AI VIETNAM
All-in-One Course

Algorithm design and analysis

Recursion – Two Pointers

Nguyen Quoc Thai

Year 2023

CONTENT

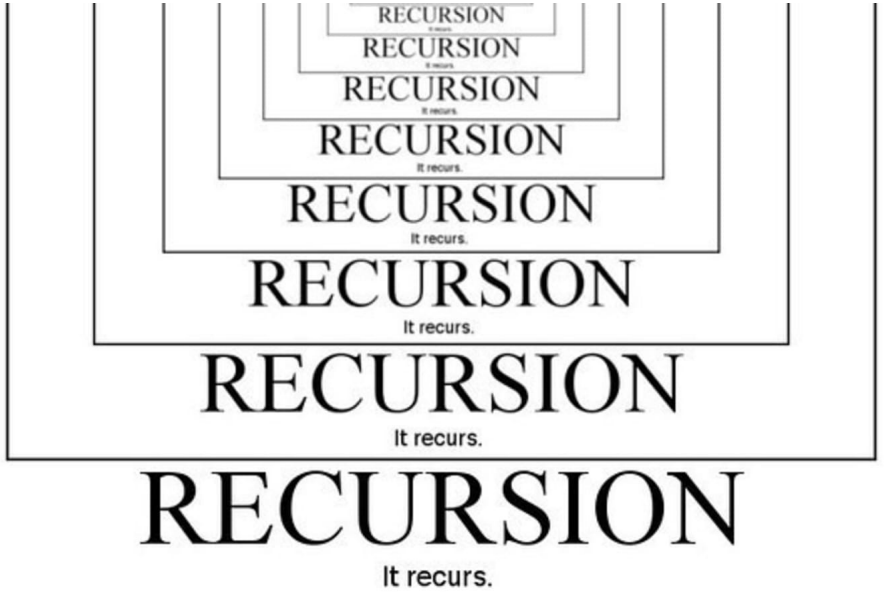
1 – Recursion

- A programming technique that breaks down a complex problem into smaller manageable pieces
- Recursive solutions solve a problem by applying the same algorithm to each piece and then combining the results
- Example:

  Sum Function

  Factorial Function

  Fibonacci Number

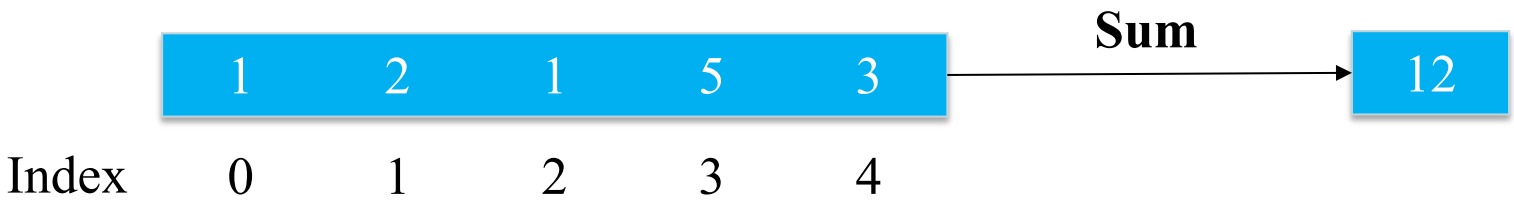


1 – Recursion

1.1. Sum Function

Sum of numbers in a list

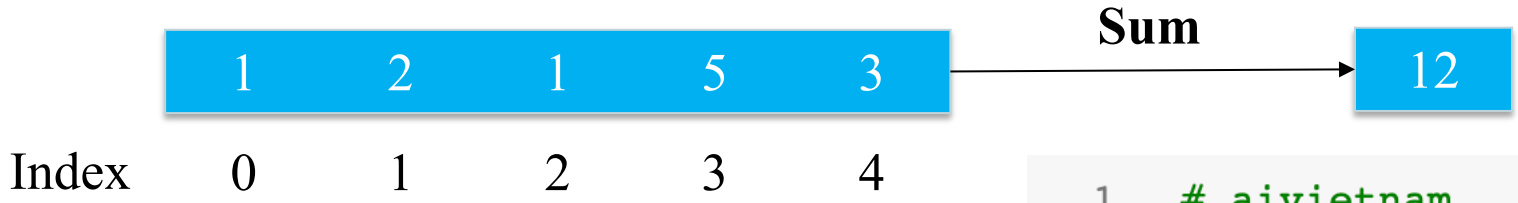
- Input: a list of n integer number $<a_1, a_2, \ldots, a_n>$
- Output: sum of the numbers in the list



1 – Recursion

1.1. Sum Function

Implementing Iteration



Pre-Loop

total → 0

list → 1 2 1 5 3

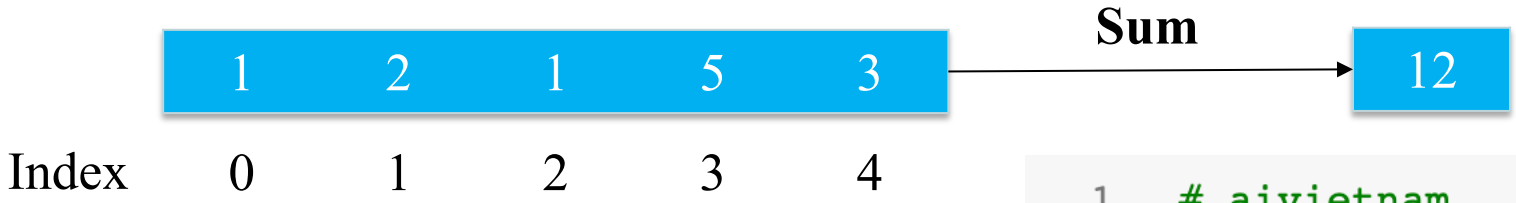
```python
# aivietnam

def sum_list(arr):
    total = 0
    n = len(arr)
    for i in range(n):
        total = total + arr[i]
    return total

arr = [1, 2, 1, 5, 3]
sum_list(arr)
```

12

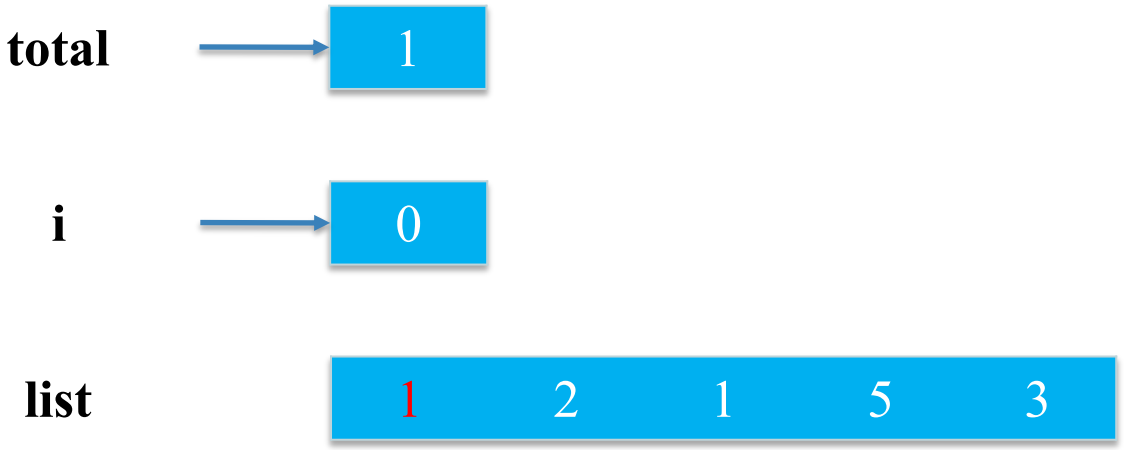
1 – Recursion

1.1. Sum Function

Implementing Iteration



1st iteration



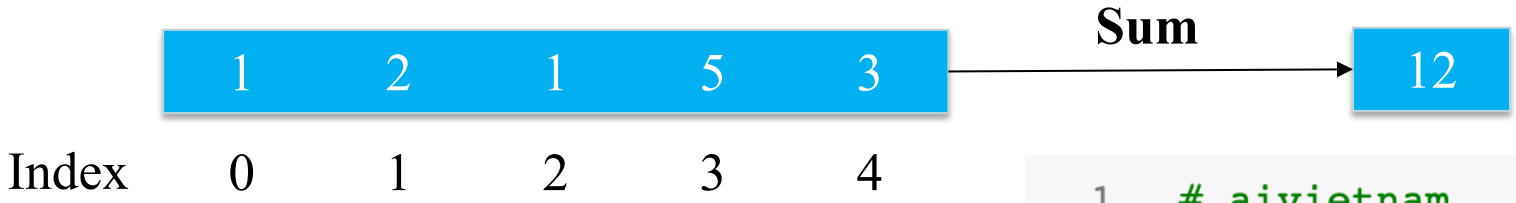

```python
# aivietnam

def sum_list(arr):
    total = 0
    n = len(arr)
    for i in range(n):
        total = total + arr[i]
    return total

arr = [1, 2, 1, 5, 3]
sum_list(arr)
```

```
12
```

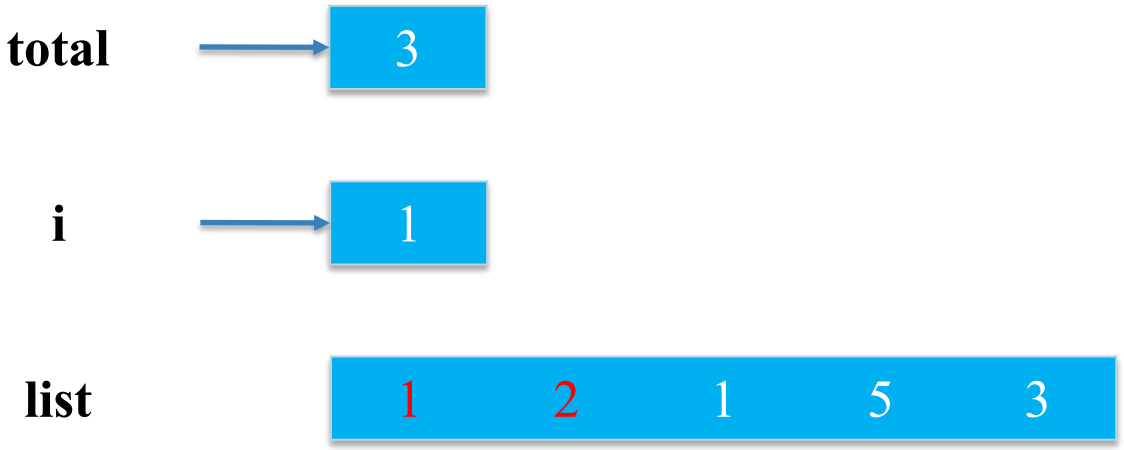
1 – Recursion

1.1. Sum Function

Implementing Iteration



2nd iteration



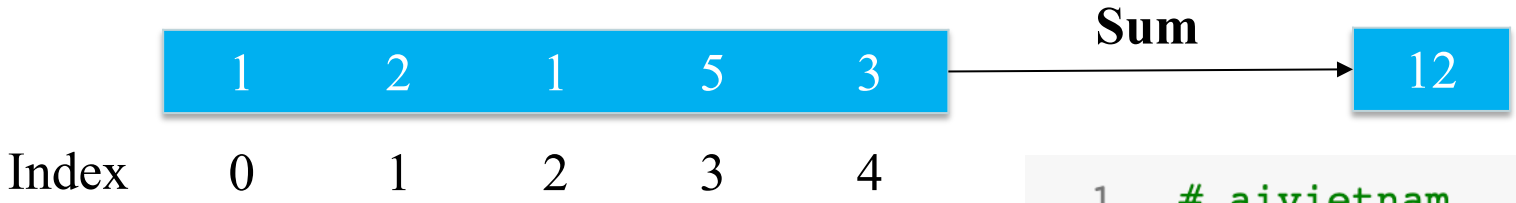

```python
# aivietnam

def sum_list(arr):
    total = 0
    n = len(arr)
    for i in range(n):
        total = total + arr[i]
    return total

arr = [1, 2, 1, 5, 3]
sum_list(arr)
```

```
12
```

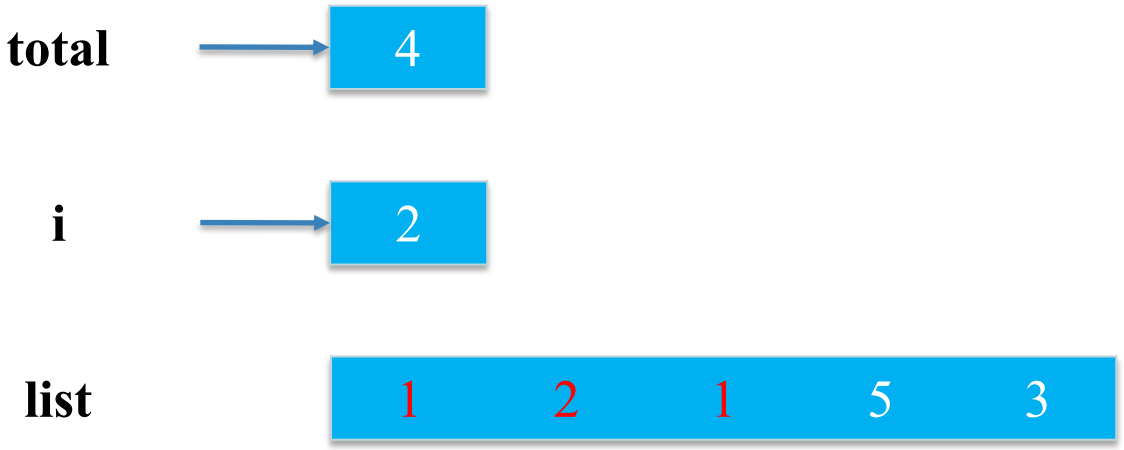
1 – Recursion

1.1. Sum Function

Implementing Iteration



3rd iteration



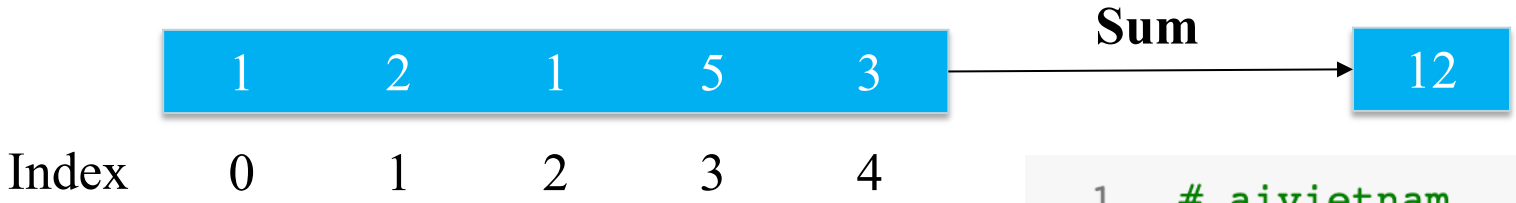

```python
# aivietnam

def sum_list(arr):
    total = 0
    n = len(arr)
    for i in range(n):
        total = total + arr[i]
    return total

arr = [1, 2, 1, 5, 3]
sum_list(arr)
```

```
12
```

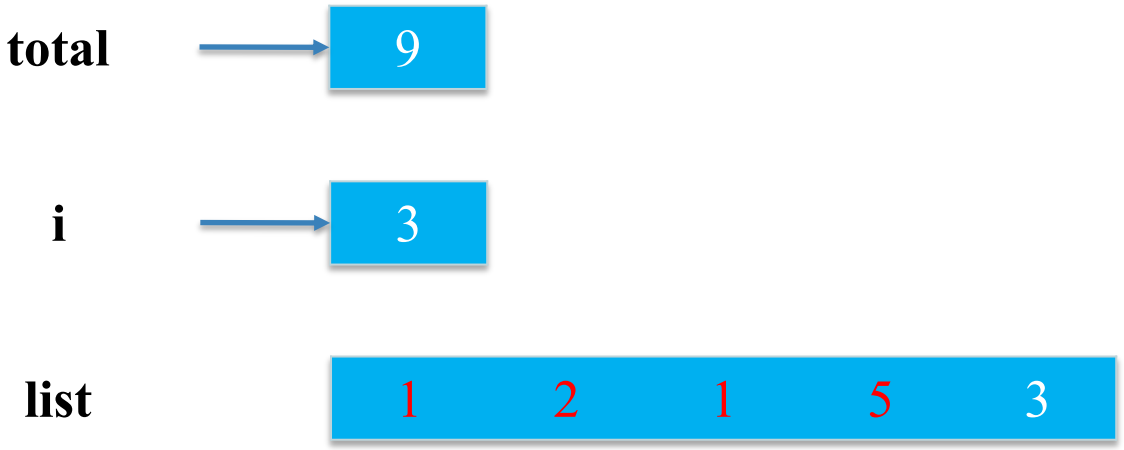
1 – Recursion

1.1. Sum Function

Implementing Iteration



4th iteration



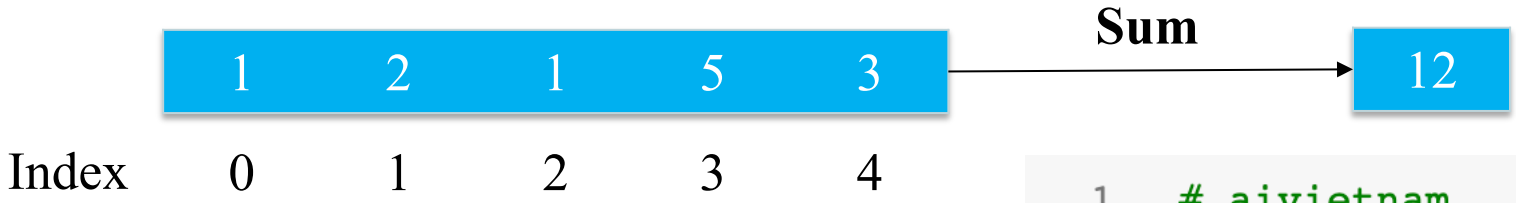

```python
# aivietnam

def sum_list(arr):
    total = 0
    n = len(arr)
    for i in range(n):
        total = total + arr[i]
    return total

arr = [1, 2, 1, 5, 3]
sum_list(arr)
```

```
12
```

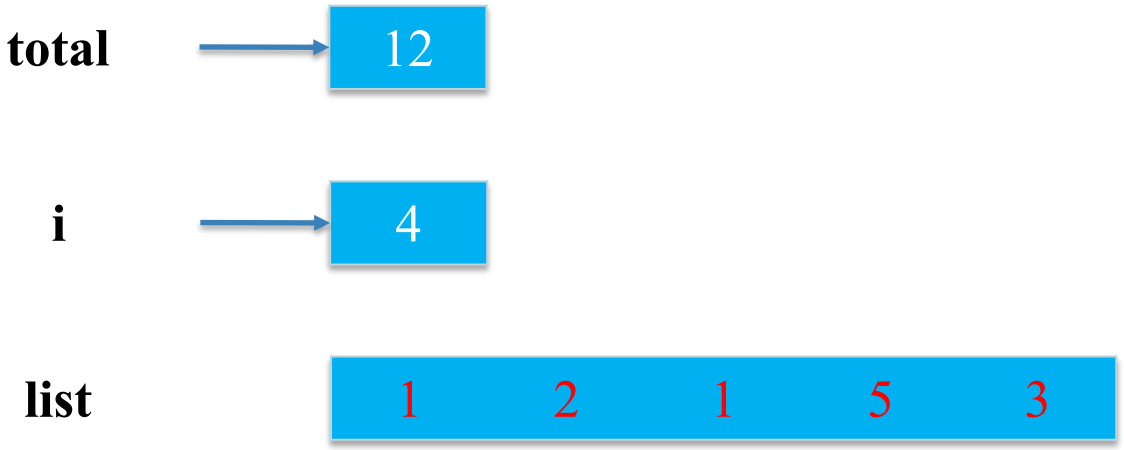
1 – Recursion

1.1. Sum Function

Implementing Iteration



5th iteration




```python
# aivietnam

def sum_list(arr):
    total = 0
    n = len(arr)
    for i in range(n):
        total = total + arr[i]
    return total

arr = [1, 2, 1, 5, 3]
sum_list(arr)
```

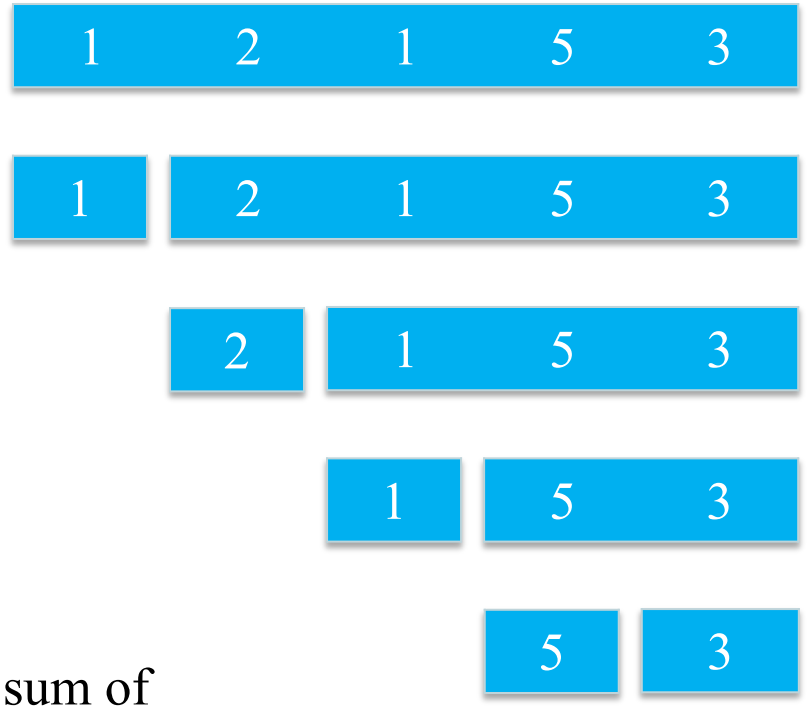
```
12
```

1 – Recursion

1.1. Sum Function

Implementing Recursion

- Break down a complex problem into smaller
- Recursion: a function makes one or more calls to itself during execution



total → 12

1 + 11

Smaller problem: sum of numbers in smaller list

list 1 | 2 1 5 3

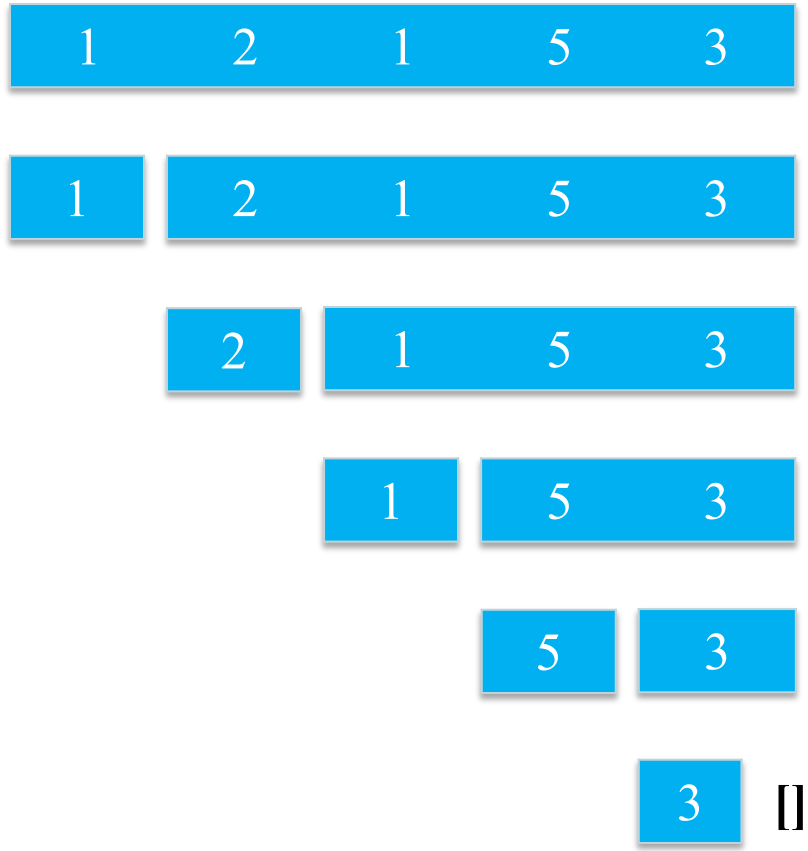
How small problem is enough?

1 – Recursion

1.1. Sum Function

Implementing Recursion

- Break down a complex problem into smaller
- Recursion: a function makes one or more calls to itself during execution
- Base case: solve the problem without recursion
- Combining the results



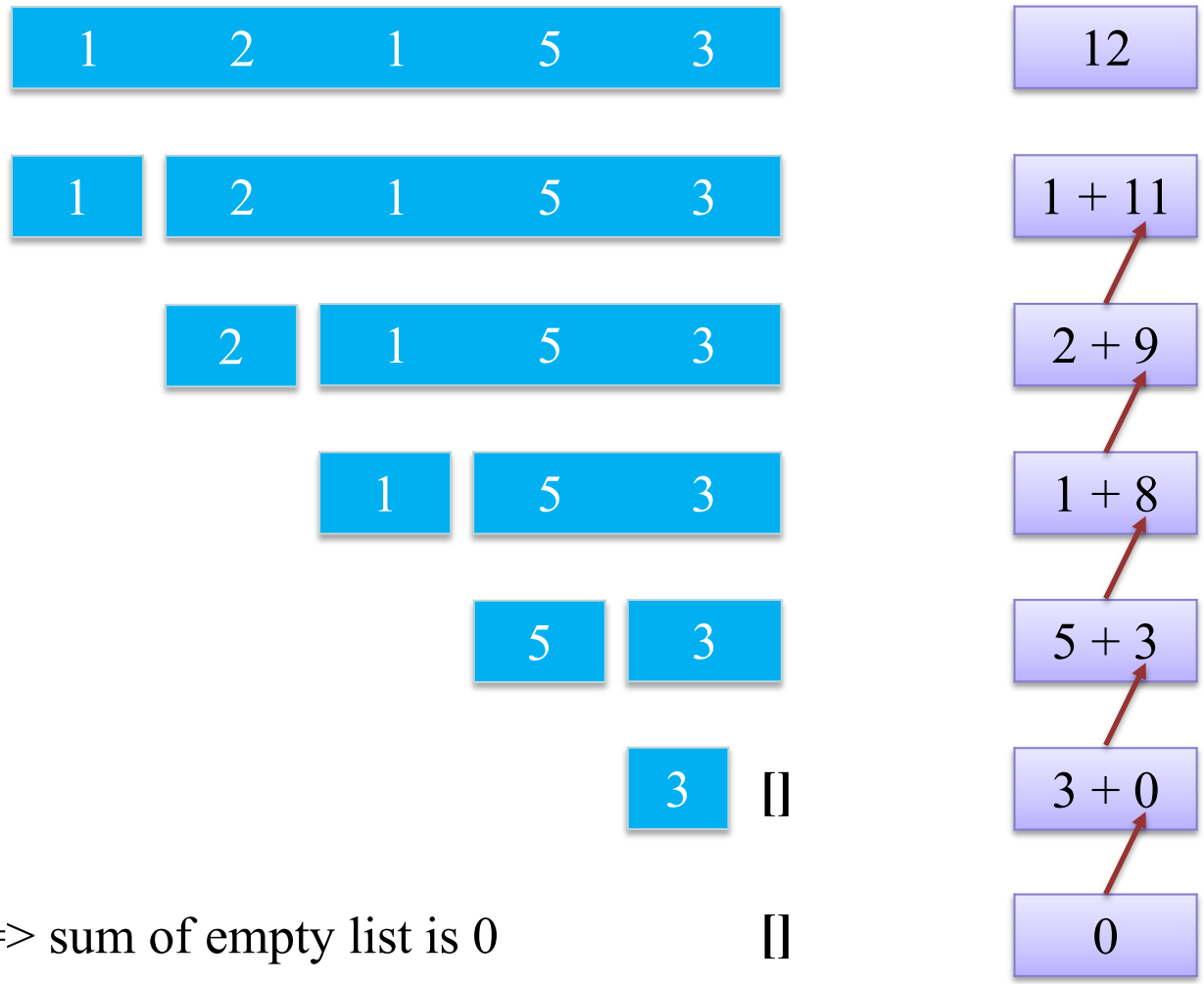
Base case: empty list => sum of empty list is 0 []

1 – Recursion

1.1. Sum Function

Implementing Recursion

- Combining the results
- From base case



Base case: empty list => sum of empty list is 0

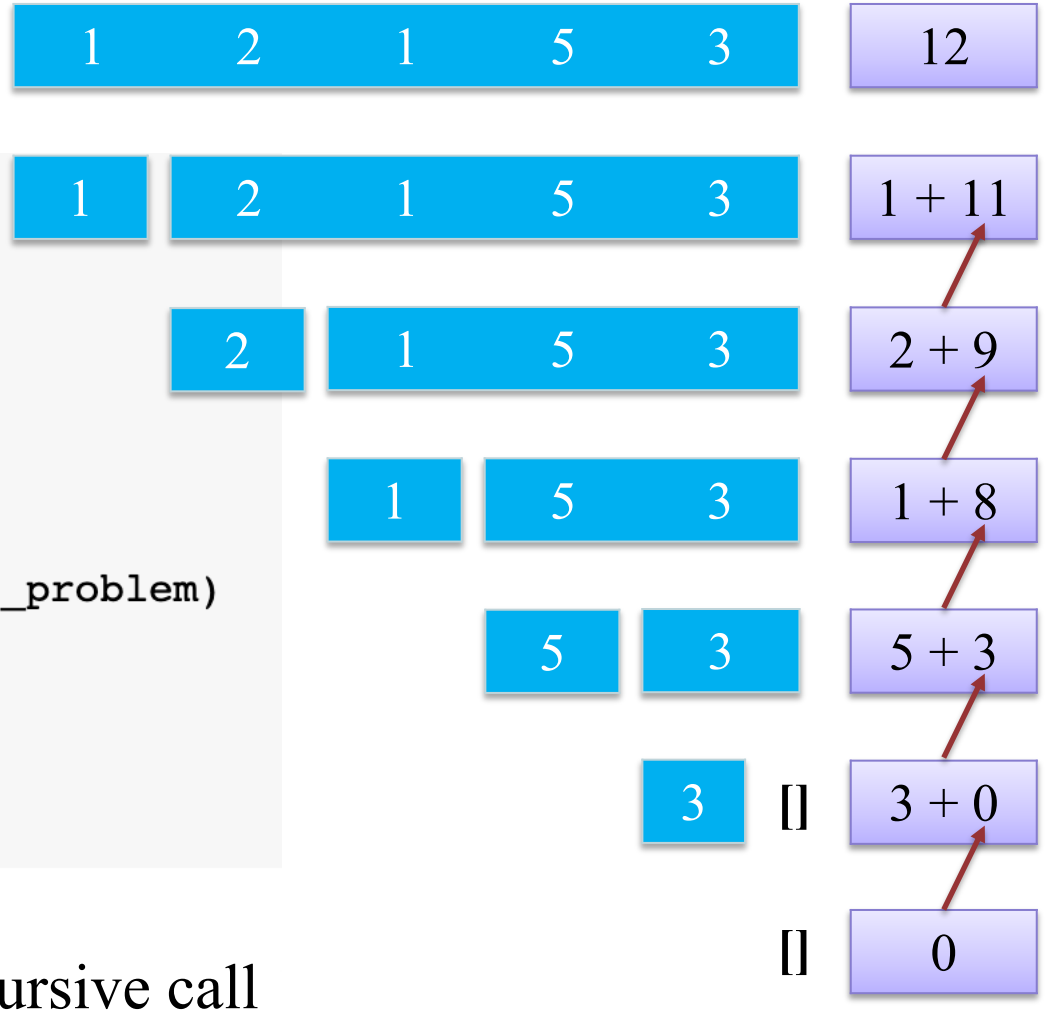
1 – Recursion

1.1. Sum Function

Implementing Recursion

```
1   # aivietnam
2
3   def sum_recursion(arr):
4       if arr == []:
5           return 0
6       else:
7           smaller_problem = arr[1:]
8           smaller_result = sum_recursion(smaller_problem)
9           return arr[0] + smaller_result
10
11  arr = [1, 2, 1, 5, 3]
12  sum_recursion(arr)
```

```
12
```



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

1 – Recursion

1.1. Sum Function

(1) Base case (non-recursive)

One or more simple cases that can be solved directly (Avoid infinite recursion)

(2) Recursive case

One or more simple cases that require solving "simpler" version

Call to itself with smaller instance size

```
# aivietnam

def recursive_function(problem):
    #base case
    if problem is base_case:
        return ___
    else:
        # smaller intance size
        smaller_problem = ___
        # solve the smaller problem
        smaller_result = recursive_function(smaller_problem)
        # combine with the smaller problem
        return ___
```

1 – Recursion

1.2. Factorial Function

- For any integer $n \geq 0$

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

- Example:

$0! = 1$

$1! = 1$

$3! = 3 \times 2 \times 1 = 6$

$4! = 4 \times 3 \times 2 \times 1 = 24$

1 – Recursion

1.2. Factorial Function

(1) Base case (non-recursive)

One or more simple cases that can be solved directly (Avoid infinite recursion)

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

(2) Recursive case

One or more simple cases that require solving “simpler” version

Call to itself with smaller instance size

AI VIET NAM
@aivietnam.edu.vn

1 – Recursion

1.2. Factorial Function

(1) Base case (non-recursive)

One or more simple cases that can be solved directly (Avoid infinite recursion)

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

(2) Recursive case

One or more simple cases that require solving "simpler" version

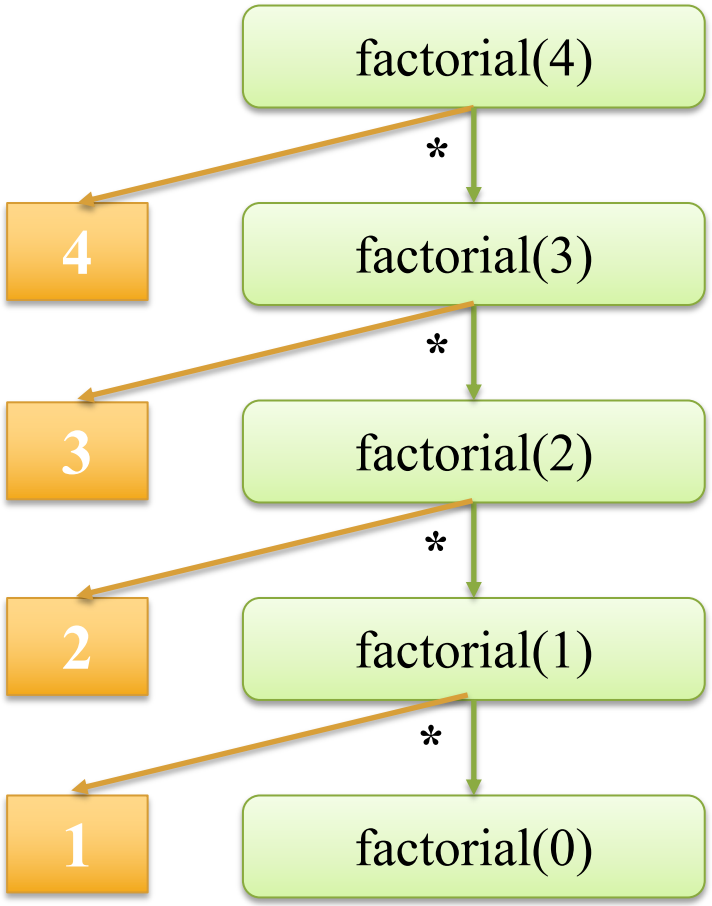
Call to itself with smaller instance size

```python
# aivietnam

def factorial(n):
    if n == 0:
        return 1
    else:
        smaller_fac = n-1
        smaller_res = factorial(smaller_fac)
        return n * smaller_res

n = 4
factorial(n)
```

```
24
```

1 – Recursion

1.2. Factorial Function

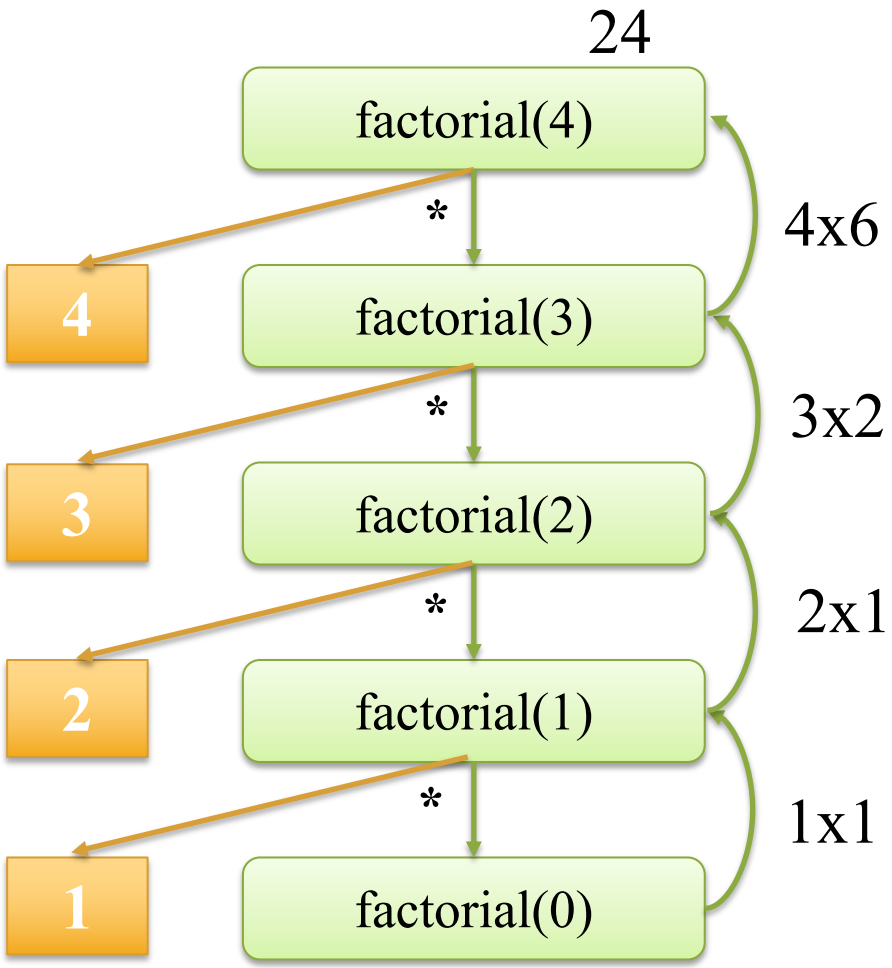



```python
# aivietnam

def factorial(n):
    if n == 0:
        return 1
    else:
        smaller_fac = n-1
        smaller_res = factorial(smaller_fac)
        return n * smaller_res

n = 4
factorial(n)
```

```
24
```

1 – Recursion

1.2. Factorial Function

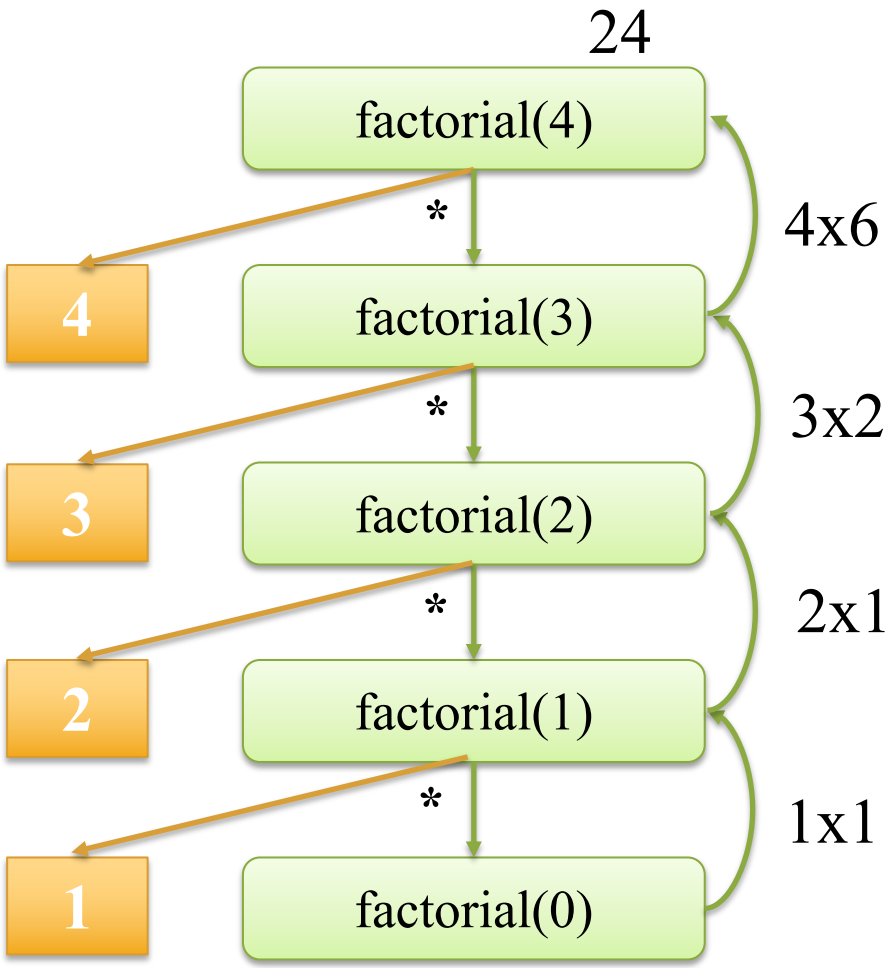



```python
# aivietnam

def factorial(n):
    if n == 0:
        return 1
    else:
        smaller_fac = n-1
        smaller_res = factorial(smaller_fac)
        return n * smaller_res

n = 4
factorial(n)
```

```
24
```

1 – Recursion

1.2. Factorial Function




```python
# aivietnam

def factorial(n):
    if n == 0:
        return 1
    else:
        smaller_fac = n-1
        smaller_res = factorial(smaller_fac)
        return n * smaller_res

n = 4
factorial(n)
```

```
24
```

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

1 – Recursion

1.3. Fibonacci Number

- For any integer $n \geq 0$

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

- Example:

$F(0) = 0$

$F(1) = 1$

$F(2) = F(1) + F(0) = 1 + 0 = 1$

$F(3) = F(2) + F(1) = 1 + 1 = 2$

1 – Recursion

1.3. Fibonacci Number

(1) Base case (non-recursive)

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 2 \end{cases}$$

(2) Recursive case

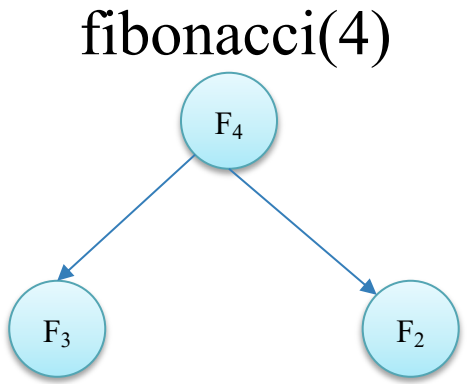
Small instance size: (n-1) and (n-2)

```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

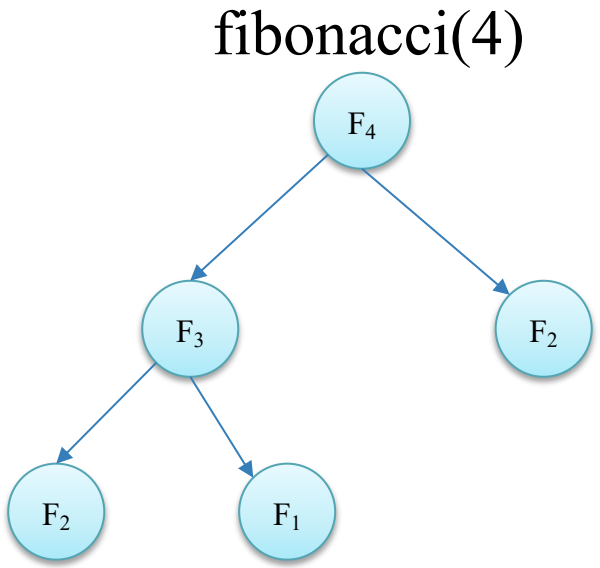



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

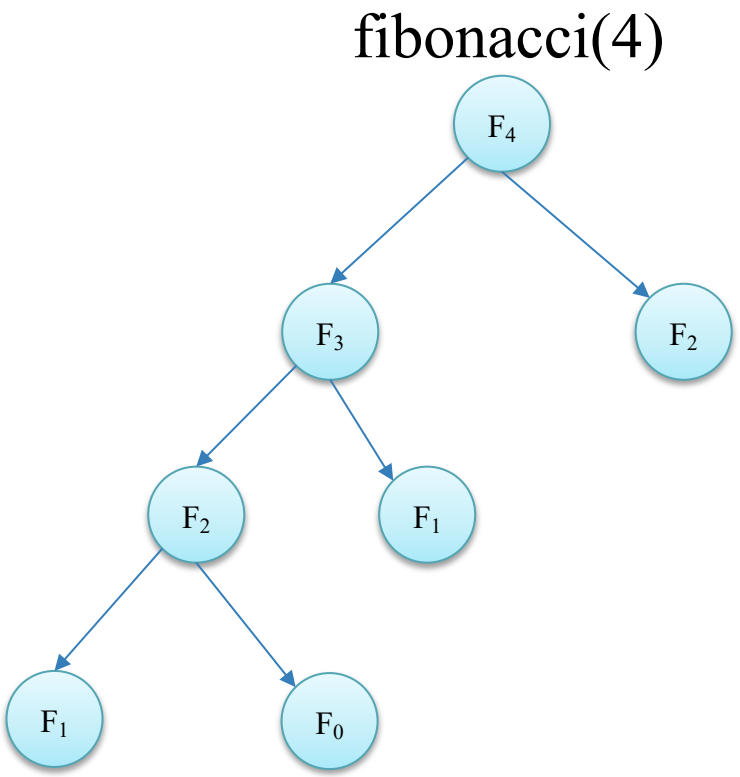



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

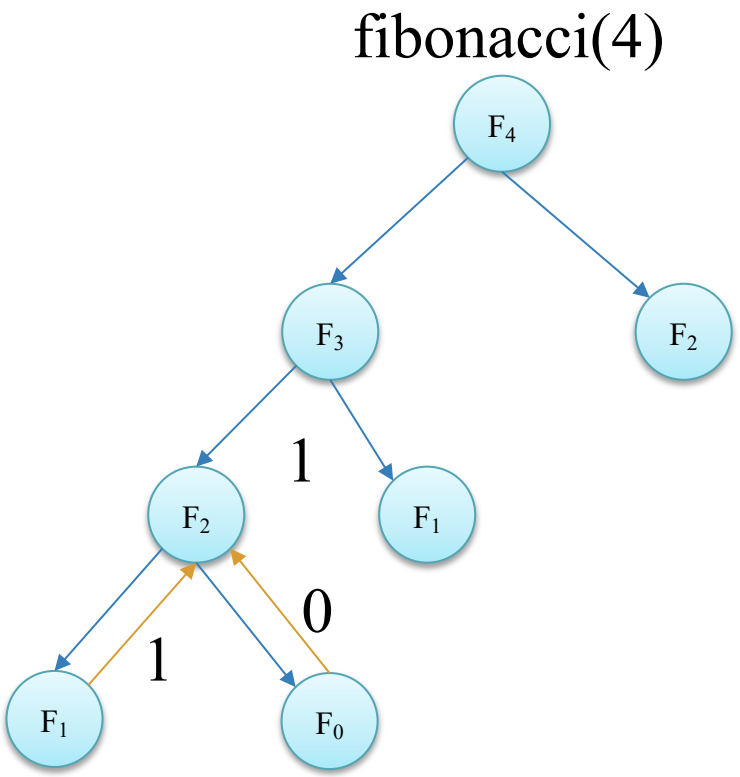



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

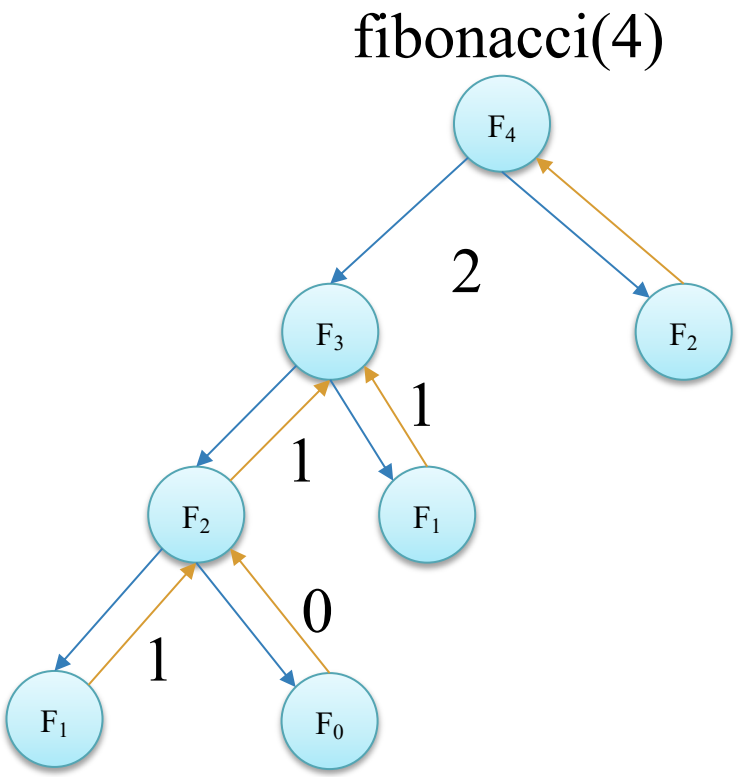



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

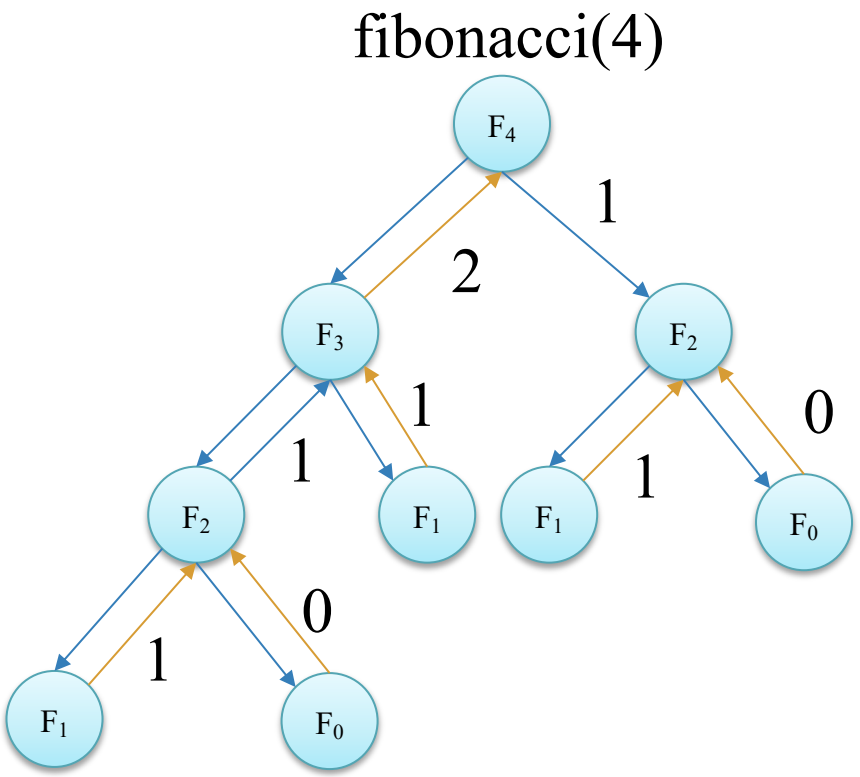



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

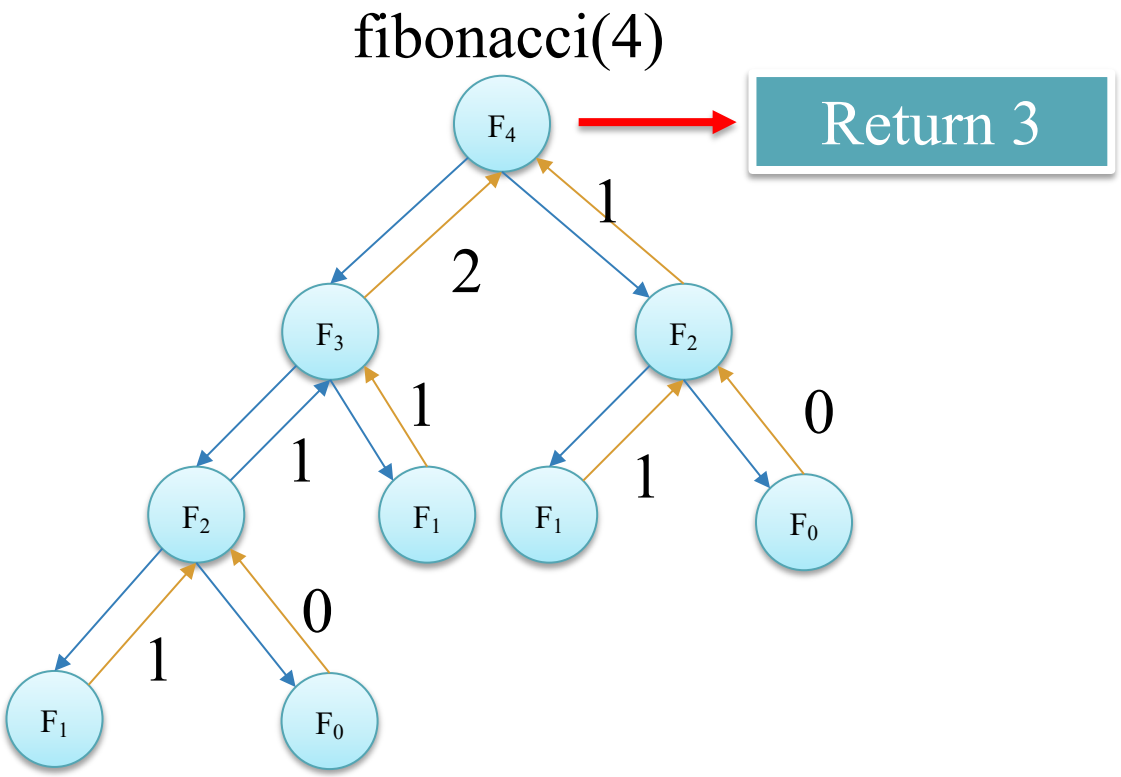



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number

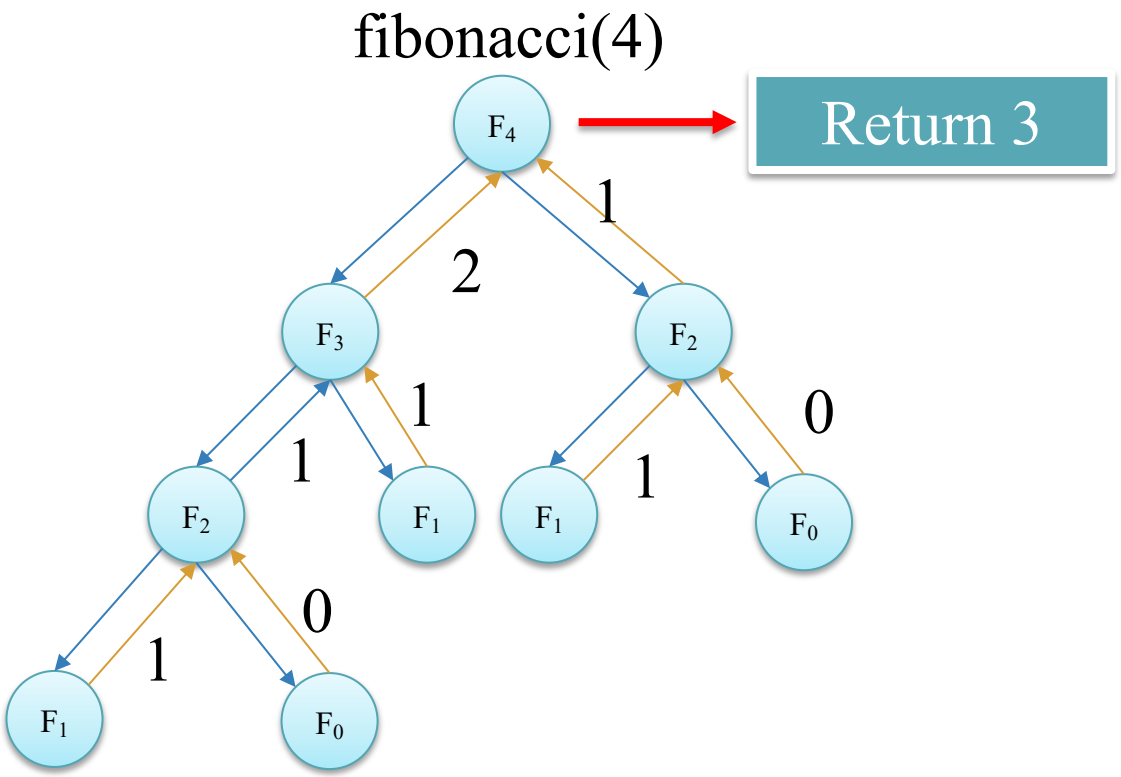



```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

1 – Recursion

1.3. Fibonacci Number



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

```python
# aivietnam

def fibonacci(n):
    # base case
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # recursive func
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 4
fibonacci(n)
```

```
3
```

2 – Two Pointers

2.1. Two Pointers Technique

Access Python List Elements

	1	2	1	5	3	
Positive Index	0	1	2	3	4	
	-5	-4	-3	-2	-1	Negative Index

```
1   #aivietnam
2
3   arr = [1, 2, 1, 5, 3]
4   print(arr[0], arr[-5])
5   print(arr[1], arr[-4])
6   print(arr[4], arr[-1])
```

```
1 1
2 2
3 3
```

2 – Two Pointers

2.1. Two Pointers Technique

- Technique pointer represent either **index** or an iteration attribute like node's next

  => Access, keep track of array or string indices

"Two are better than one if they act as one"

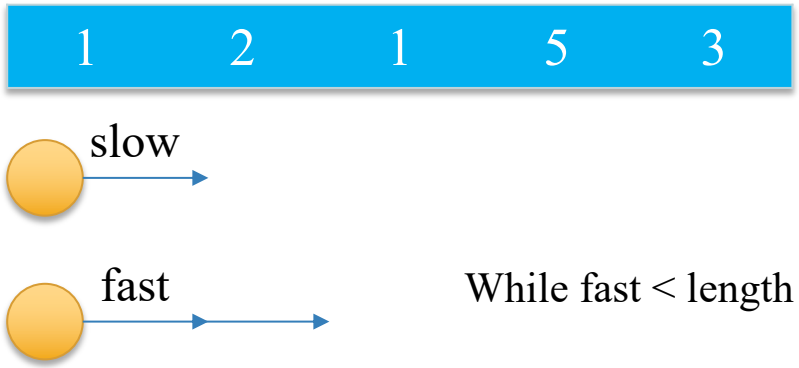
AI VIET NAM
@aivietnam.edu.vn

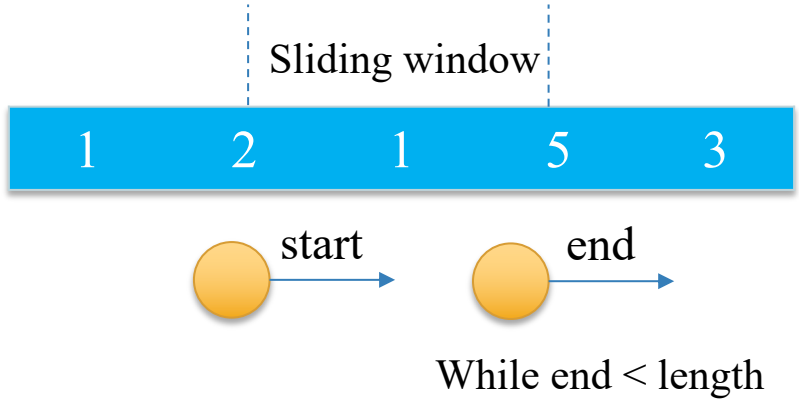
2 – Two Pointers

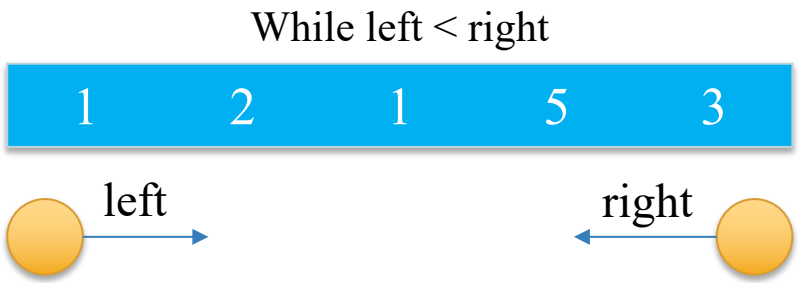
2.1. Two Pointers Technique

Three main steps

- Pointer Initialization
- Pointer movement
- Stop condition







2 – Two Pointers

2.2. Reverse String [Leetcode 344]

Write a function that reverses a string

Input	Output
“hello”	“olleh”
“Hannah”	“hannaH”

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.

2 – Two Pointers

2.2. Reverse String [Leetcode 344]

Using for loop

T(n) is O(n)

```python
# aivietnam

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

s = 'hello'
print(reverse_str_using_for(s))
s = 'Hannah'
print(reverse_str_using_for(s))
```

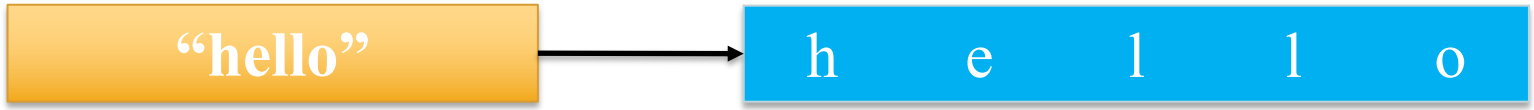
```
olleh
hannaH
```

2 – Two Pointers

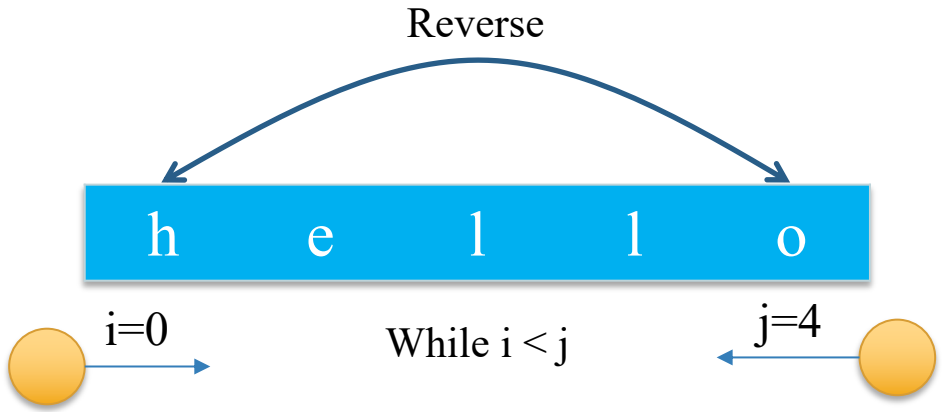
2.2. Reverse String [Leetcode 344]

Using Two Pointer

- Convert string to a list



- Pointer Initialization: "left-right" pointers
- Pointer Movement
- Stop Condition



- Convert the list to string

2 – Two Pointers

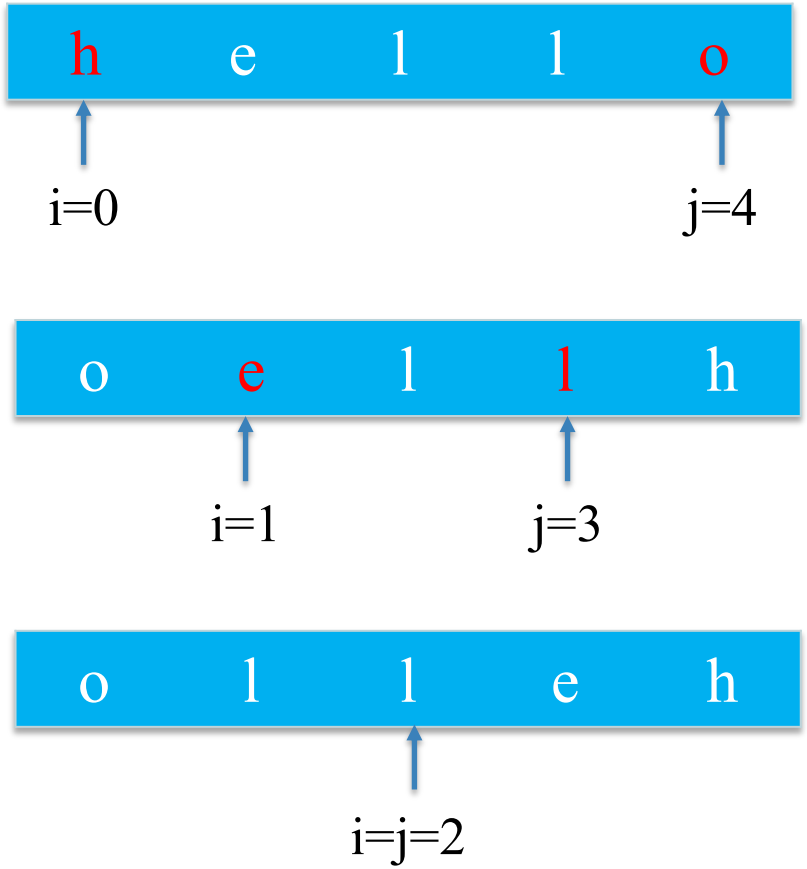
2.2. Reverse String [Leetcode 344]

Using Two Pointer

```
1   # aivietnam
2
3   def reverse_str_two_pointers(s):
4       s = list(s)
5       i = 0
6       j = len(s) - 1
7       while i < j:
8           s[i], s[j] = s[j], s[i]
9           i = i + 1
10          j = j - 1
11      return "".join(s)
12
13  s = 'hello'
14  print(reverse_str_two_pointers(s))
15  s = 'Hannah'
16  print(reverse_str_two_pointers(s))
```

```
olleh
hannaH
```

T(n) is O(n)



2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Given an array of integers *nums* (nums[i] >= 0) 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
```

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using for loop

Key = 5

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

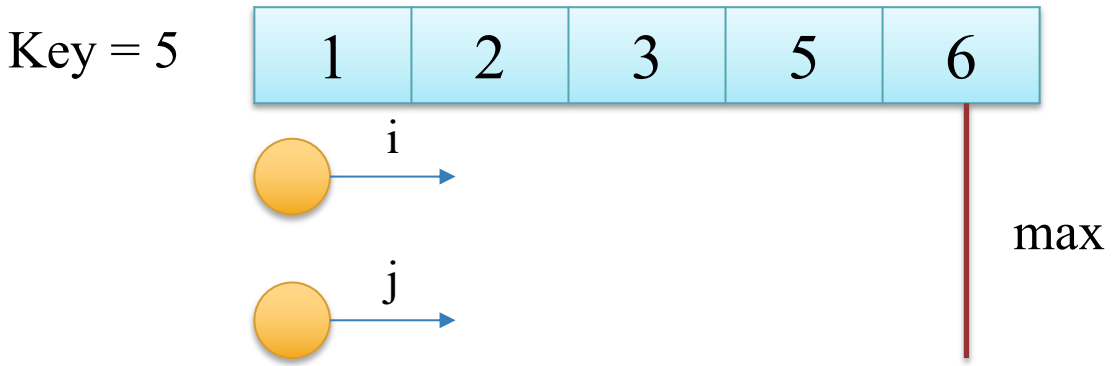
```python
# aivietnam

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 Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

- Pointer Initialization
- Pointer Movement
- Stop Condition



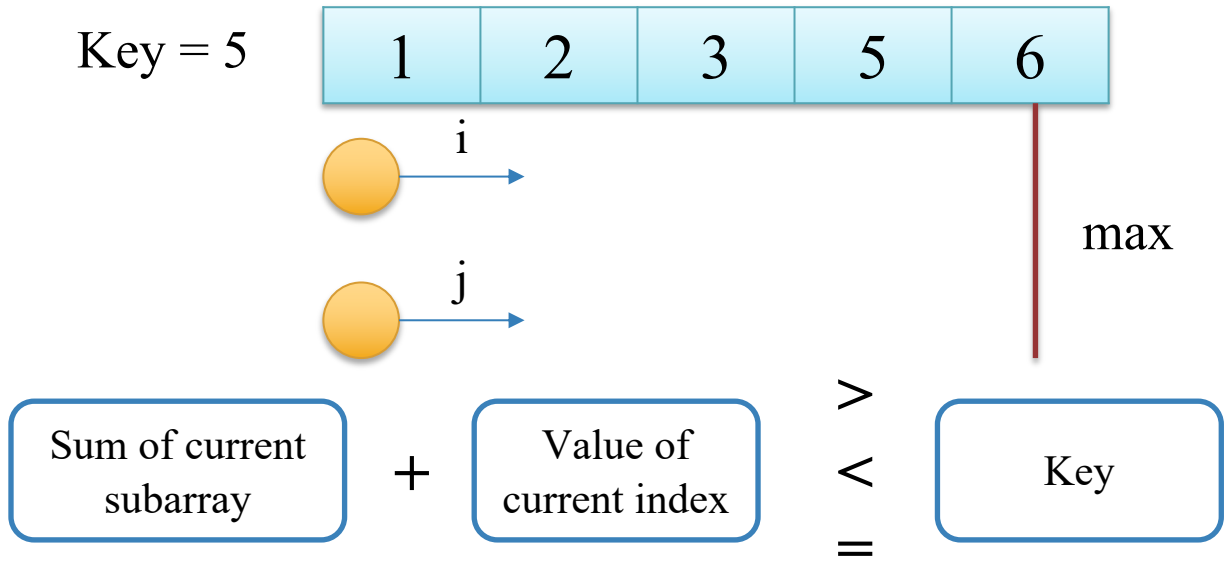

```python
# aivietnam

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 Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

- Pointer Initialization
- Pointer Movement
- Stop Condition



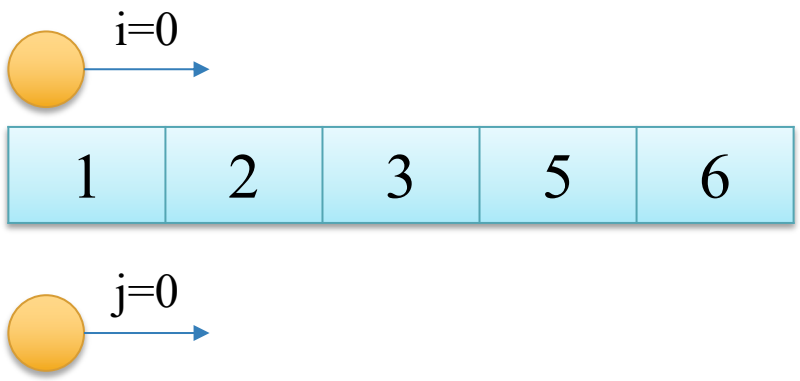

```python
# aivietnam

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

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 0

count = 0

2 – Two Pointers

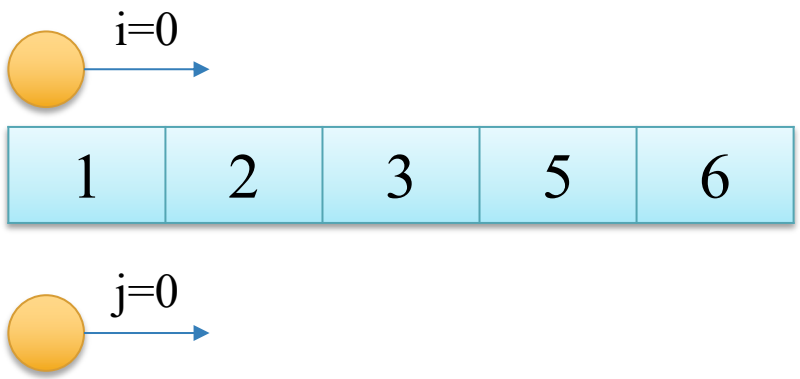
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 0

count = 0



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

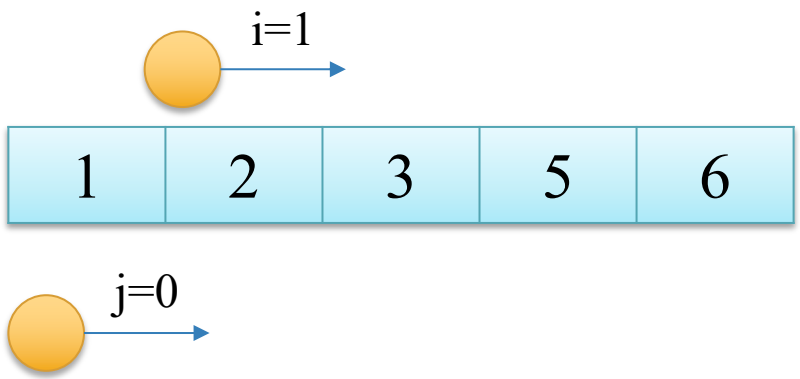
total = total + array[i] = 0 + 1 = 1

i = i + 1 = 0 + 1 = 1

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 1

count = 0

2 – Two Pointers

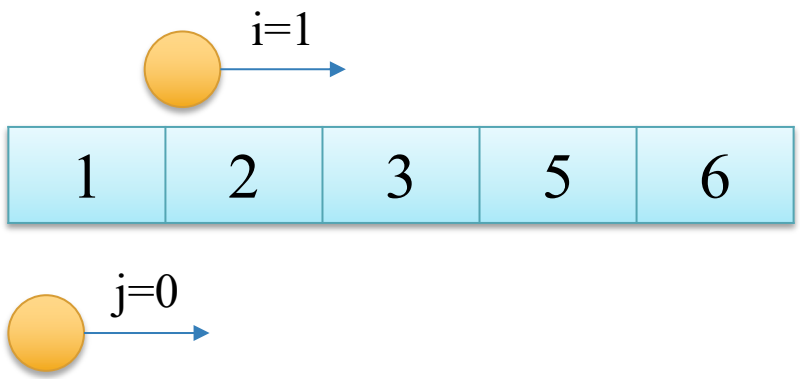
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 1

count = 0



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

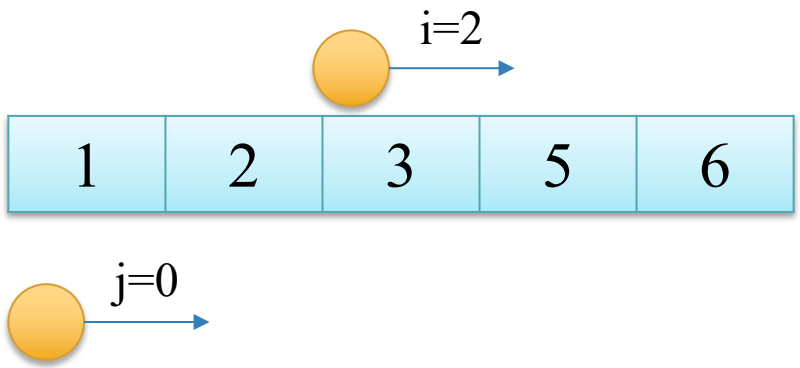
total = total + array[i] = 1 + 2 = 3

i = i + 1 = 1 + 1 = 2

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 3

count = 0

2 – Two Pointers

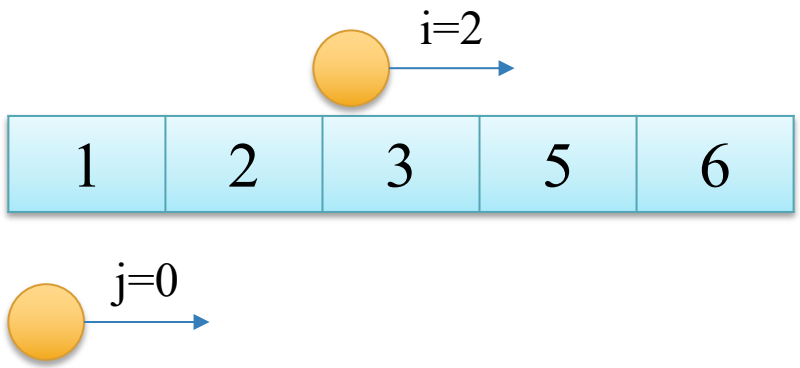
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 3

count = 0



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

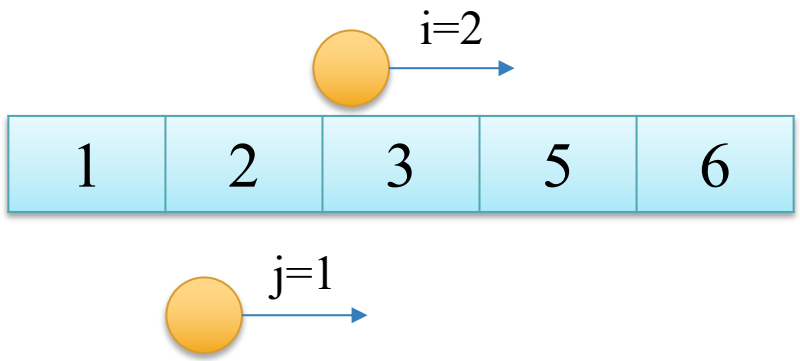
total = total - array[j] = 3 - 1= 2

j = j + 1 = 0 + 1 = 1

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 2

count = 0

2 – Two Pointers

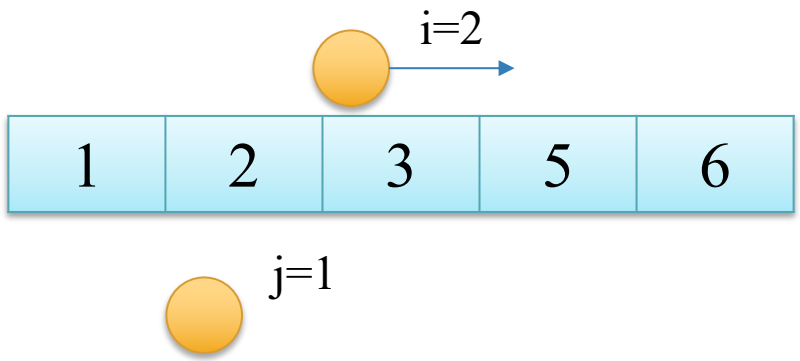
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 2

count = 0



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

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

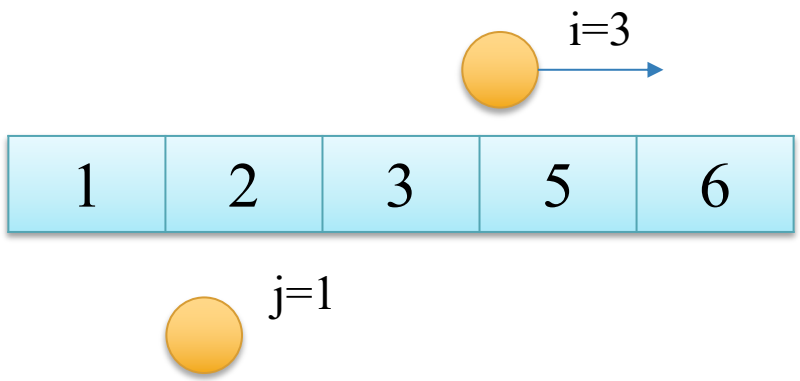
count = count + 1 = 0 + 1 = 1

i = i + 1 = 2 + 1 = 3

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 5

count = 1

2 – Two Pointers

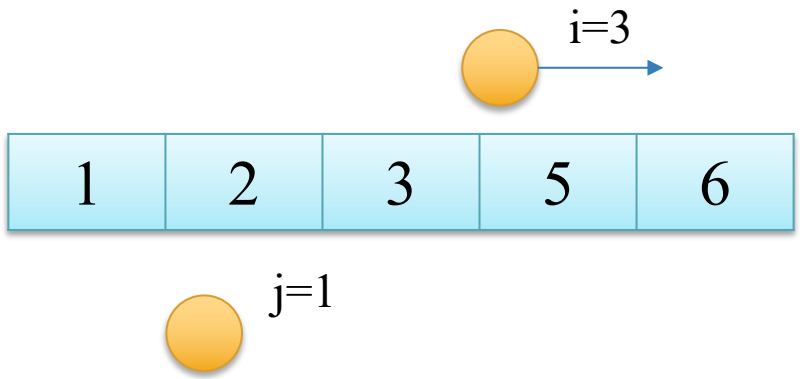
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 5

count = 1



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

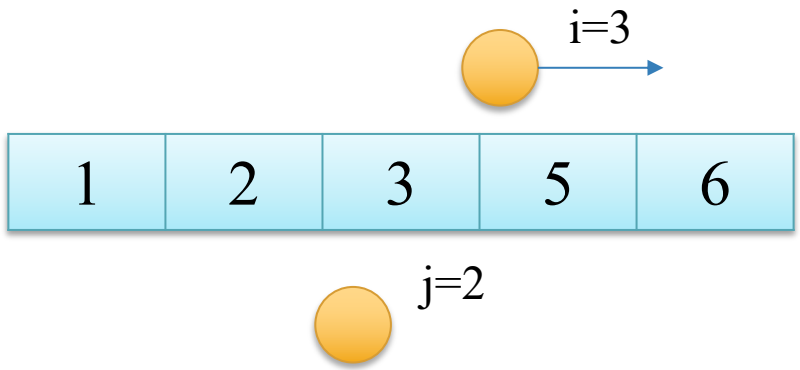
total = total - array[j] = 5 - 2 = 3

j = j + 1 = 1 + 1 = 2

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 3

count = 1

2 – Two Pointers

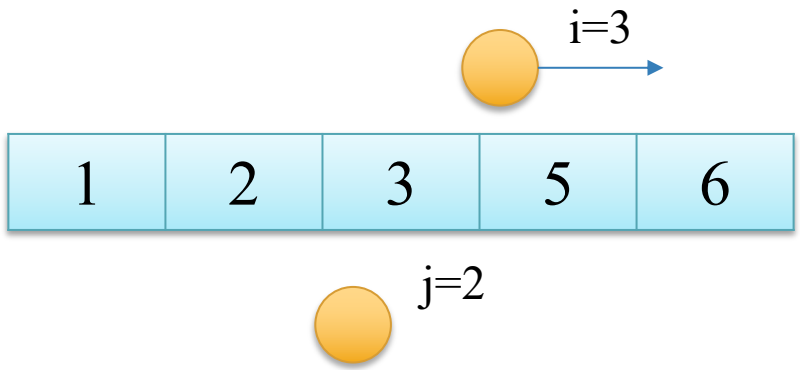
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 3

count = 1



total + array[i] = 3 + 5 = 8 > 5

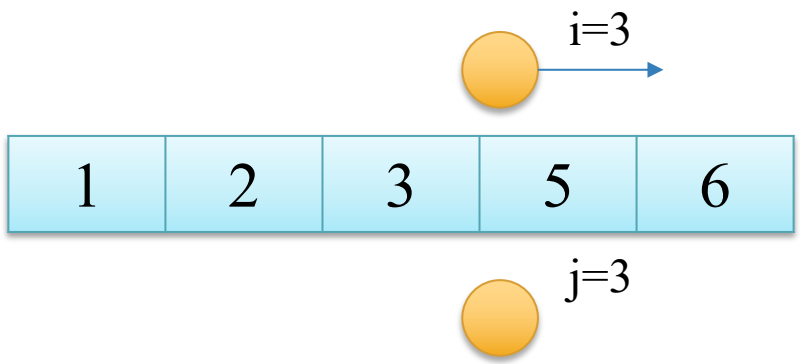
total = total - array[j] = 3 - 3 = 0

j = j + 1 = 2 + 1 = 3

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer



Key = 5

total = 0

count = 1

2 – Two Pointers

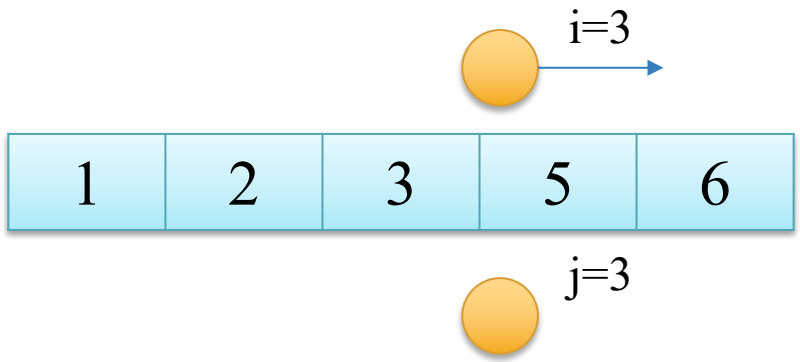
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 0

count = 1



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

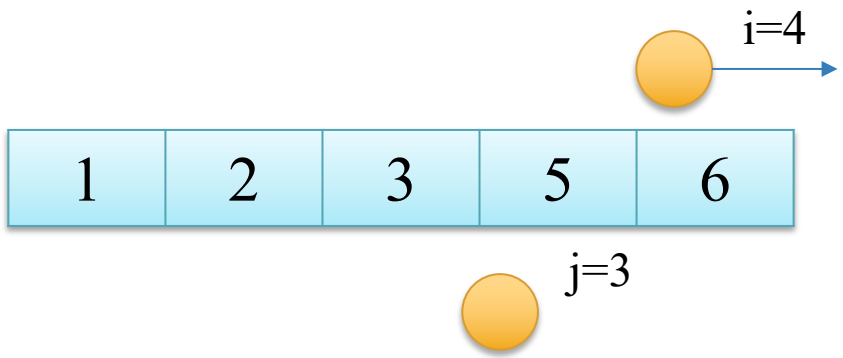
- total = total + array[i] = 0 + 5 = 5
- count = count + 1 = 1 + 1 = 2
- i = i + 1 = 3 + 1 = 4

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5



total = 5

count = 2

2 – Two Pointers

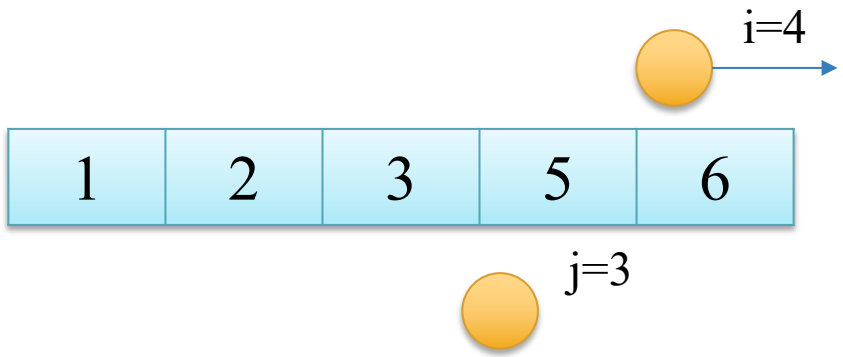
2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5

total = 5

count = 2



total + array[i] = 5 + 6 = 11 > 5

total = total - array[j] = 5 – 5 = 0

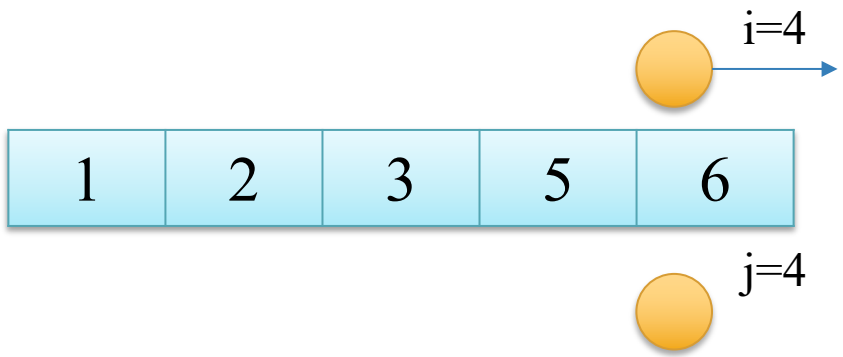
j = j + 1 = 3 + 1 = 4

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5



total = 0

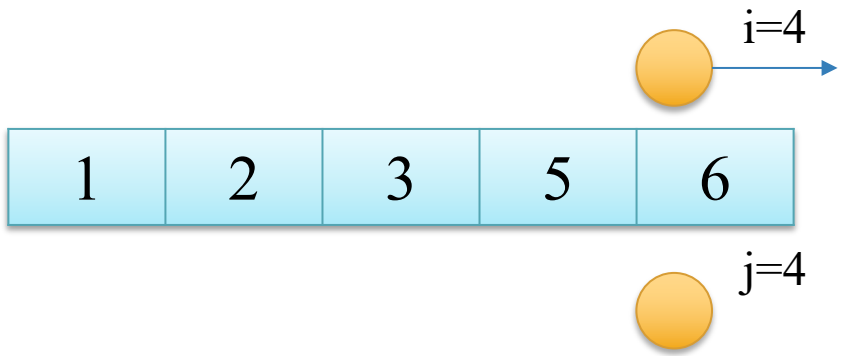
count = 2

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

Key = 5



total = 0

count = 2

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

total = total - array[j] = 0 – 6 = -6

Stop While Loop

j = j + 1 = 4 + 1 = 5

2 – Two Pointers

2.3. Subarray Sum Equals K [Leetcode 560]

Using Two Pointer

T(n) is O(n)

```python
#aivietnam

def subarray_sum_two_pointer(nums, key):
    i = 0
    j = 0
    count = 0
    total = 0
    n = len(nums)
    while (i < n) and (j < n):
        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 Recursion

(1) Base case (non-recursive)

(2) Recursive case

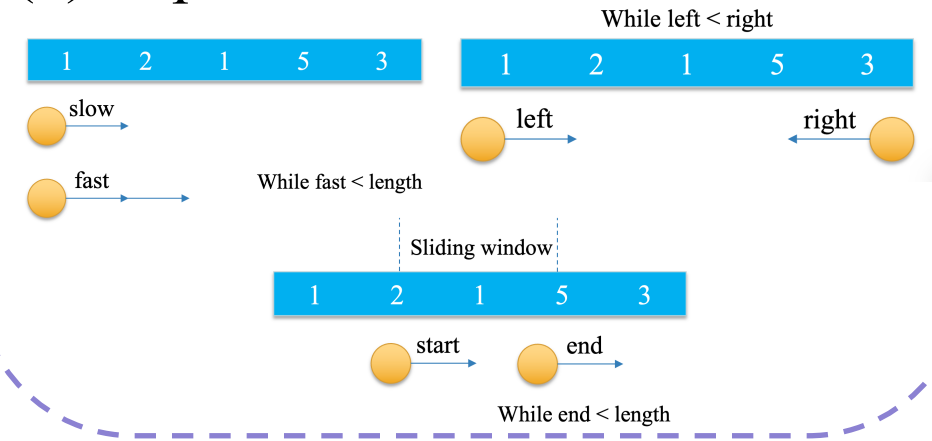
```
# aivietnam

def recursive_function(problem):
    #base case
    if problem is base_case:
        return ___
    else:
        # smaller intance size
        smaller_problem = ___
        # solve the smaller problem
        smaller_result = recursive_function(smaller_problem)
        # combine with the smaller problem
        return ___
```

2 Two pointers

(1) Pointer Initialization

(2) Pointer movement

(3)Stop condition



Optional Exercise

- Leetcode Two Sum
- Leetcode 3Sum
- Leetcode 3Sum Closet
- Leetcode 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.

- 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

Reference

(1) Introduction to Algorithms, 3rd Edition; Thomas H.Cormen et al; 2009

(2) Data Structures & Algorithms; Michael T.Goodrich et al; 2013

(3) Algorithms, 4th; Robert Sedgewick et al; 2011

(4) https://towardsdatascience.com/two-pointer-approach-python-code-f3986b602640

AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?