AI VIET NAM

@aivietnam.edu.vn

Algorithm design and analysis

Recursion – Two pointer

Nguyễn Quốc Thái

AI VIET NAM
@aivietnam.edu.vn

CONTENT

(1) – Brute Force - Exhaustive

(2) – Recursion

(3) – Two Pointer

Brute Force

- A straightforward approach usually based on problem statement and definitions
- The simplest to apply
- General approach: applicable to a very wide variety of problems
- No limitation on instance size
- Acceptable speed
- Useful for solving small-size instances of a problem

Brute Force

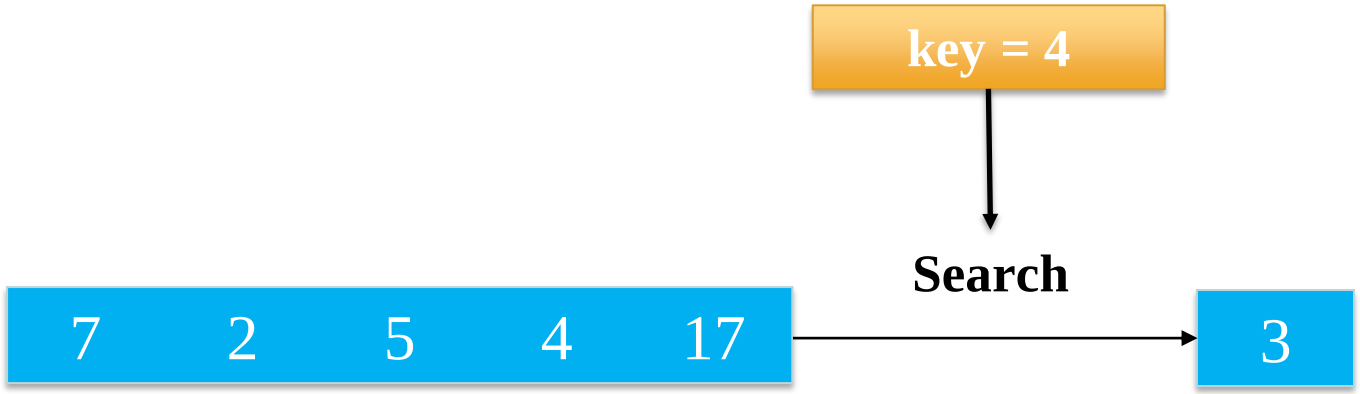
- A straightforward approach usually based on problem statement and definitions
- Example:
  - Computing a^n (a, b > 0, n integer)
  - Computing n!
  - Multiple two b by b matrices
  - Sequential search
  - Selection Sort

Brute Force

Sequential Search (Linear Search)

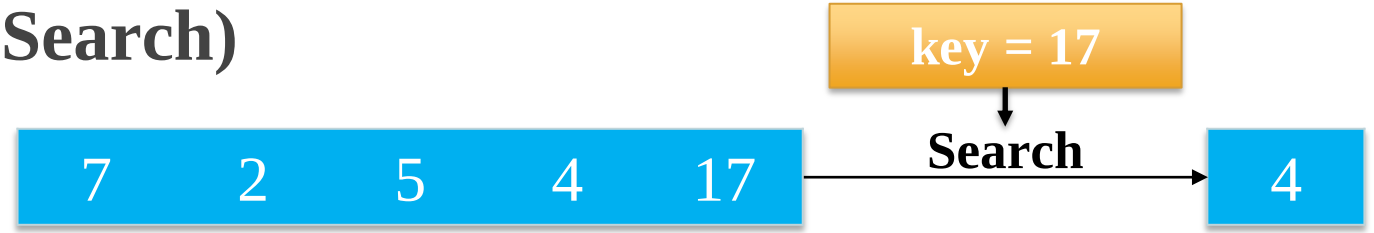
Input: a sequence of n number <a_1, a_2,…, a_n> and key

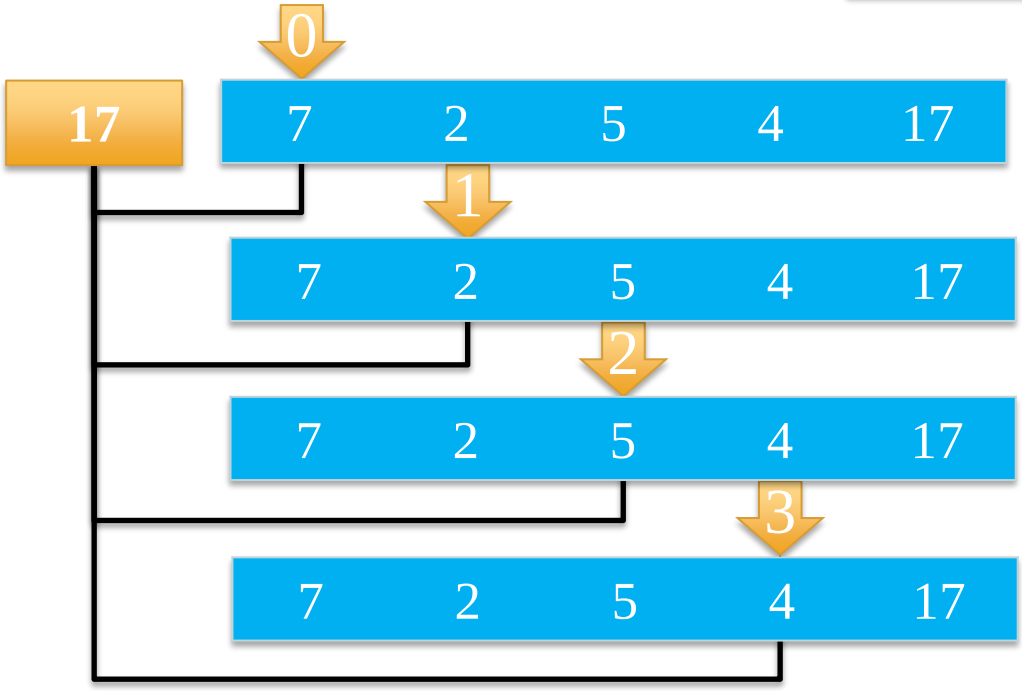
Output: index of key in the sequence if exist, -1 if not exist



Brute Force

Sequential Search (Linear Search)





Stop => Output = 3

```
[15] def linear_search(arr, key):

        n = len(arr)
        for idx in range(n):

            element = arr[idx]
            if element == key:
                return idx

        return -1
```

T(n) is O(n)

Brute Force

Selection Sort

Input: a sequence of n number $<a_1, a_2, \ldots, a_n>$ and *key*

Output: a permutation (reordering) $<a'_1, a'_2, \ldots, a'_n>$

such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Brute Force

Selection Sort

Input: a sequence of n number $<a_1, a_2, \ldots, a_n>$ and *key*

Output: a permutation (reordering) $<a'_1, a'_2, \ldots, a'_n>$

such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

- Select the smallest element from an unsorted list in each iteration
- Place that element at the beginning of the unsorted list

AI VIET NAM
@aivietnam.edu.vn

Brute Force

Selection Sort

- Set the first element as minimum

- Compare minimum with the other element. If the other element is smaller than minimum, assign the second element as minimum.

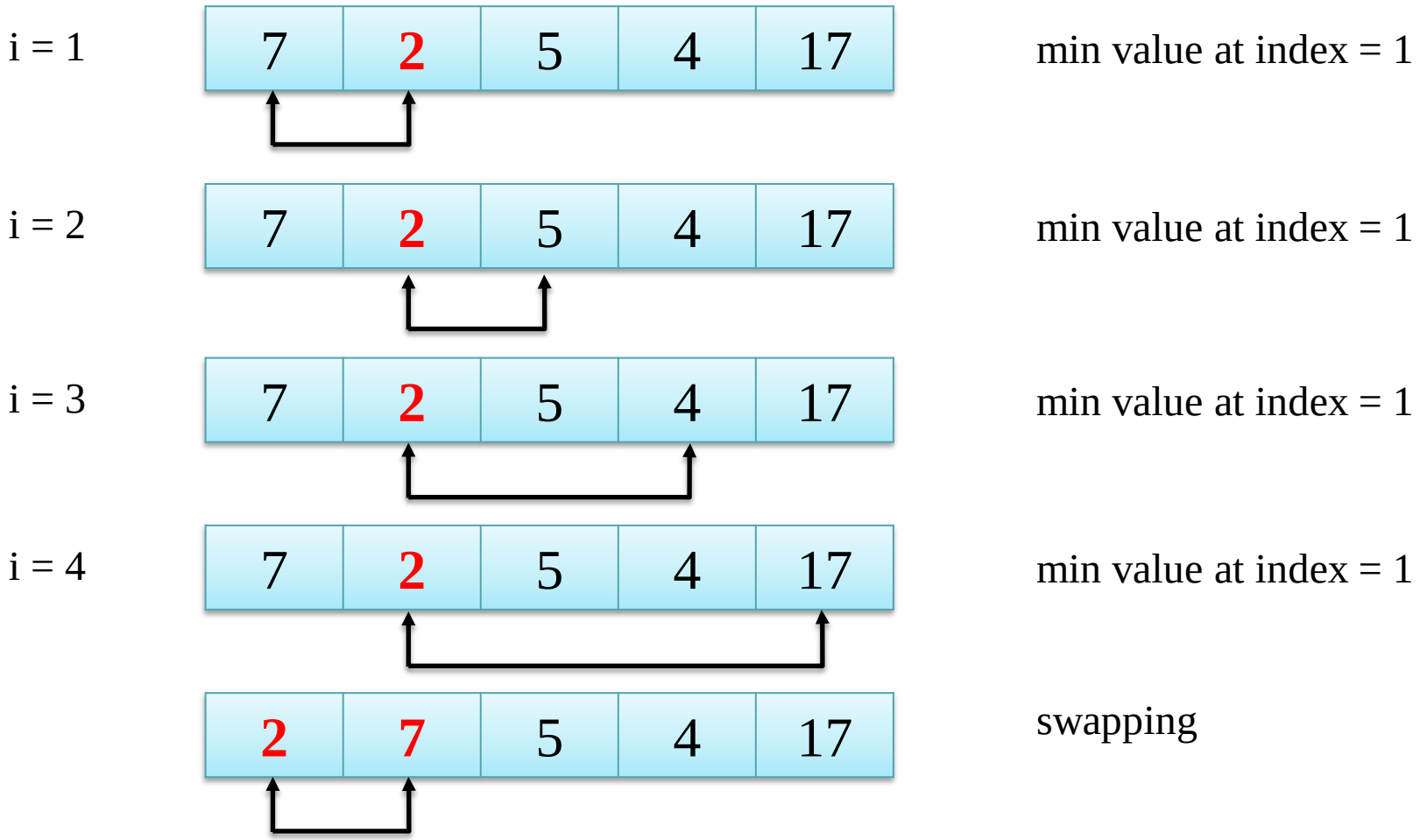
- After each iteration, minimum is placed in the front of the unsorted list

AI VIET NAM
@aivietnam.edu.vn

Brute Force

Selection Sort

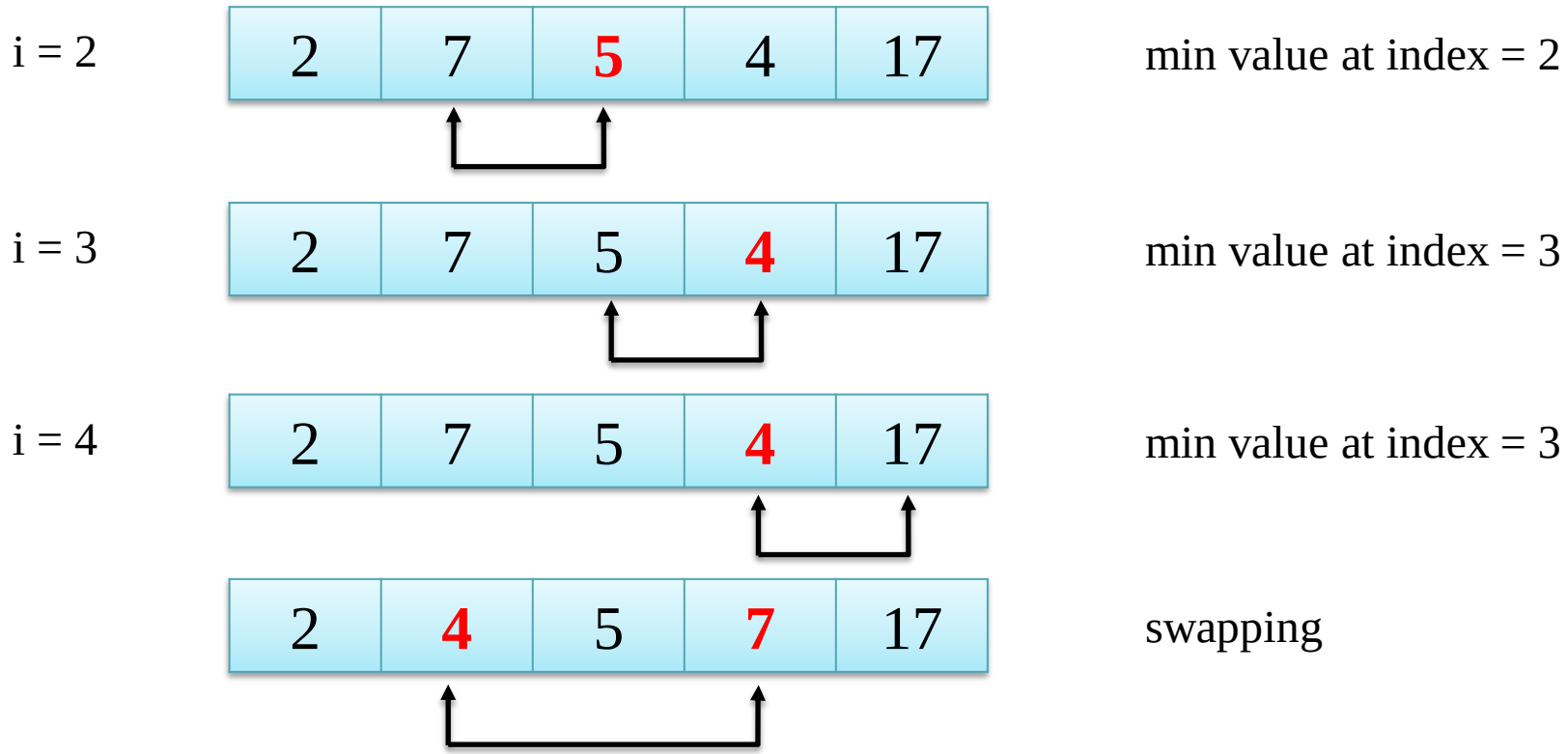
Step 0



Brute Force

Selection Sort

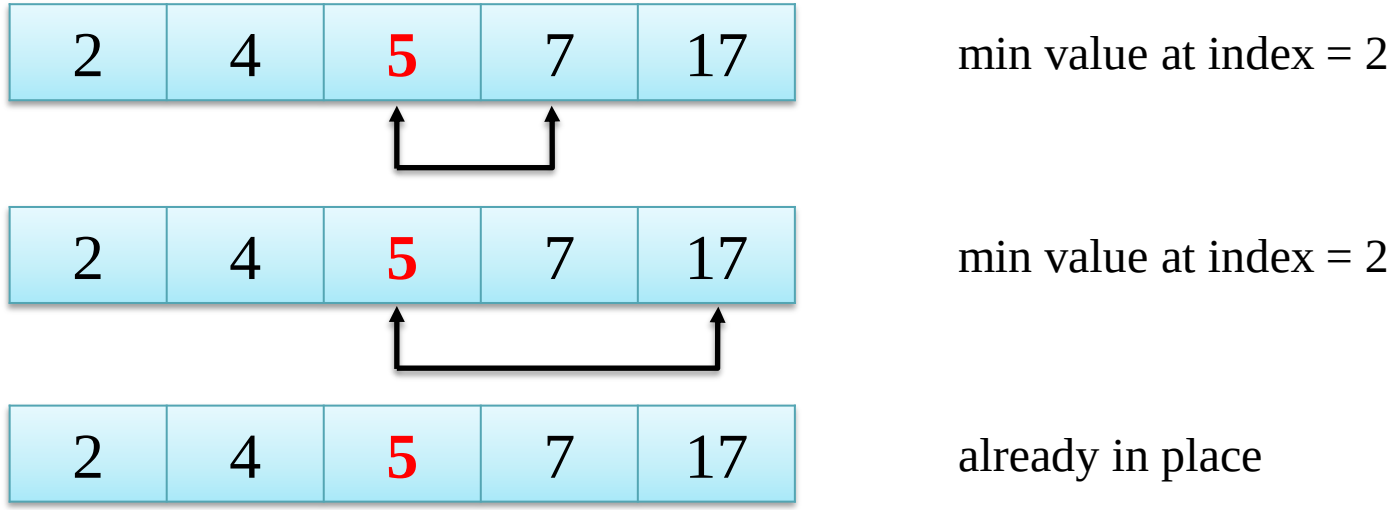
Step 1



Brute Force

Selection Sort

Step 2

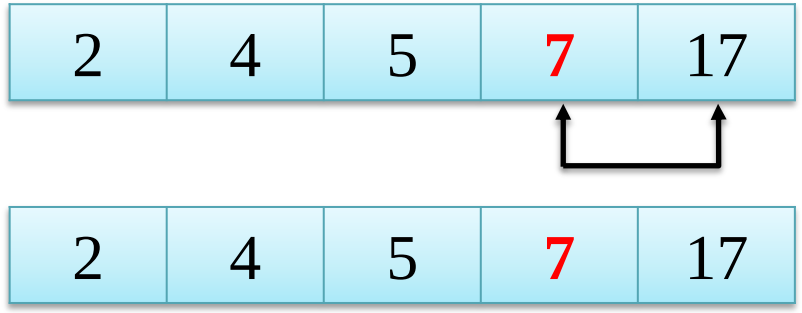


Brute Force

Selection Sort

Step 3

i = 4

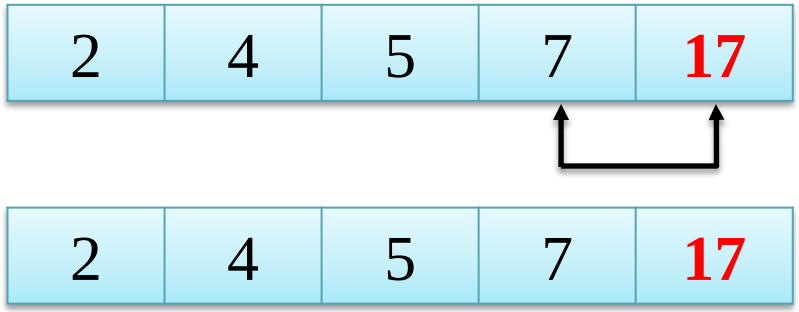


min value at index = 3

already in place

Step 4

i = 4



min value at index = 4

already in place

Brute Force

Selection Sort

```
[3]  def selection_sort(array):
         n = len(array)

         for step in range(n):
             min_idx = step

             for i in range(step + 1, n):

                 if array[i] < array[min_idx]:
                     min_idx = i

             array[step], array[min_idx] = array[min_idx], array[step]

         return array

     arr = [7, 2, 5, 4, 17]
     selection_sort(arr)
```

```
[2, 4, 5, 7, 17]
```

Brute Force

Selection Sort

```
[3]  def selection_sort(array):
         n = len(array)

         for step in range(n):
             min_idx = step

             for i in range(step + 1, n):

                 if array[i] < array[min_idx]:
                     min_idx = i

             array[step], array[min_idx] = array[min_idx], array[step]

         return array

     arr = [7, 2, 5, 4, 17]
     selection_sort(arr)
```

```
[2, 4, 5, 7, 17]
```

T(n) is $O(n^2)$

Worst case
Best case
Average case

AI VIET NAM
@aivietnam.edu.vn

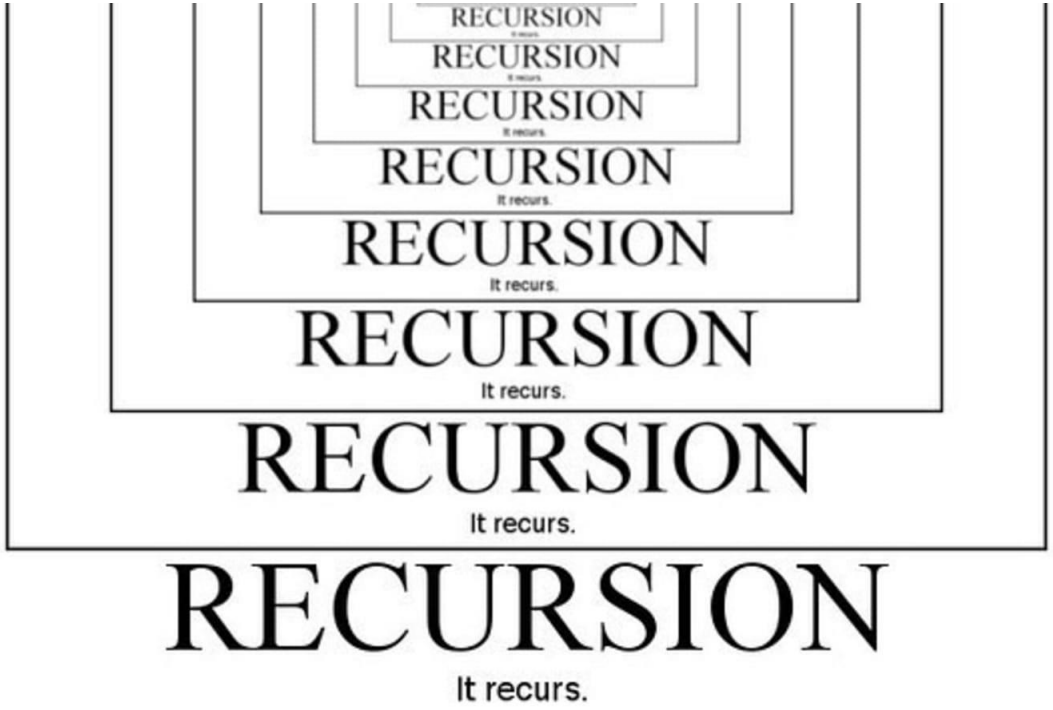
Brute Force

Summary

- Scanning all element (objective)
- Each element: check with the condition
  - If yes, stop
  - If no, continue
- Simple
- Not optimized
- The first problem solving model to think for every problem

Recursion

- Recursion: a function makes one or more calls to itself during execution
- Powerful for performing repetitive tasks
- Example:
  - Factorial Function
  - Binary Search
  - Fibonacci Number



AI VIET NAM
@aivietnam.edu.vn

Recursion

Factorial Function

- For any integer $n \geq 0$

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) * \cdots * 2 * 1 & \text{if } n \geq 1 \end{cases}$$

- Recursive definition:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n \geq 1 \end{cases}$$

Base case

- Input: integer $n \geq 0$
- Output: value of n!

AI VIET NAM
@aivietnam.edu.vn

Recursion

Factorial Function

- Recursive definition:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n \geq 1 \end{cases}$$

```
[4]  def factorial(n):
         if n == 0:
             return 1
         else:
             return n*factorial(n-1)

     factorial(5)
```

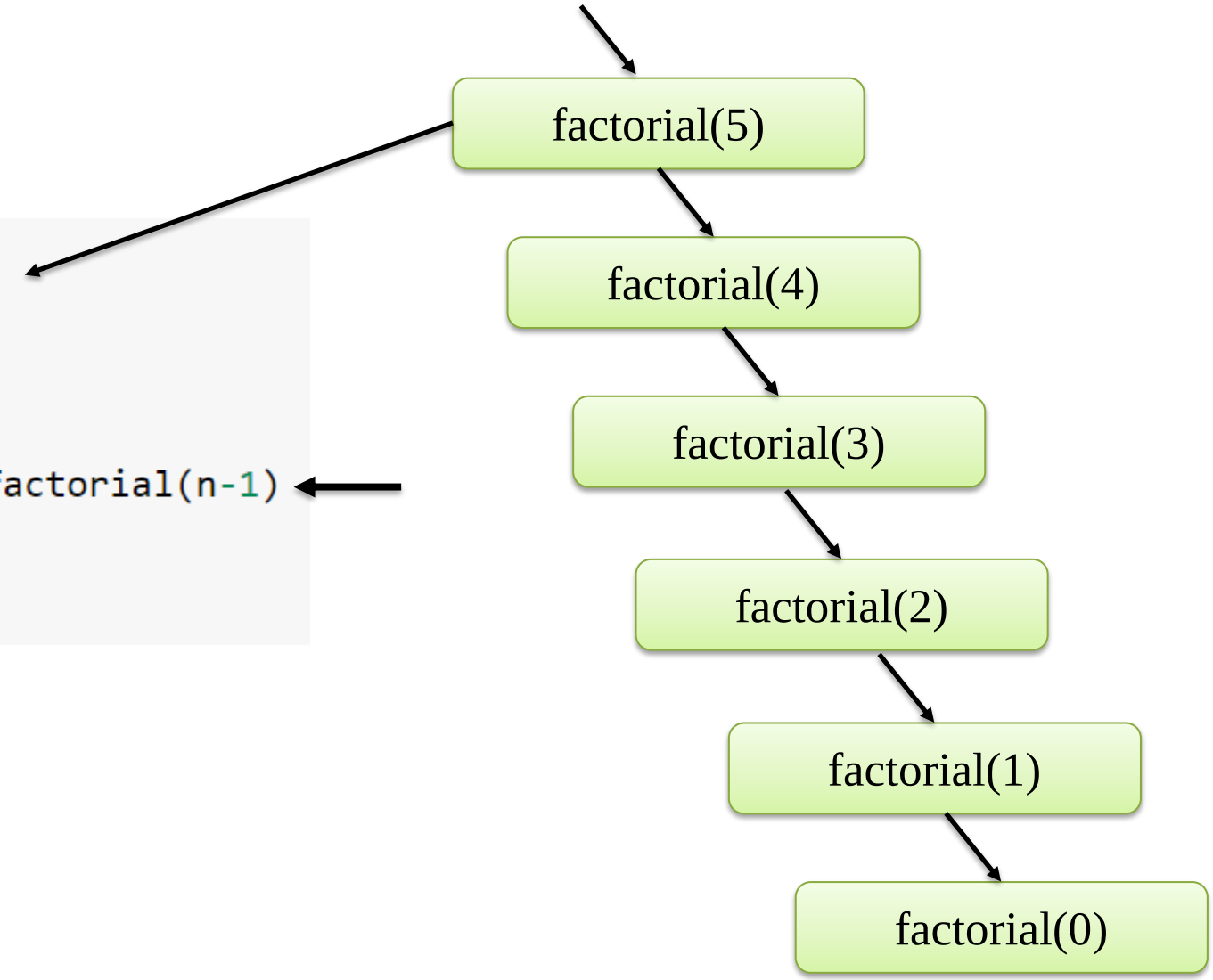
```
120
```

Recursion

Factorial Function

```
[4]  def factorial(n):
         if n == 0:
             return 1
         else:
             return n*factorial(n-1)

     factorial(5)
```

```
120
```



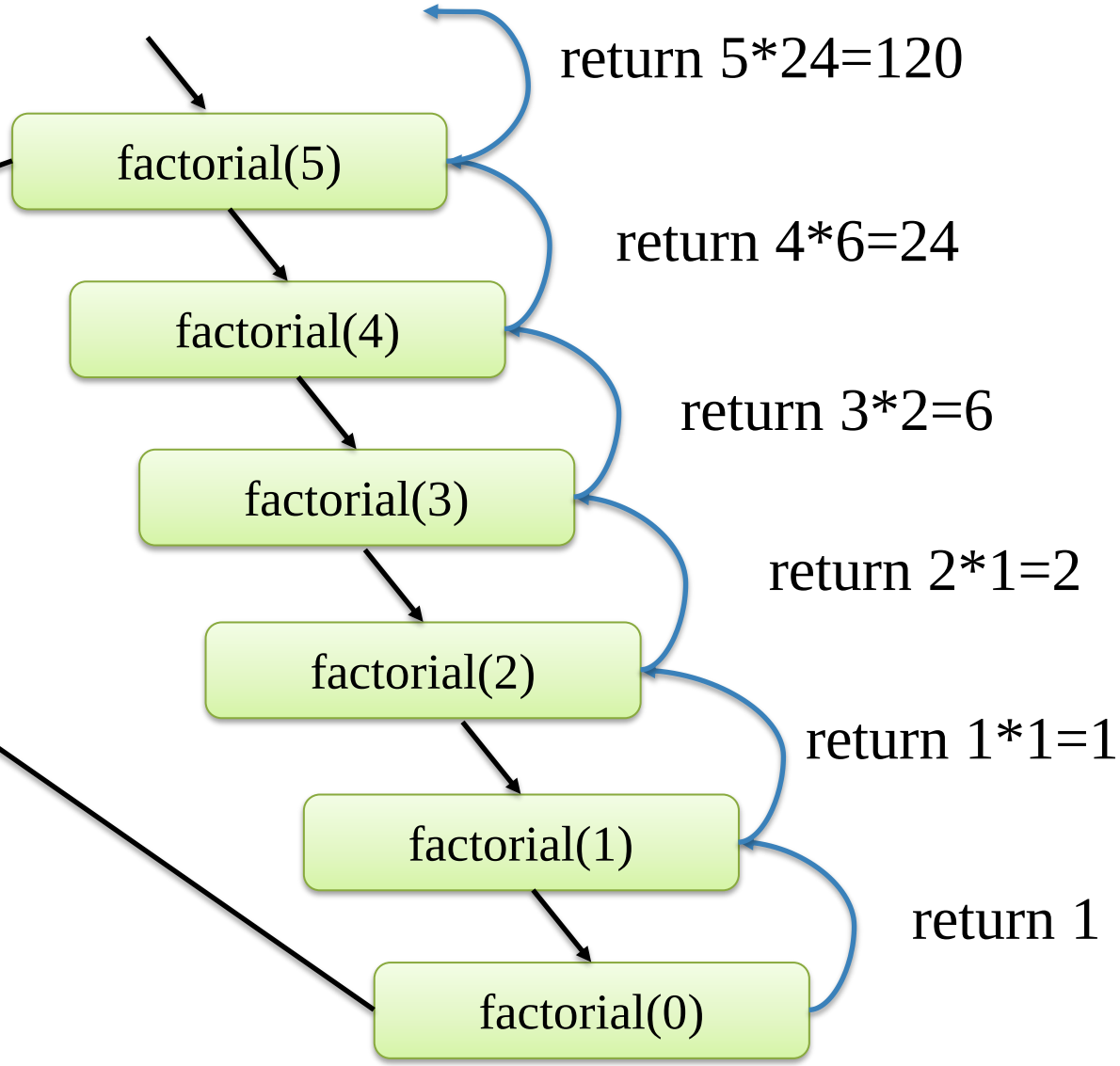
AI VIET NAM
@aivietnam.edu.vn

Recursion

Factorial Function

```
[4]  def factorial(n):
         if n == 0:
             return 1
         else:
             return n*factorial(n-1)

     factorial(5)
```

```
120
```



AI VIET NAM
@aivietnam.edu.vn

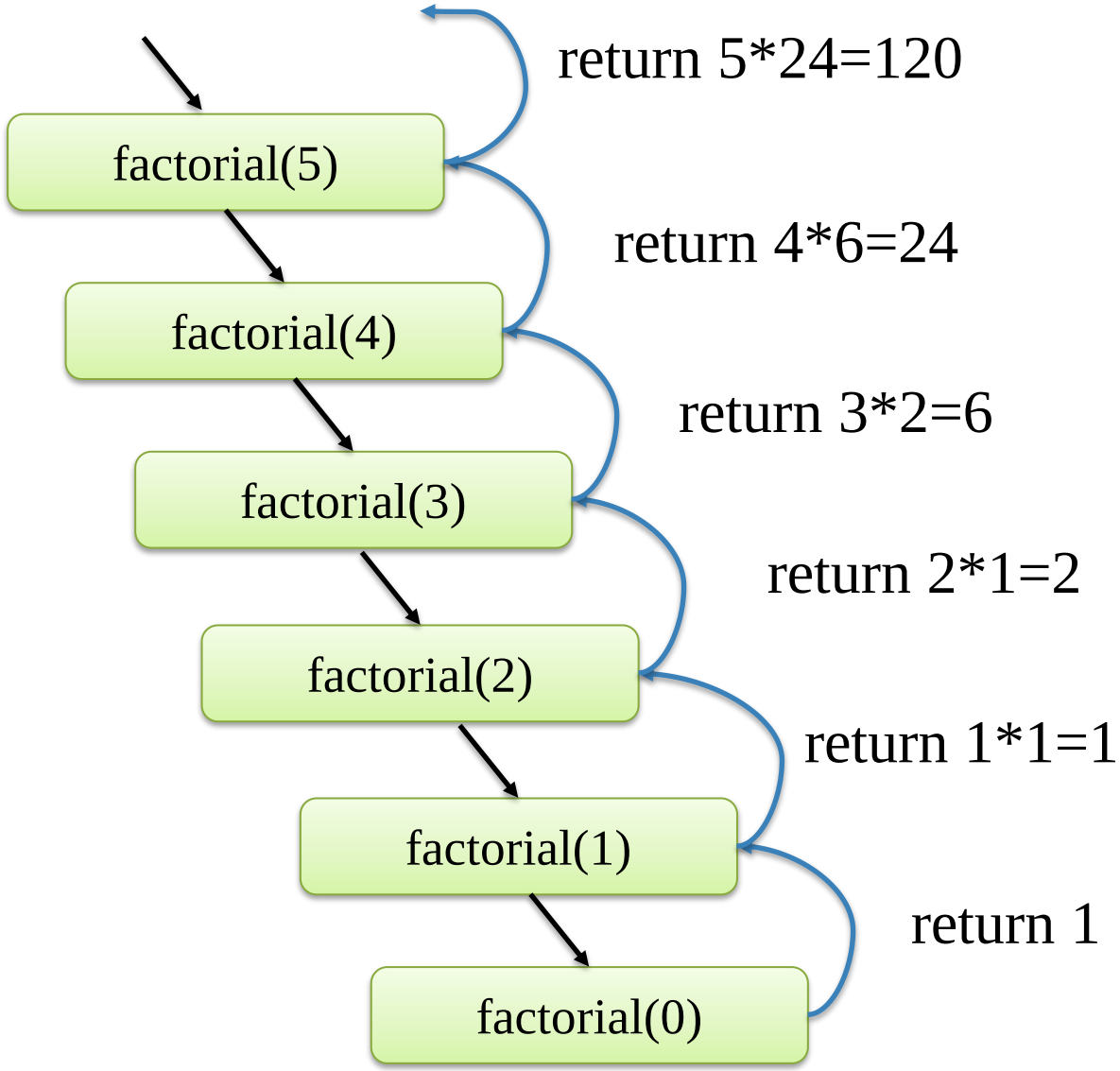
Recursion

Factorial Function

```
[4]  def factorial(n):
         if n == 0:
             return 1
         else:
             return n*factorial(n-1)

     factorial(5)
```

```
120
```

The number of recursive call



AI VIET NAM
@aivietnam.edu.vn

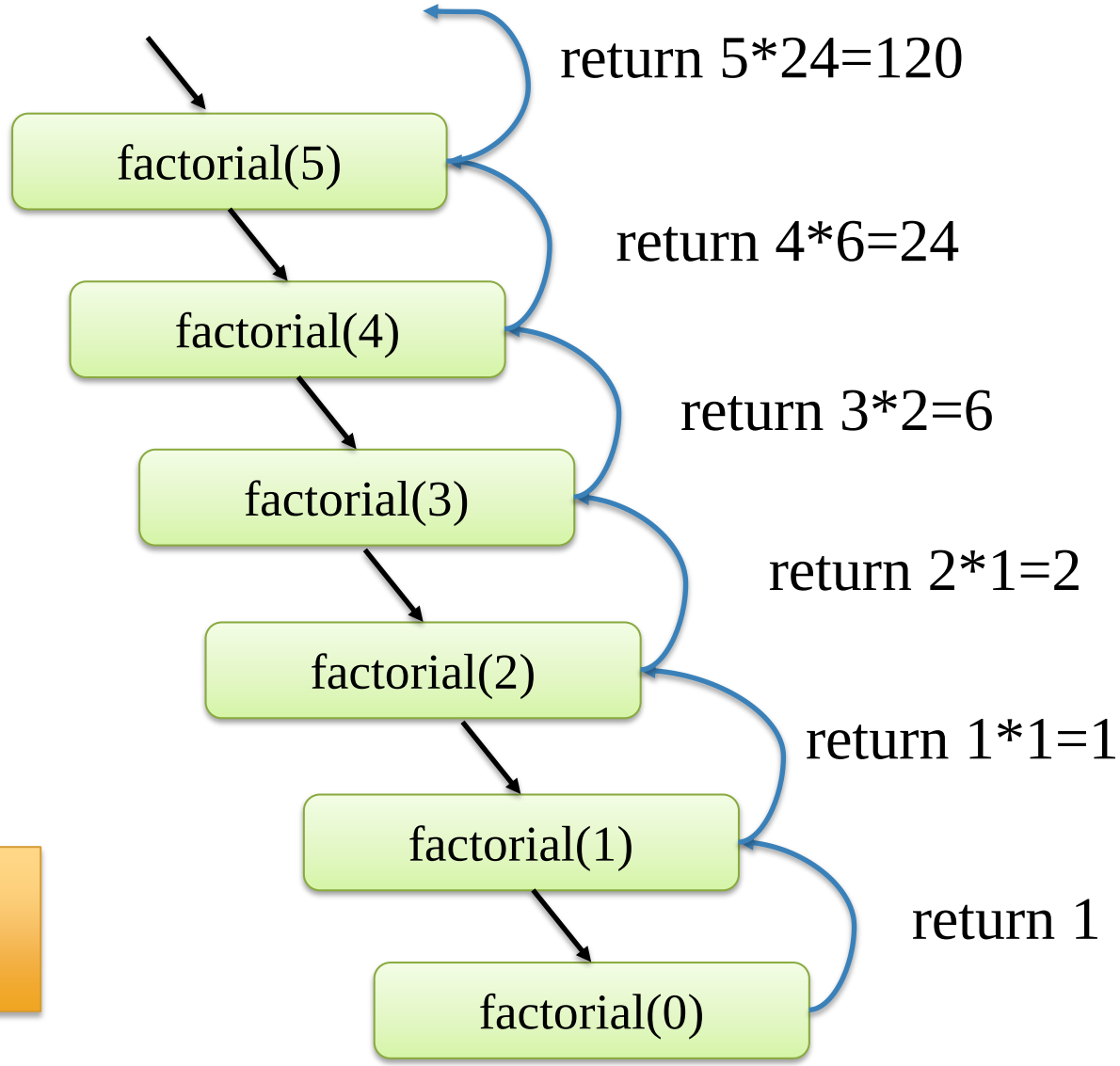
Recursion

Factorial Function

```
[4]  def factorial(n):
         if n == 0:
             return 1
         else:
             return n*factorial(n-1)

     factorial(5)
```

```
120
```

The number of recursive call is n+ 1
T(n) is O(n)



Recursion

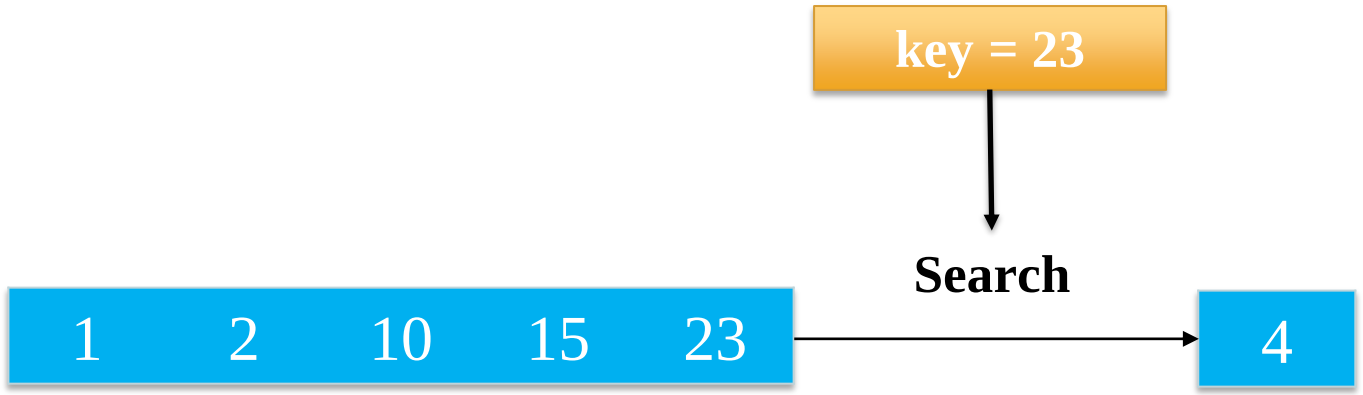
Binary Search

Searching problem

Input: a sorted sequence of n number <a_1, a_2,…, a_n>, key

Output: index of key in the sequence if exist, -1 if not exist

=> Always searched in the middle of a portion of a sequence



Recursion

Binary Search

Initially, low = 0, high = len(arr) - 1

Always searched in the middle of a portion of a sequence

$$mid = \lfloor (low + high)/2 \rfloor$$

Consider three case:

- key = arr[mid] => search successfully => return mid
- key < arr[mid] => find key in arr[low, mid-1]
- key > arr[mid] => find key in arr[mid+1, high]

Recursion

Binary Search

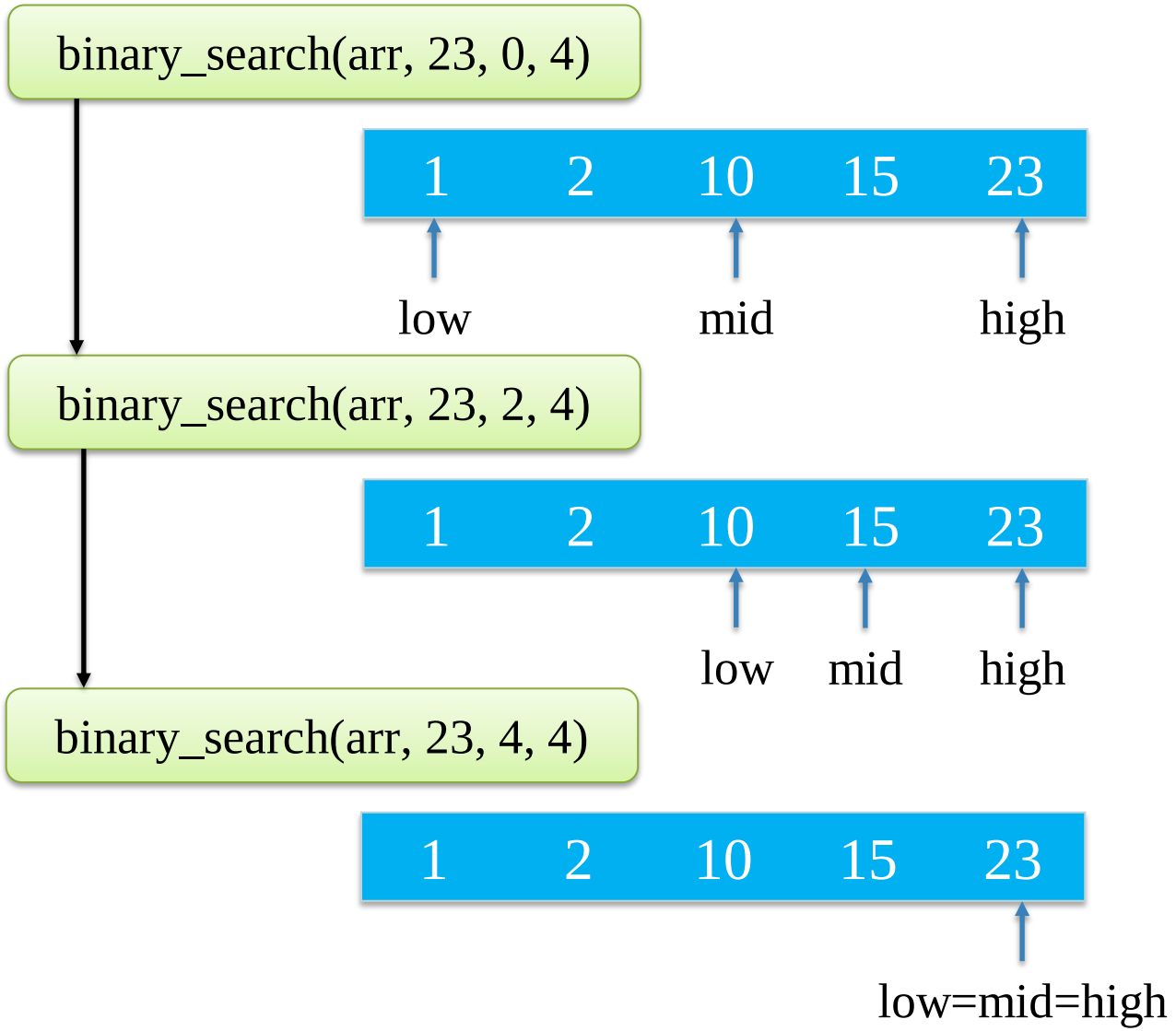
```python
def binary_search(array, key, low, high):
    if high >= low:
        mid = low + (high - low)//2
        if array[mid] == key:
            return mid
        elif array[mid] > key:
            return binary_search(array, key, low, mid-1)
        else:
            return binary_search(array, key, mid + 1, high)
    else:
        return -1

arr = [1, 2, 10, 15, 23]
key = 23
low = 0
high = len(arr) - 1
binary_search(arr, key, low, high)
```

AI VIET NAM
@aivietnam.edu.vn

Recursion

Binary Search

```python
def binary_search(array, key, low, high):
    if high >= low:
        mid = low + (high - low)//2
        if array[mid] == key:
            return mid
        elif array[mid] > key:
            return binary_search(array, key, low, mid-1)
        else:
            return binary_search(array, key, mid + 1, high)
    else:
        return -1

arr = [1, 2, 10, 15, 23]
key = 23
low = 0
high = len(arr) - 1
binary_search(arr, key, low, high)
```



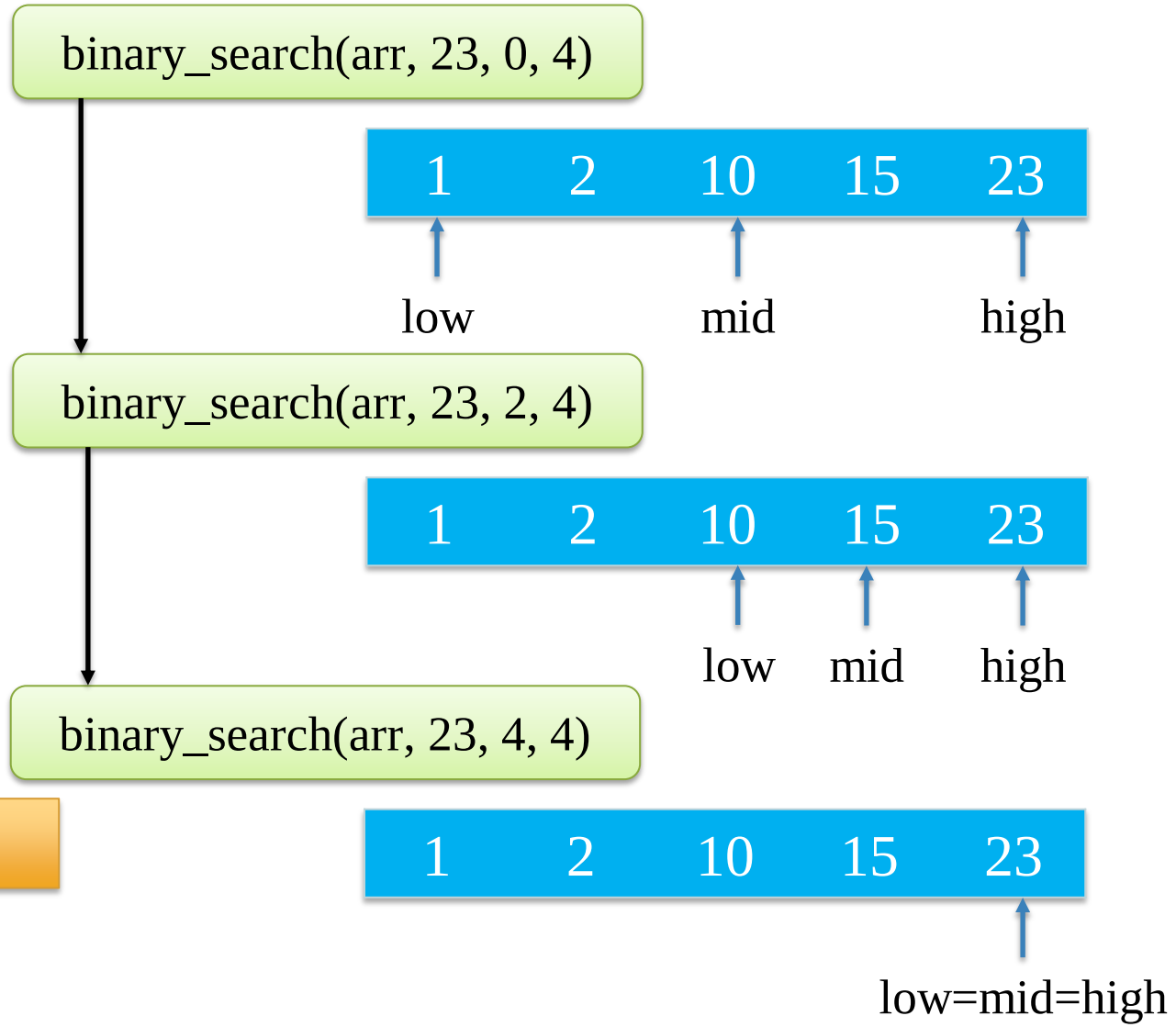
AI VIET NAM
@aivietnam.edu.vn

Recursion

Binary Search

```python
def binary_search(array, key, low, high):
    if high >= low:
        mid = low + (high - low)//2
        if array[mid] == key:
            return mid
        elif array[mid] > key:
            return binary_search(array, key, low, mid-1)
        else:
            return binary_search(array, key, mid + 1, high)
    else:
        return -1

arr = [1, 2, 10, 15, 23]
key = 23
low = 0
high = len(arr) - 1
binary_search(arr, key, low, high)
```

The number of recursive call ?



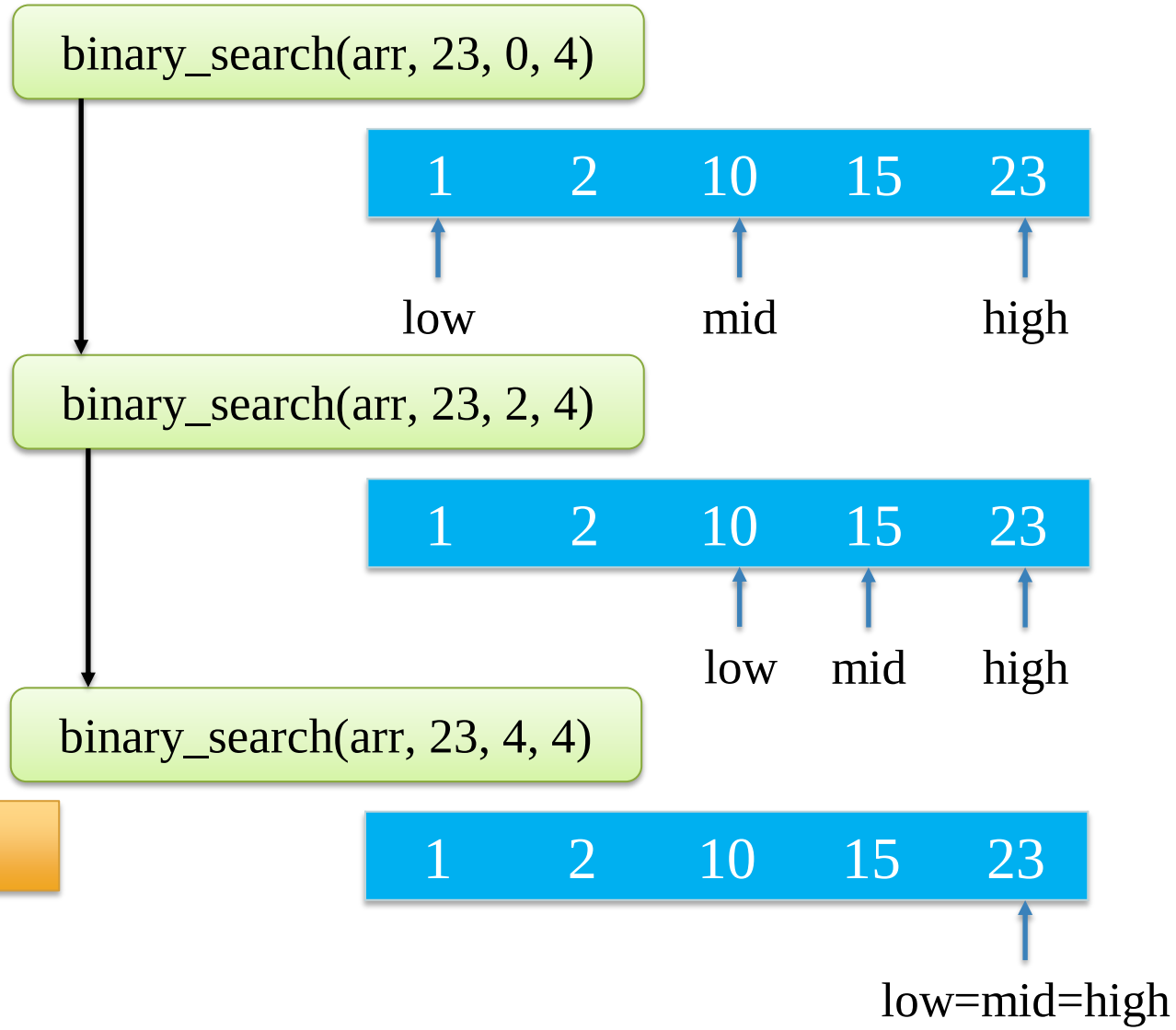
AI VIET NAM
@aivietnam.edu.vn

Recursion

Binary Search

```python
def binary_search(array, key, low, high):
    if high >= low:
        mid = low + (high - low)//2
        if array[mid] == key:
            return mid
        elif array[mid] > key:
            return binary_search(array, key, low, mid-1)
        else:
            return binary_search(array, key, mid + 1, high)
    else:
        return -1

arr = [1, 2, 10, 15, 23]
key = 23
low = 0
high = len(arr) - 1
binary_search(arr, key, low, high)
```

T(n) is O(logn)



Recursion

Fibonacci Number [509] Leetcode

$F_0 = 0$

$F_1 = 1$

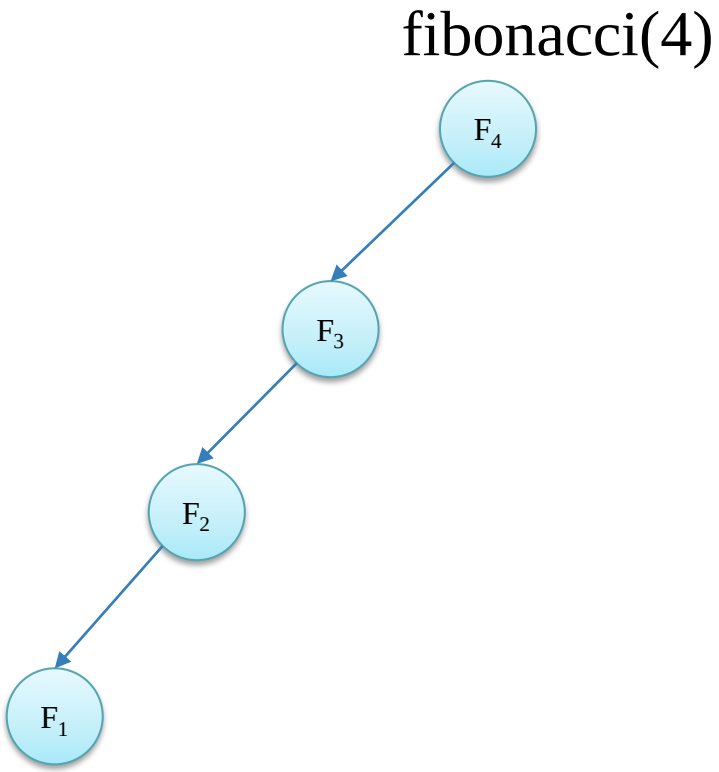
$F_n = F_{n-2} + F_{n-1}$ for $n > 1$

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number

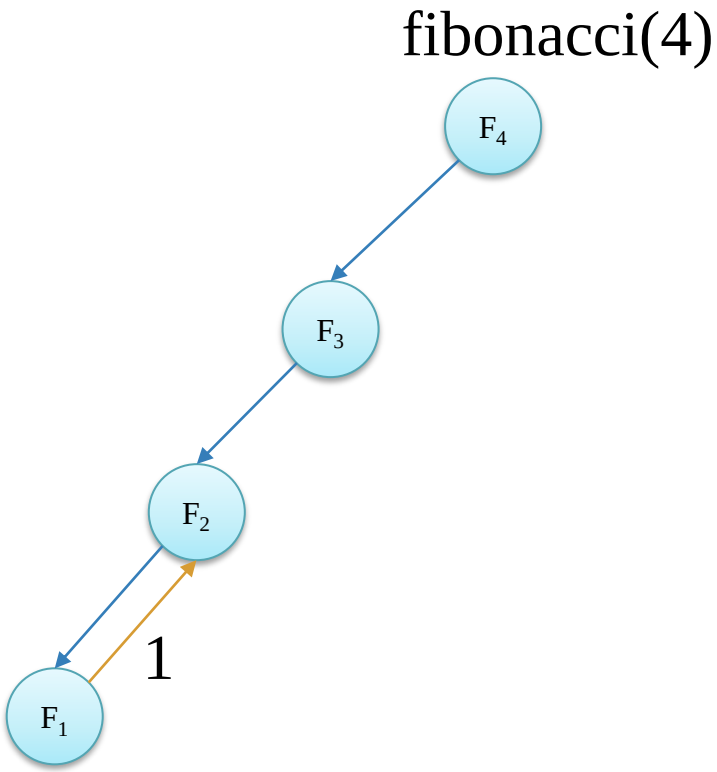



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number

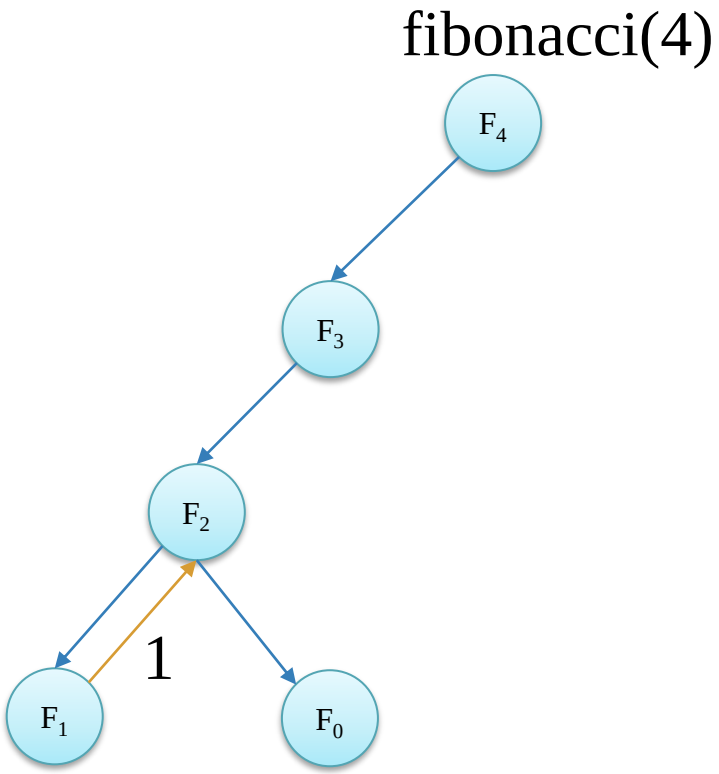



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number

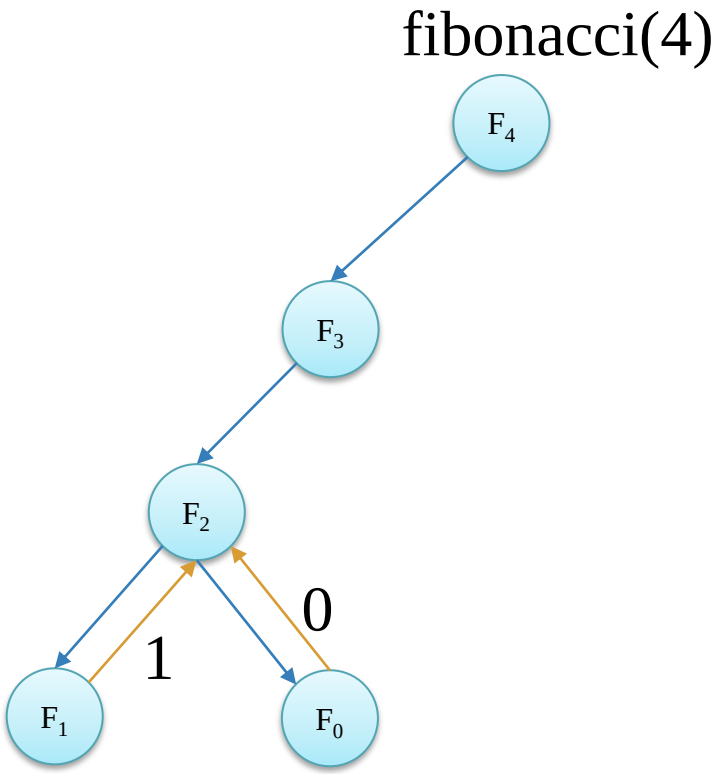



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number



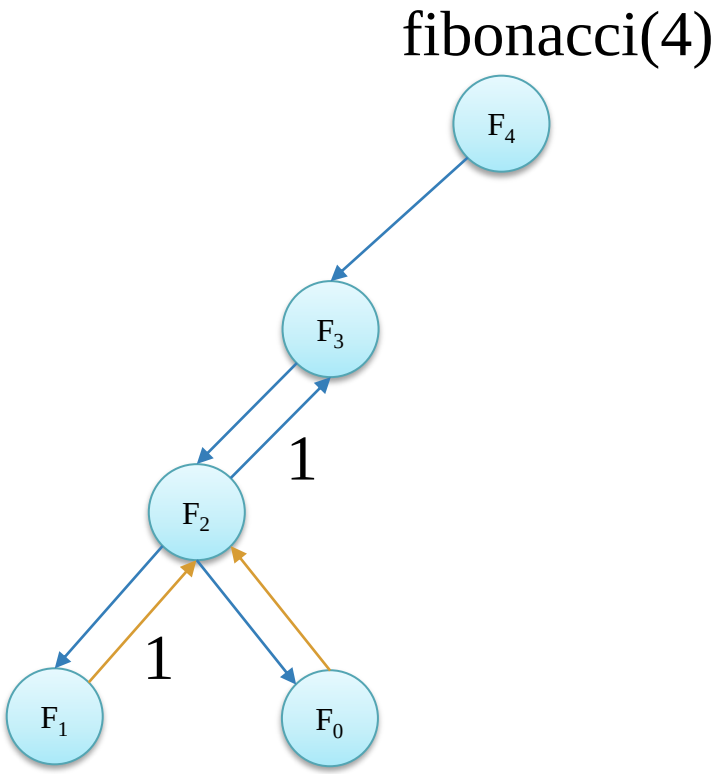

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

AI VIET NAM
@aivietnam.edu.vn

Recursion

Fibonacci Number



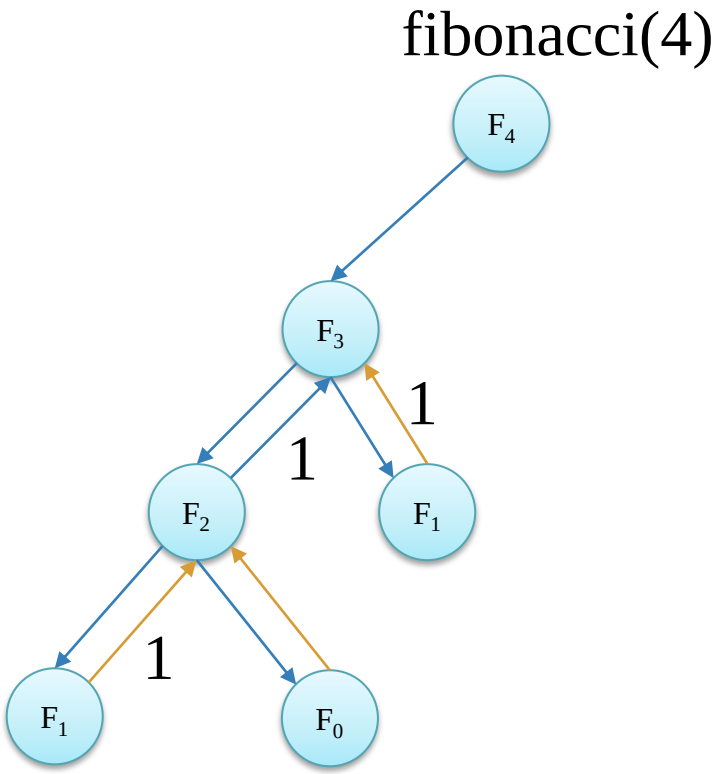

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

AI VIET NAM
@aivietnam.edu.vn

Recursion

Fibonacci Number

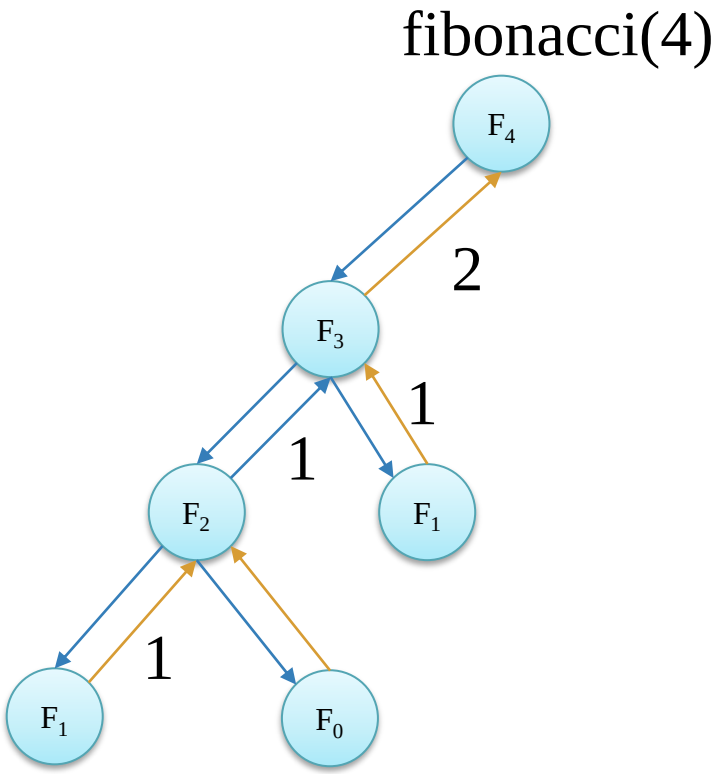



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number

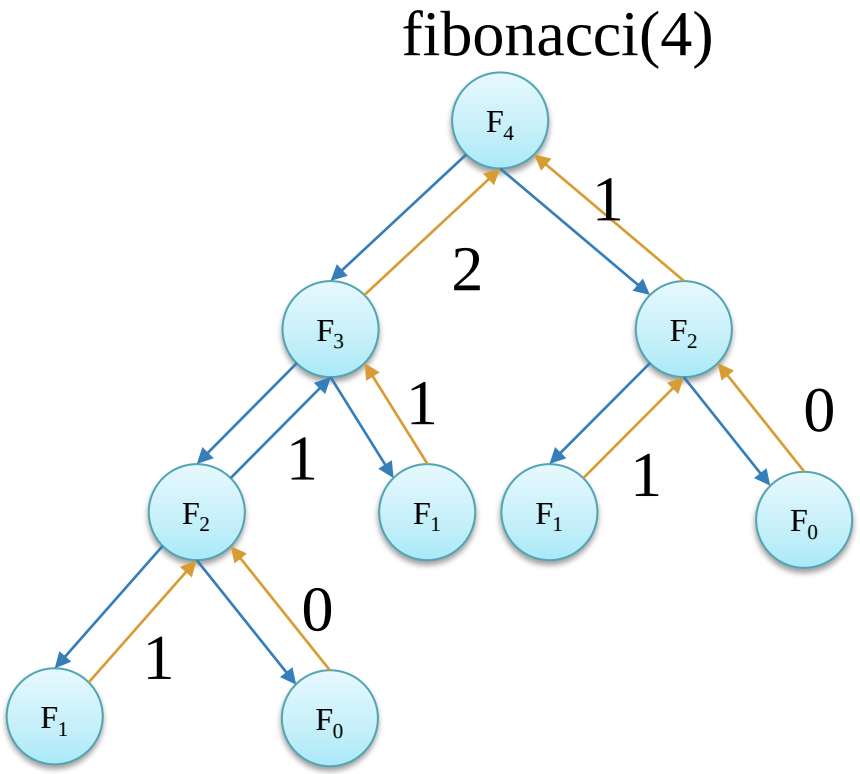



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number

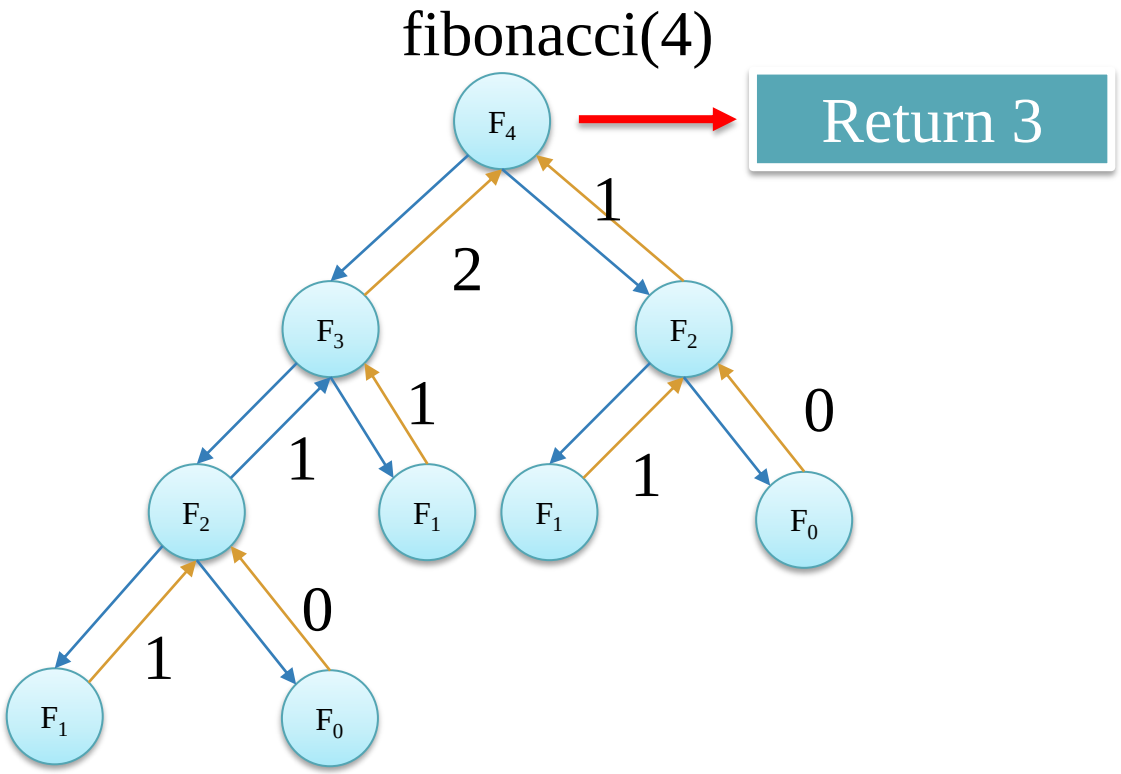



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number

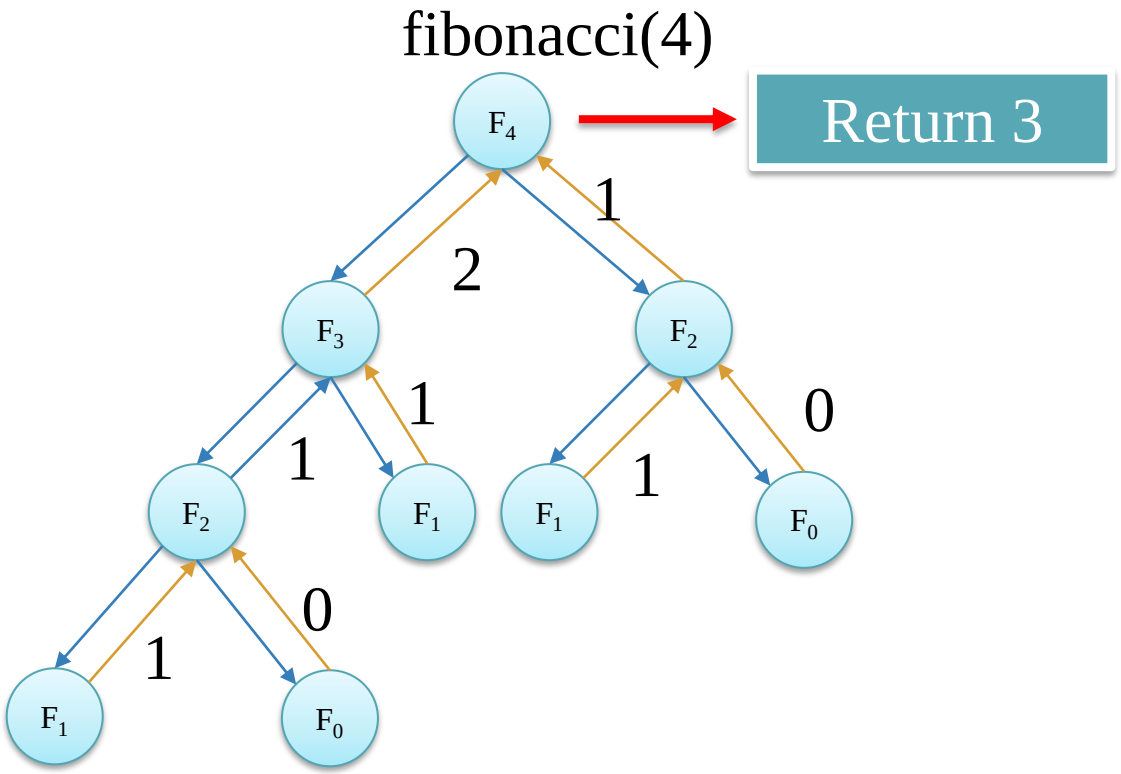



```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

Recursion

Fibonacci Number



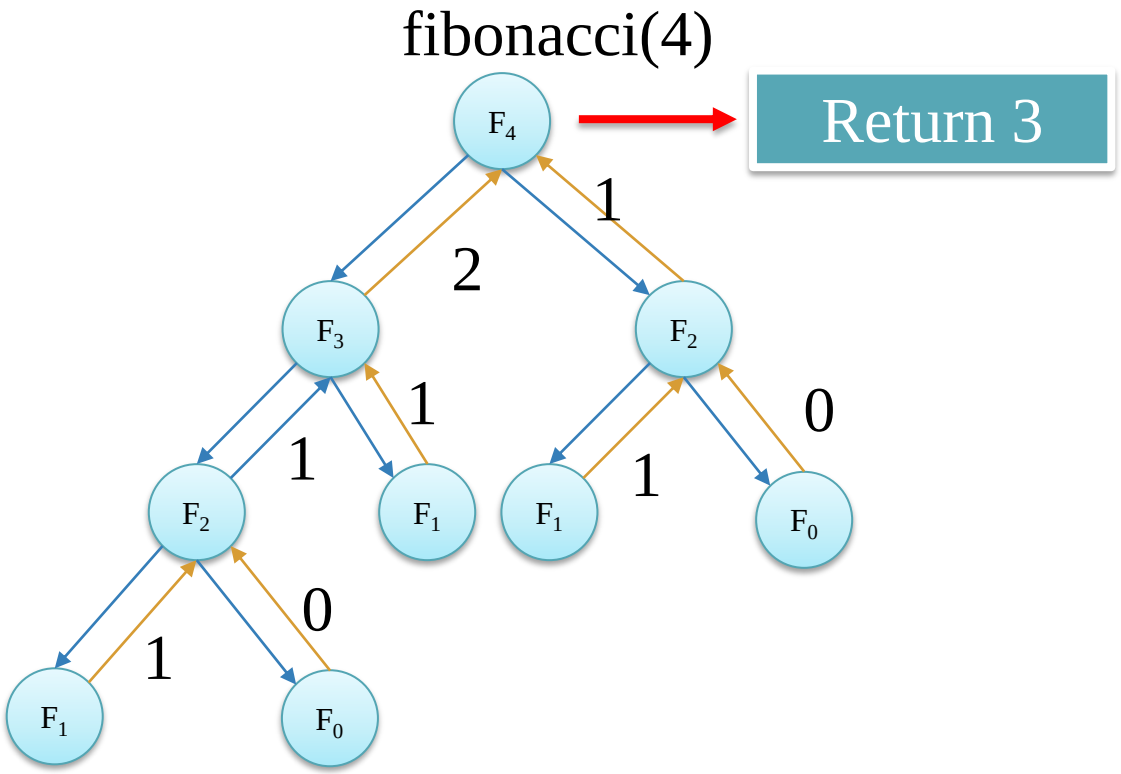

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

The number of recursive call is ?

Recursion

Fibonacci Number



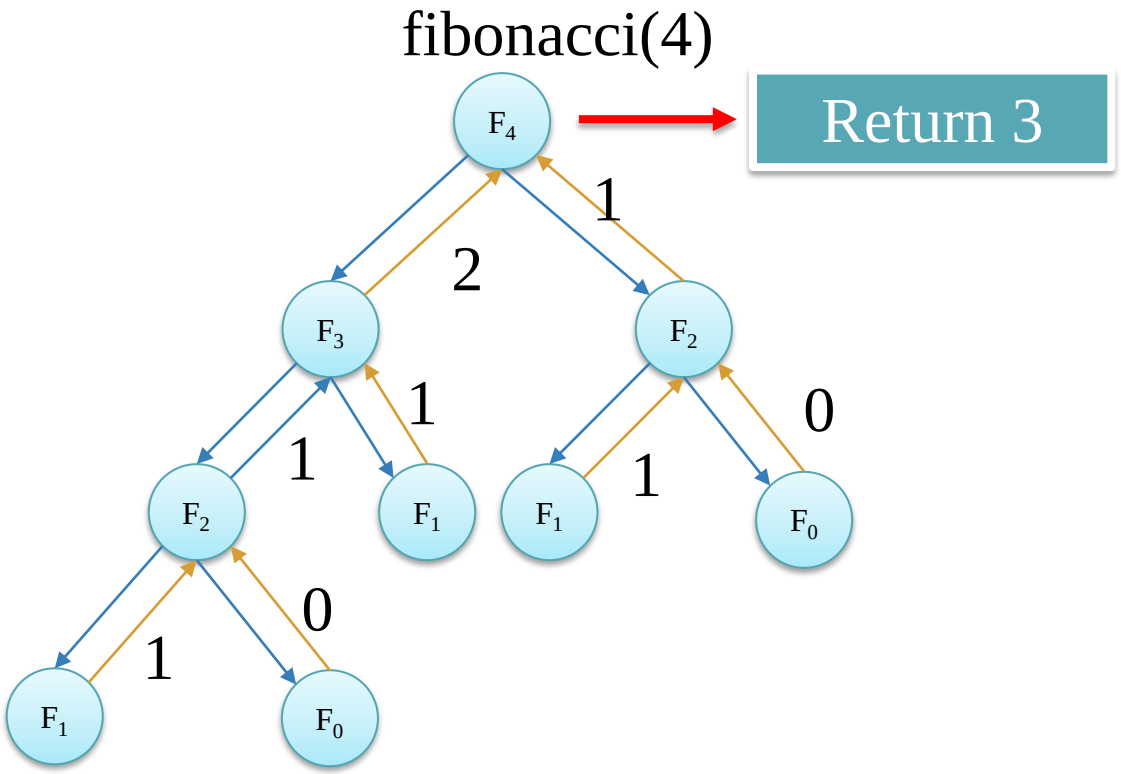

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

T(n) is $O(2^n)$

Recursion

Fibonacci Number




```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

fibonacci(4)
```

```
3
```

T(n) is $O(2^n)$

T(n) is O(n) ???

Recursion

Fibonacci Number

Efficient recursion with O(n)

```
[34] def good_fibonacci(n):
         if n == 0:
             return (0, 0)
         elif n == 1:
             return (1, 0)
         else:
             (a, b) = good_fibonacci(n-1)
             return (a+b, a)

     good_fibonacci(4), good_fibonacci(4)[0]
```

```
((3, 2), 3)
```

AI VIET NAM
@aivietnam.edu.vn

Recursion

SUMMAR

Base case
Avoid infinite recursion

```
[34] def good_fibonacci(n):
         if n == 0:
             return (0, 0)
         elif n == 1:
             return (1, 0)
         else:
             (a, b) = good_fibonacci(n-1)
             return (a+b, a)

     good_fibonacci(4), good_fibonacci(4)[0]

     ((3, 2), 3)
```

Call to itself

Smaller instance size

AI VIET NAM
@aivietnam.edu.vn

Two pointer

Reverse String [344] Leetcode

Example 1:

```
Input: s = ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
```

Example 2:

```
Input: s = ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]
```

Constraints:

- `1 <= s.length <= 10^5`
- `s[i]` is a printable ascii character.

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Reverse String

Using for loop

```
[68] def reverse_str_1(s):
         tg_s = ''
         for i in range(len(s)-1, -1, -1):
             tg_s += s[i]
         return tg_s

     s = 'abcde'
     reverse_str_1(s)
```

```
'edcba'
```

Two Pointer

Reverse String

Using recursion

```
[67] def reverse_str_2(s, low, high):
         if low < high -1:
             s[low], s[high-1] = s[high-1], s[low]
             reverse_str_2(s, low + 1, high - 1)

     s = 'abcde'
     s = list(s)
     low = 0
     high = len(s)
     reverse_str_2(s, low, high)
     "".join(s)
```

```
'edcba'
```

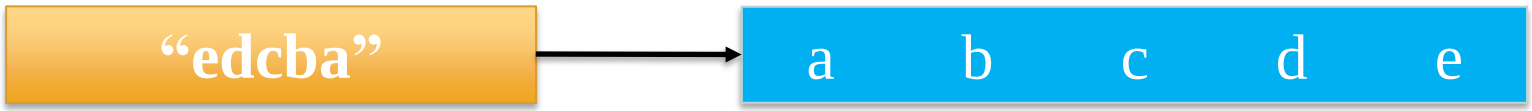
AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Reverse String

Using two pointer

- Convert string to list



- Init two pointer: i: index of first element, j: index of last element

- While loop with i <=j. swapping value of i and j, add i and 1. sub j and 1

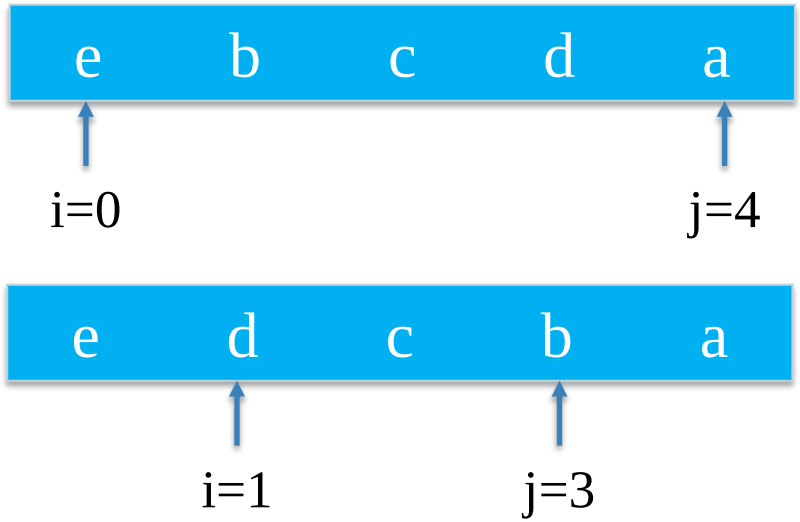
AI VIET NAM
@aivietnam.edu.vn

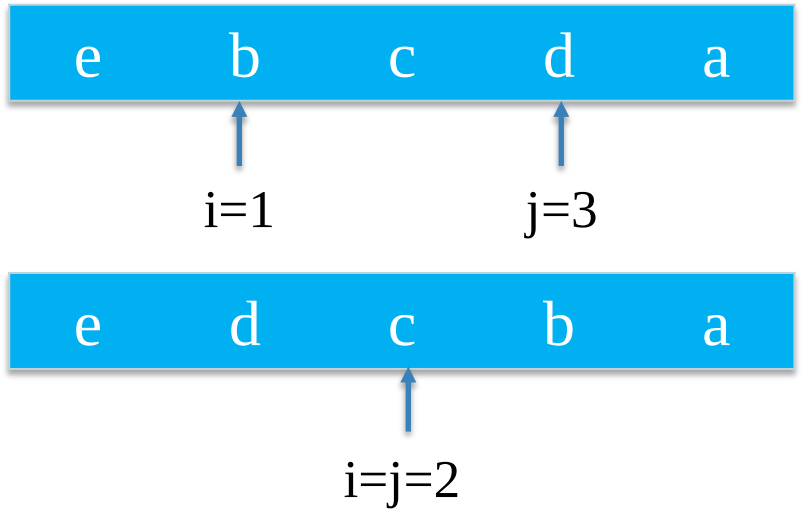
Two Pointer

Reverse String

Using two pointer

- While loop with i <=j. swapping value of i and j, add i and 1. sub j and 1





- Convert list to string

AI VIET NAM
@aivietnam.edu.vn

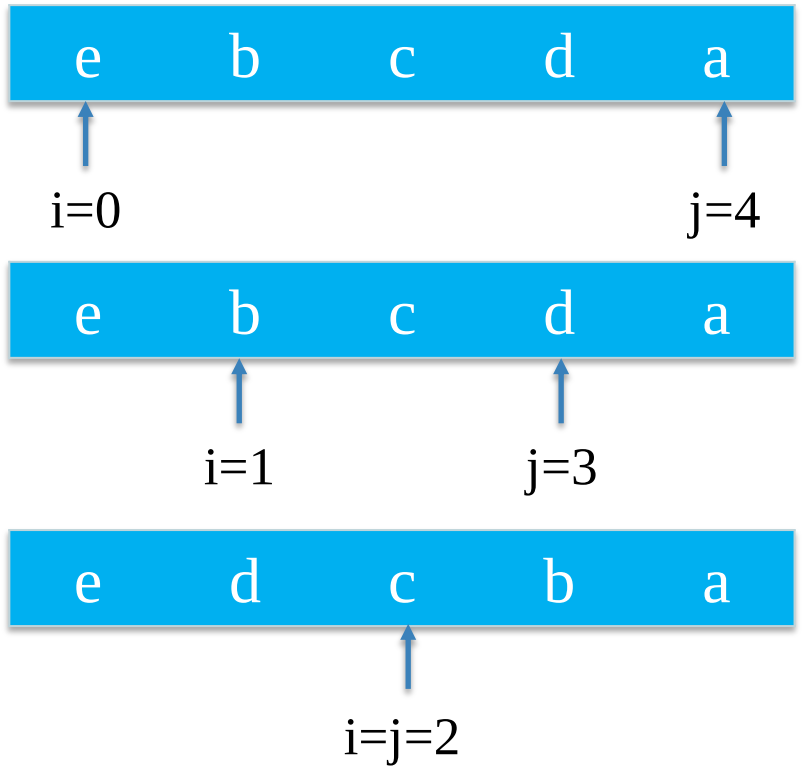
Two Pointer

Reverse String

Using two pointer

```python
def reverse_str_3(s):
    i = 0
    j = len(s) - 1
    while i <= j:
        s[i], s[j] = s[j], s[i]
        i = i + 1
        j = j - 1
    return "".join(s)

s = 'abcde'
s = list(s)
reverse_str_3(s)
```

```
'edcba'
```



AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum [560] Leetcode

Given an array of integers *nums* and an integer *k*, return the total number of subarrays whose sum equals to *k*.

A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

```
Input: nums = [1,1,1], k = 2
Output: 2
```

Example 2:

```
Input: nums = [1,2,3], k = 3
Output: 2
```

Constraints:

- $1 <= nums.length <= 2 * 10^4$
- $-1000 <= nums[i] <= 1000$
- $-10^7 <= k <= 10^7$

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Simple solution

T(n) is ?

```python
def subarray_sum(nums, key):
        n = len(nums)
        count = 0
        for i in range(n):
            total = 0
            for j in range(i, n):
                total += nums[j]
                if total == key:
                    count += 1
        return count


nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum(nums, key)
```

```
2
```

Two Pointer

Find Subarray with Given Sum

Simple solution

T(n) is $O(n^2)$

```python
def subarray_sum(nums, key):
    n = len(nums)
    count = 0
    for i in range(n):
        total = 0
        for j in range(i, n):
            total += nums[j]
            if total == key:
                count += 1
    return count


nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum(nums, key)
```

```
2
```

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

i = ?
j = ?
stop condition ?

```python
def subarray_sum(nums, key):
    n = len(nums)
    count = 0
    for i in range(n):
        total = 0
        for j in range(i, n):
            total += nums[j]
            if total == key:
                count += 1
    return count


nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum(nums, key)
```

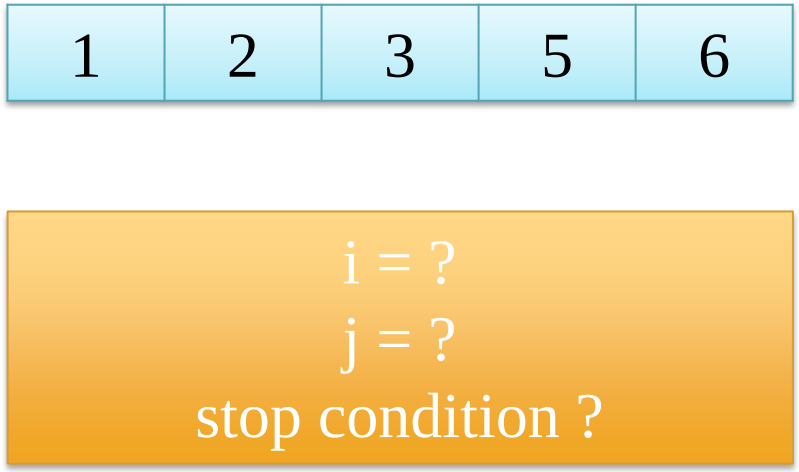
```
2
```

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer




```python
def subarray_sum(nums, key):
    n = len(nums)
    count = 0
    for i in range(n):
        total = 0
        for j in range(i, n):
            total += nums[j]
            if total == key:
                count += 1
    return count


nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum(nums, key)
```

```
2
```

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

i = ?
j = ?
stop condition ?
total : current subarray sum
count: number of subarray

```python
def subarray_sum(nums, key):
    n = len(nums)
    count = 0
    for i in range(n):
        total = 0
        for j in range(i, n):
            total += nums[j]
            if total == key:
                count += 1
    return count


nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum(nums, key)
```

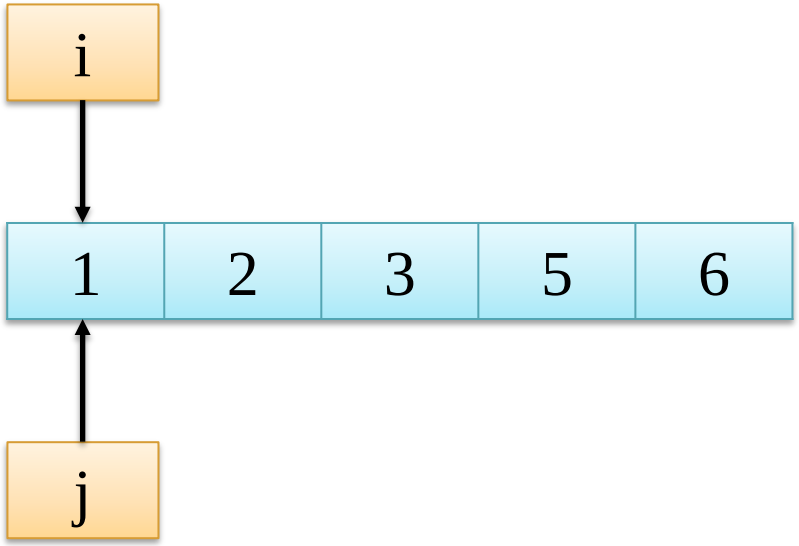
```
2
```

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 0

count: 0

total + array[i] = 0 + 1 = 1 < 5

AI VIET NAM
@aivietnam.edu.vn

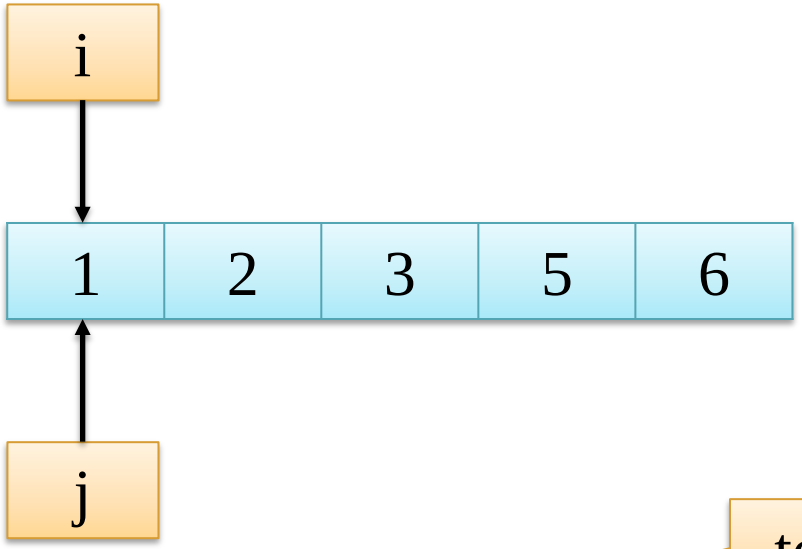
Two Pointer

Find Subarray with Given Sum

Using two pointer

total: 0

count: 0



total + array[i] = 0 + 1 = 1 < 5

total = total + array[i] = 0 + 1 = 1

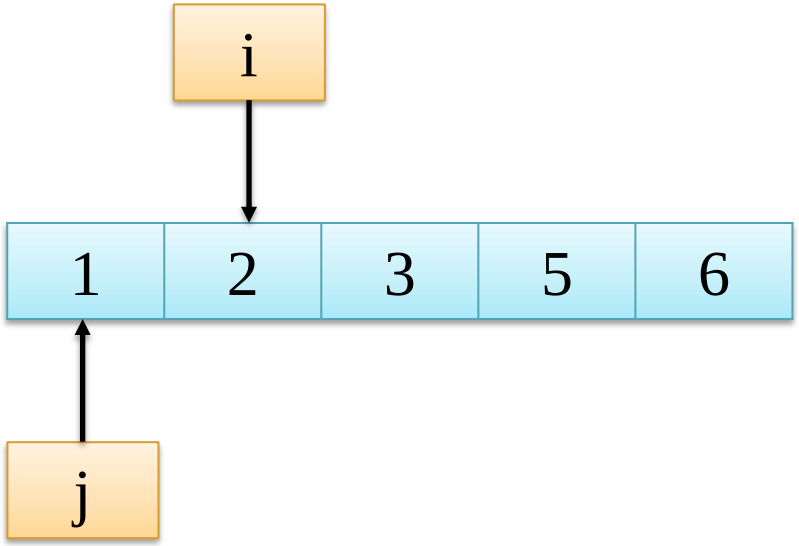
i = i + 1 = 0 + 1 = 1

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 1

count: 0

AI VIET NAM
@aivietnam.edu.vn

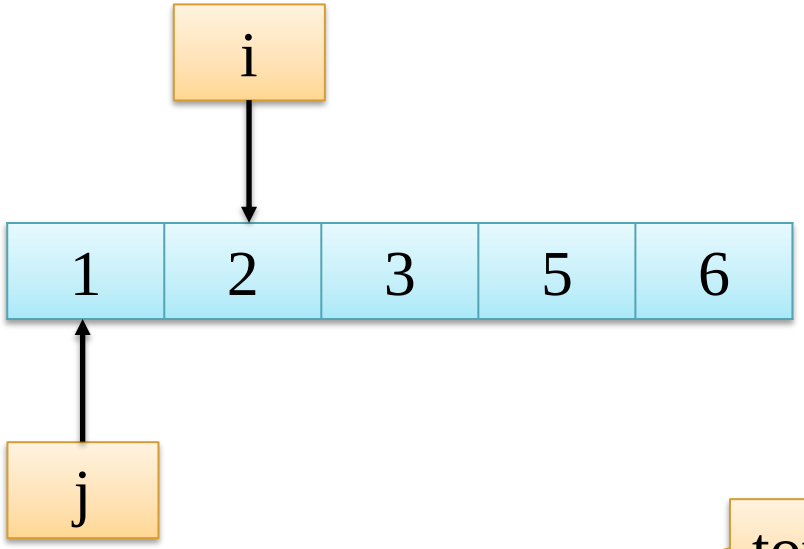
Two Pointer

Find Subarray with Given Sum

Using two pointer

total: 1

count: 0



total = total + array[i] = 1 + 2 = 3

total + array[i] = 1 + 2 = 3 < 5

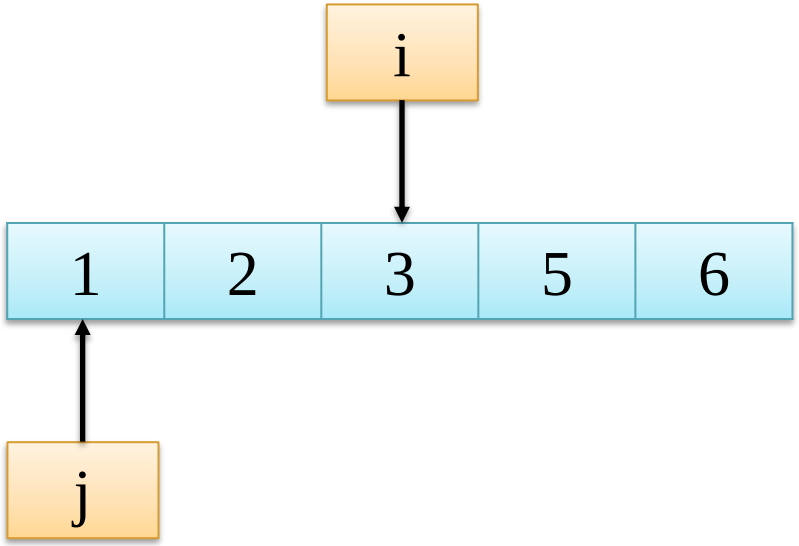
i = i + 1 = 1 + 1 = 2

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 3

count: 0

AI VIET NAM
@aivietnam.edu.vn

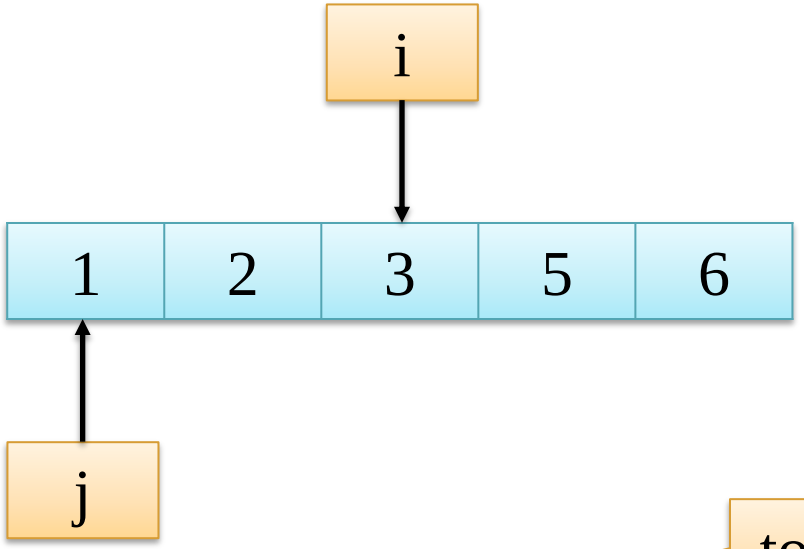
Two Pointer

Find Subarray with Given Sum

Using two pointer

total: 3

count: 0



total = total – array[j] = 3 – 1 = 2

total + array[i] = 3 + 3 = 6 > 5

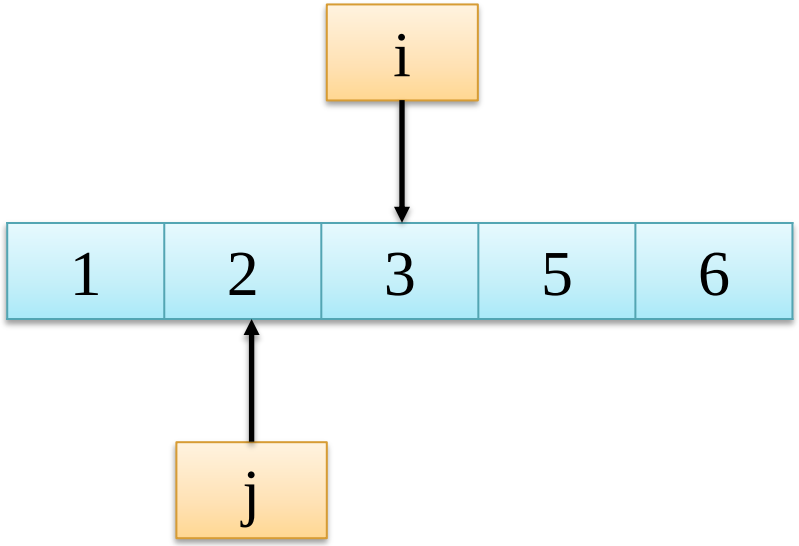
j = j + 1 = 0 + 1 = 1

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 2

count: 0

AI VIET NAM
@aivietnam.edu.vn

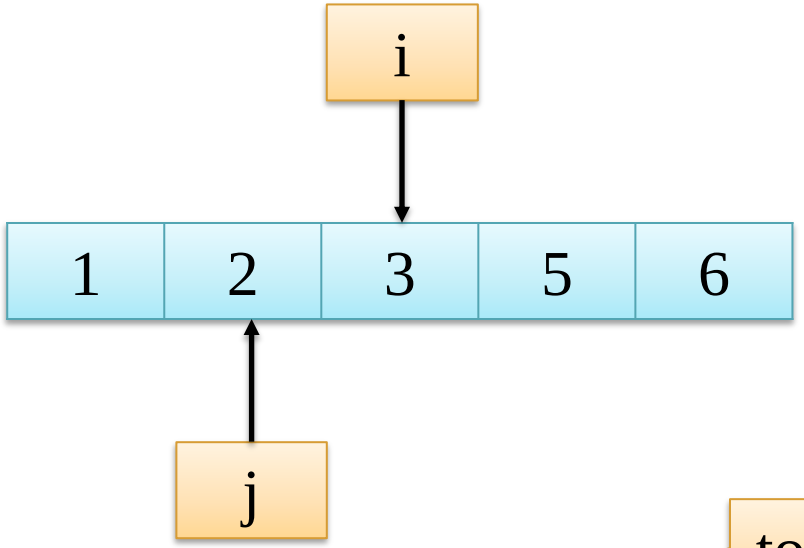
Two Pointer

Find Subarray with Given Sum

Using two pointer

total: 2

count: 0



total + array[i] = 2 + 3 = 5 = 5

total = total + array[i] = 2 + 3 = 5

count = count + 1 = 0 + 1 = 1

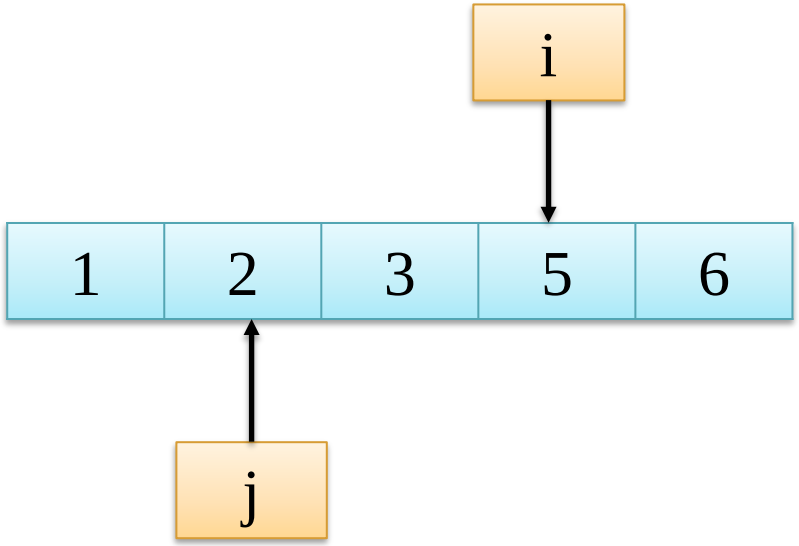
i = i + 1 = 2 + 1 = 3

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 5

count: 1

AI VIET NAM
@aivietnam.edu.vn

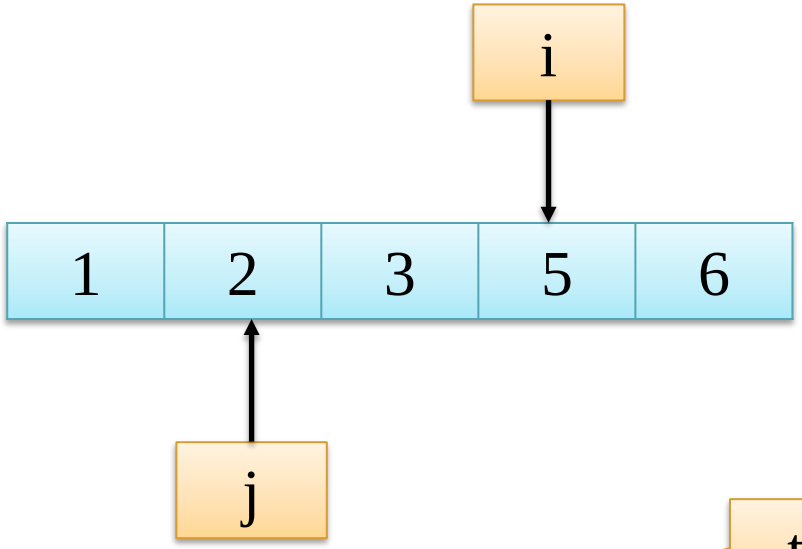
Two Pointer

Find Subarray with Given Sum

Using two pointer

total: 5

count: 1



total + array[i] = 5 + 5 = 10 > 5

total = total - array[j] = 5 - 2 = 3

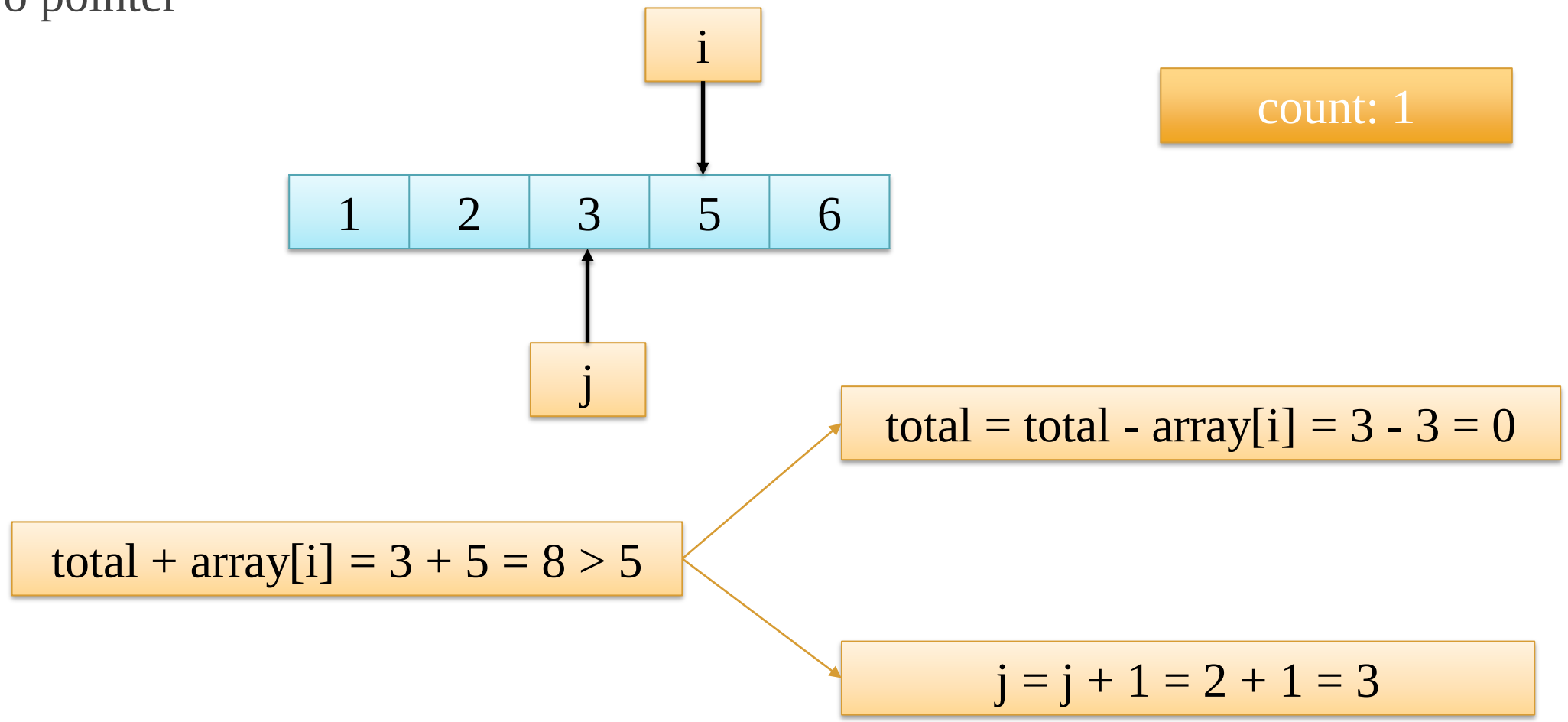
j = j + 1 = 1 + 1 = 2

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

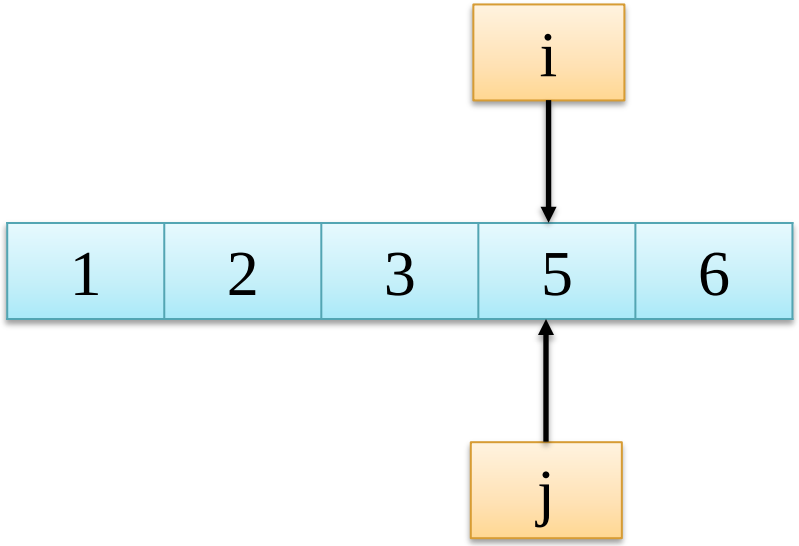


AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 0

count: 1

AI VIET NAM
@aivietnam.edu.vn

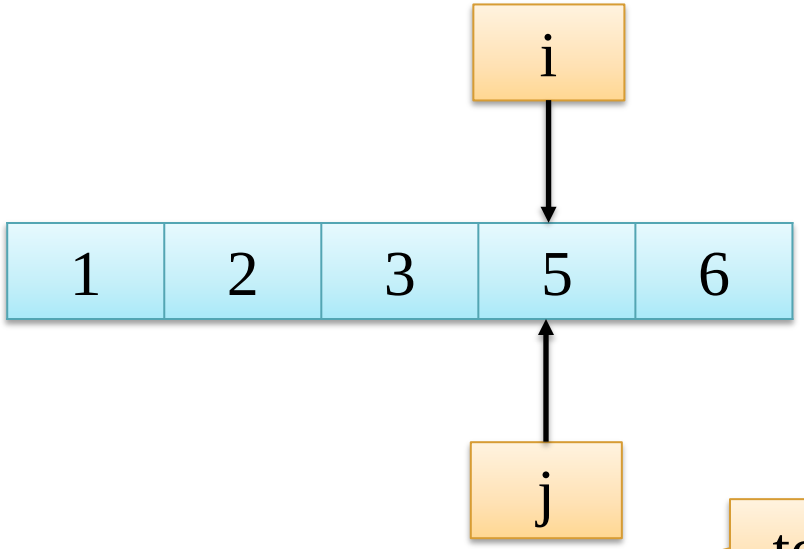
Two Pointer

Find Subarray with Given Sum

Using two pointer

total: 0

count: 1



total = total + array[i] = 0 + 5 = 5

total + array[i] = 0 + 5 = 5 = 5

count = count + 1 = 1 + 1 = 2

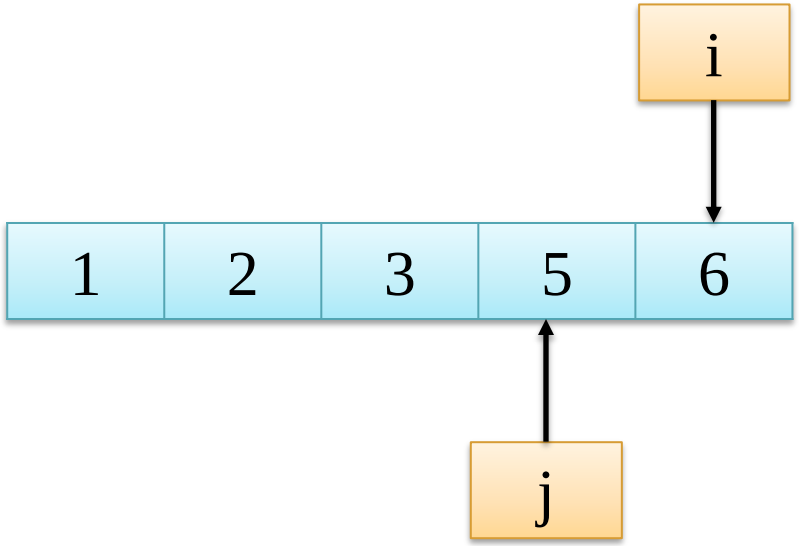
i = i + 1 = 3 + 1 = 4

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 5

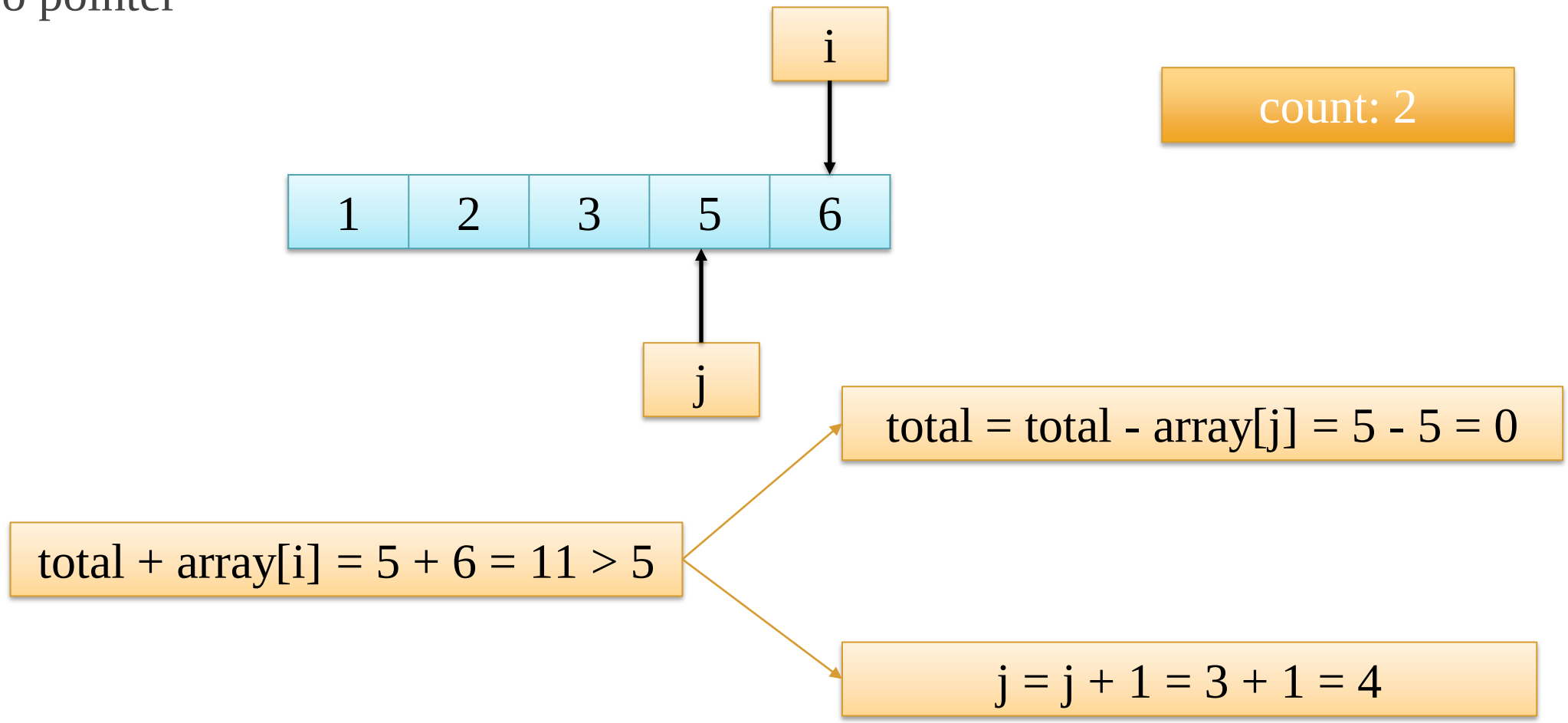
count: 2

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

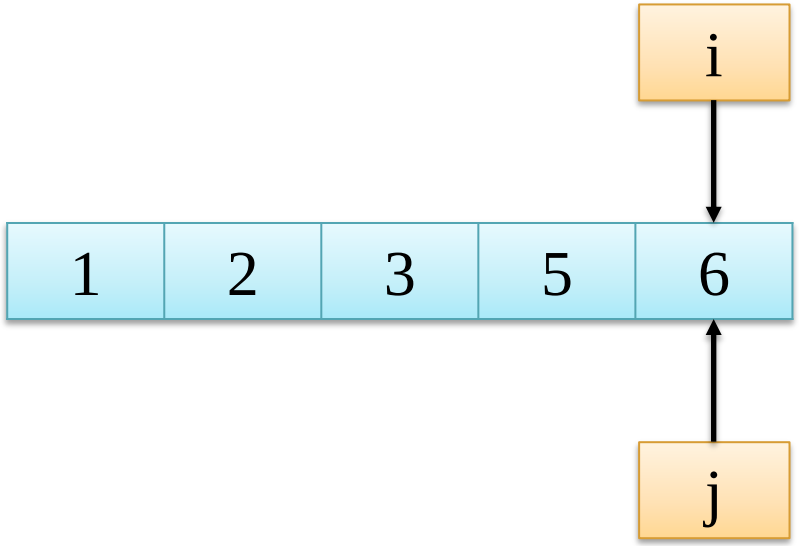


AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 0

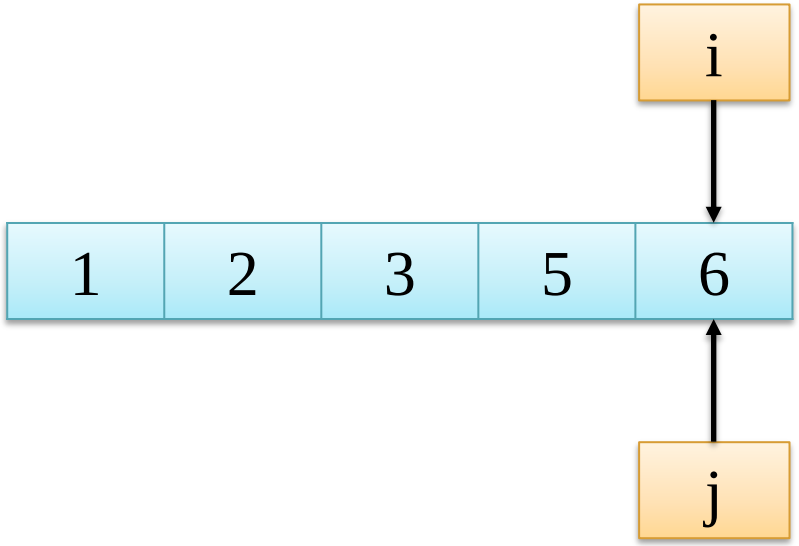
count: 2

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer



total: 0

count: 2

total + array[i] = 0 + 6 = 6 > 5

Stop loop

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

```python
def subarray_sum_two_pointer(nums, key):
    i = 0
    j = 0
    count = 0
    total = 0
    while i < len(nums):
        if total + nums[i] < key:
            total += nums[i]
            i += 1
        elif total + nums[i] > key:
            total -= nums[j]
            j += 1
        else:
            count += 1
            total += nums[i]
            i += 1
    return count

nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum_two_pointer(nums, key)
```

```
2
```

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

T(n) is ?

```python
def subarray_sum_two_pointer(nums, key):
    i = 0
    j = 0
    count = 0
    total = 0
    while i < len(nums):
        if total + nums[i] < key:
            total += nums[i]
            i += 1
        elif total + nums[i] > key:
            total -= nums[j]
            j += 1
        else:
            count += 1
            total += nums[i]
            i += 1
    return count

nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum_two_pointer(nums, key)
```

```
2
```

AI VIET NAM
@aivietnam.edu.vn

Two Pointer

Find Subarray with Given Sum

Using two pointer

T(n) is O(n)

```python
def subarray_sum_two_pointer(nums, key):
    i = 0
    j = 0
    count = 0
    total = 0
    while i < len(nums):
        if total + nums[i] < key:
            total += nums[i]
            i += 1
        elif total + nums[i] > key:
            total -= nums[j]
            j += 1
        else:
            count += 1
            total += nums[i]
            i += 1
    return count

nums = [1, 2, 3, 5, 6]
key = 5
subarray_sum_two_pointer(nums, key)
```

```
2
```

SUMMARY

(1) – Brute Force – Exhaustive
- **Sequential Search**
- **Selection Sort**
- **(Optional) Bubble Sort**

(2) – Recursion
- **Factorial Function**
- **Fibonacci Number**

(3) – Two Pointer
- **Reverse String**
- **Find Subarray With Given Sum**

AI VIET NAM
@aivietnam.edu.vn

(Optional) Exercise

- Leetcode [1] Two Sum
- Leetcode [15] 3Sum
- Leetcode [16] 3Sum Closet
- Leetcode [867] Transpose Matrix

(Optional) Exercise

- Fibonacci Word

  F(0) = A

  F(1) = B

  F(n) = F(n-1) + F(n-2)

  EX: F(2) = F(1) + F(0) = BA => len(F(n)) = 2

  F(3) = F(2) + F(1) = BAB => len(F(n)) = 3

  Given a integer *target*. Returns the value indexed at (*target* - 1) in the string F(n) found, such that the longest length of string F(n) is closest to *target*.

(Optional) Exercise

- Fibonacci Word

  F(0) = A

  F(1) = B

  F(n) = F(n-1) + F(n-2)

  EX: F(2) = F(1) + F(0) = BA => len(F(n)) = 2

  F(3) = F(2) + F(1) = BAB => len(F(n)) = 3

  Given a integer *target*. Returns the value indexed at (*target* - 1) in the string F(n) found, such that the longest length of string F(n) is closest to *target*.

(Optional) Exercise

- Fibonacci Word

Input: target = 2

Output: A

Explanation: target = 2 => n = 2 because F(2) = BA, len(F(2) = 2 >= target => F(2)[target-1) = F(2)[1] = A

Input: target = 4

Output: B

Explanation: target = 4 => n = 4 because F(4) = BABBA, len(F(4) = 5 >= target => F(4)[target-1) = F(4)[3] = B

(Optional) Exercise

- Fibonacci Word

Input: target = 5

Output: A

Explanation: target = 5 => n = 4 because F(4) = BABBA, len(F(4) = 5 >= target => F(4)[target-1) = F(4)[4] = A

Input: target = 7

Output: A

Input: target = 1000000000

Output: A

AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?