RECURSION

Data Structures and Algorithms
Waheed Iqbal

Department of Data Science, FCIT
University of the Punjab, Lahore, Pakistan

Introduction

Introduction (Cont.)

- Recursion is a technique that solves a problem by solving a smaller problem of the same type

- In recursion a method call itself repeatedly to solve a specific problem

Requirements for Recursive Solution

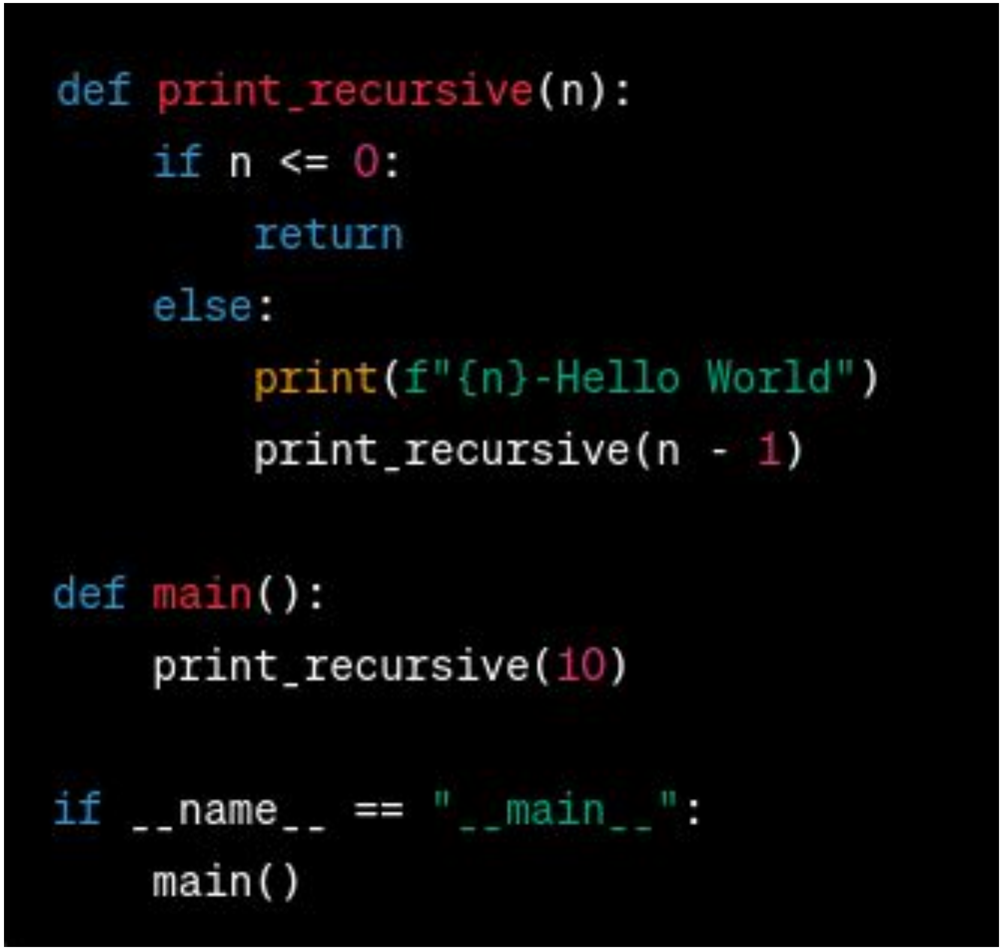
- At least one "*small*" case that you can solve directly
- A way of *breaking* a larger problem down into:
  - One or more *smaller* subproblems
  - Each of the *same kind* as the original
- A way of *combining* subproblem results into an overall solution to the larger problem

General Recursive Design Strategy

- Identify the *base case(s)* (for direct solution)
- Devise a problem *splitting strategy*
  - Subproblems must be smaller
  - Subproblems must work towards a base case
- Devise a solution *combining strategy*

Recursive Hello World!

Let's try to write a recursive hello world



```python
def print_recursive(n):
    if n <= 0:
        return
    else:
        print(f"{n}-Hello World")
        print_recursive(n - 1)

def main():
    print_recursive(10)

if __name__ == "__main__":
    main()
```

Factorial

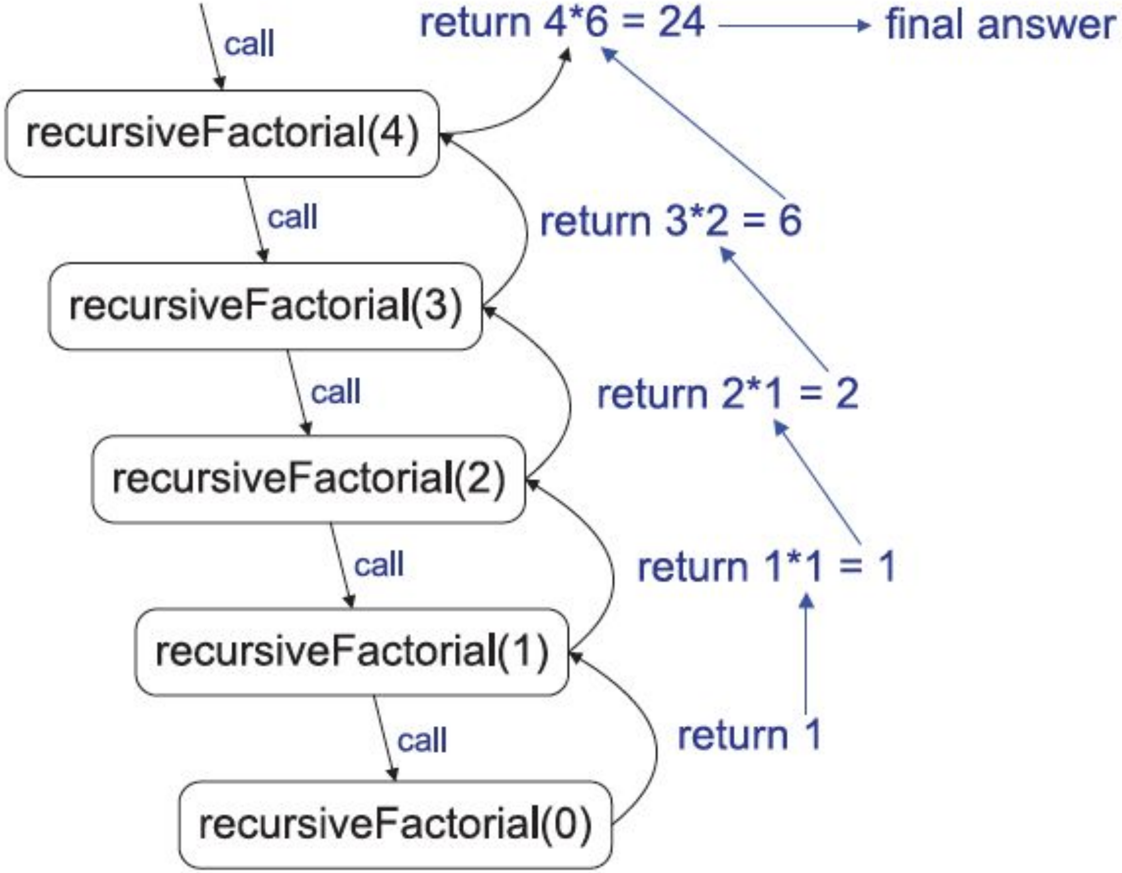
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

$$\text{factorial}(5) = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 5 \cdot \text{factorial}(4).$$

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{factorial}(n-1) & \text{if } n \geq 1. \end{cases}$$

Recursive Factorial

```
int recursiveFactorial(int n) {               // recursive factorial function
  if (n == 0) return 1;                       // basis case
  else return n * recursiveFactorial(n−1);    // recursive case
}
```



call

recursiveFactorial(4)

call

recursiveFactorial(3)

call

recursiveFactorial(2)

call

recursiveFactorial(1)

call

recursiveFactorial(0)

return 1

return 1*1 = 1

return 2*1 = 2

return 3*2 = 6

return 4*6 = 24 → final answer

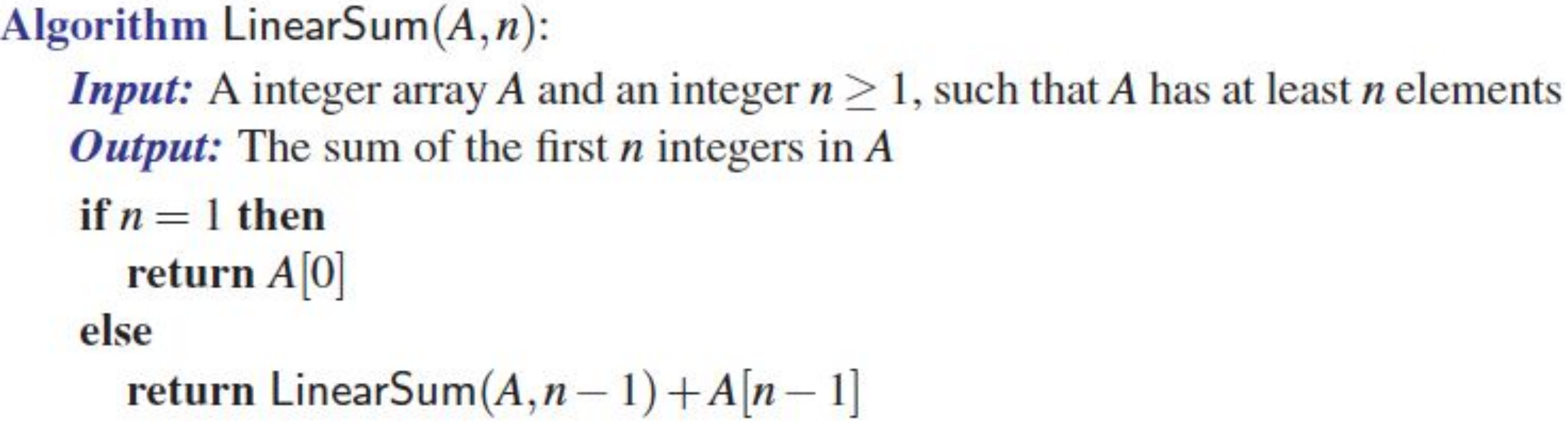
A recursion trace for the call **recursiveFactorial(4)**

Array Sum

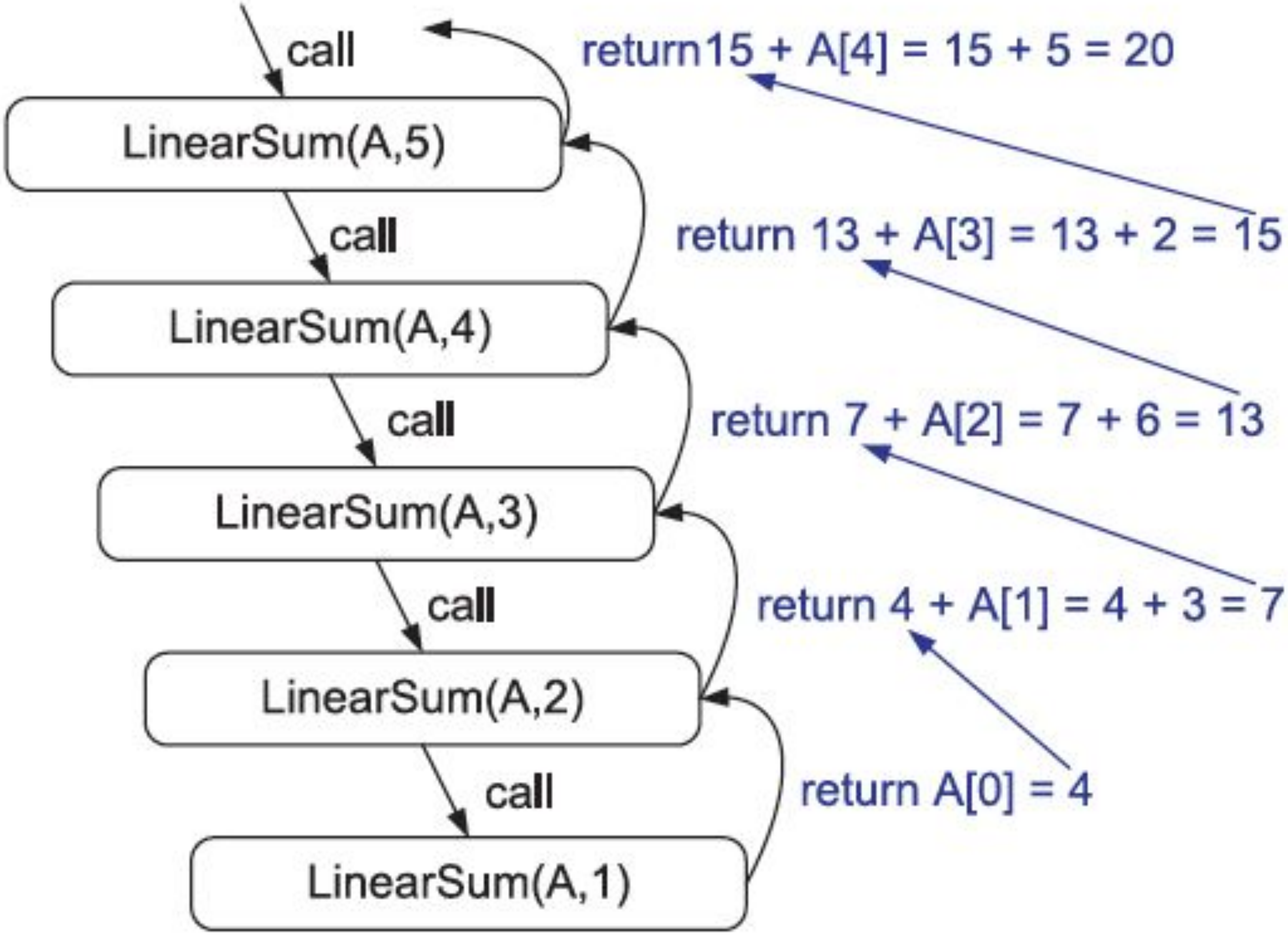
- we are given an array, **A**, of **n** integers that we want to sum together using recursion!

Array Sum

- we are given an array, **A**, of **n** integers that we want to sum together using recursion!

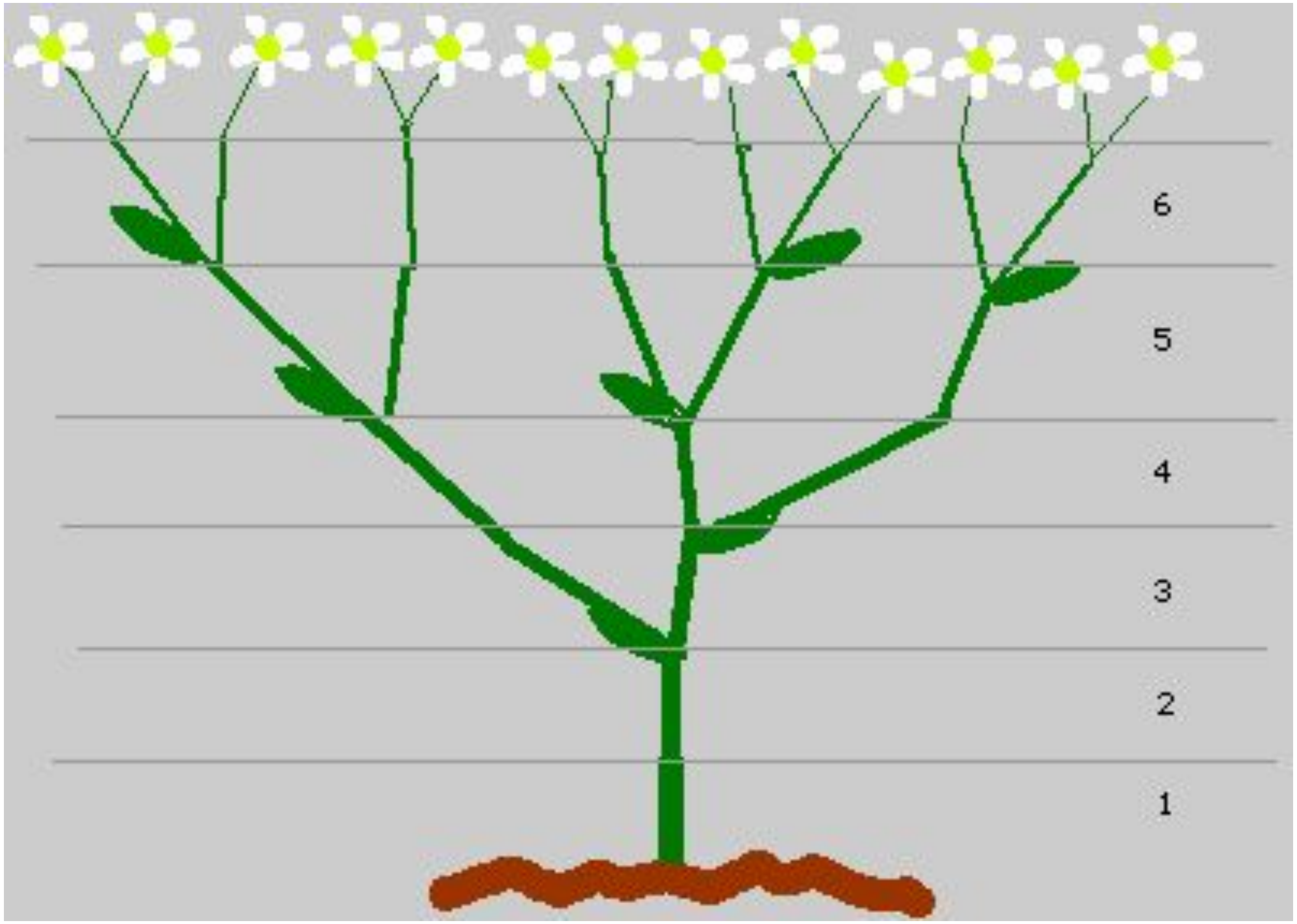


```
Algorithm LinearSum($A, n$):
    Input: A integer array $A$ and an integer $n \geq 1$, such that $A$ has at least $n$ elements
    Output: The sum of the first $n$ integers in $A$
    if $n = 1$ then
        return $A[0]$
    else
        return LinearSum($A, n - 1$) $+ A[n - 1]$
```


call
return15 + A[4] = 15 + 5 = 20
LinearSum(A,5)
call
return 13 + A[3] = 13 + 2 = 15
LinearSum(A,4)
call
return 7 + A[2] = 7 + 6 = 13
LinearSum(A,3)
call
return 4 + A[1] = 4 + 3 = 7
LinearSum(A,2)
call
return A[0] = 4
LinearSum(A,1)

Recursion trace for an execution of **LinearSum(A,n)** with input parameters **A = {**4**,**3**,**6**,**2**,**5**}** and **n =** 5.

Fibonacci Series



Recursive Fibonacci

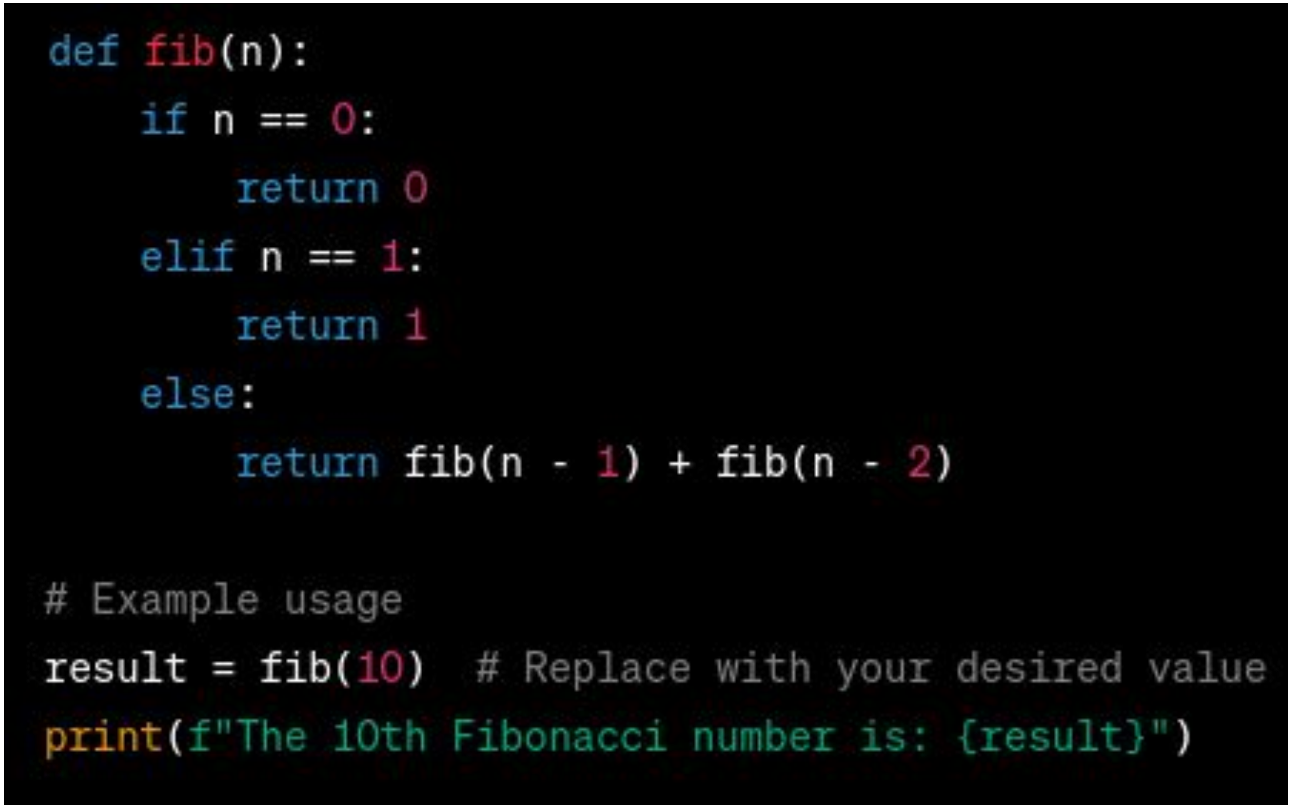
1. Write a code to calculate nth Fibonacci series element

2. Now write a recursive implementation to do the same task

Does the following code generate 4th Fibonacci term?

```
n = 4
a, b = 0, 1
for i in range(n):
    c = a +b
    a = b
    b = c
print(a)
```

Recursive Fibonacci



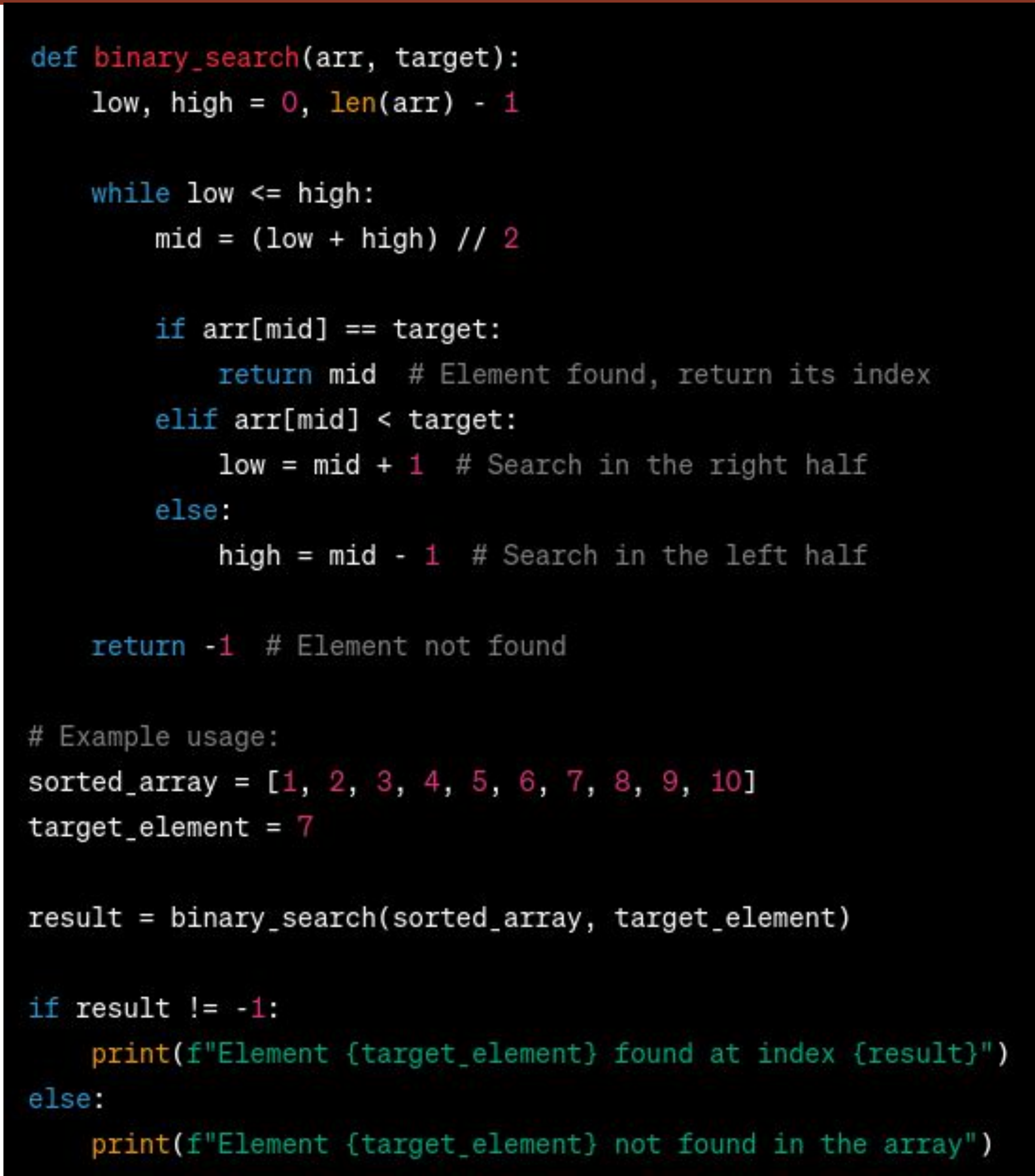
```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

# Example usage
result = fib(10)  # Replace with your desired value
print(f"The 10th Fibonacci number is: {result}")
```

Exercise: Draw recursion tree for fib(4)

Binary Search

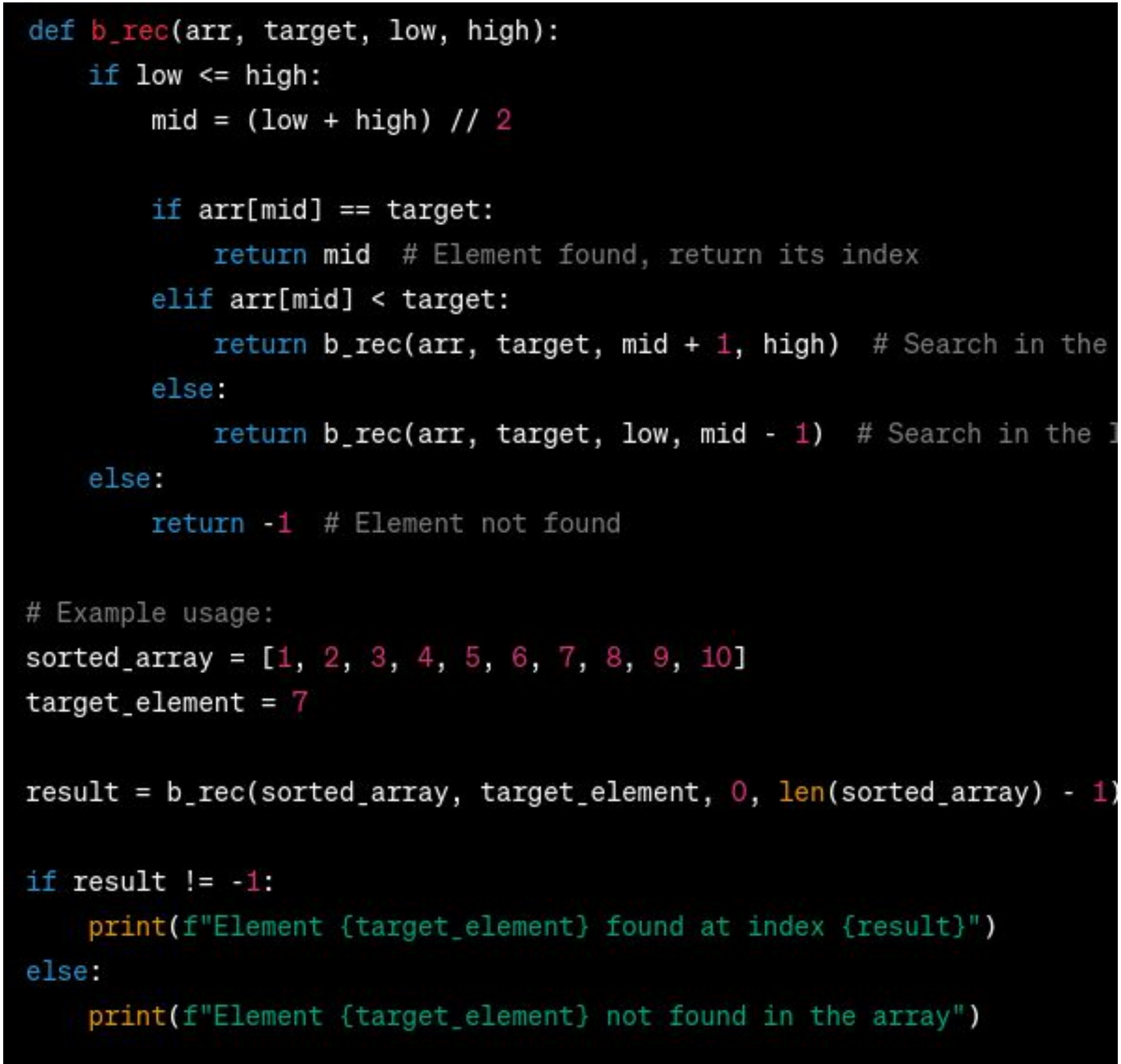
- Given a sorted array of length n, find an element by value.

Iterative Binary Search



```python
def binary_search(arr, target):
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == target:
            return mid  # Element found, return its index
        elif arr[mid] < target:
            low = mid + 1  # Search in the right half
        else:
            high = mid - 1  # Search in the left half

    return -1  # Element not found

# Example usage:
sorted_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
target_element = 7

result = binary_search(sorted_array, target_element)

if result != -1:
    print(f"Element {target_element} found at index {result}")
else:
    print(f"Element {target_element} not found in the array")
```

Recursive Binary Search



```
def b_rec(arr, target, low, high):
    if low <= high:
        mid = (low + high) // 2

        if arr[mid] == target:
            return mid  # Element found, return its index
        elif arr[mid] < target:
            return b_rec(arr, target, mid + 1, high)  # Search in the
        else:
            return b_rec(arr, target, low, mid - 1)  # Search in the l
    else:
        return -1  # Element not found

# Example usage:
sorted_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
target_element = 7

result = b_rec(sorted_array, target_element, 0, len(sorted_array) - 1)

if result != -1:
    print(f"Element {target_element} found at index {result}")
else:
    print(f"Element {target_element} not found in the array")
```