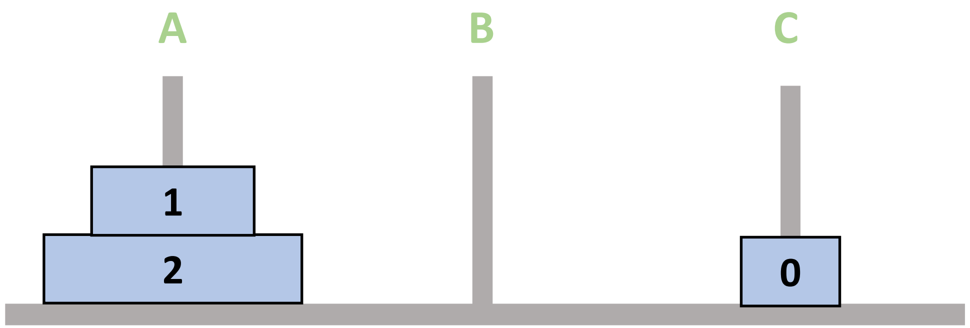

hanoi(0, A, B, C)

hanoi(1, A, C, B)

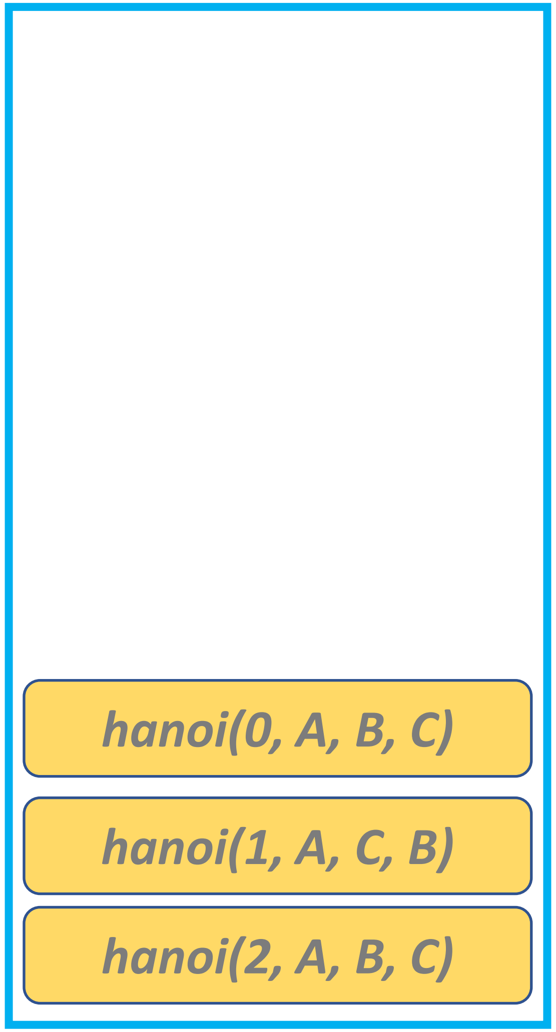
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



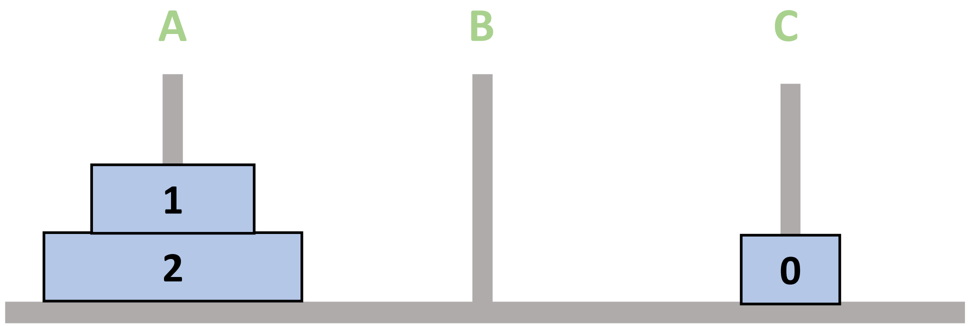
hanoi(0, A, B, C)

hanoi(1, A, C, B)

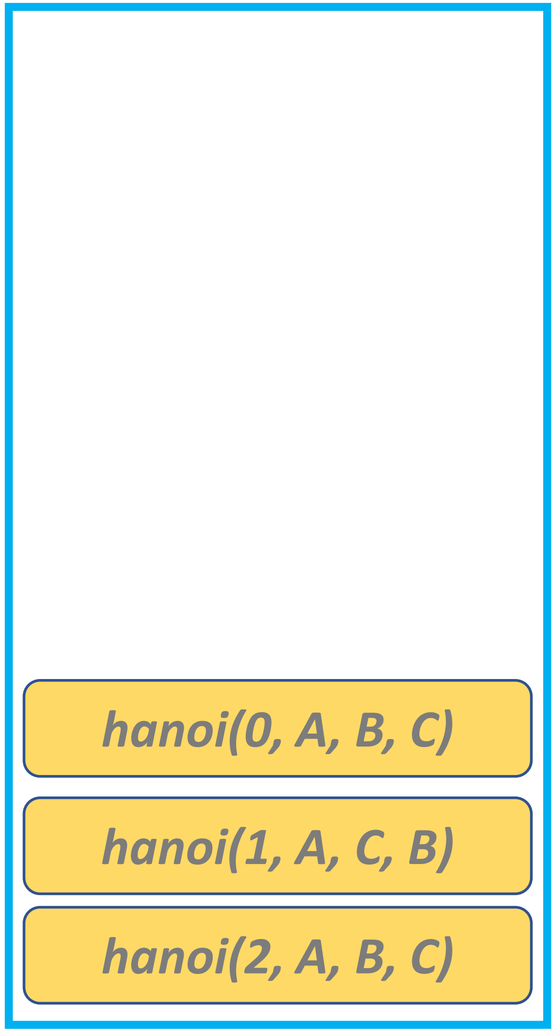
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



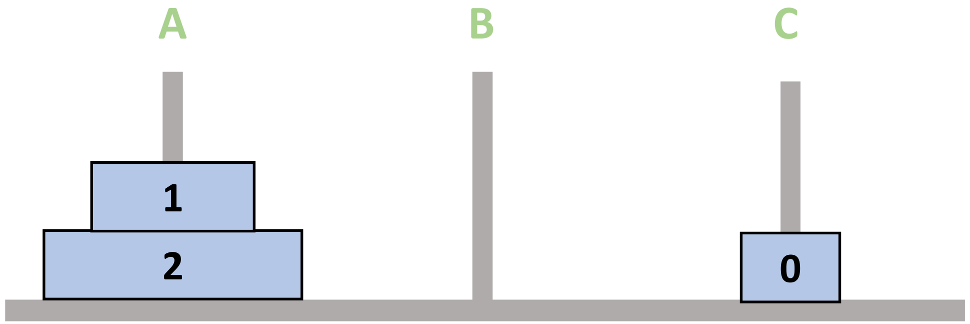
hanoi(0, A, B, C)

hanoi(1, A, C, B)

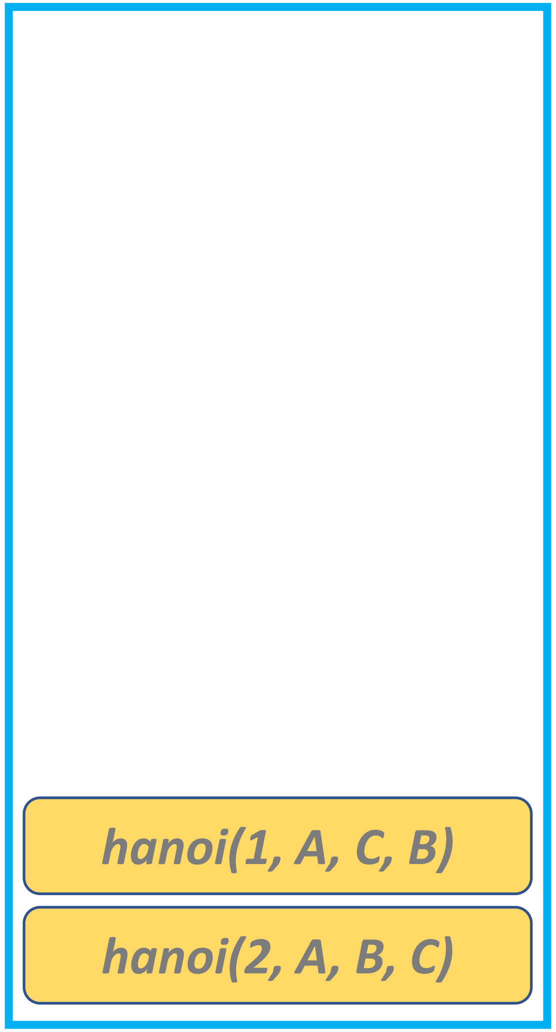
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

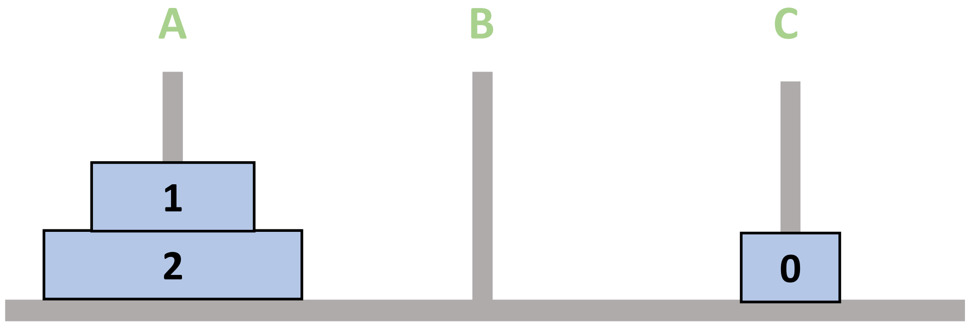


hanoi(1, A, C, B)

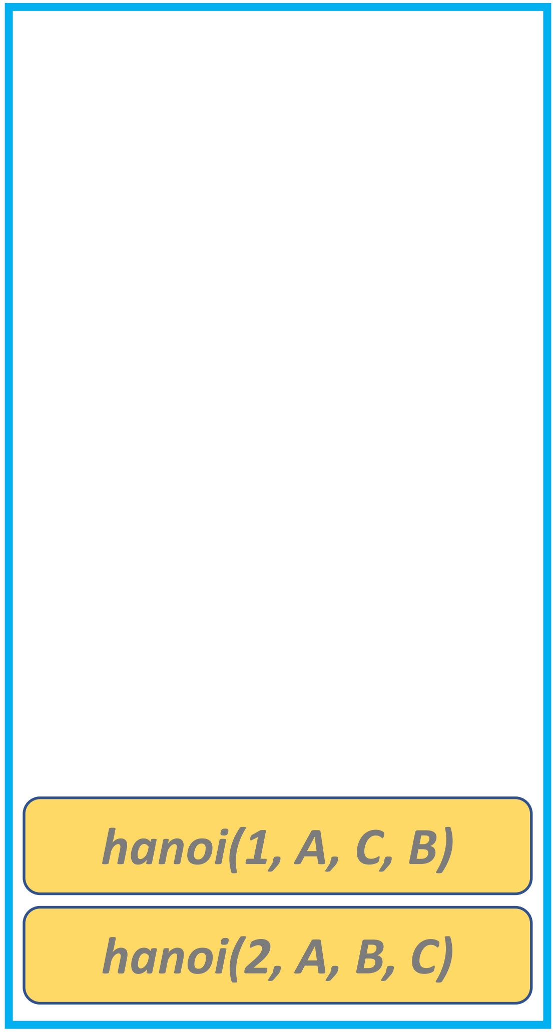
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

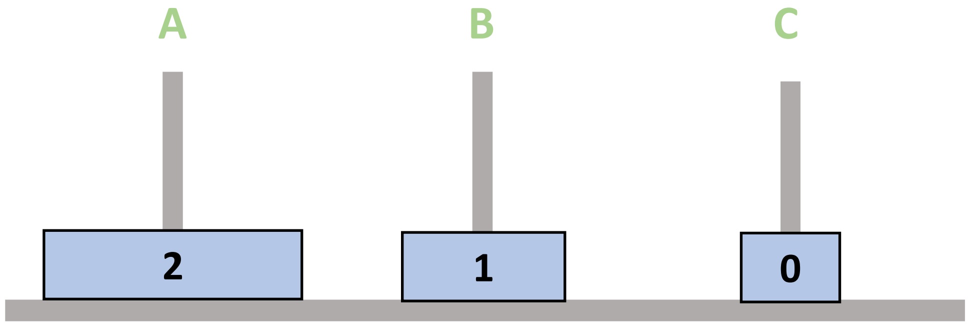


hanoi(1, A, C, B)

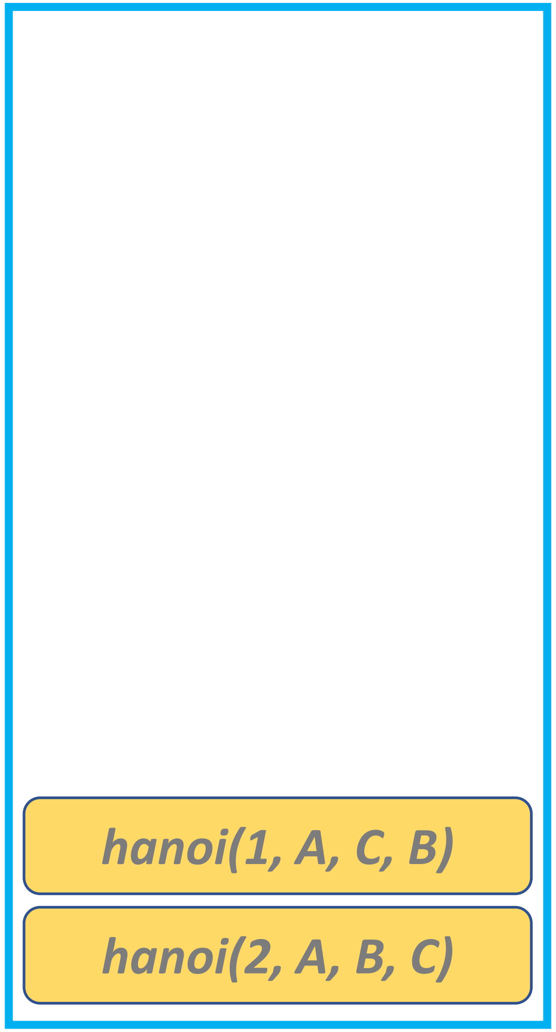
hanoi(2, A, B, C)

STACK

Towers of Hanoi



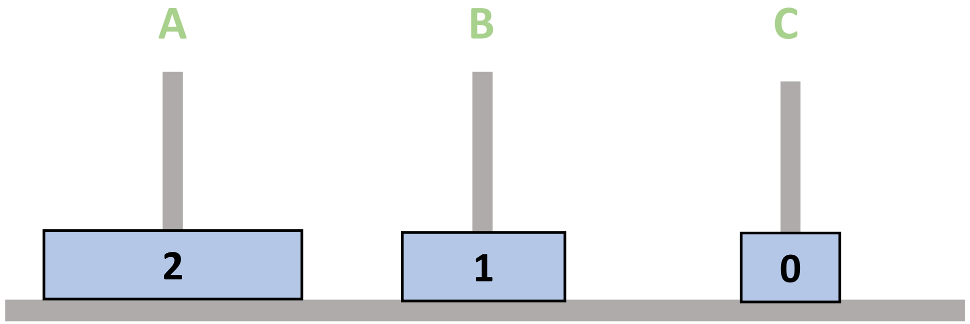
```
def hanoi(disk, source, middle, dest):

          if n==0:
             move disk from source to dest
             return

          hanoi(disk-1, source, dest, middle)
          move disk from source to dest
          hanoi(disk-1, middle, source, dest)
```


hanoi(1, A, C, B)

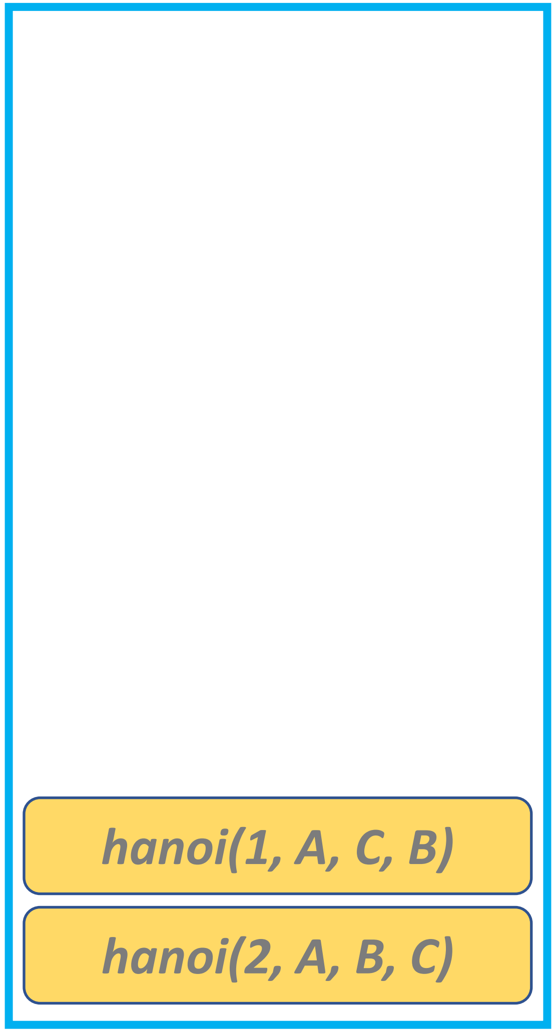
hanoi(2, A, B, C)

STACK

Towers of Hanoi

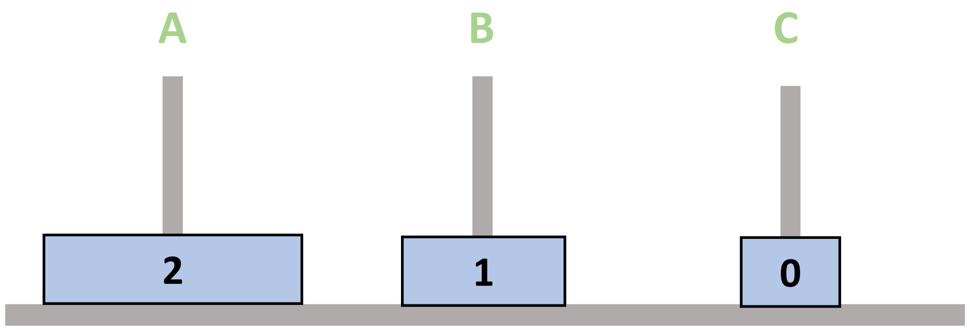


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

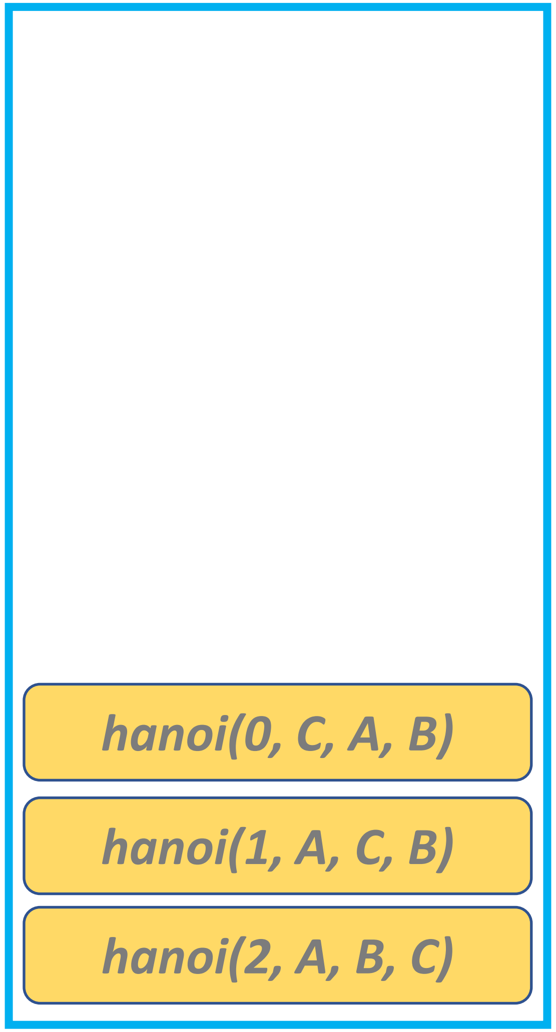


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

          if n==0:
             move disk from source to dest
             return

          hanoi(disk-1, source, dest, middle)
          move disk from source to dest
          hanoi(disk-1, middle, source, dest)
```

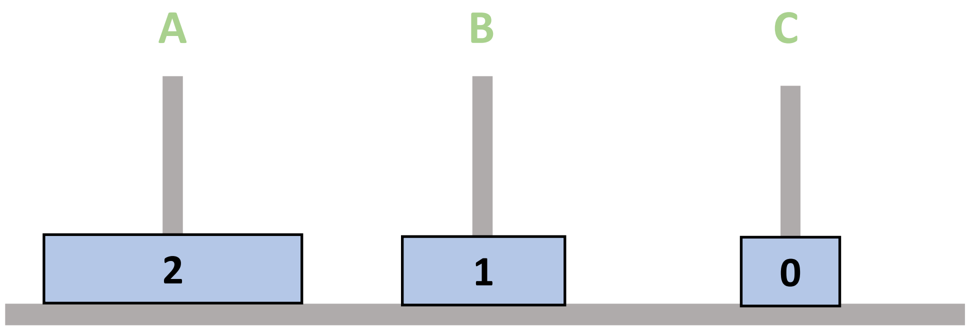

hanoi(0, C, A, B)

hanoi(1, A, C, B)

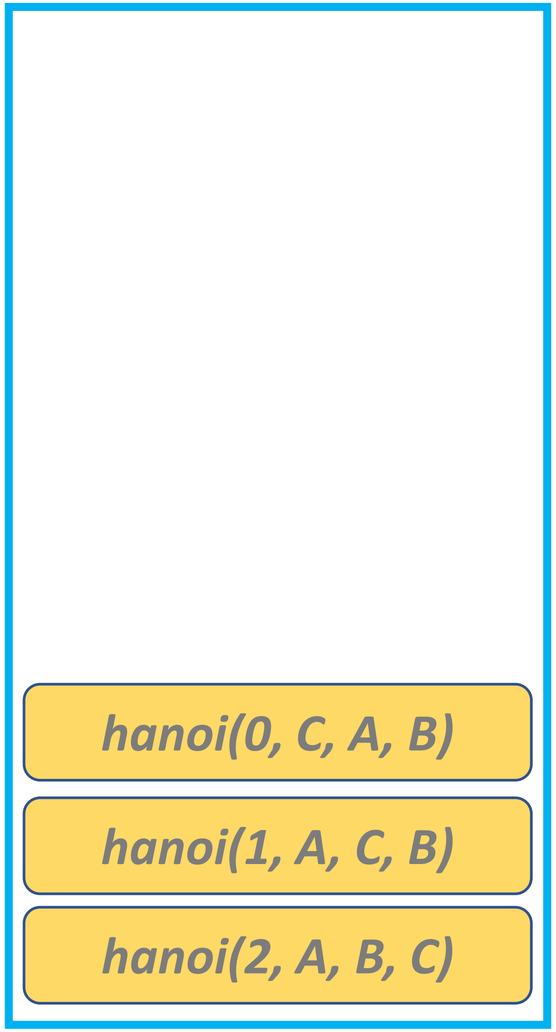
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



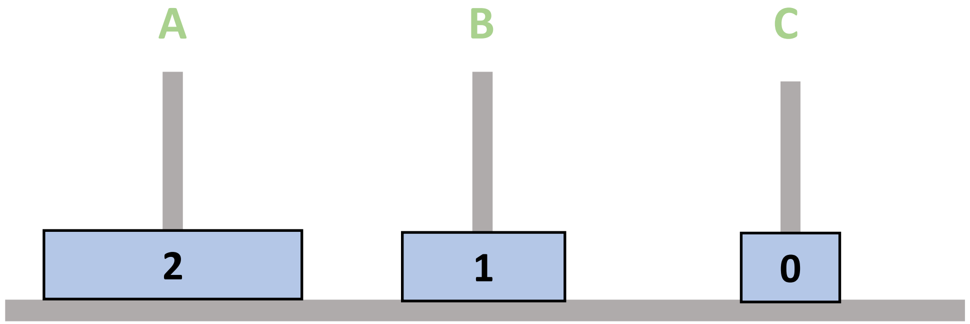
hanoi(0, C, A, B)

hanoi(1, A, C, B)

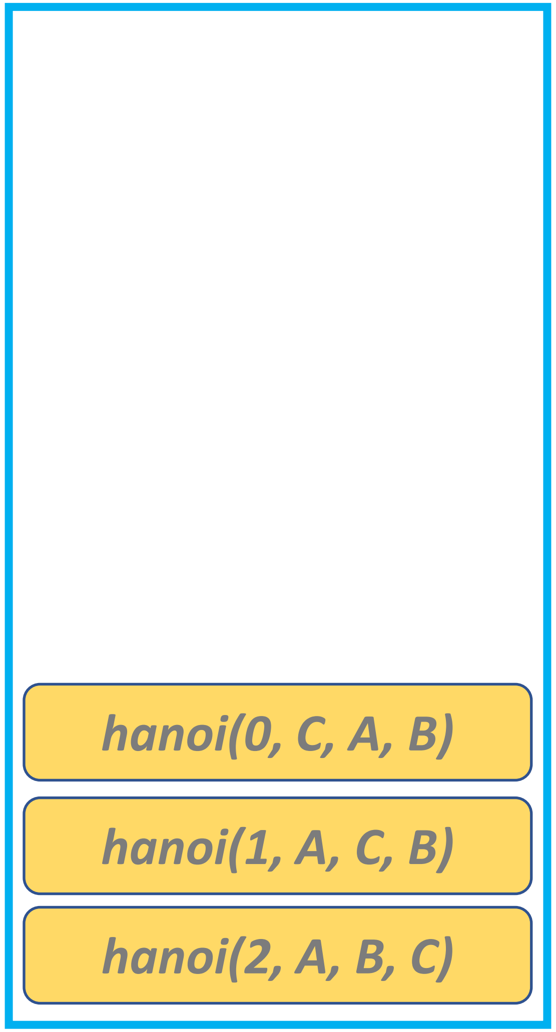
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



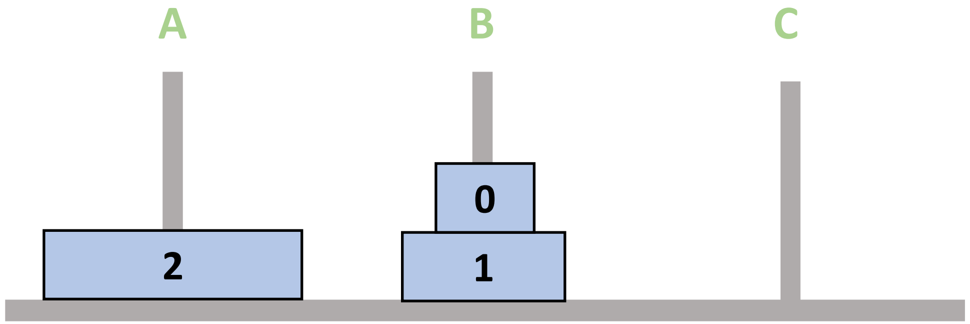
hanoi(0, C, A, B)

hanoi(1, A, C, B)

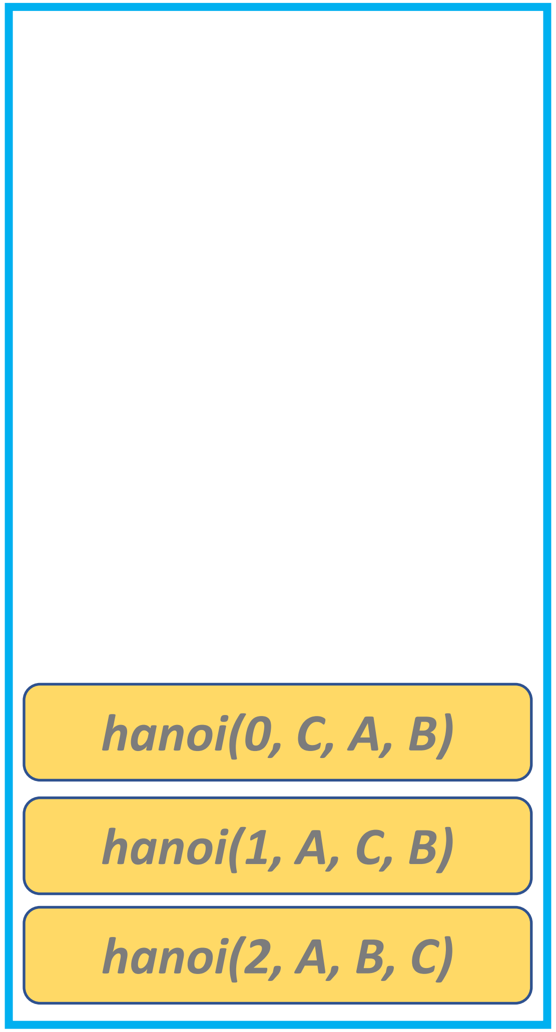
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



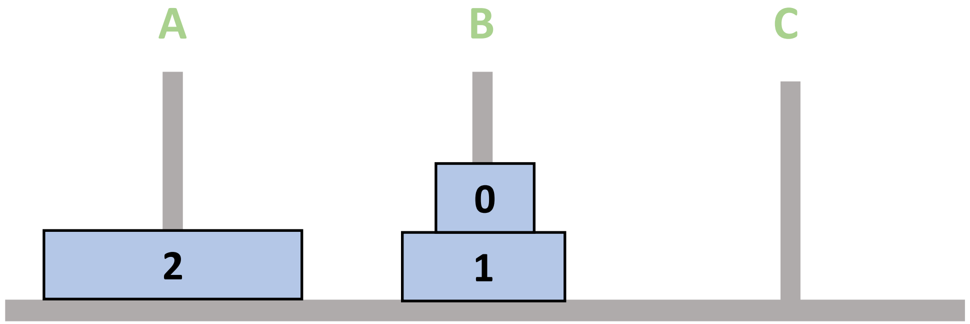
hanoi(0, C, A, B)

hanoi(1, A, C, B)

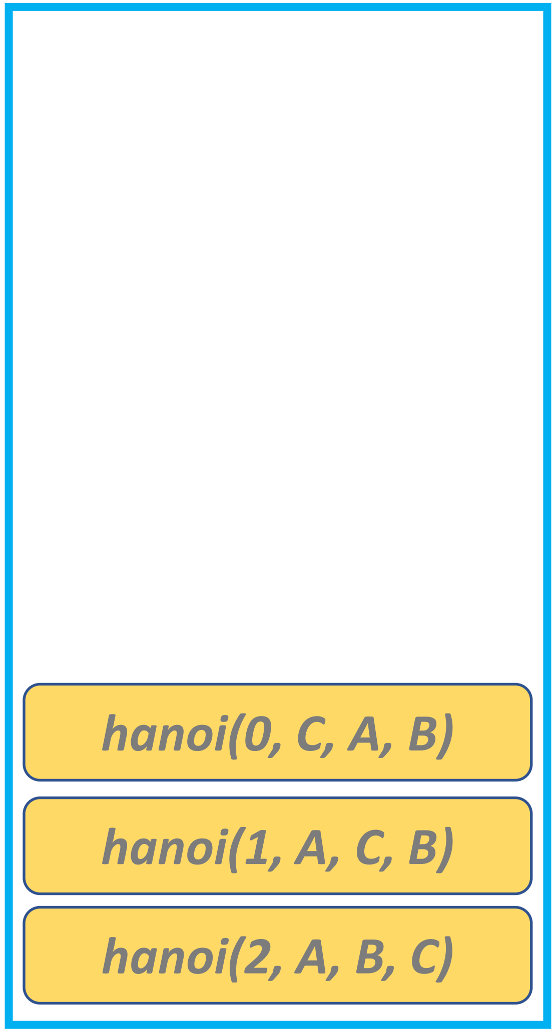
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



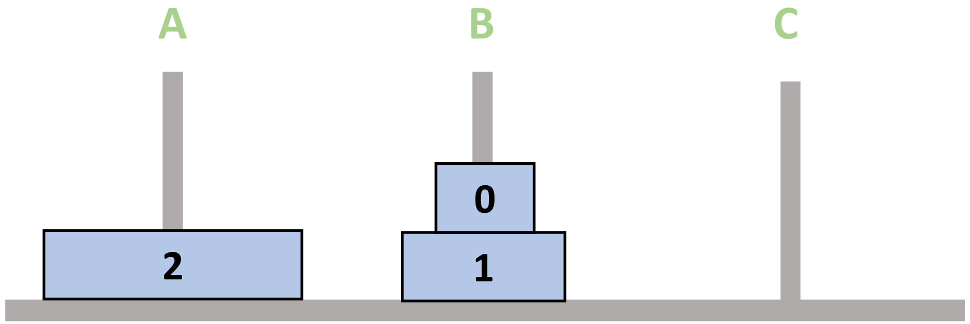
hanoi(0, C, A, B)

hanoi(1, A, C, B)

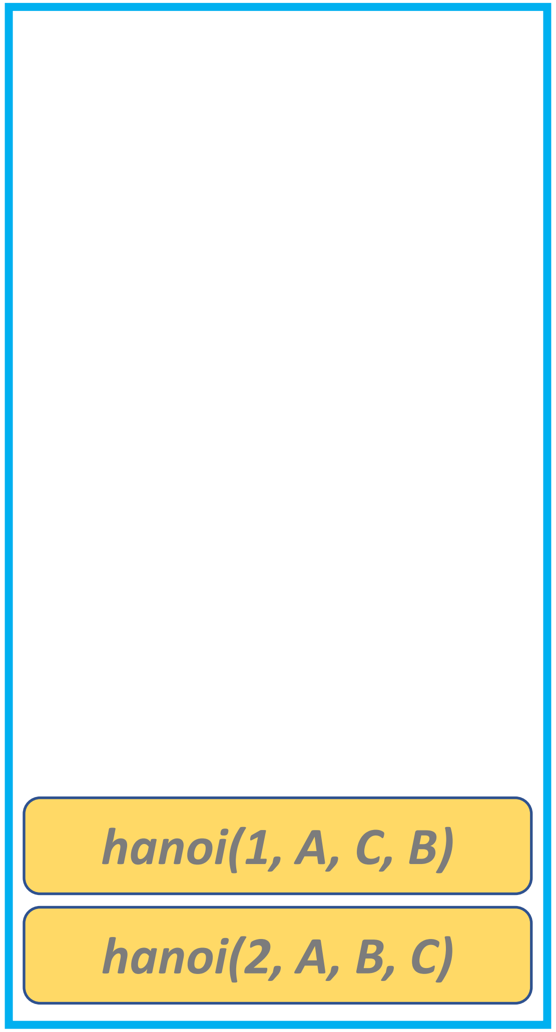
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

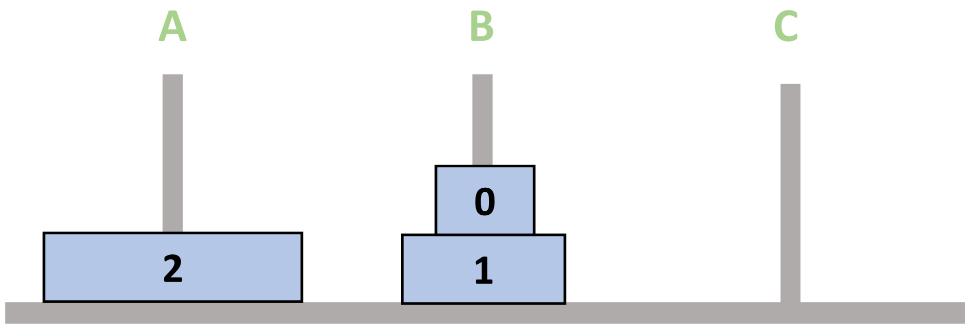


hanoi(1, A, C, B)

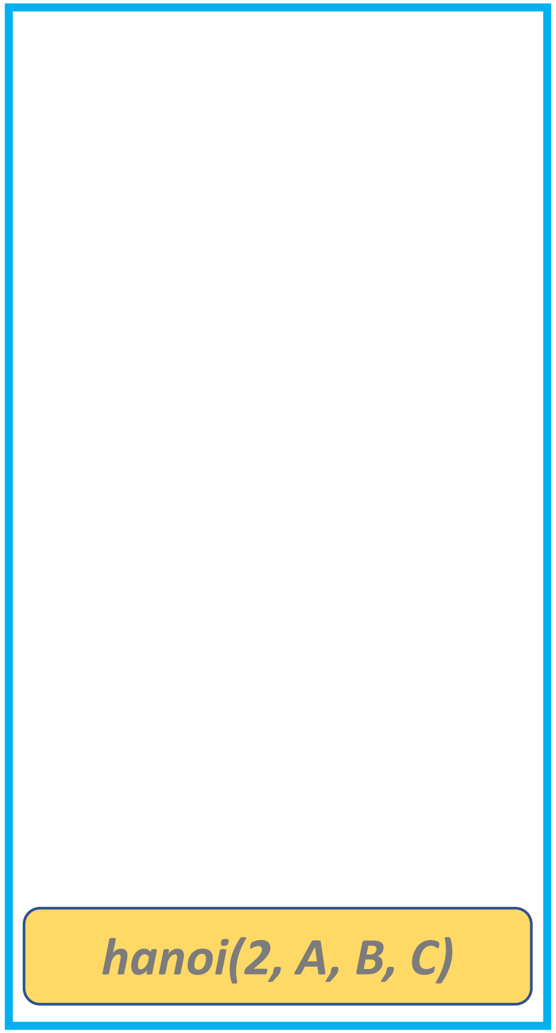
hanoi(2, A, B, C)

STACK

Towers of Hanoi

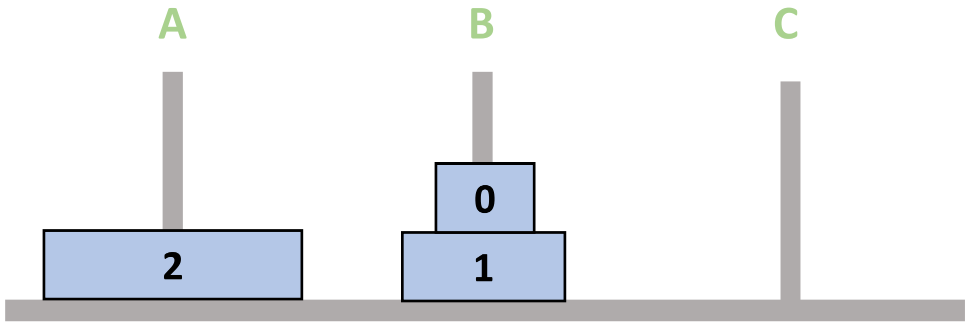


```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

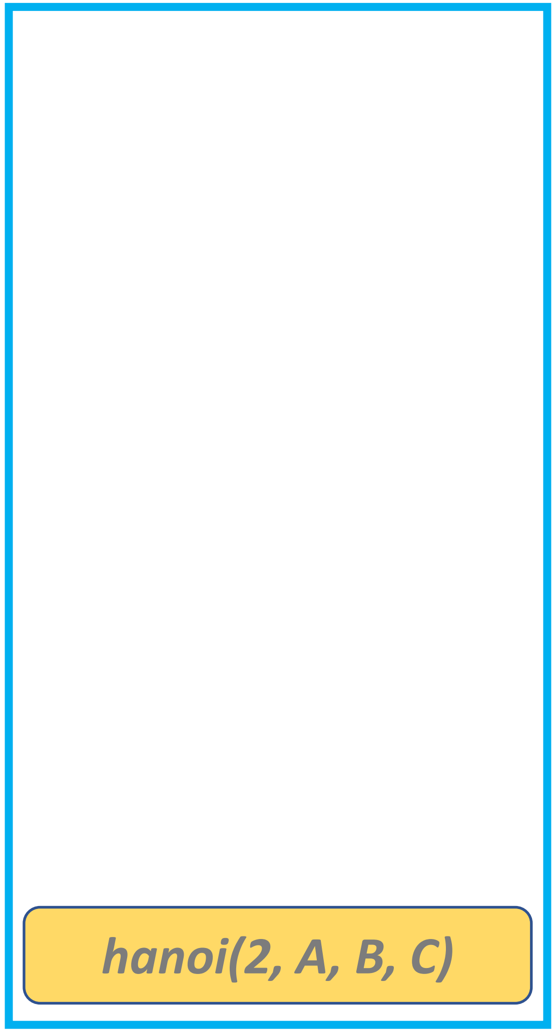


STACK

Towers of Hanoi

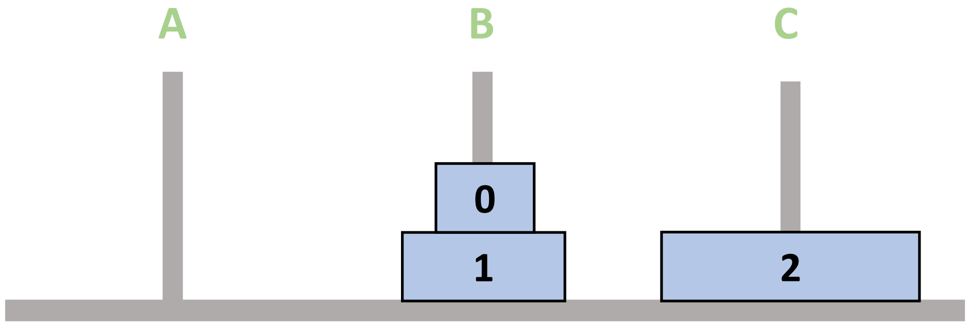


```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

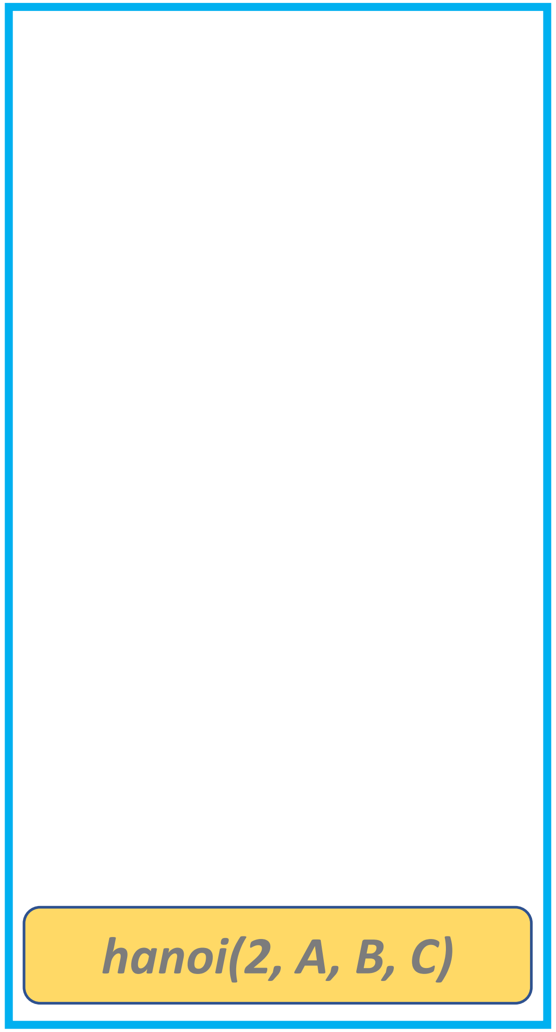


STACK

Towers of Hanoi



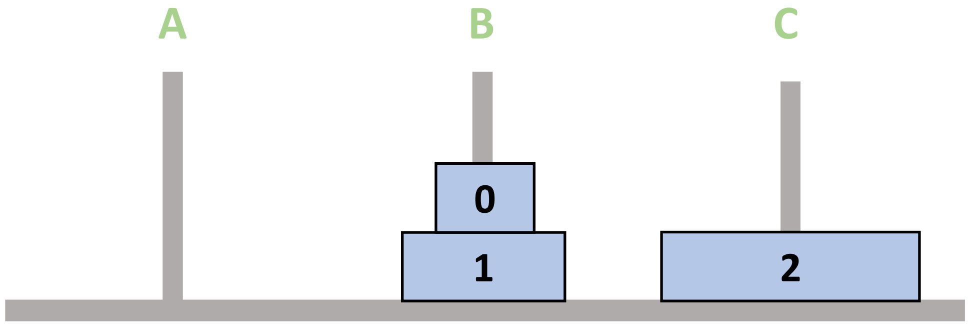
```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```



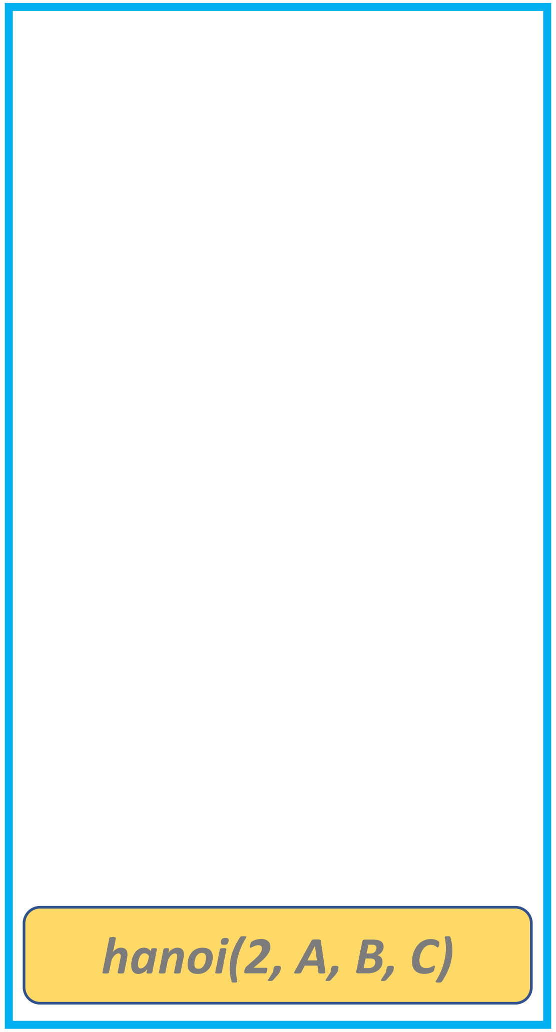
hanoi(2, A, B, C)

STACK

Towers of Hanoi

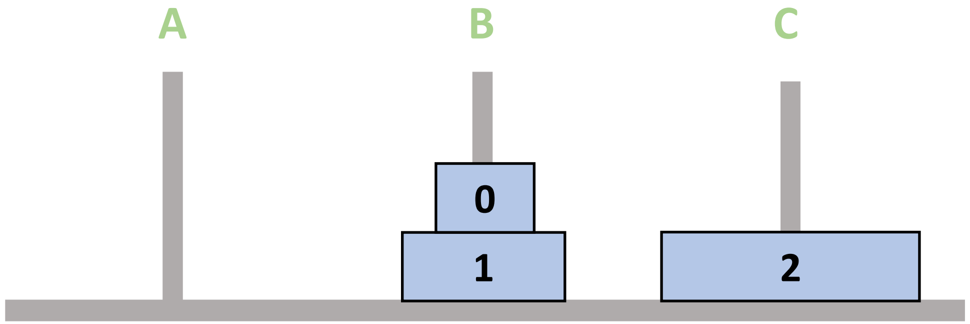


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

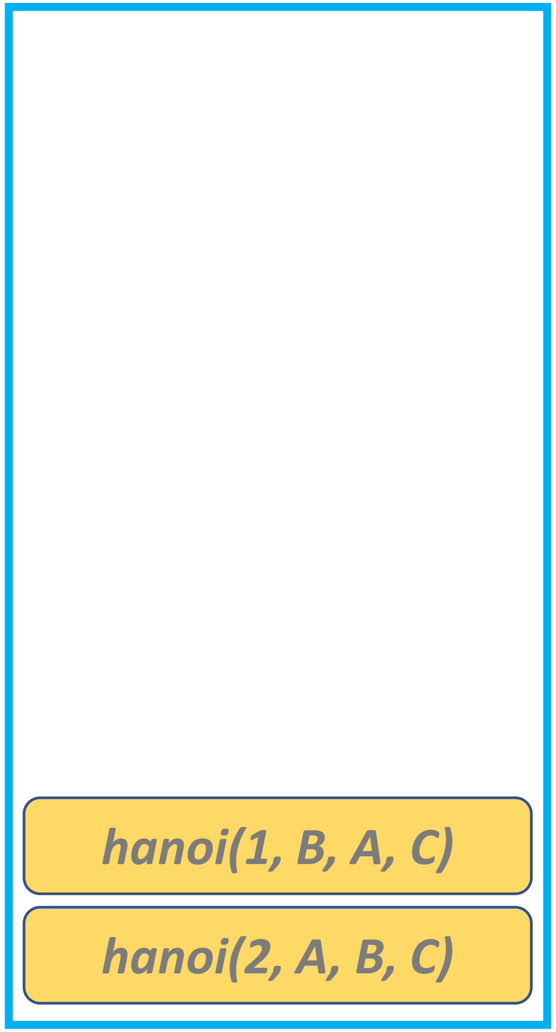


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

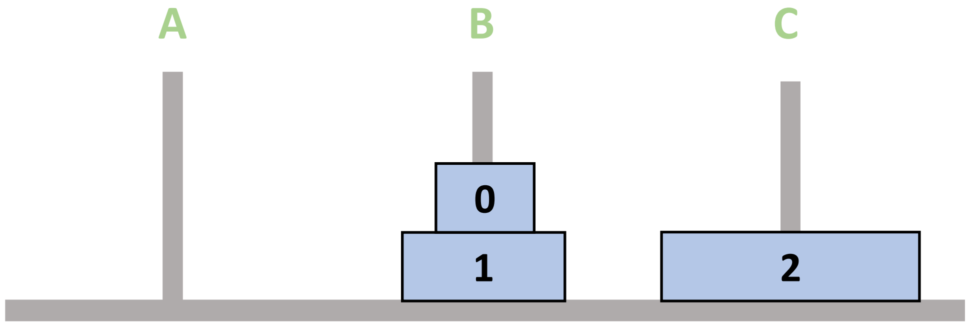


hanoi(1, B, A, C)

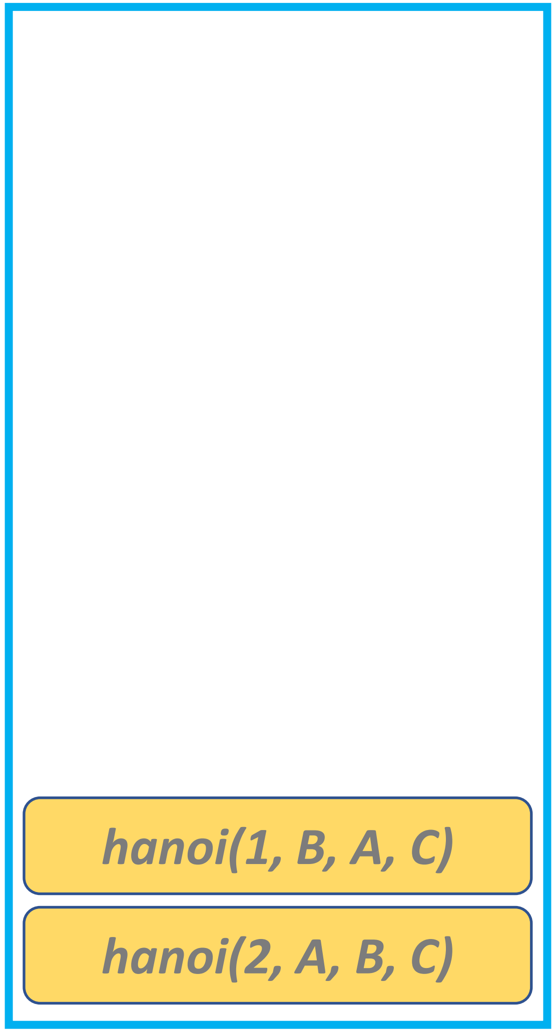
hanoi(2, A, B, C)

STACK

Towers of Hanoi

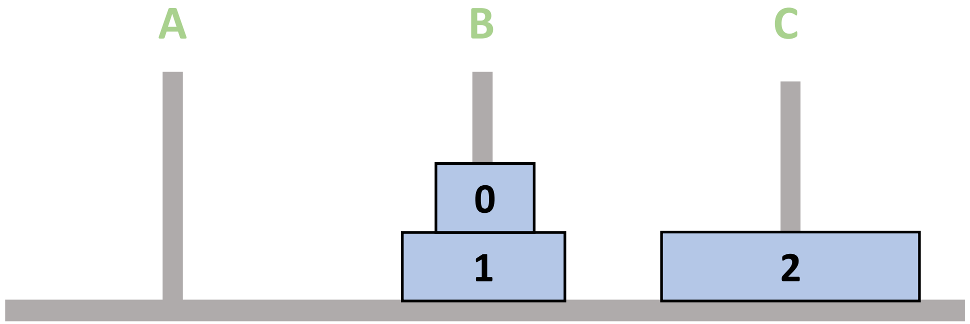


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

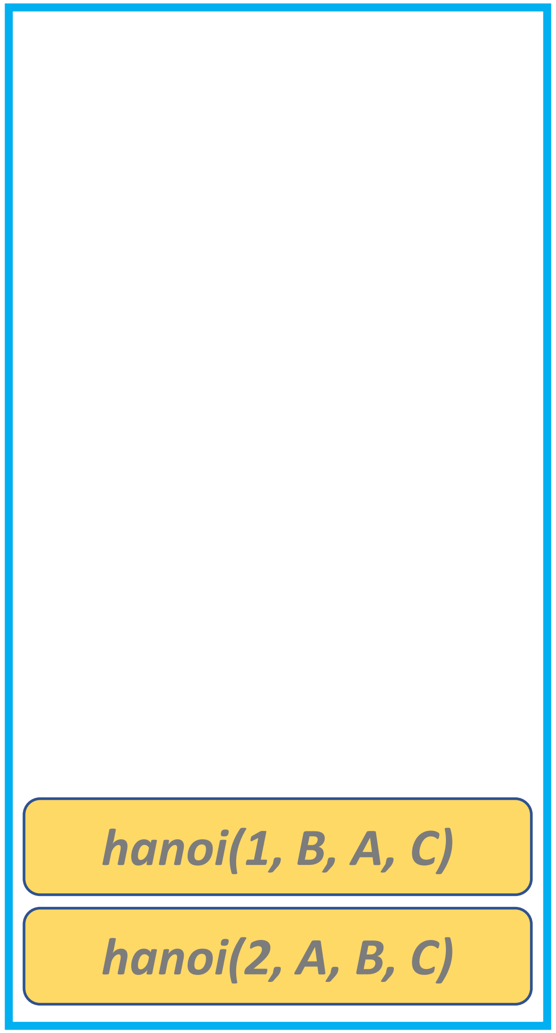


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

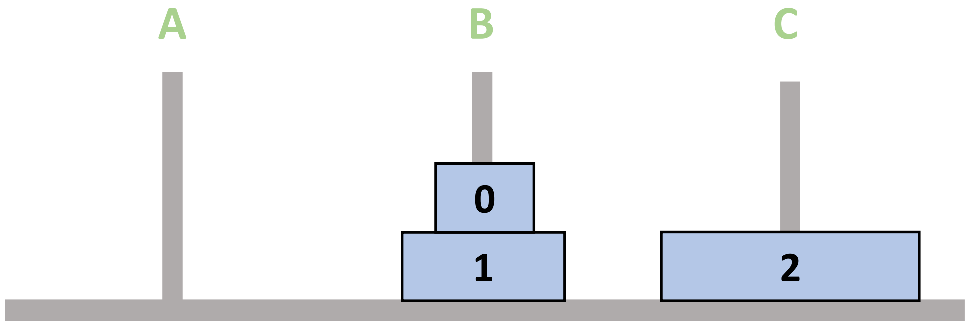


hanoi(1, B, A, C)

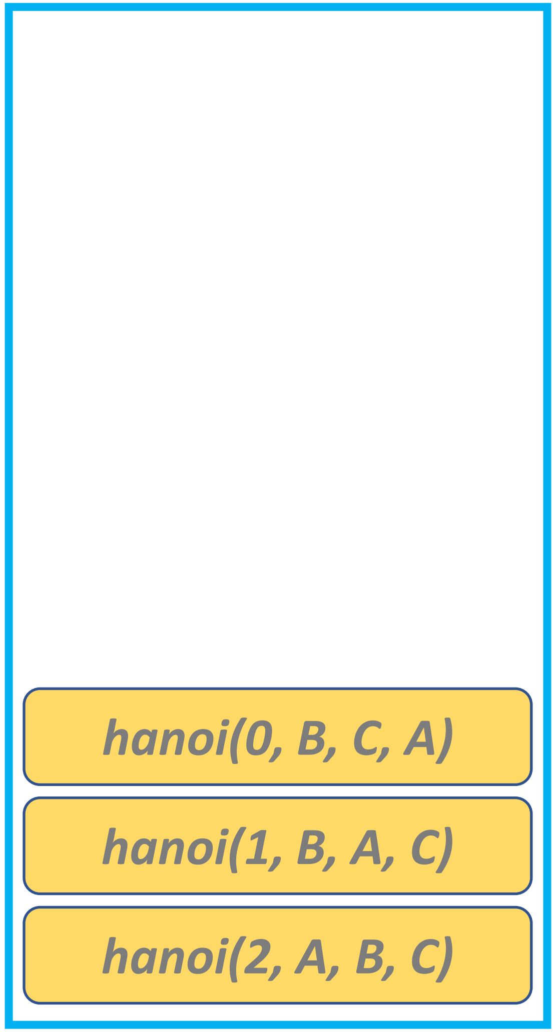
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```



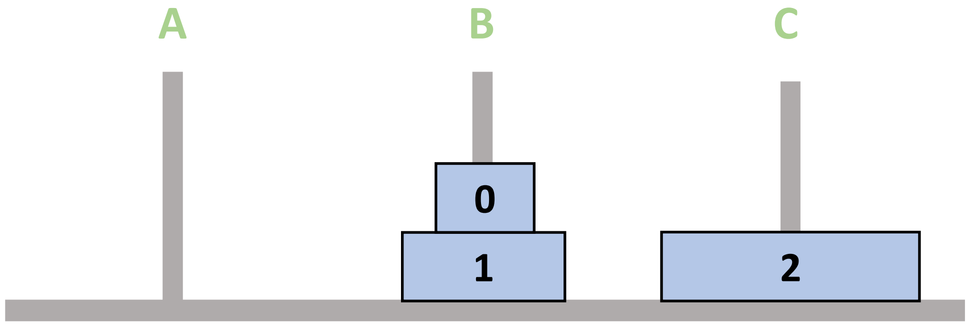
hanoi(0, B, C, A)

hanoi(1, B, A, C)

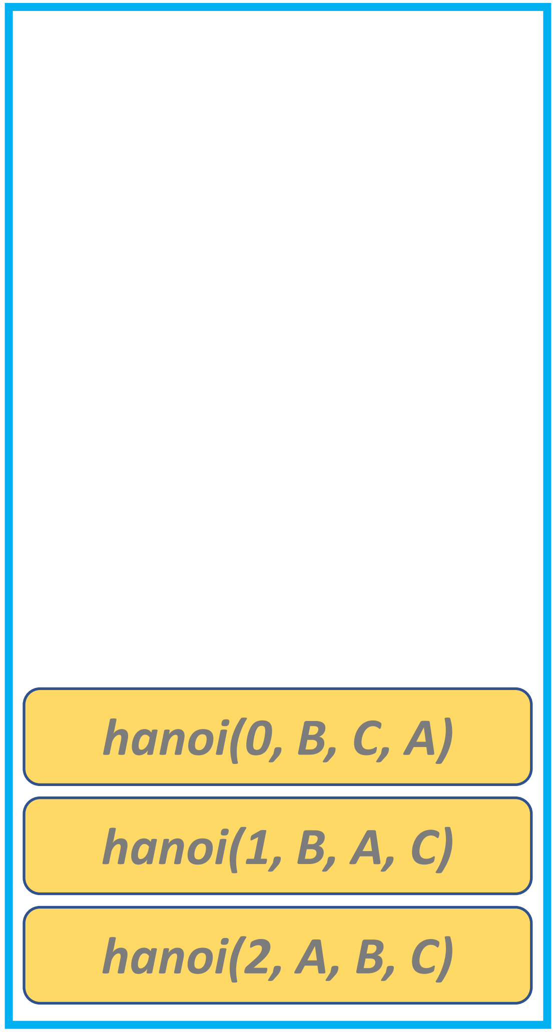
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```



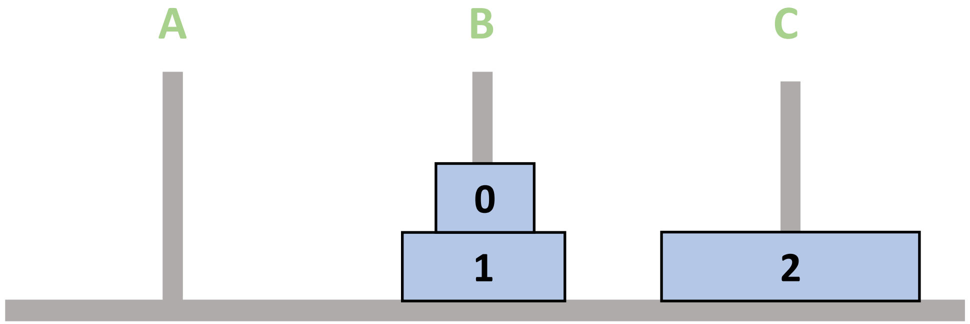
hanoi(0, B, C, A)

hanoi(1, B, A, C)

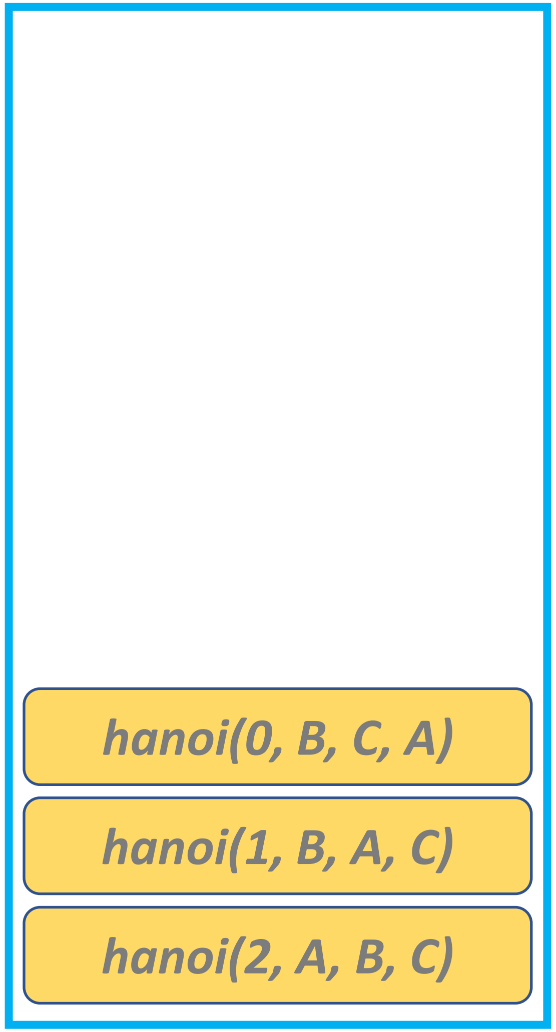
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```



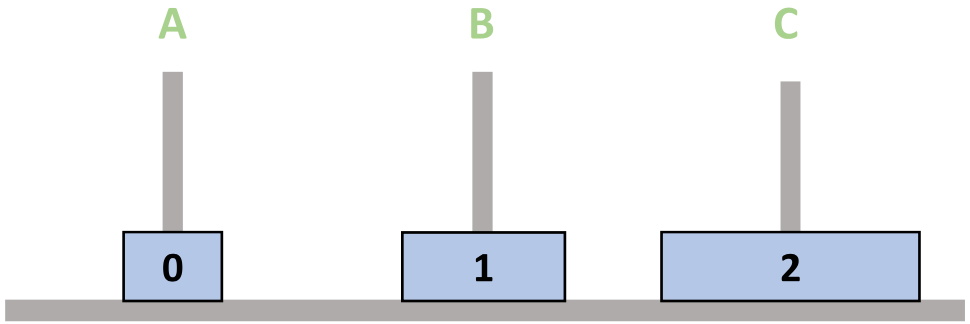
hanoi(0, B, C, A)

hanoi(1, B, A, C)

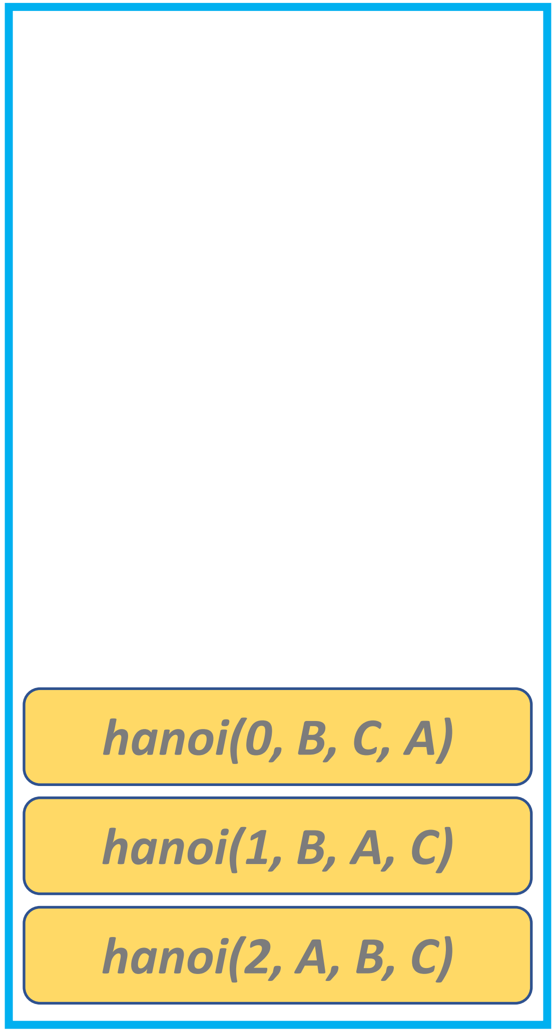
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



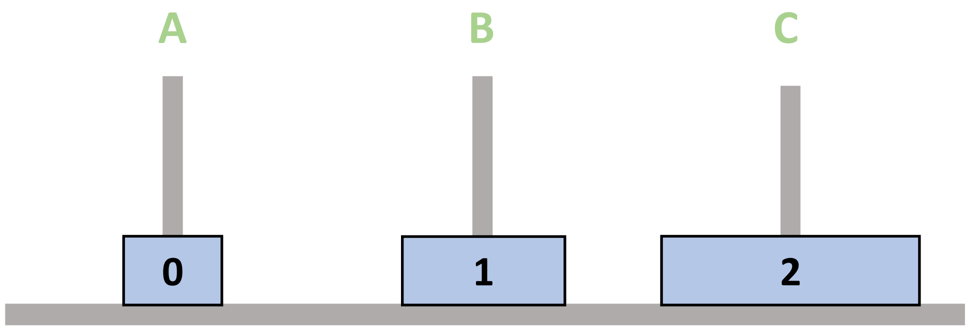
hanoi(0, B, C, A)

hanoi(1, B, A, C)

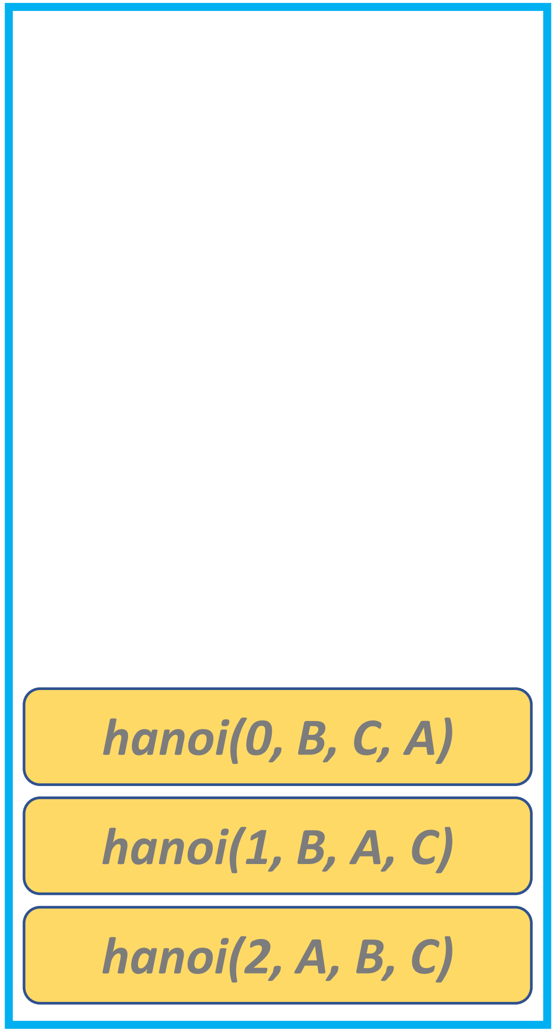
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

          if n==0:
             move disk from source to dest
             return

          hanoi(disk-1, source, dest, middle)
          move disk from source to dest
          hanoi(disk-1, middle, source, dest)
```



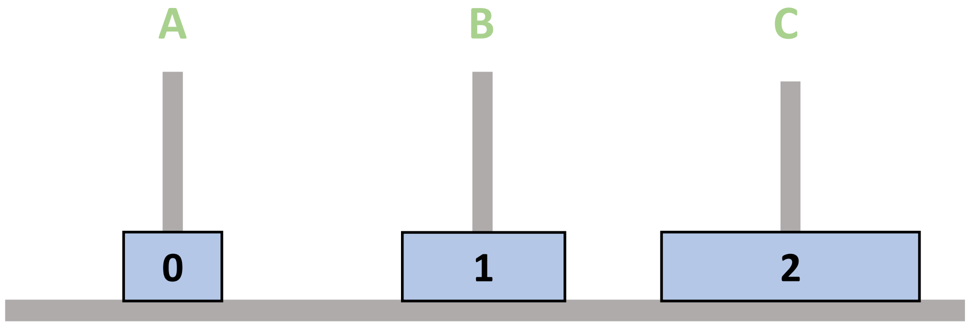
hanoi(0, B, C, A)

hanoi(1, B, A, C)

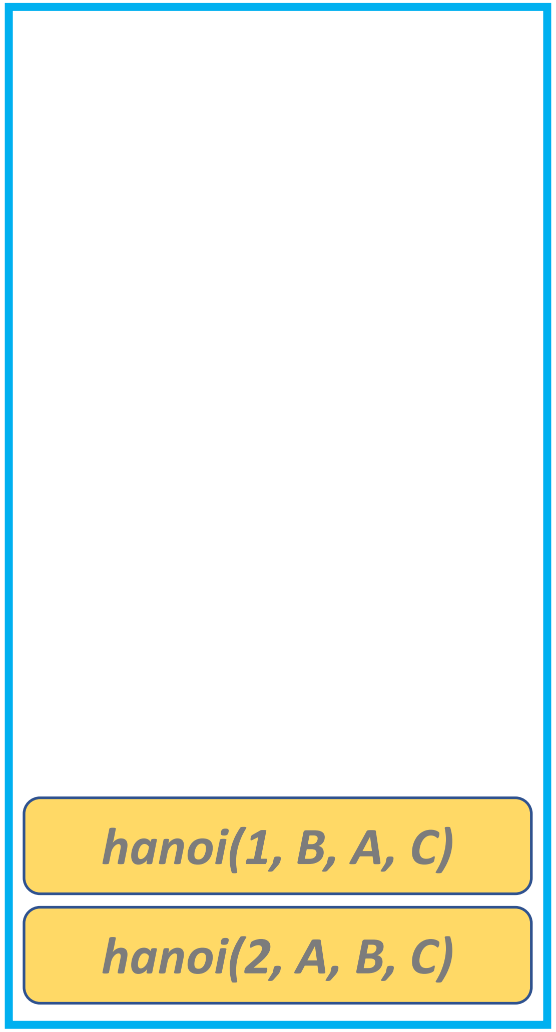
hanoi(2, A, B, C)

STACK

Towers of Hanoi

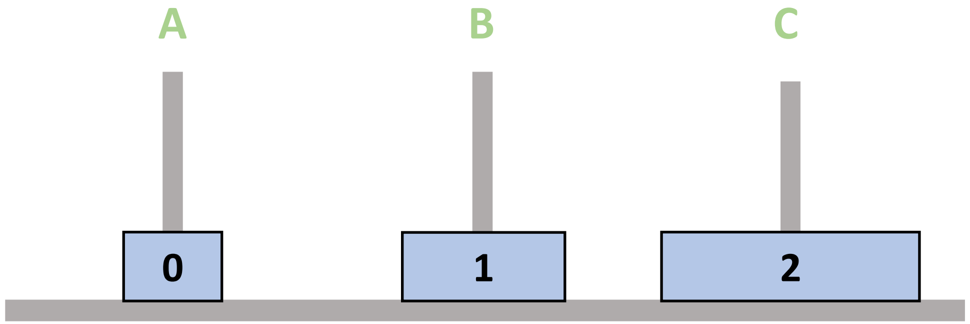


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

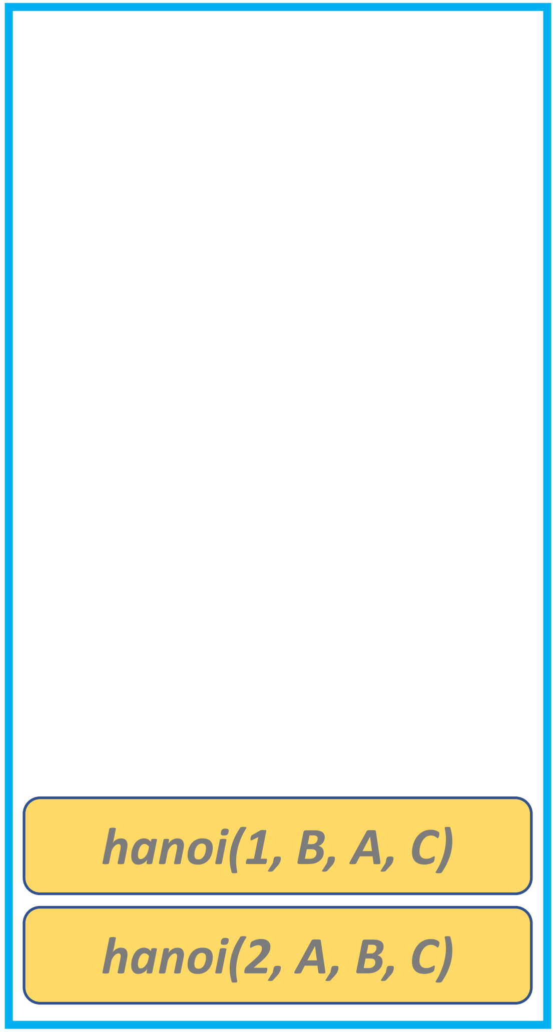


STACK

Towers of Hanoi



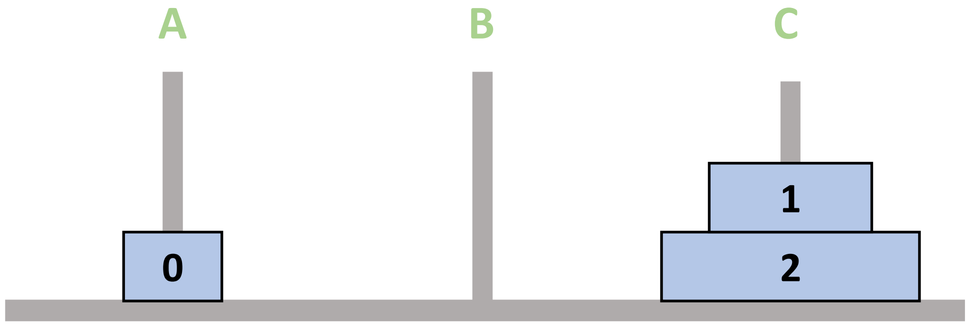
```
def hanoi(disk, source, middle, dest):

          if n==0:
             move disk from source to dest
             return

          hanoi(disk-1, source, dest, middle)
          move disk from source to dest
          hanoi(disk-1, middle, source, dest)
```


hanoi(1, B, A, C)

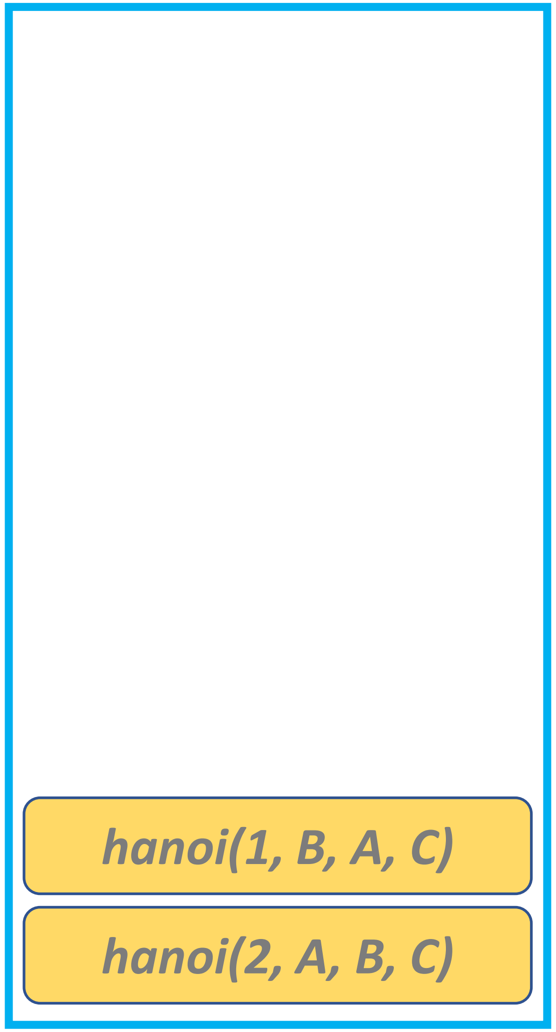
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

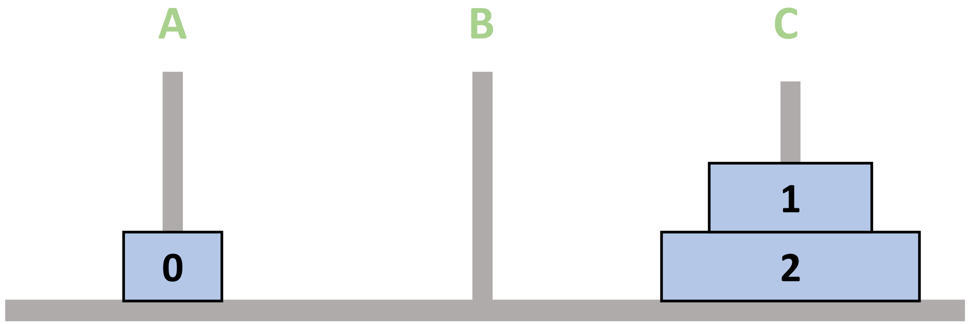


hanoi(1, B, A, C)

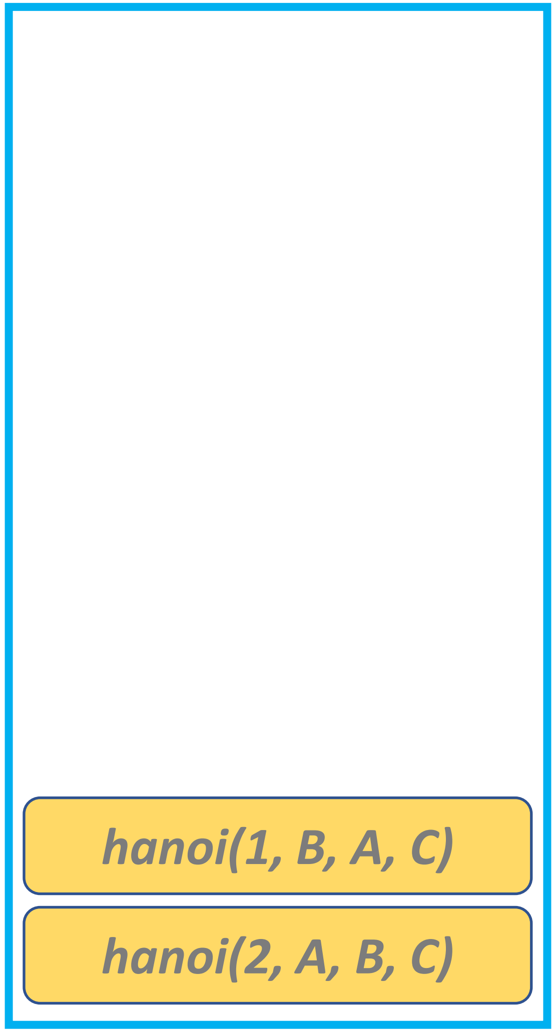
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

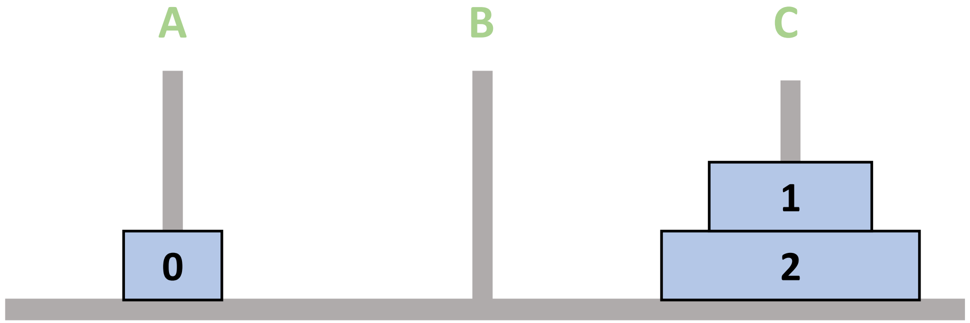


hanoi(1, B, A, C)

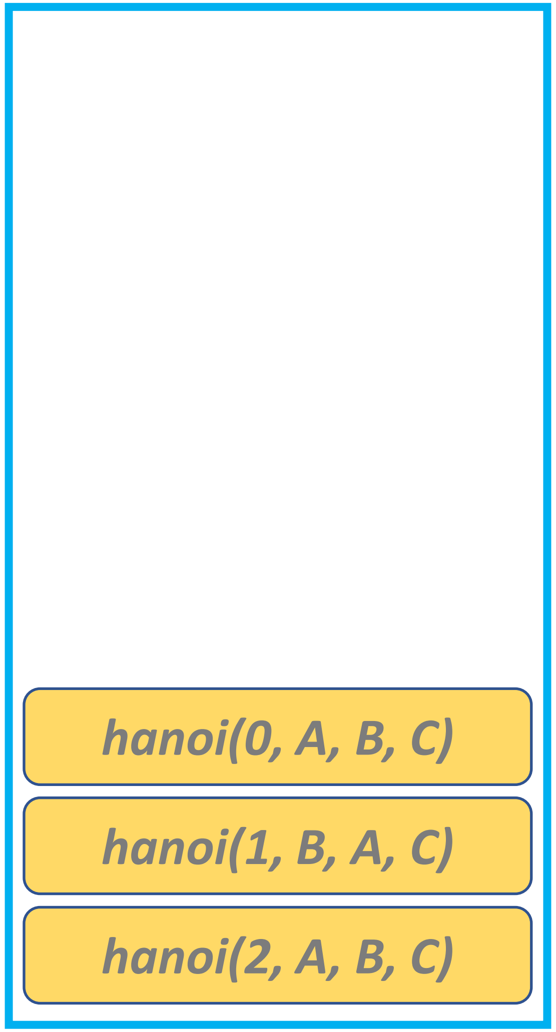
hanoi(2, A, B, C)

STACK

Towers of Hanoi

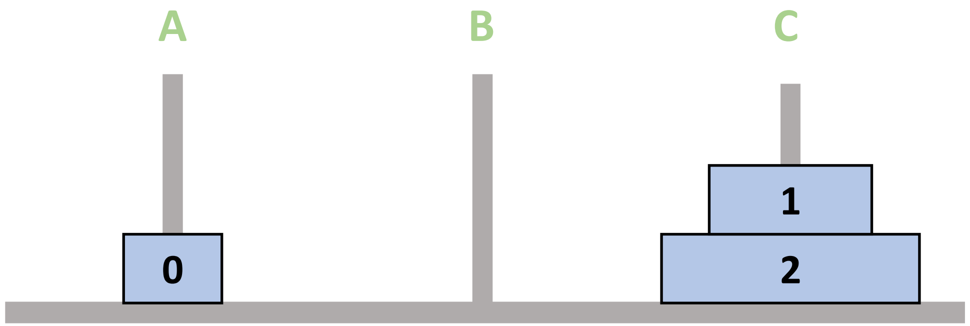


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

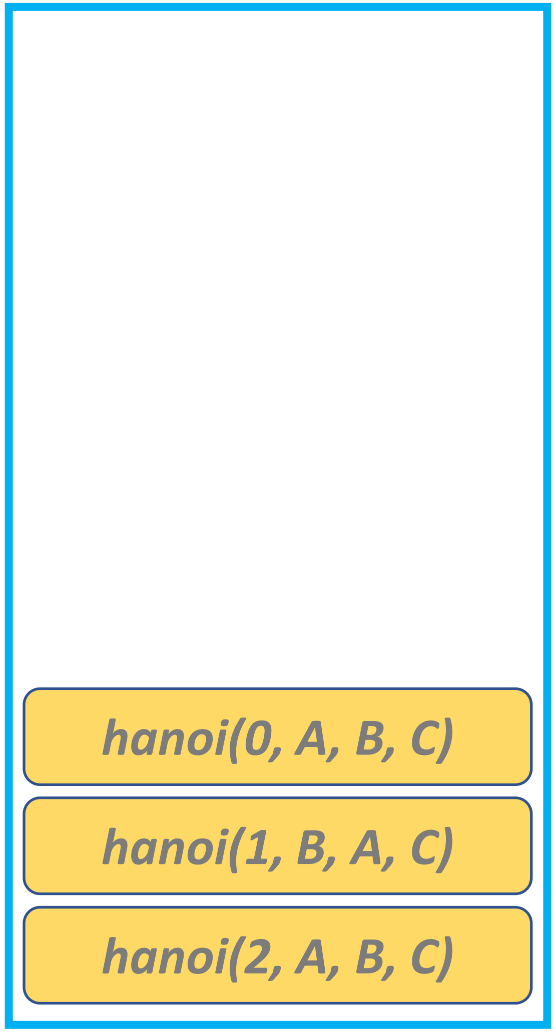


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



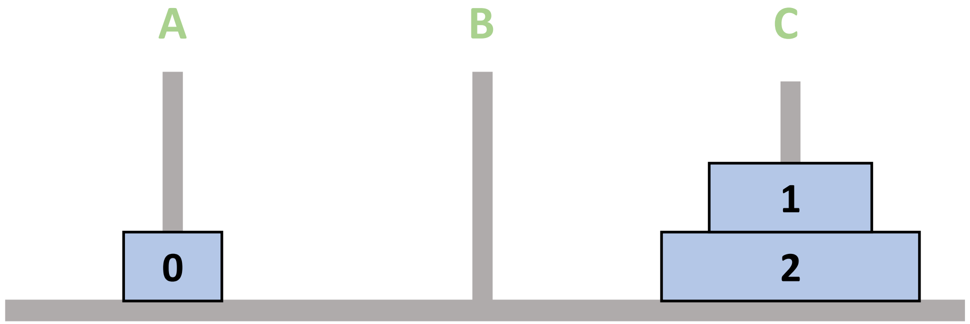
hanoi(0, A, B, C)

hanoi(1, B, A, C)

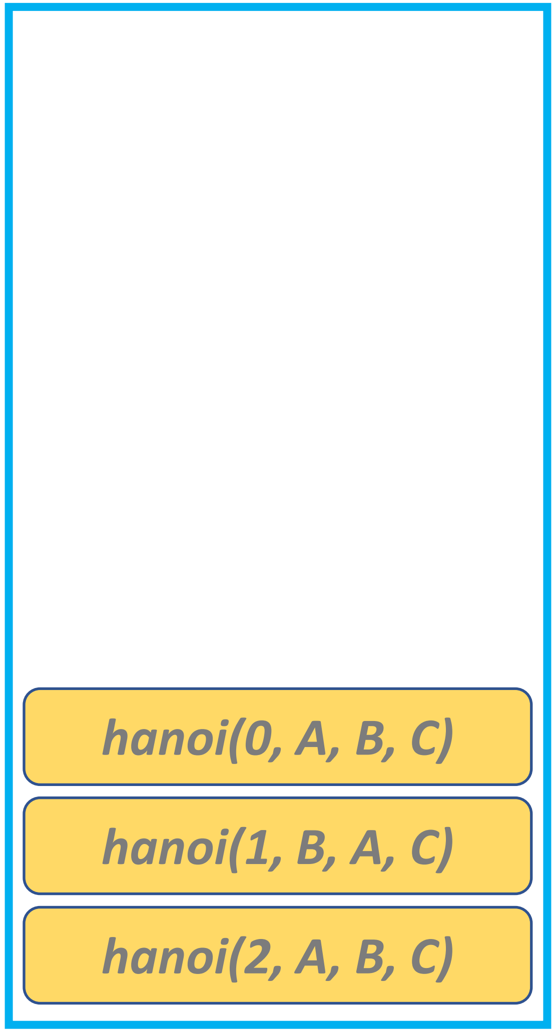
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



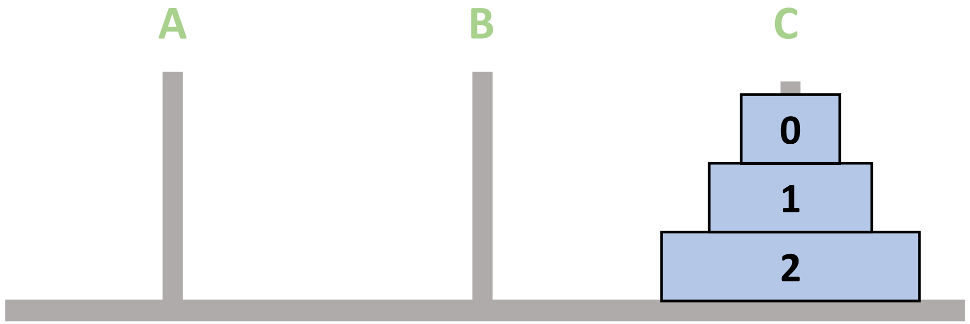
hanoi(0, A, B, C)

hanoi(1, B, A, C)

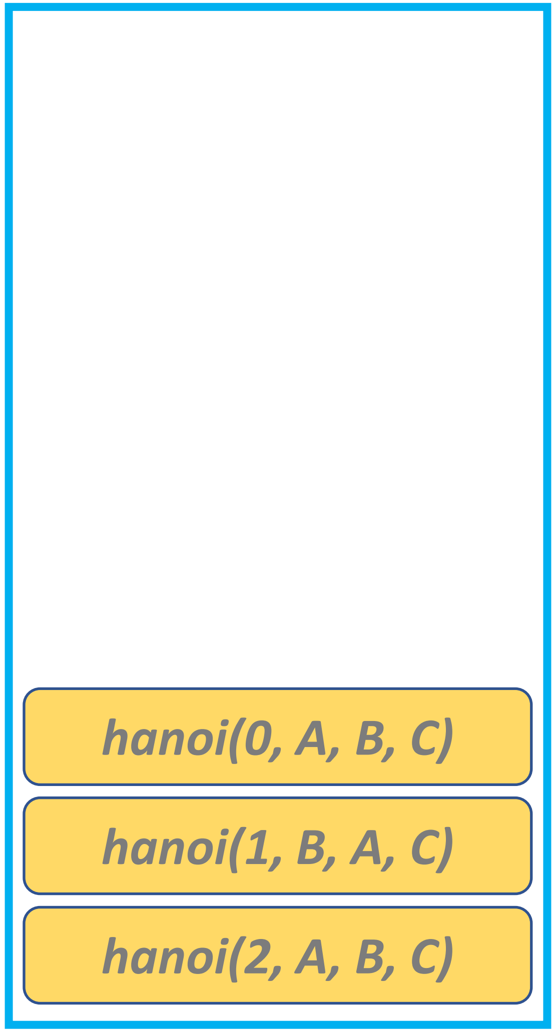
hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



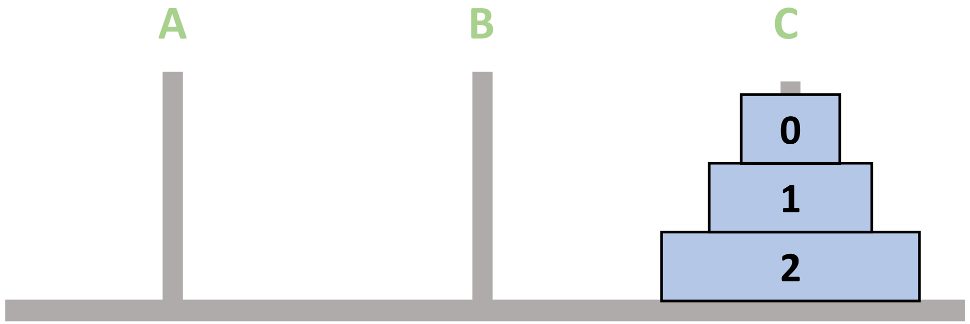
hanoi(0, A, B, C)

hanoi(1, B, A, C)

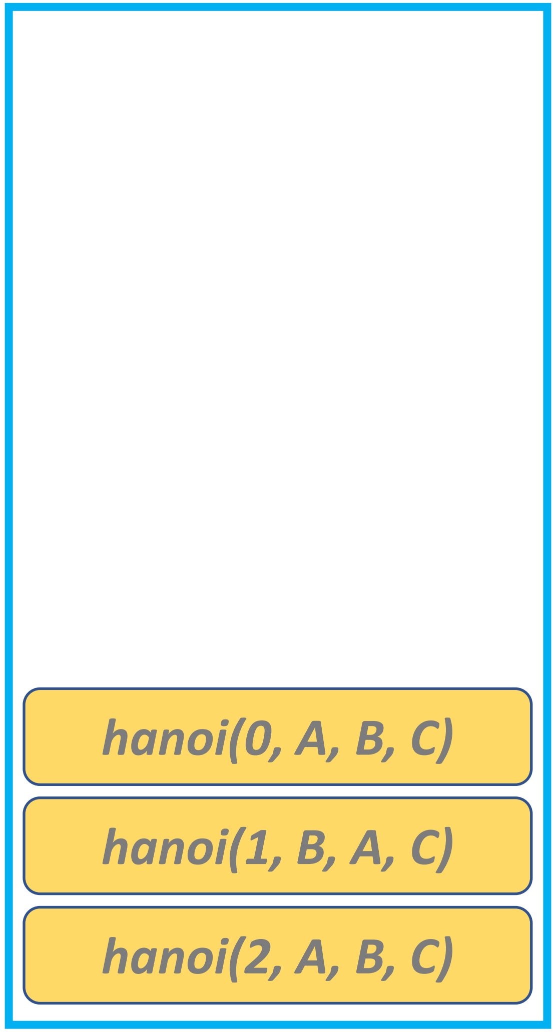
hanoi(2, A, B, C)

STACK

Towers of Hanoi

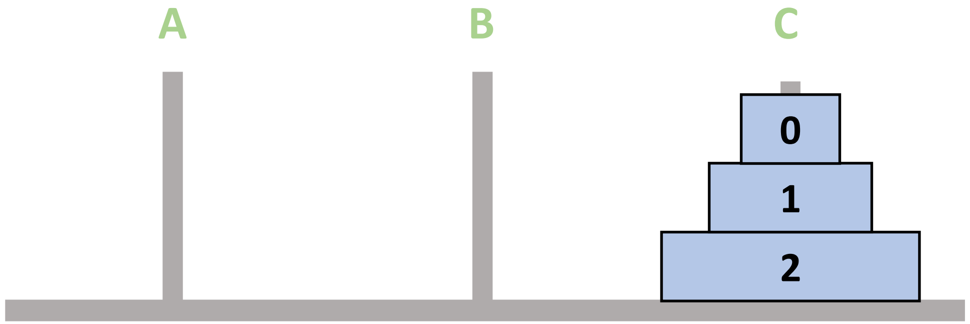


```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

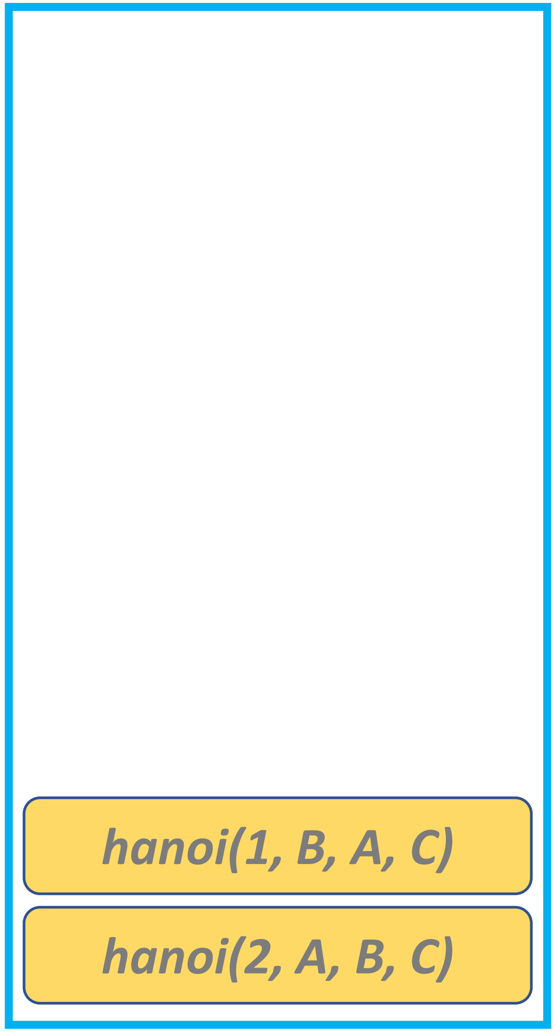


STACK

Towers of Hanoi

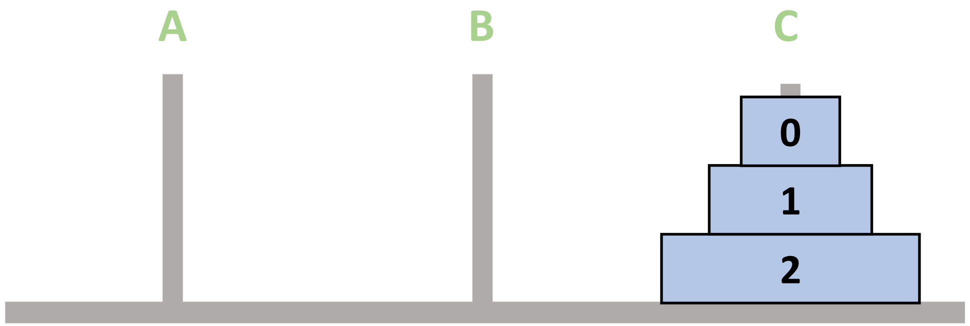


```
def hanoi(disk, source, middle, dest):

    if n==0:
        move disk from source to dest
        return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

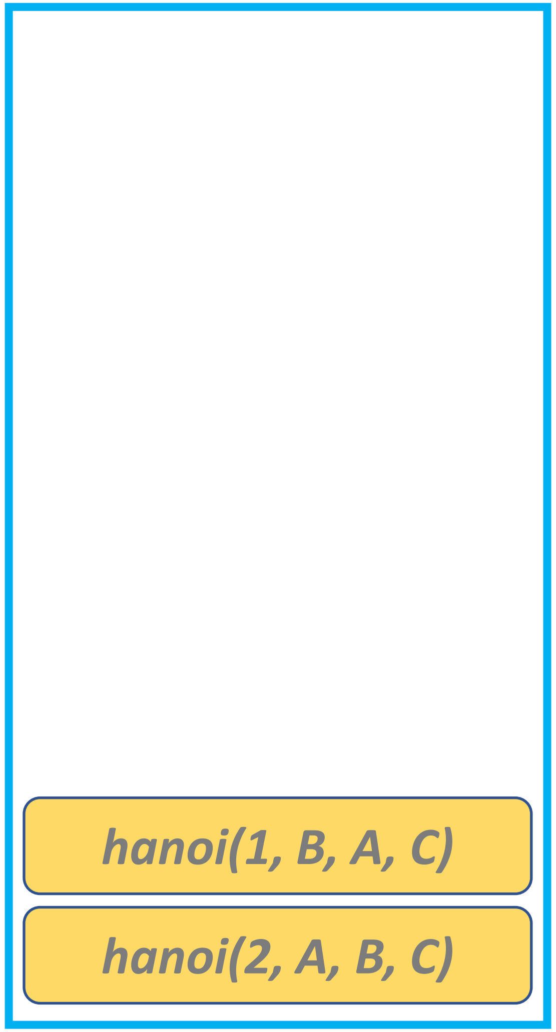


STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```

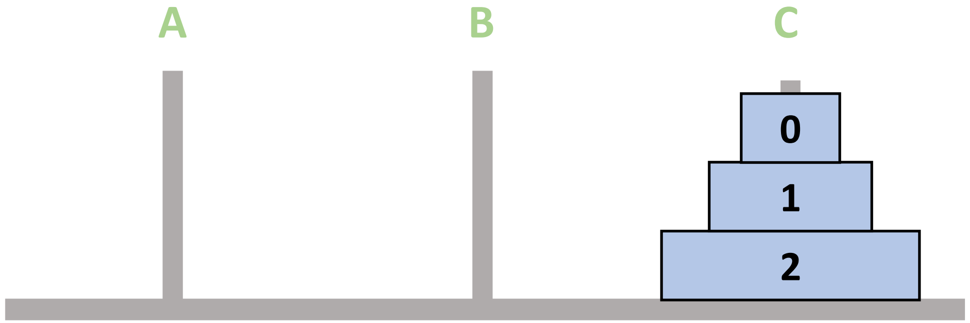


hanoi(1, B, A, C)

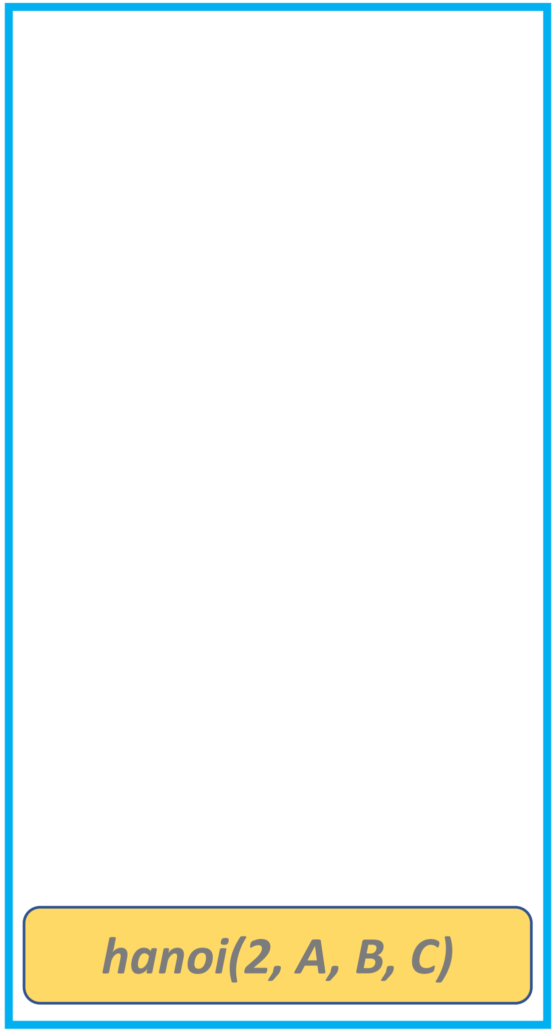
hanoi(2, A, B, C)

STACK

Towers of Hanoi



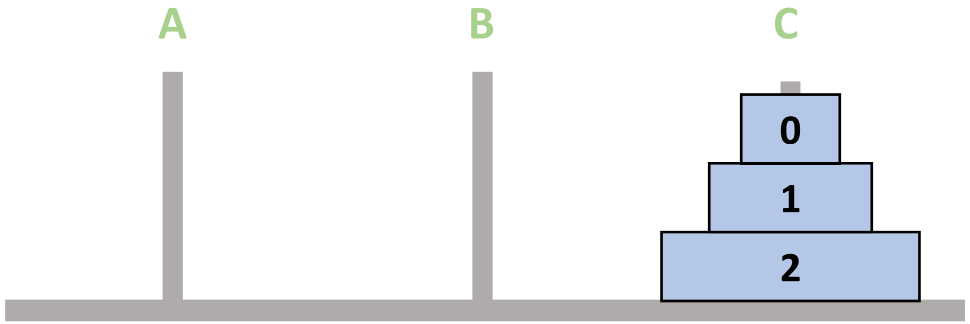
```
def hanoi(disk, source, middle, dest):

        if n==0:
            move disk from source to dest
            return

        hanoi(disk-1, source, dest, middle)
        move disk from source to dest
        hanoi(disk-1, middle, source, dest)
```



hanoi(2, A, B, C)

STACK

Towers of Hanoi



```
def hanoi(disk, source, middle, dest):

    if n==0:
      move disk from source to dest
      return

    hanoi(disk-1, source, dest, middle)
    move disk from source to dest
    hanoi(disk-1, middle, source, dest)
```

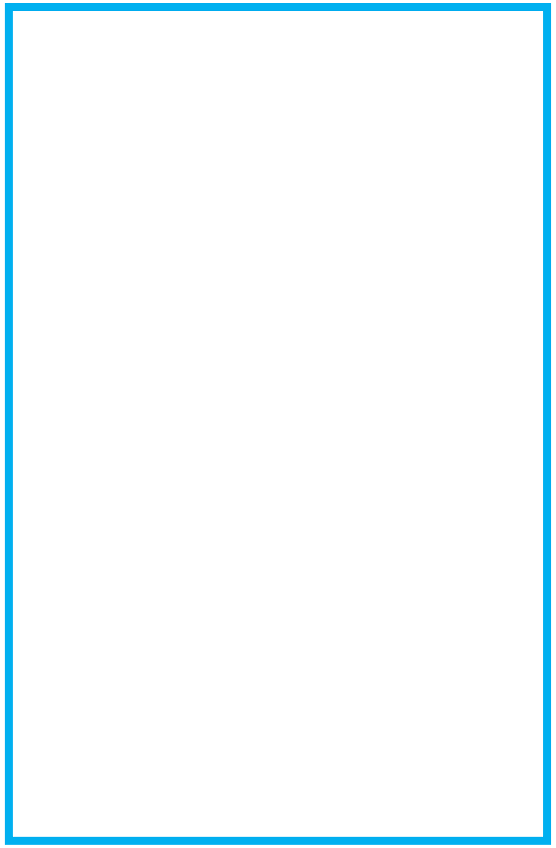
STACK

Factorial Function

(Recursion Visualization)

Factorial Function

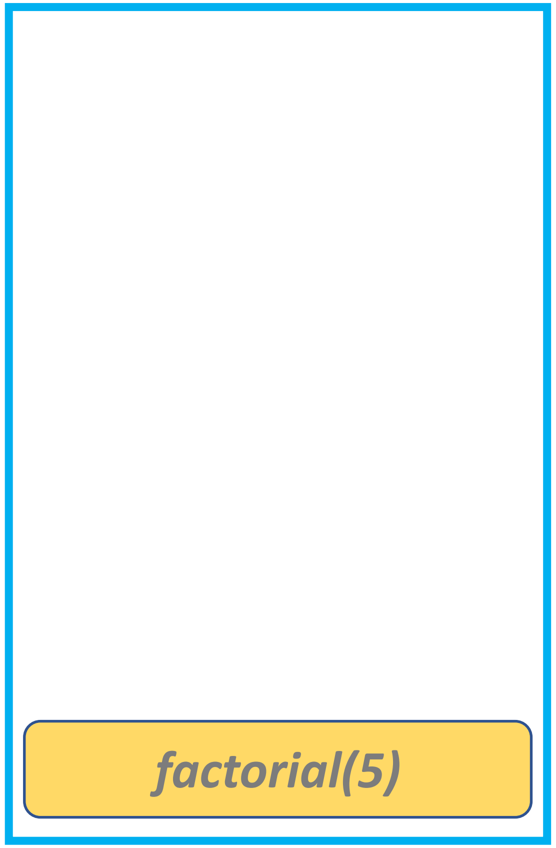
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

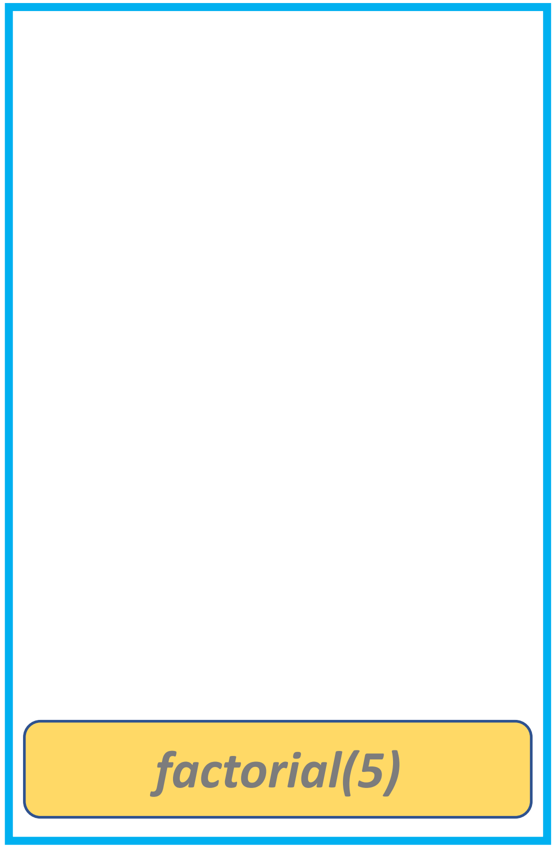
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

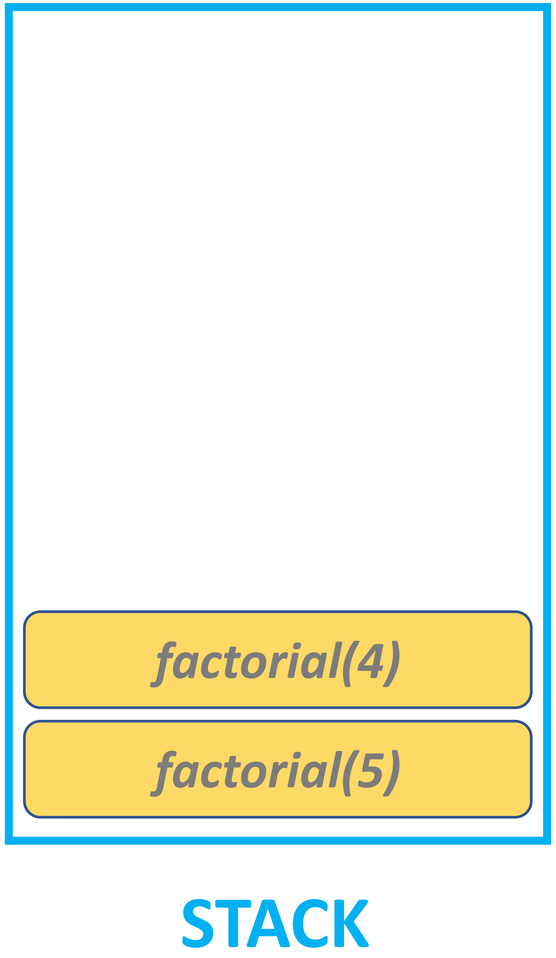
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

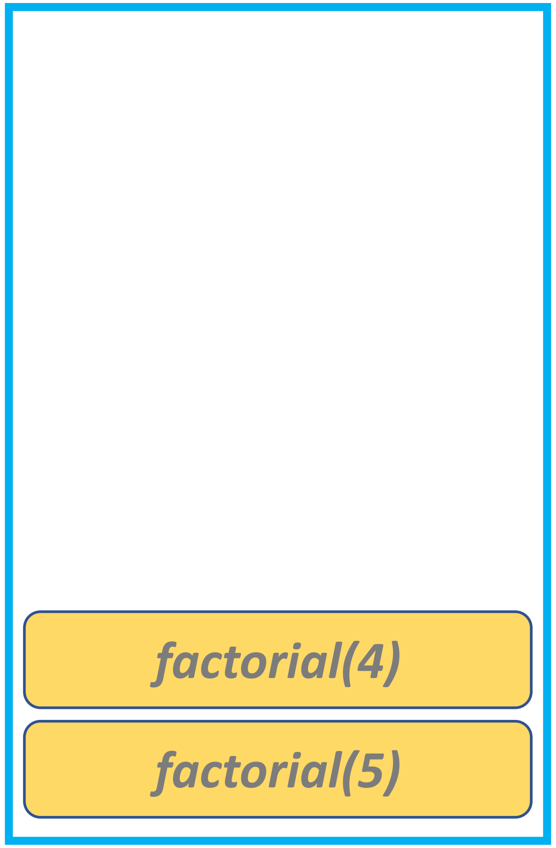
Factorial Function

```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



Factorial Function

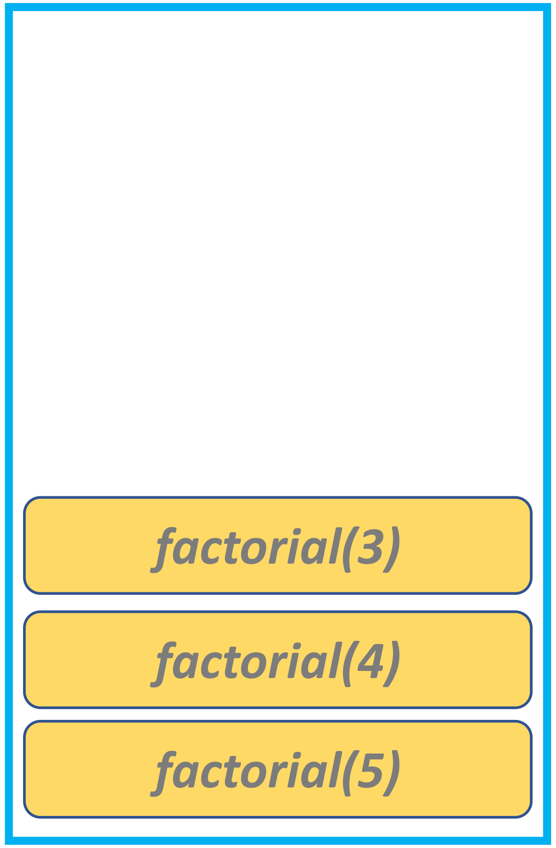
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

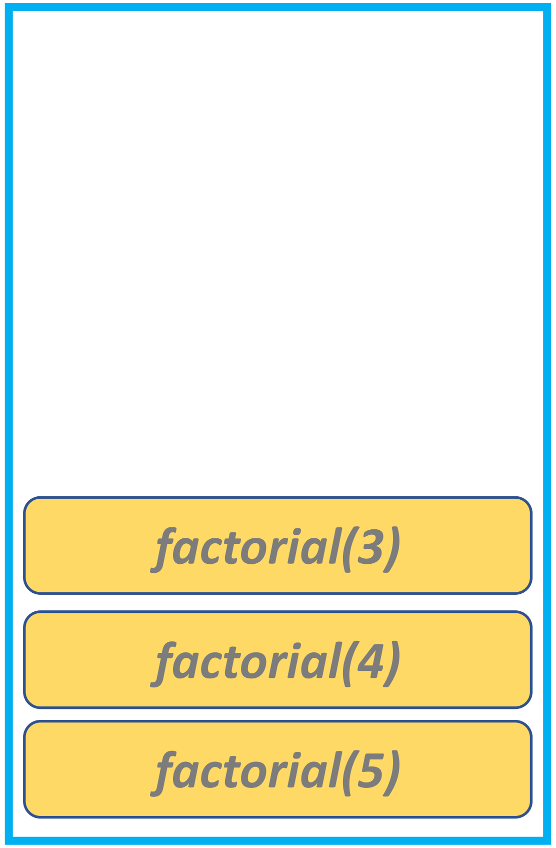
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

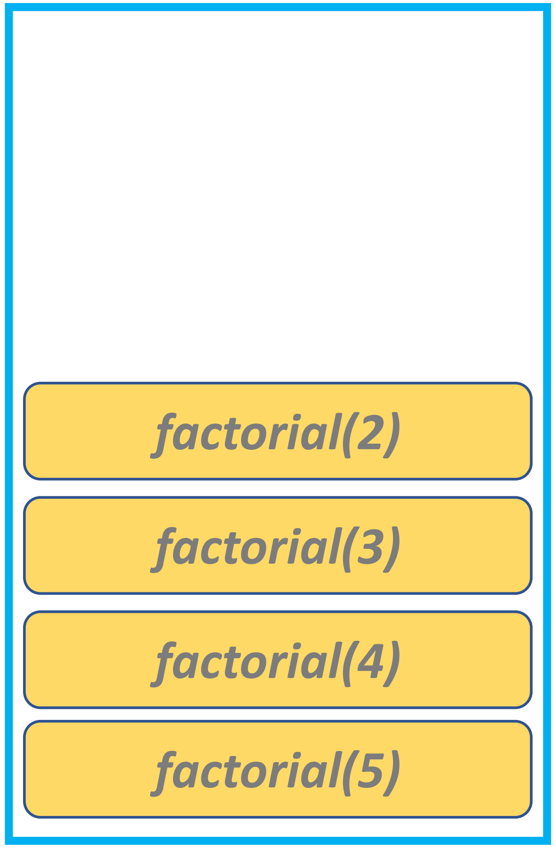
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

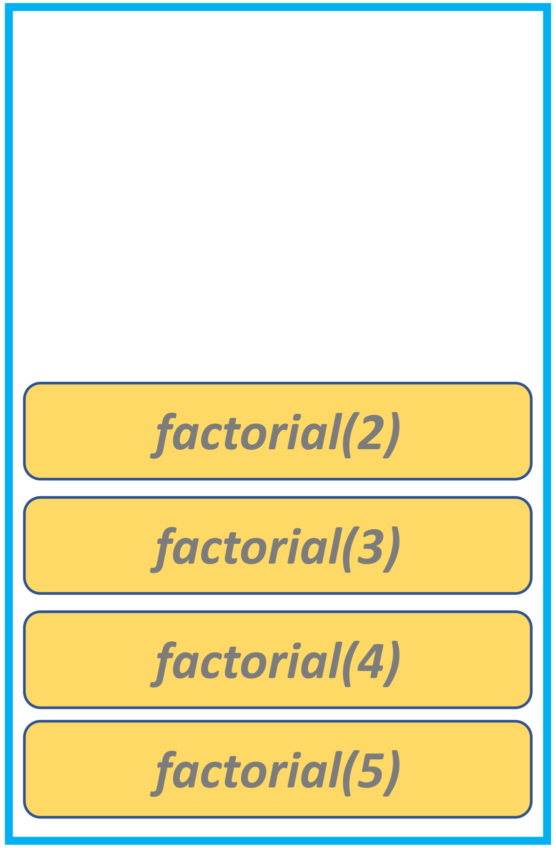
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

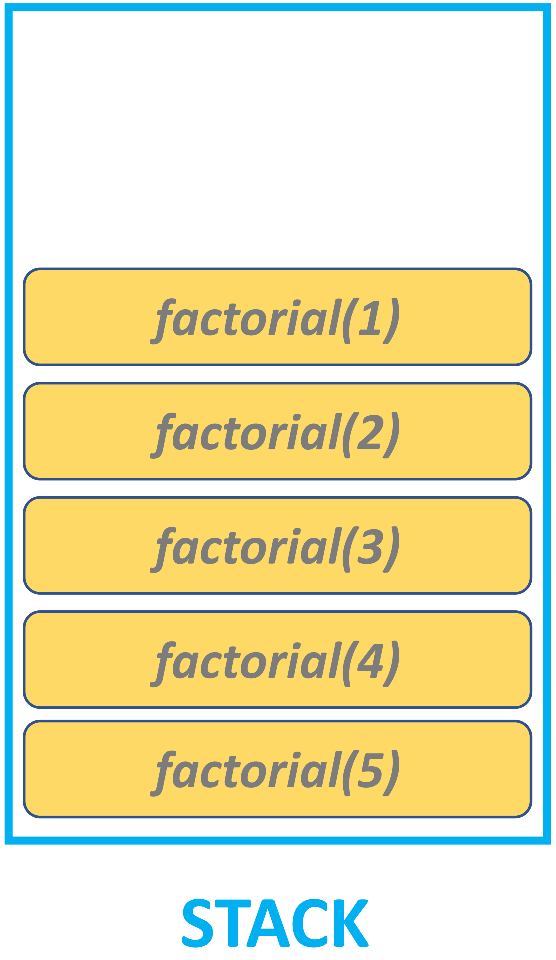
```
def factorial(n):

        if n==1:
          return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



factorial(1)

factorial(2)

factorial(3)

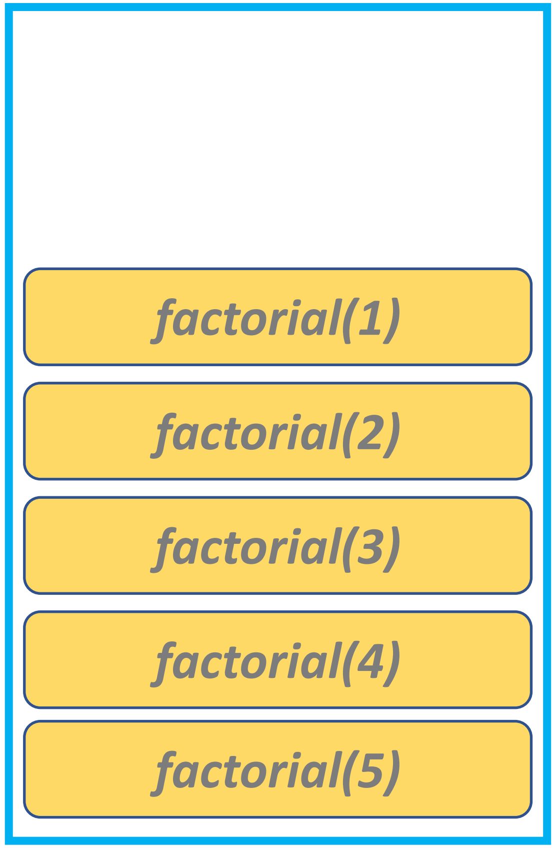
factorial(4)

factorial(5)

STACK

Factorial Function

```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



factorial(1)

factorial(2)

factorial(3)

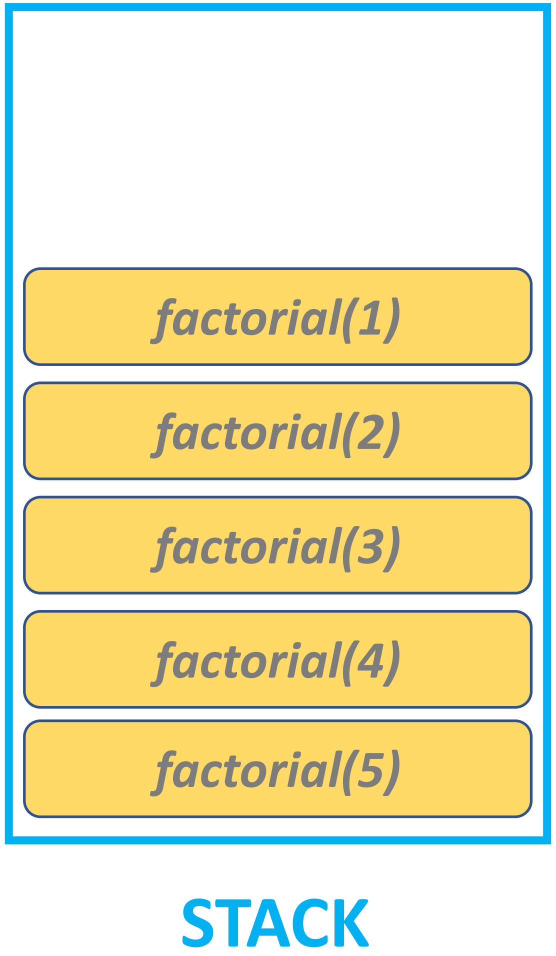
factorial(4)

factorial(5)

STACK

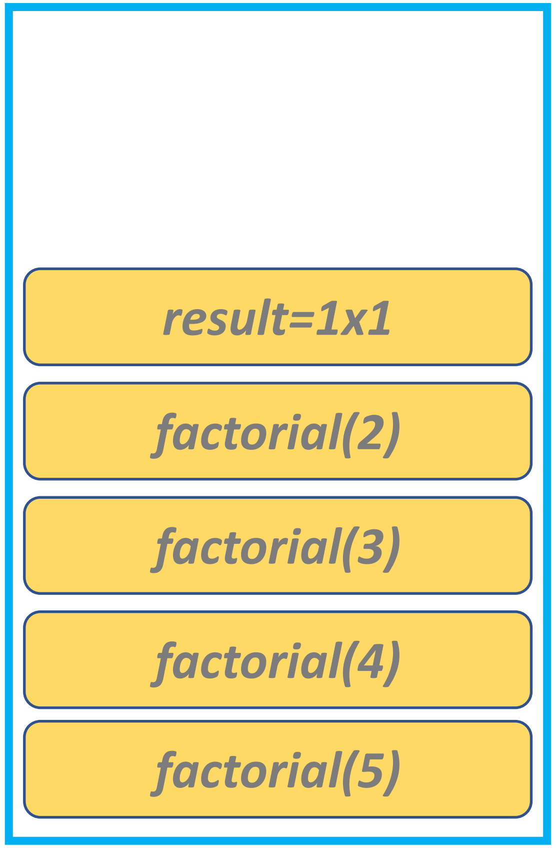
Factorial Function

```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



Factorial Function

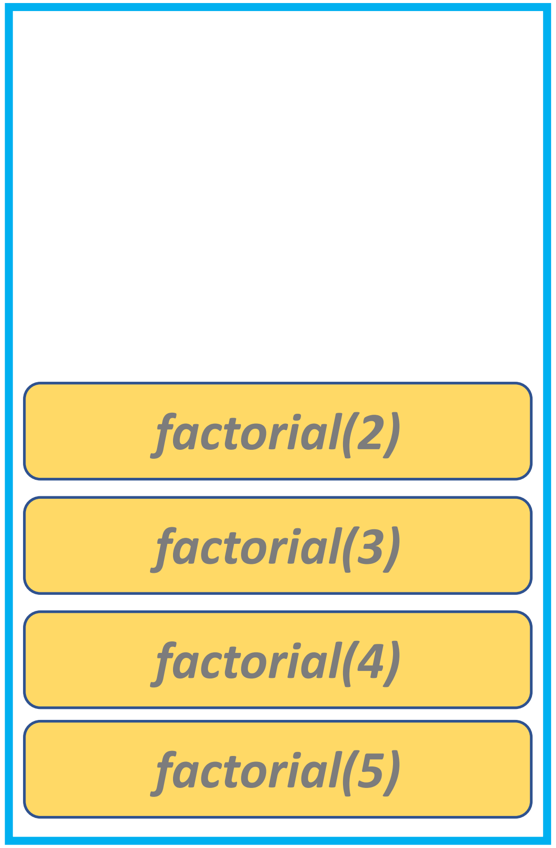
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

```
def factorial(n):

    if n==1:
       return 1

    res = factorial(n-1)
    result = n * res
    return result
```



factorial(2)

factorial(3)

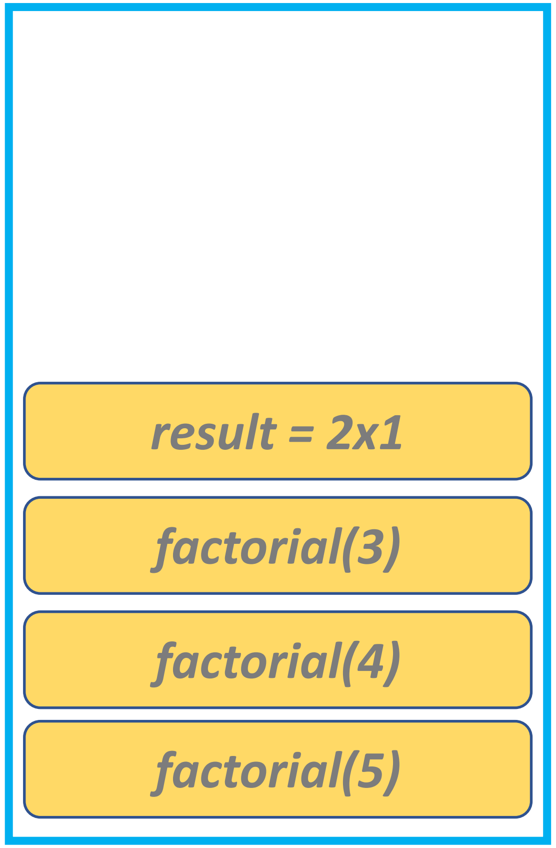
factorial(4)

factorial(5)

STACK

Factorial Function

```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



result = 2x1

factorial(3)

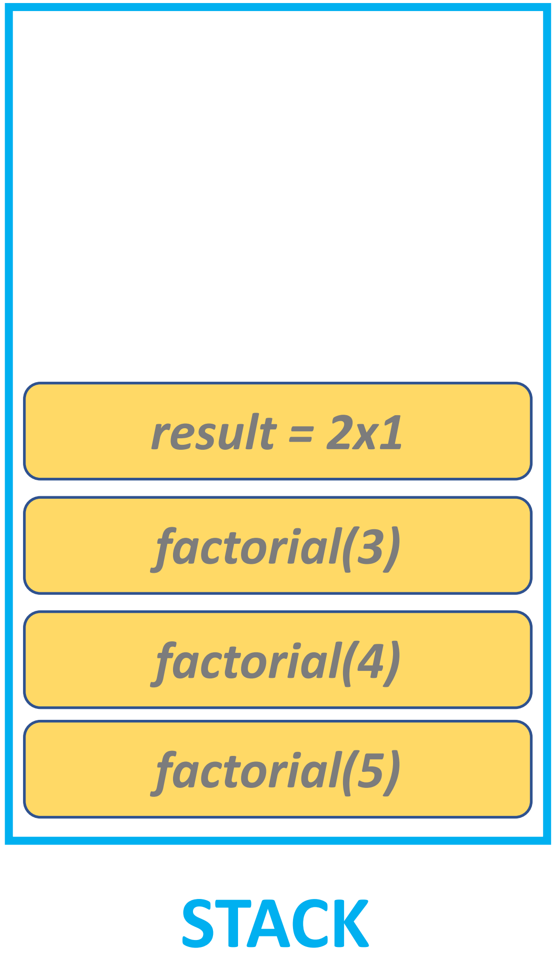
factorial(4)

factorial(5)

STACK

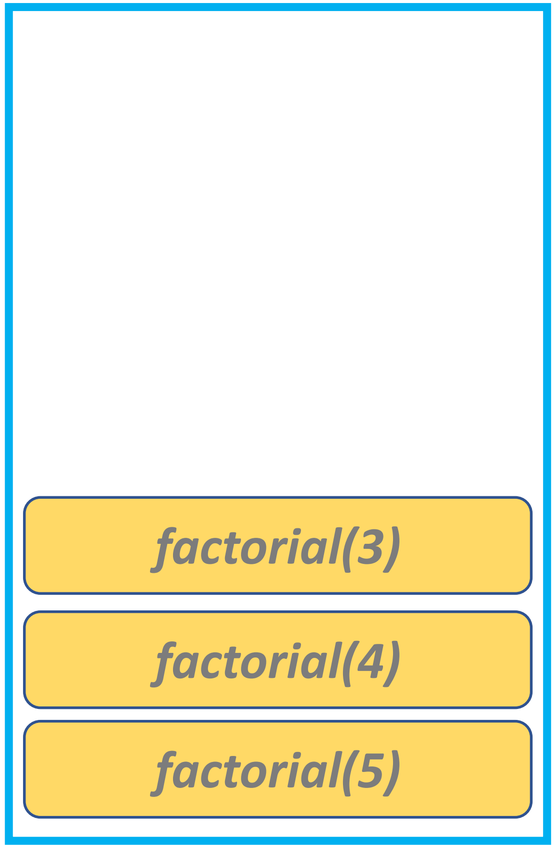
Factorial Function

```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



Factorial Function

```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



factorial(3)

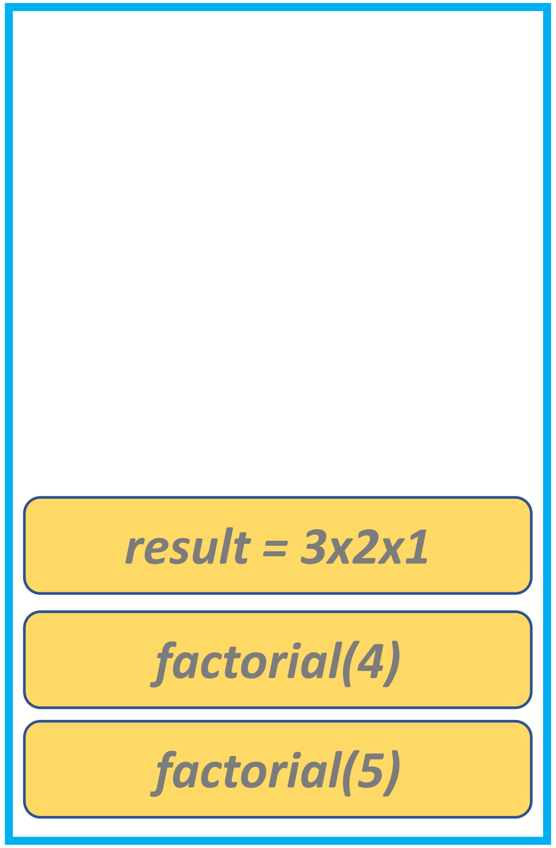
factorial(4)

factorial(5)

STACK

Factorial Function

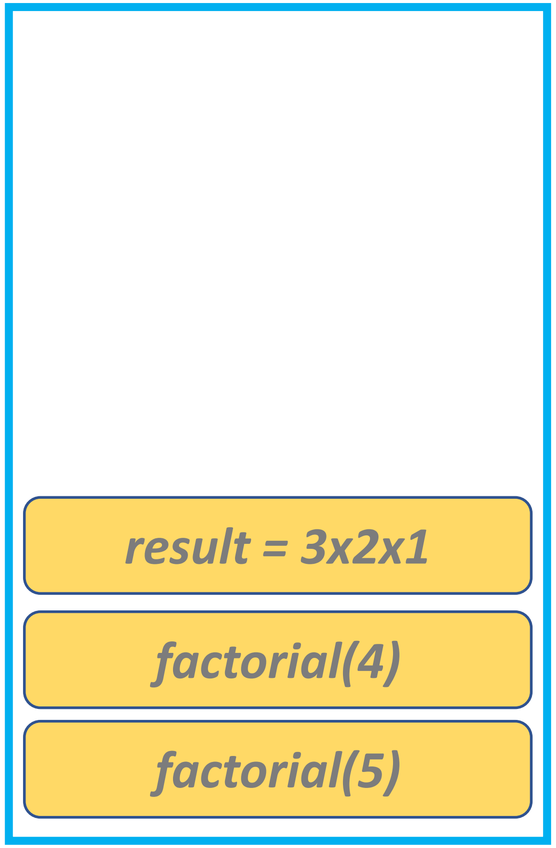
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

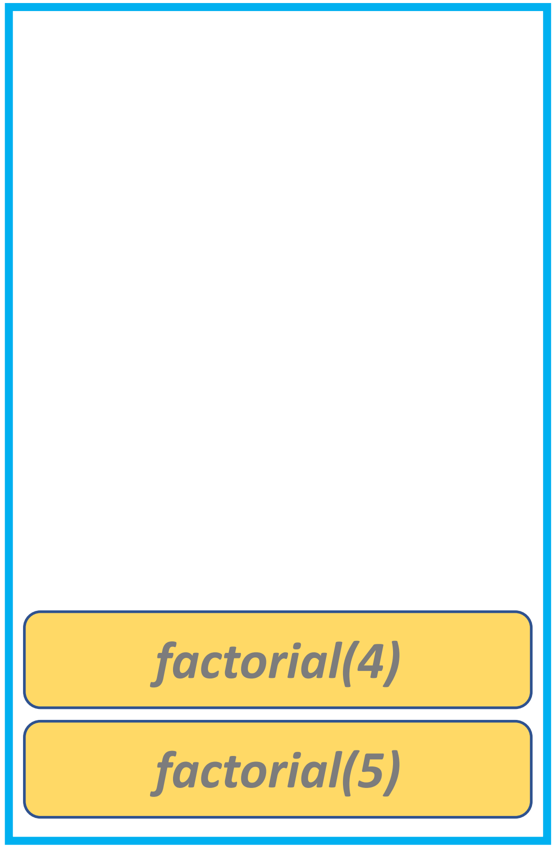
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

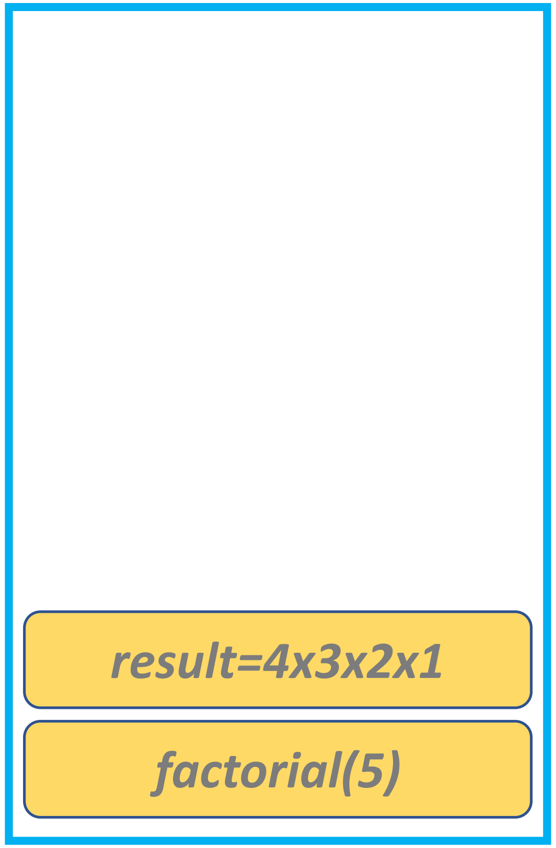
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

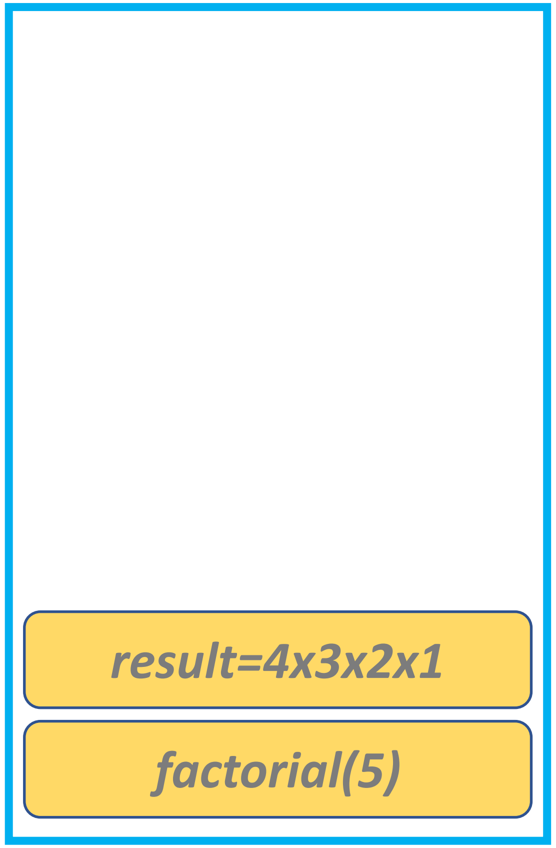
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



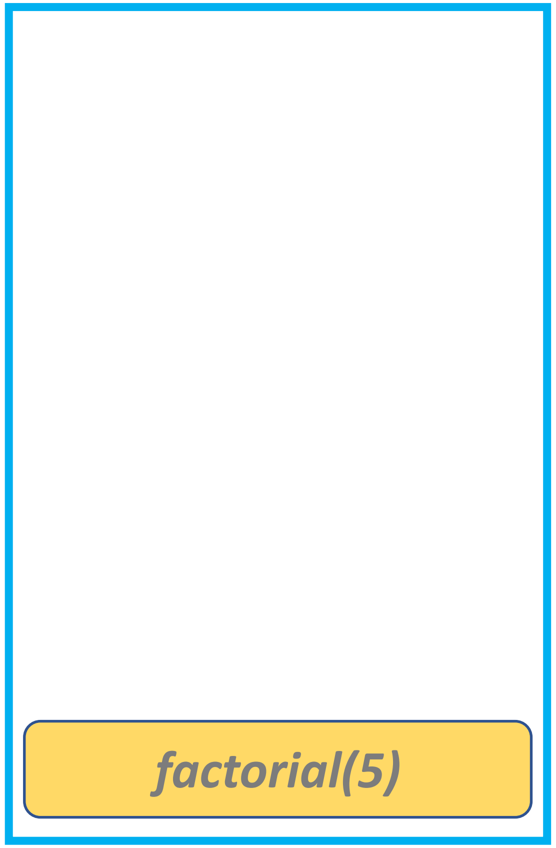
result=4x3x2x1

factorial(5)

STACK

Factorial Function

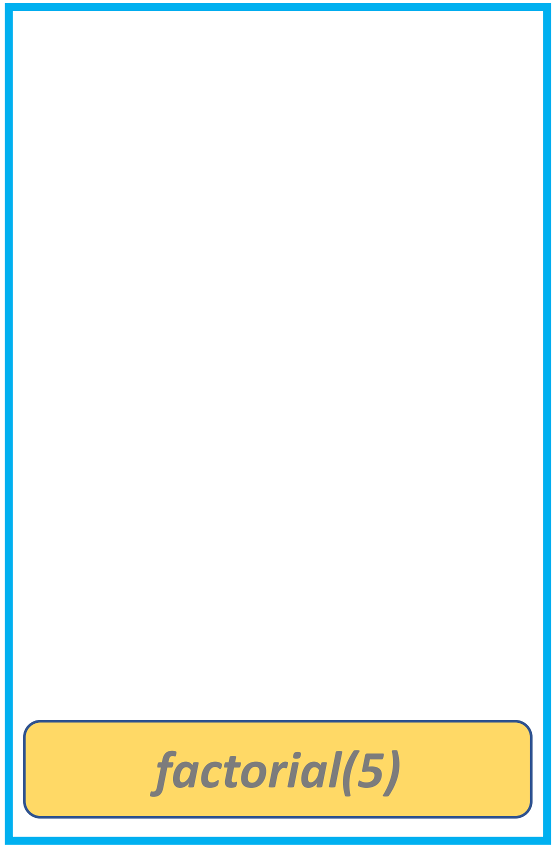
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

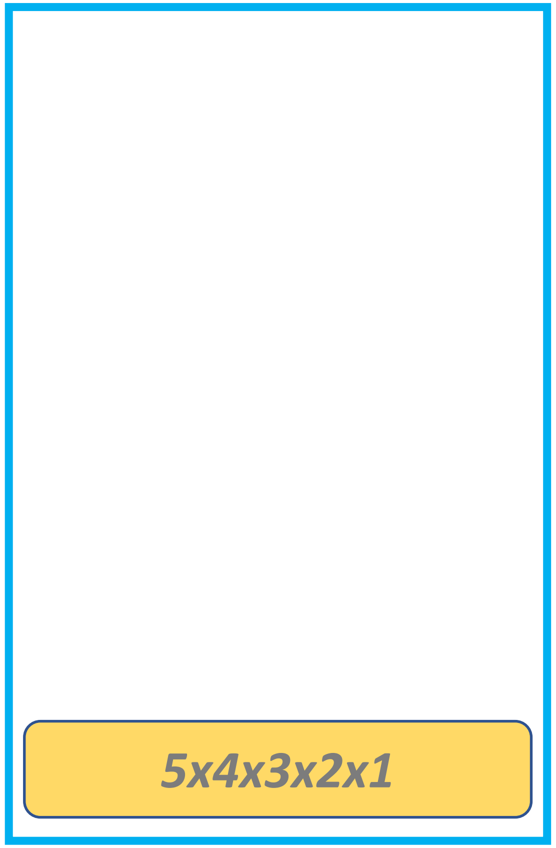
```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



STACK

Factorial Function

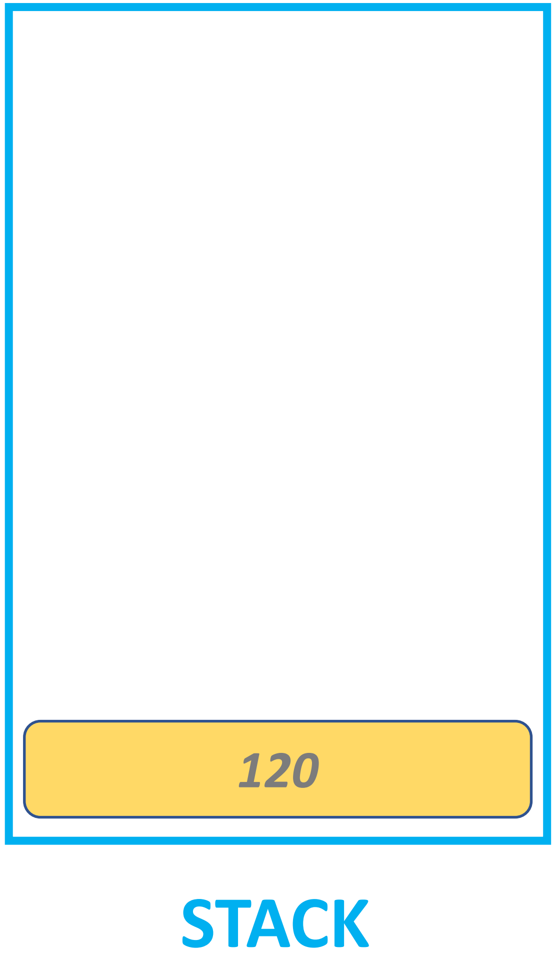
```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```



STACK

Factorial Function

```
def factorial(n):

        if n==1:
            return 1

        res = factorial(n-1)
        result = n * res
        return result
```

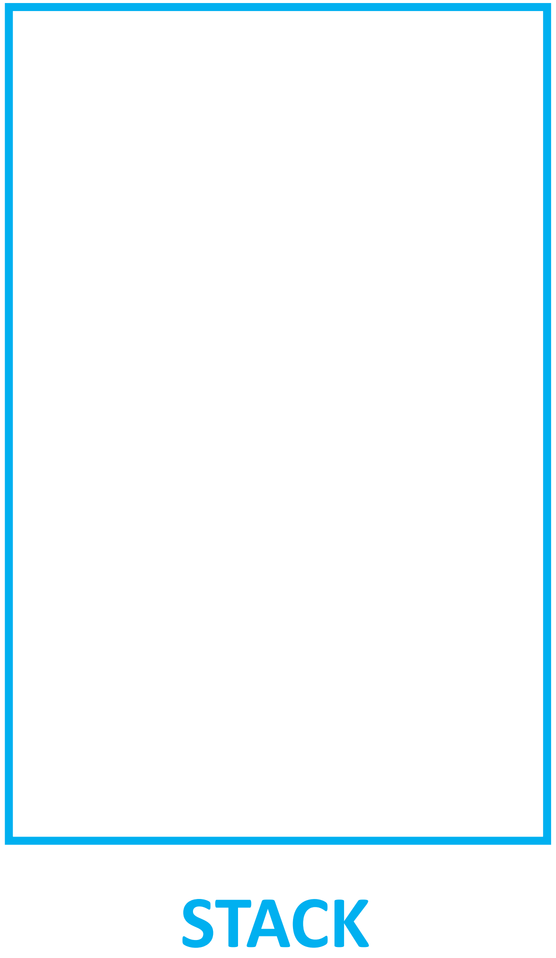


120

STACK

Factorial Function

```
def factorial(n):

    if n==1:
        return 1

    res = factorial(n-1)
    result = n * res
    return result
```



Search Algorithms

(Linear Search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

5	**-4**	**8**	**2**	**1**	**12**	**77**	**43**	**1**	**24**	**-1**	**0**

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

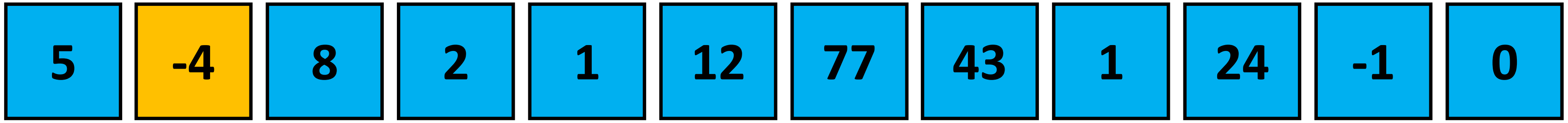
- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

5	**-4**	**8**	**2**	**1**	**12**	**77**	**43**	**1**	**24**	**-1**	**0**

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

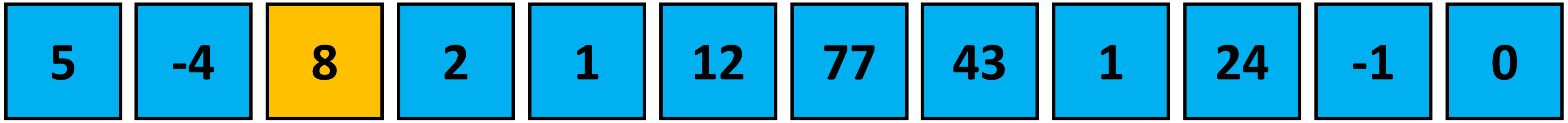


5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

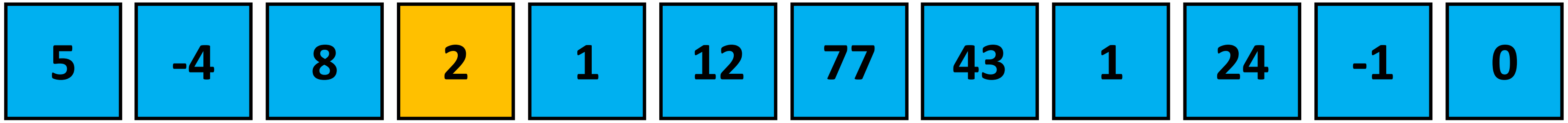


5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

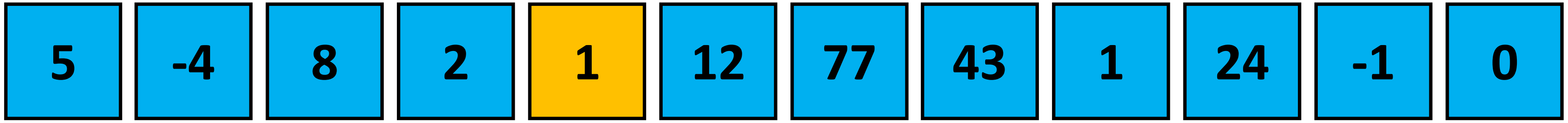


5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

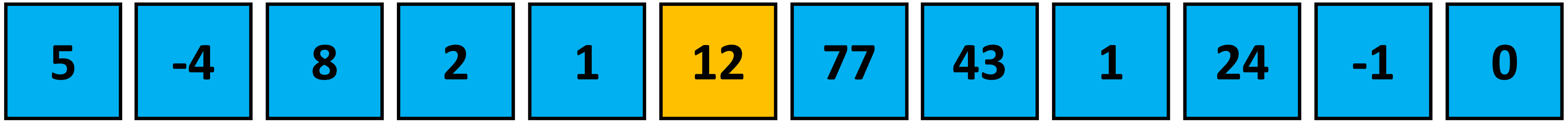


5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

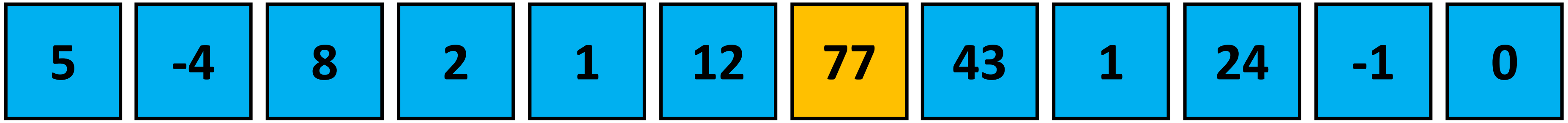


5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

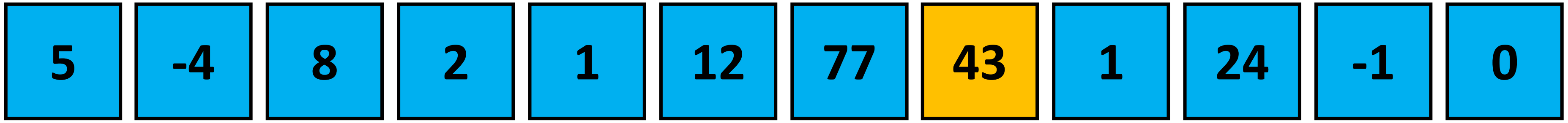
- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables



if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables

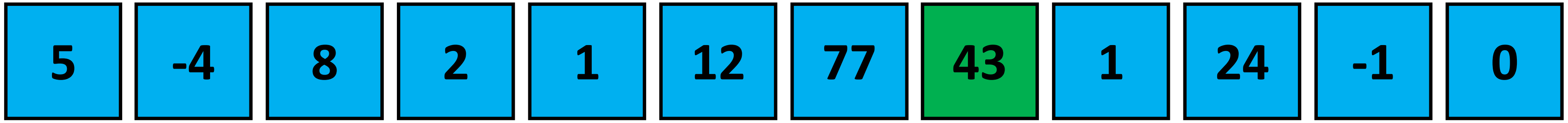


5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Linear Search

- **linear search** (sequential search) is a method for finding an item (element) in an unsorted list
- the algorithm makes **N** comparisons in worst-case
- hence the running time complexity is **O(N)** linear
- not that practical as we can achieve **O(logN)** or even **O(1)** running time with binary search and hash-tables



5	-4	8	2	1	12	77	43	1	24	-1	0

if we want to find an unknown item in a one-dimensional array (linear search)

Search Algorithms

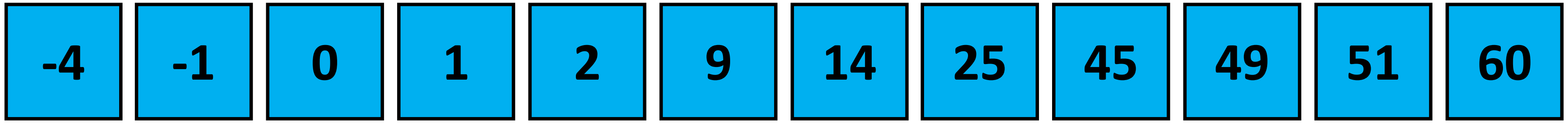
(Binary Search)

Binary Search

- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time

Binary Search

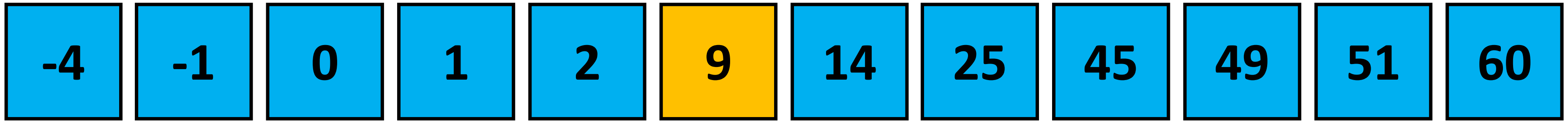
- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60

Binary Search

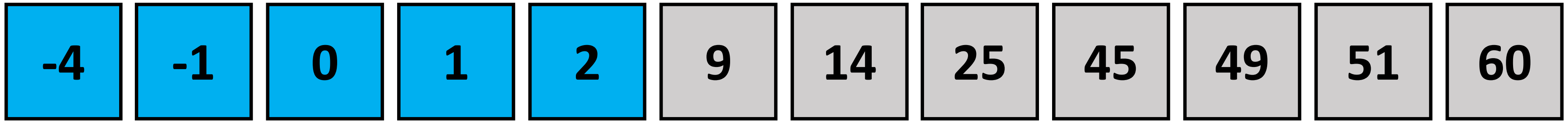
- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60

Binary Search

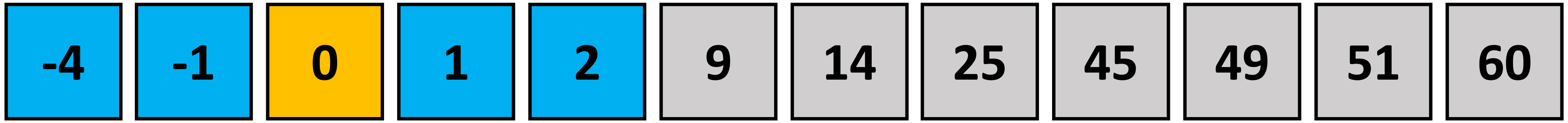
- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60

Binary Search

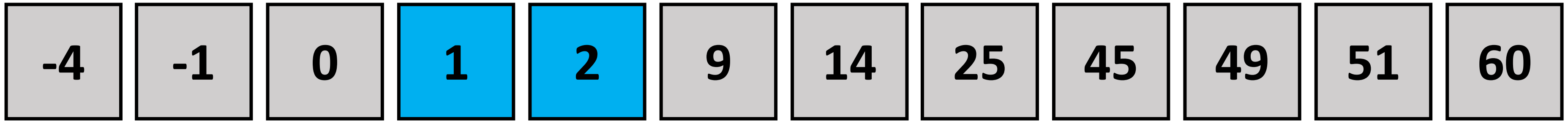
- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60

Binary Search

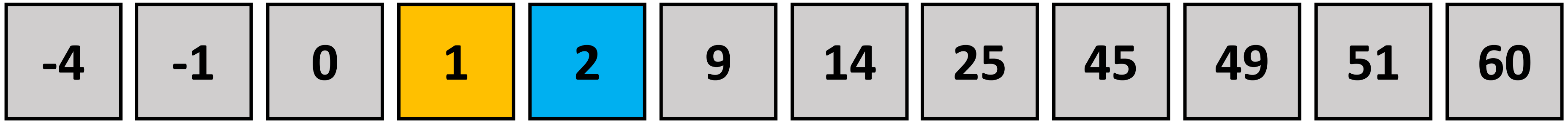
- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a sorted list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60

Binary Search

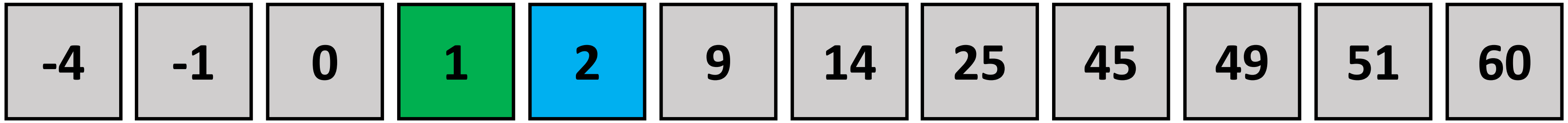
- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60

Binary Search

- **binary search** (logarithmic search) is a method for finding the position of an item (element) in a **sorted** list
- the algorithm makes **logN** comparisons in worst-case
- hence the running time complexity is **O(logN)** linear
- it has practical and real-world applications as **O(logN)** running time is quite favorable – it is close to **O(1)** constant running time



-4	-1	0	1	2	9	14	25	45	49	51	60