Analysis and Design of Algorithms

Lecture 3

Analysis of Algorithms II

Dr. Mohamed Loey
Lecturer, Faculty of Computers and Information
Benha University
Egypt

Table of Contents

Maximum Pairwise Product

Maximum Pairwise Product

- Given a sequence of non-negative integers $a_0,\ldots,a_{n-1}$, find the **maximum pairwise product**, that is, the largest integer that can be obtained by multiplying two different elements from the sequence (or, more formally, max $a_i a_j$ where $0 \le i \ne j \le n-1$). Different elements here mean a_i and a_j with $i \ne j$ (it can be the case that $a_i = a_j$).

Maximum Pairwise Product

- Constraints $2 \leq n \leq 2 * 10^5$ & $0 \leq a_0, \ldots, a_{n-1} \leq 10^5$.

Maximum Pairwise Product

- ❑ **Sample 1**
- ❖ Input: 1 2 3
- ❖ Output:6
- ❑ **Sample 2**
- ❖ Input: 7 5 14 2 8 8 10 1 2 3
- ❖ Output:140

Maximum Pairwise Product

- ❑ **Sample 3**

- ❖ Input: 4 6 2 6 1
- ❖ Output:36

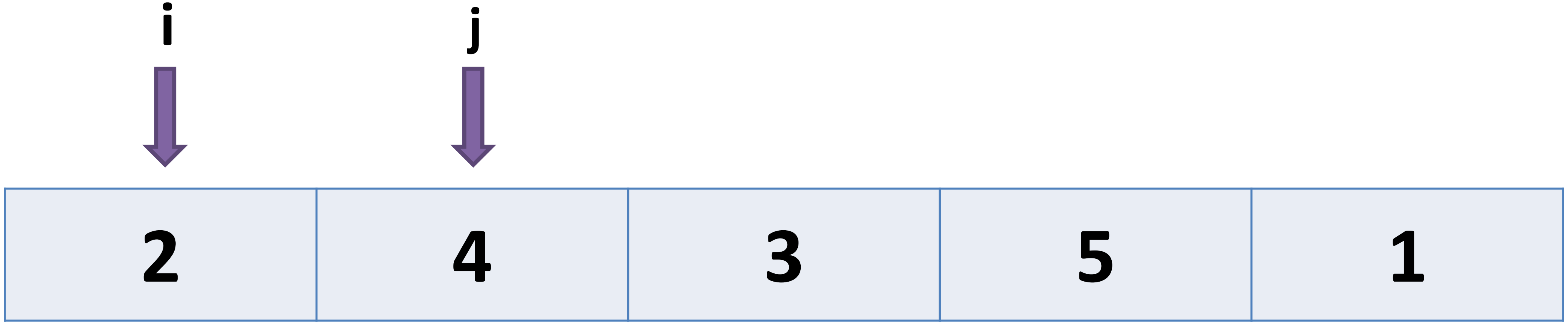
Maximum Pairwise Product

- Assume the following array

2	**4**	**3**	**5**	**1**

Maximum Pairwise Product

- Assume the following array

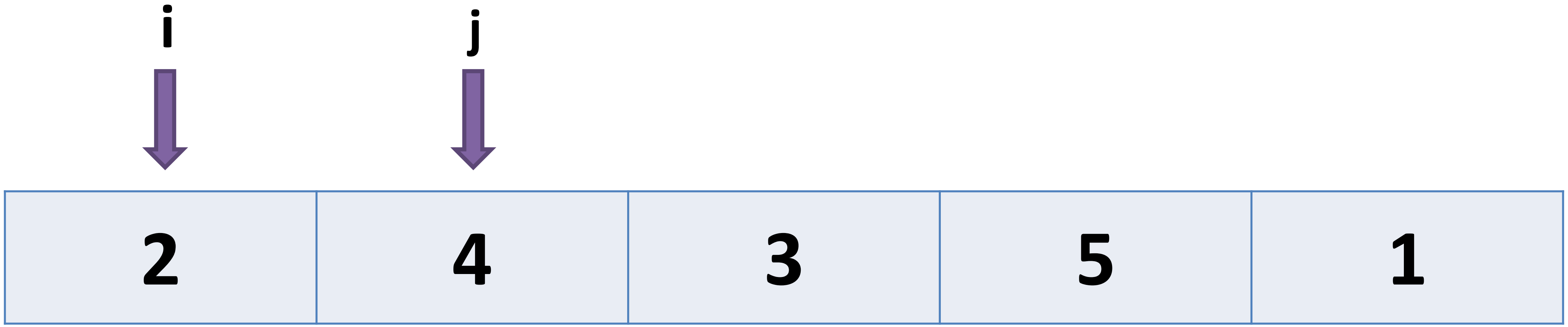


2	4	3	5	1

Result=0

Maximum Pairwise Product

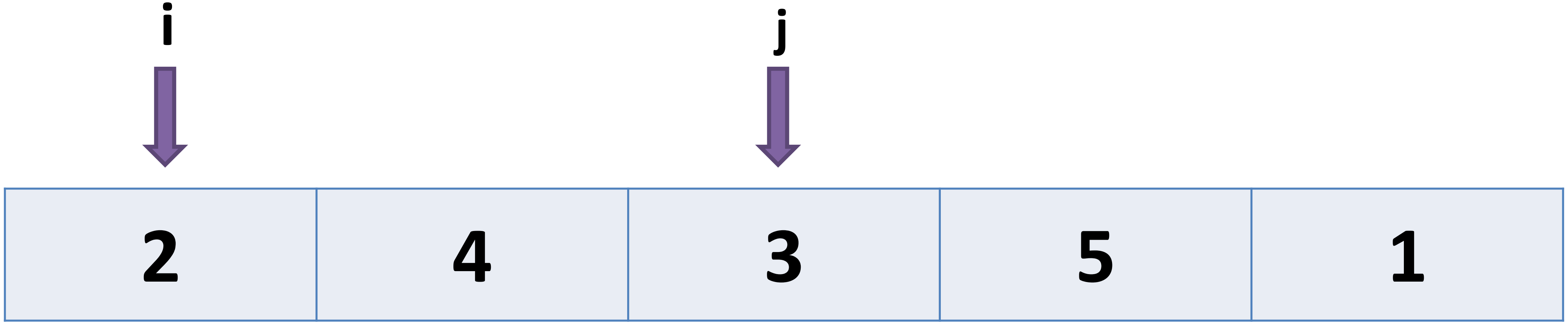
- ❑ Assume the following array



If a[i]*a[j] > result
result=a[i]*a[j]=8

Maximum Pairwise Product

- ❑ Assume the following array



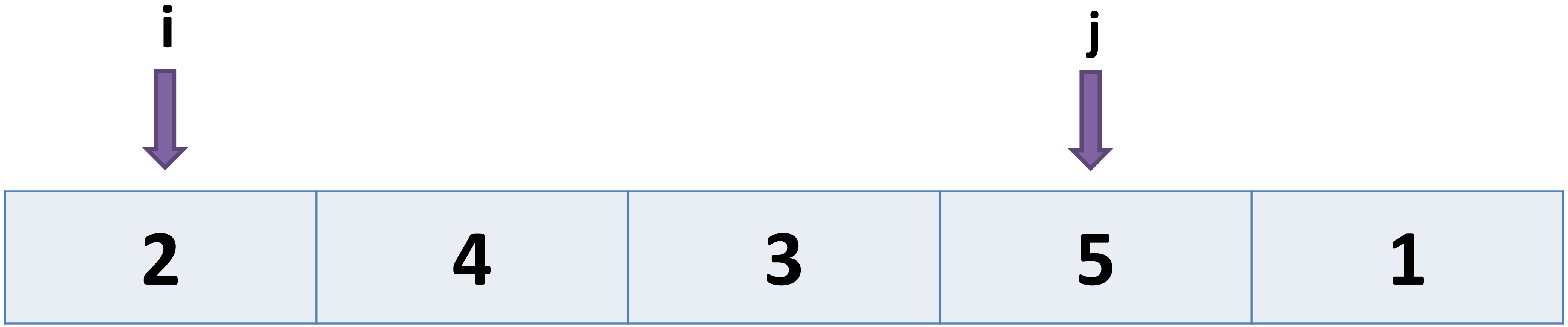
2	4	3	5	1

If a[i]*a[j] > result

result=8

Maximum Pairwise Product

- ❑ Assume the following array



2	4	3	5	1

If a[i]*a[j] > result
result= a[i]*a[j] =10

Maximum Pairwise Product

- Naive algorithm

```
def max_pairwise_product(a):
    result = 0
    for i in range(0, len(a)):
        for j in range(i+1, len(a)):
            if a[i]*a[j] > result:
                result = a[i]*a[j]
    return result
```

Maximum Pairwise Product

- ❑ Python Code:

```python
a=[2,6,4,5,2]
print(max_pairwise_product(a))
```

```
30
```

Maximum Pairwise Product

- Time complexity $O(n^2)$

```python
def max_pairwise_product(a):
    result = 0
    for i in range(0, len(a)):
        for j in range(i+1, len(a)):
            if a[i]*a[j] > result:
                result = a[i]*a[j]
    return result
```

Maximum Pairwise Product

- **we need a faster algorithm.** This is because our program performs about n^2 steps on a sequence of length n. For the maximal possible value n=200,000 = $2*10^5$, the number of steps is 40,000,000,000 = $4*10^{10}$. This is too much. Recall that modern machines can perform roughly 10^9 basic operations per second

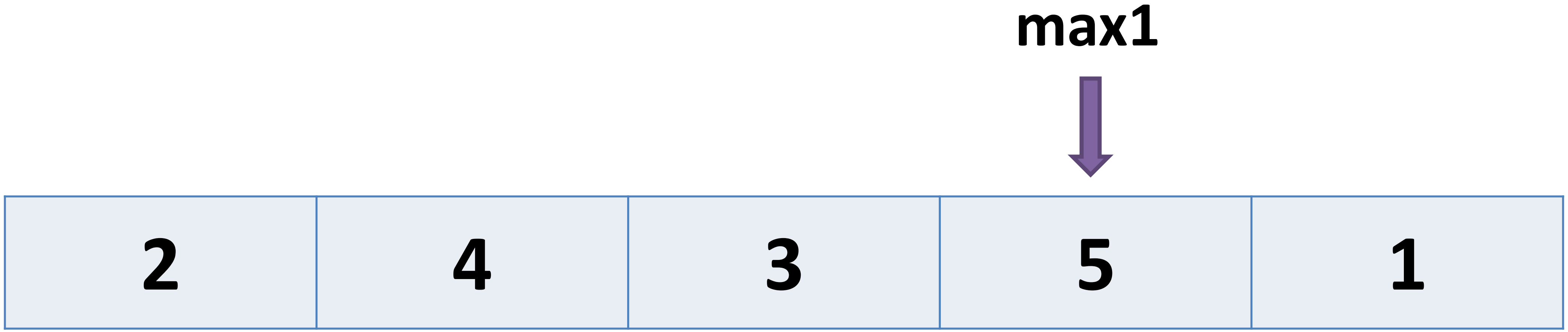
Maximum Pairwise Product

- Assume the following array

2	**4**	**3**	**5**	**1**

Maximum Pairwise Product

- ☐ Find maximum number1

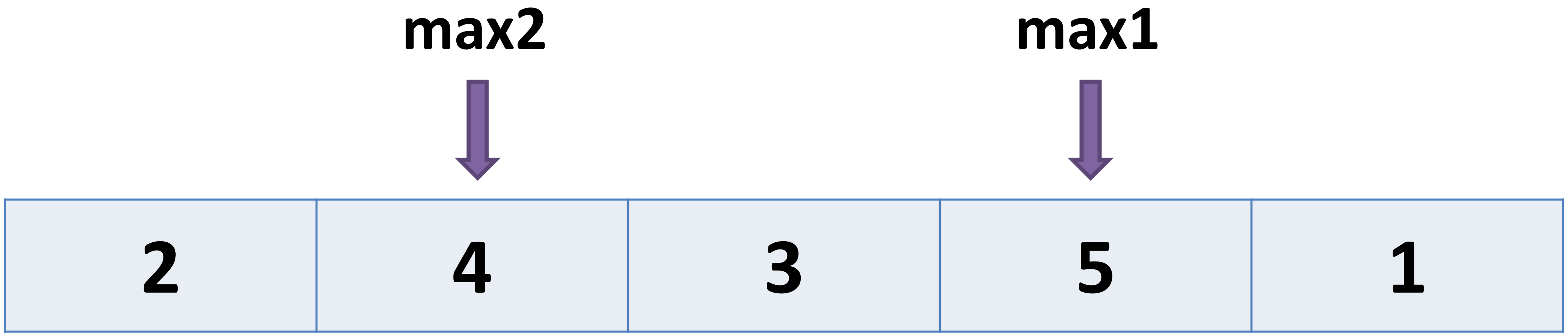


max1

2	4	3	5	1

Maximum Pairwise Product

- Find maximum number2 but not maximum number1



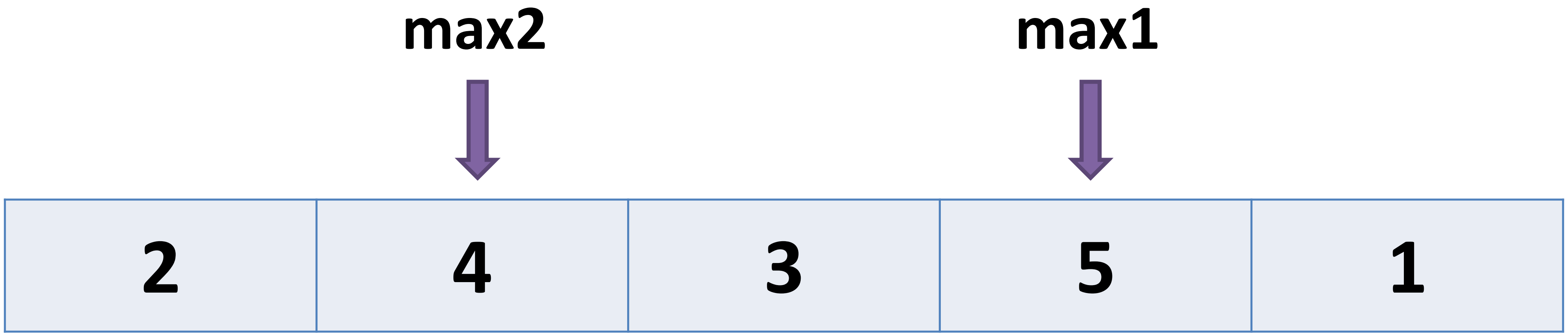
max2

max1

2	4	3	5	1

Maximum Pairwise Product

- ❑ Find maximum number2 but not maximum number1



max2 **max1**

2	**4**	**3**	**5**	**1**

return max1*max2

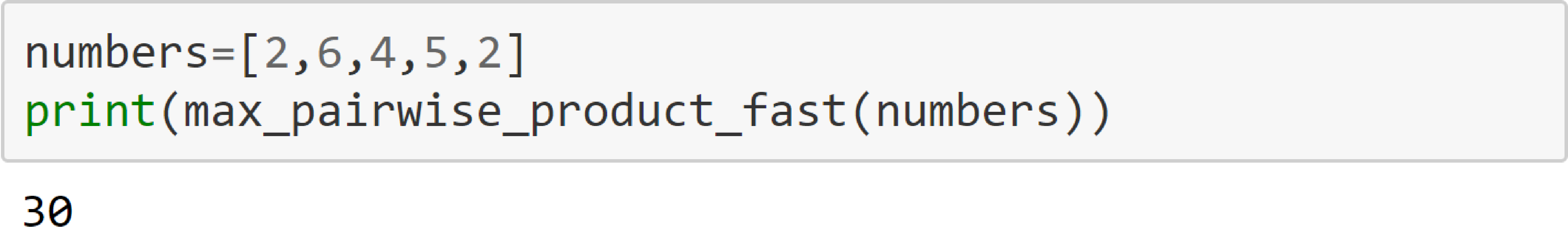
Maximum Pairwise Product

- ❑ Efficient algorithm

```python
def max_pairwise_product_fast(numbers):
    max1 = -1
    index1 = None
    max2 = -1
    for element in range(len(numbers)):
        if numbers[element] >= max1:
            max1 = numbers[element]
            index1 = element
    for element in range(len(numbers)):
        if numbers[element] >= max2:
            if element != index1:
                max2 = numbers[element]
    return max1 * max2
```

Maximum Pairwise Product

- Efficient algorithm

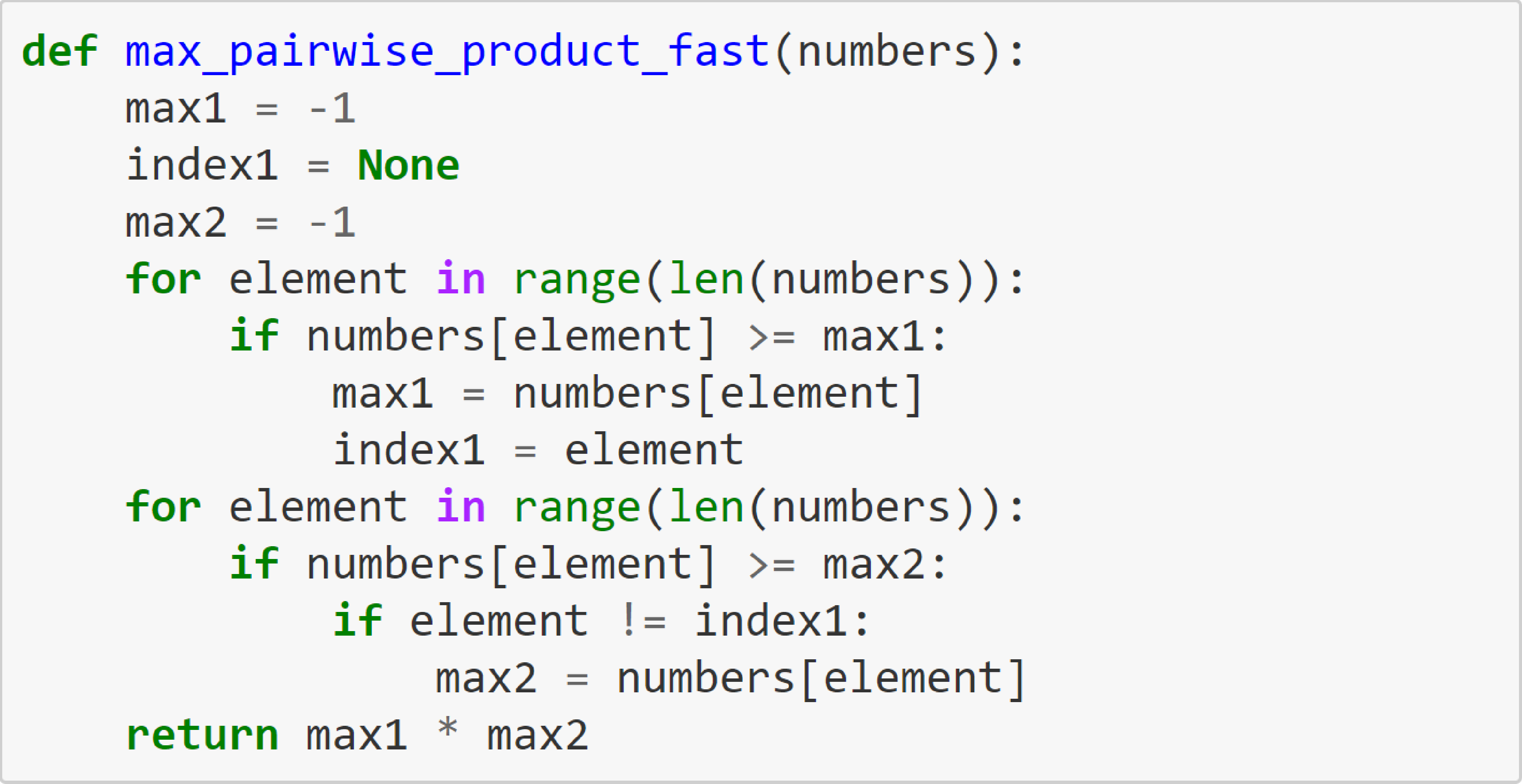


```
numbers=[2,6,4,5,2]
print(max_pairwise_product_fast(numbers))
```

```
30
```

Maximum Pairwise Product

- ❑ Time complexity O(n)



```python
def max_pairwise_product_fast(numbers):
    max1 = -1
    index1 = None
    max2 = -1
    for element in range(len(numbers)):
        if numbers[element] >= max1:
            max1 = numbers[element]
            index1 = element
    for element in range(len(numbers)):
        if numbers[element] >= max2:
            if element != index1:
                max2 = numbers[element]
    return max1 * max2
```

Fibonacci

Fibonacci

0,1,1,2,3,5,8,13,21,34, …

Fibonacci

- *Definition:*

- $$F_n = \left\{ \begin{array}{cc} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-2} + F_{n-1} & n > 1 \end{array} \right\}$$

Fibonacci

❑ Examples:

- $F_8 = 21$
- $F_{20} = 6765$
- $F_{50} = 12586269025$
- $F_{100} = 354224848179261915075$

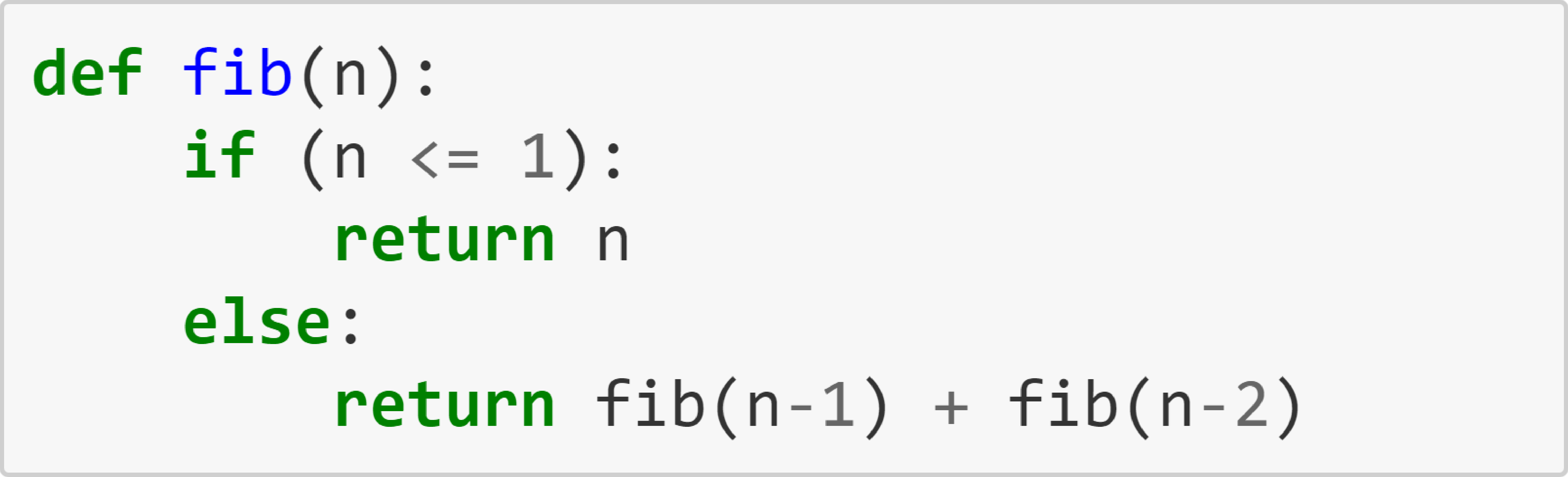
Fibonacci

- Examples:
  - $F_{500} =$

$$1394232245616978801397243828$$
$$7040728395007025658769730726$$
$$4108962948325571622863290691$$
$$557658876222521294125$$

Fibonacci

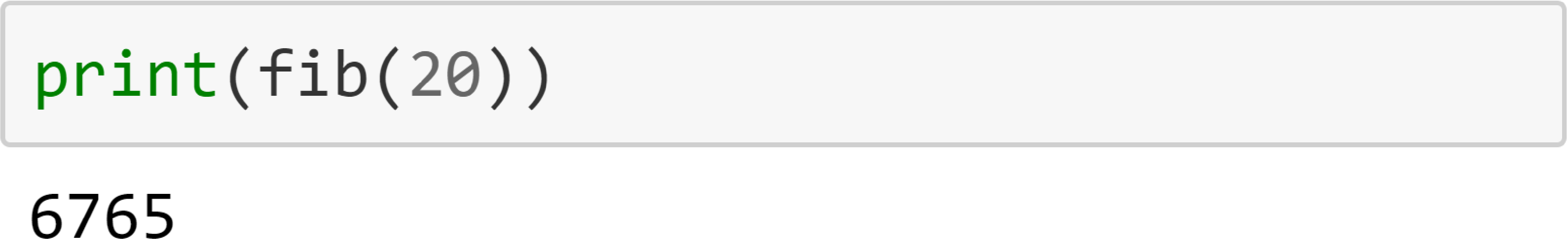
- ❑ Naive algorithm



```
def fib(n):
    if (n <= 1):
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Fibonacci

- ❑ Naive algorithm



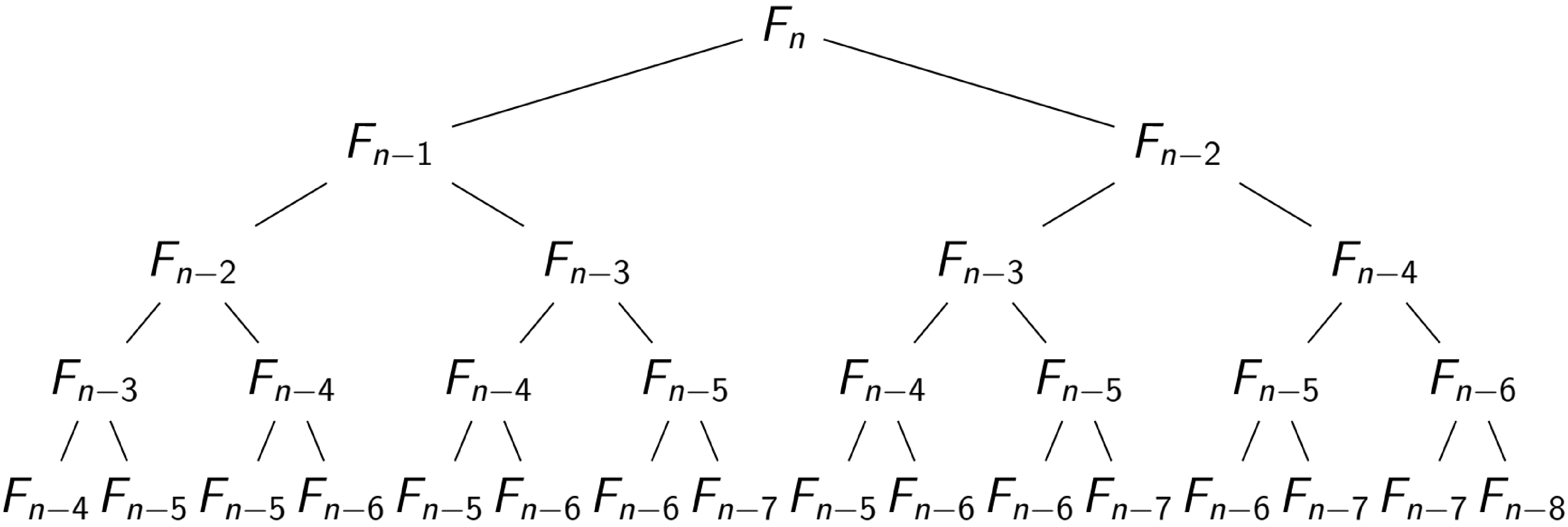
```
print(fib(20))
```

```
6765
```

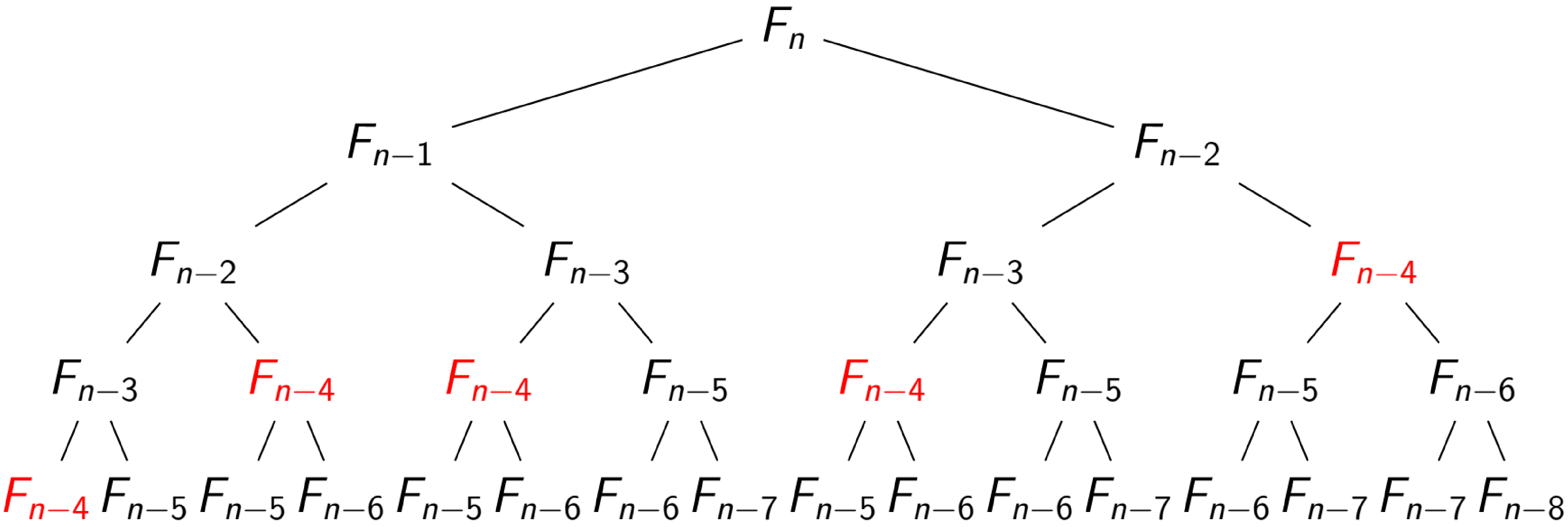
```
print(fib(100))
```

Very Long Time why????

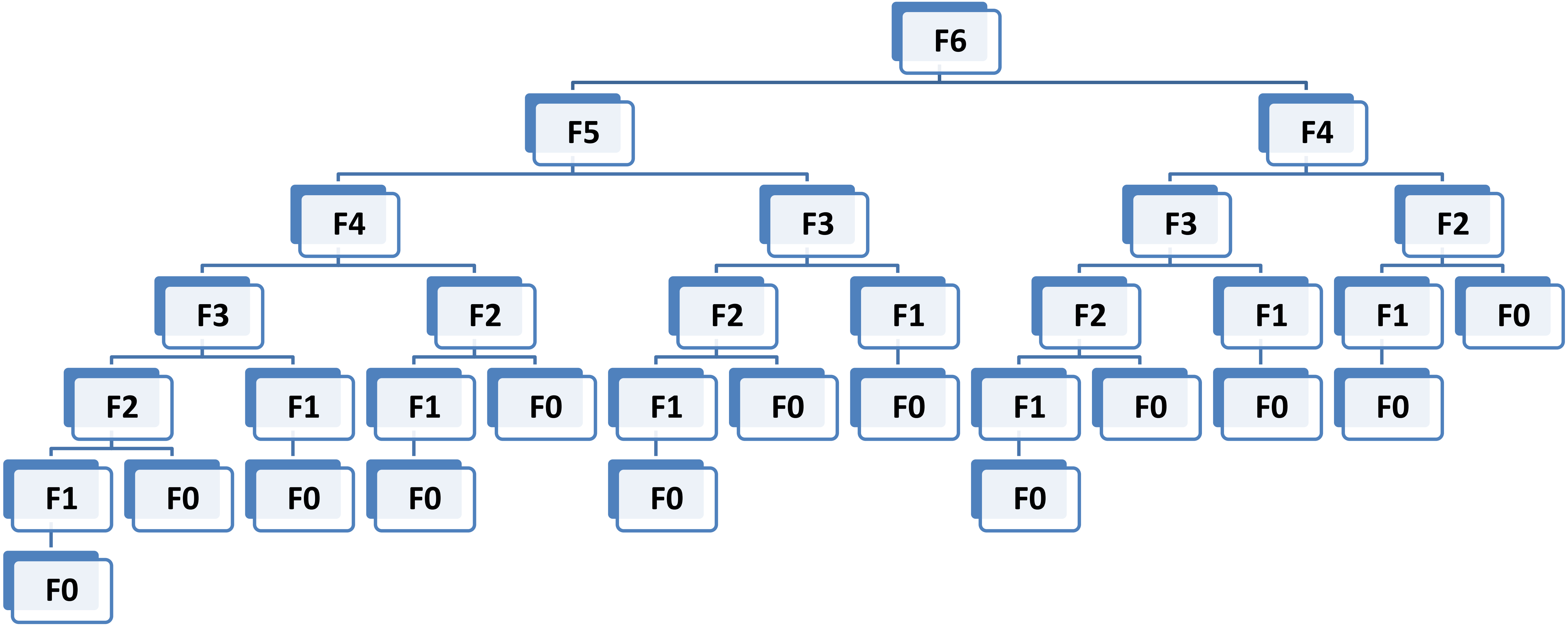
Fibonacci


F_n
F_{n-1} F_{n-2}
F_{n-2} F_{n-3} F_{n-3} F_{n-4}
F_{n-3} F_{n-4} F_{n-4} F_{n-5} F_{n-4} F_{n-5} F_{n-5} F_{n-6}
F_{n-4} F_{n-5} F_{n-5} F_{n-6} F_{n-5} F_{n-6} F_{n-6} F_{n-7} F_{n-5} F_{n-6} F_{n-6} F_{n-7} F_{n-6} F_{n-7} F_{n-7} F_{n-8}

Fibonacci


F_n
F_{n-1} F_{n-2}
F_{n-2} F_{n-3} F_{n-3} F_{n-4}
F_{n-3} F_{n-4} F_{n-4} F_{n-5} F_{n-4} F_{n-5} F_{n-5} F_{n-6}
F_{n-4} F_{n-5} F_{n-5} F_{n-6} F_{n-5} F_{n-6} F_{n-6} F_{n-7} F_{n-5} F_{n-6} F_{n-6} F_{n-7} F_{n-6} F_{n-7} F_{n-7} F_{n-8}

Fibonacci



F6

F5 F4

F4 F3 F3 F2

F3 F2 F2 F1 F2 F1 F1 F0

F2 F1 F1 F0 F1 F0 F0 F1 F0 F0 F0

F1 F0 F0 F0 F0 F0

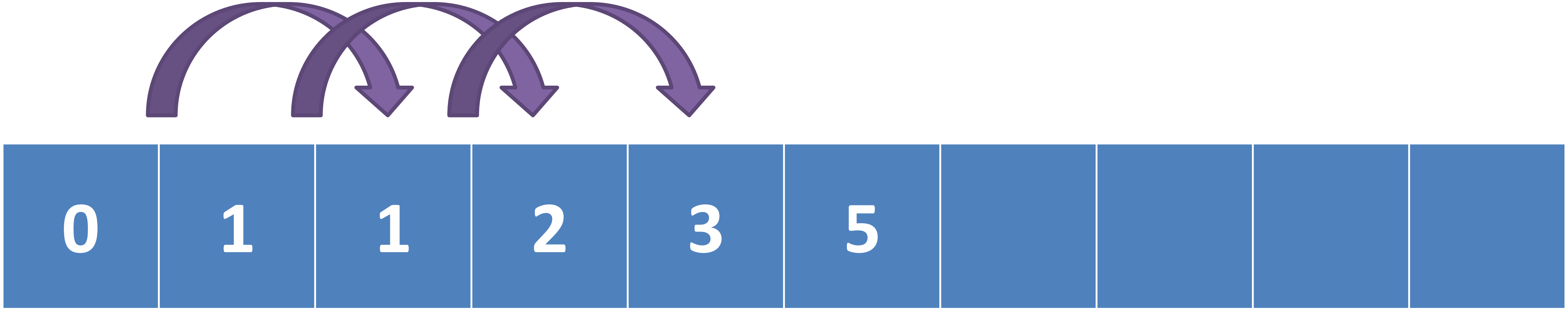
F0

Fibonacci

- ❑Fib algorithm is very slow because of recursion
- ❑Time complexity = $O(2^n)$

Fibonacci

- ❑ Efficient algorithm



0	1	1	2	3	5				

Create array then insert fibonacci

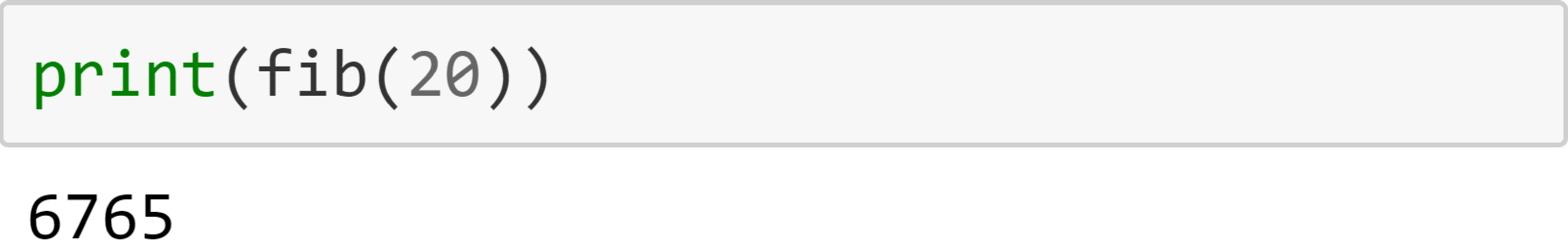
Fibonacci

- ❑ Efficient algorithm

```
def fib_fast(n):
    if (n <= 1):
        return n
    else:
        numbers = [0, 1]
        for i in range(n-1):
            numbers.append(numbers[-1]+numbers[-2])
        return numbers[-1]
```

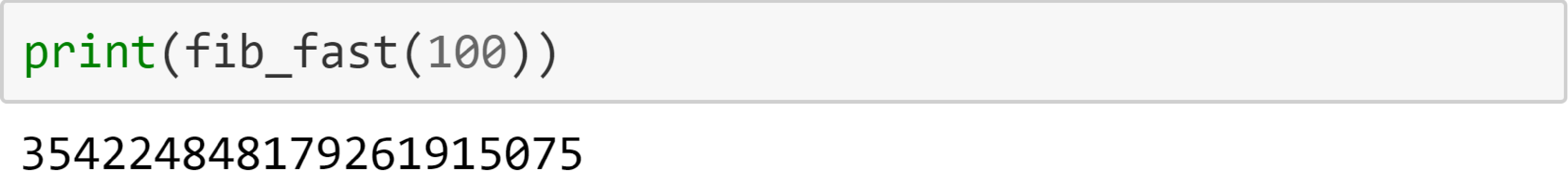
Fibonacci

- Efficient algorithm



```
print(fib(20))
```

```
6765
```



```
print(fib_fast(100))
```

```
354224848179261915075
```

short Time **why????**

Fibonacci

- Fib_Fast algorithm is fast because of loop + array
- Time complexity = $O(n^2)$

Fibonacci

- ❑ Efficient algorithm

- ➢ Try

```
print(fib_faster(1000000))
```

Very long Time why????

Fibonacci

- Advanced algorithm
  - No array
  - Need two variable + Loop

Fibonacci

- Advanced algorithm
- Compute F_6
- a=0, b=1

a	b
0	1

Fibonacci

- Advanced algorithm
- Compute F_6
- a=b, b=a+b

a	b
1	1

Fibonacci

- Advanced algorithm
- Compute F_6
- a=b, b=a+b

a	b
1	**2**

Fibonacci

- Advanced algorithm
- Compute F_6
- a=b, b=a+b

a	b
2	3

Fibonacci

- Advanced algorithm
- Compute F_6
- a=b, b=a+b

a	b
3	5

Fibonacci

- Advanced algorithm
- Compute F_6
- a=b, b=a+b

a	**b**
5	**8**

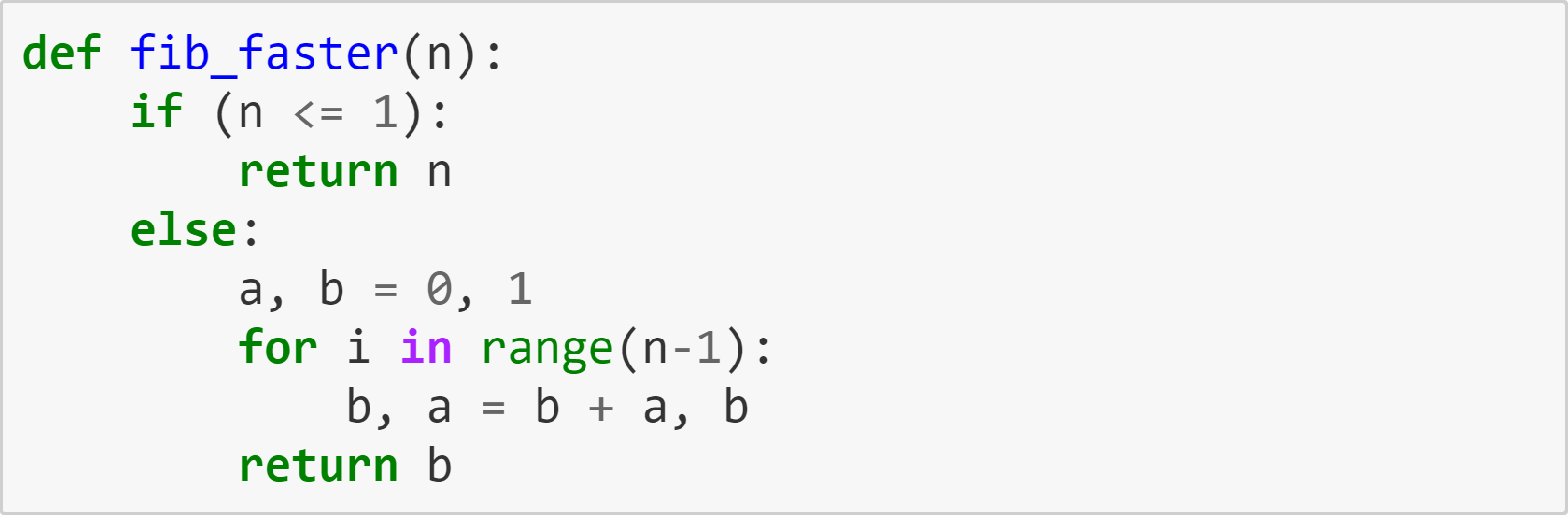
Fibonacci

- ❑ Advanced algorithm
- ❑ Compute $F_6=8$

a	b
5	8

Fibonacci

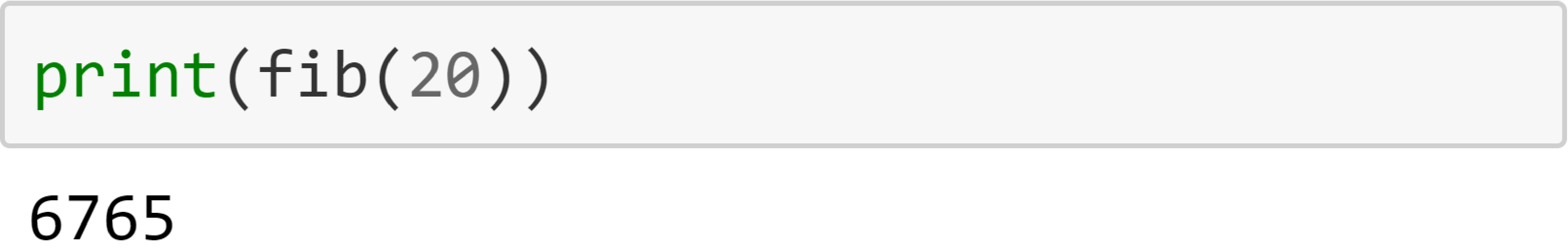
- ❑ Advanced algorithm



```
def fib_faster(n):
    if (n <= 1):
        return n
    else:
        a, b = 0, 1
        for i in range(n-1):
            b, a = b + a, b
        return b
```

Fibonacci

- ❑ Advanced algorithm



```
print(fib(20))
```

```
6765
```

```
print(fib_fast(100))
```

```
354224848179261915075
```

Very short Time **why????**

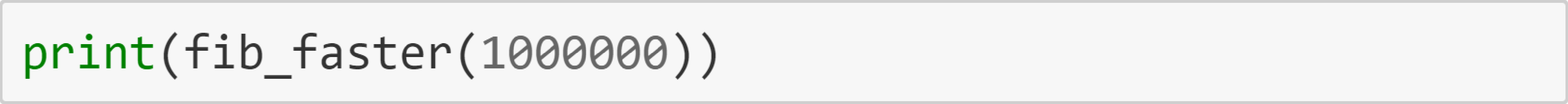
Fibonacci

- ❑Fib_Faster algorithm is faster because of loop + two variables
- ❑Time complexity = O(n)

Fibonacci

- ❑ Advanced algorithm

- ➢ Try



```
print(fib_faster(1000000))
```

Short Time why????

Greatest Common Divisors

Greatest Common Divisors

- In mathematics, the greatest common divisor (gcd) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

Greatest Common Divisors

- Input Integers a,b >=0
- Output gcd(a,b)

Greatest Common Divisors

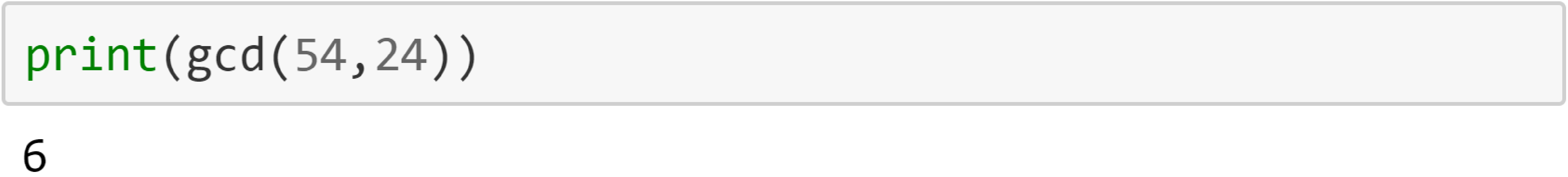
- ❑ What is the greatest common divisor of 54 and 24?
- ❑ The divisors of 54 are: 1,2,3,6,9,18,27,54
- ❑ Similarly, the divisors of 24 are: 1,2,3,4,6,8,12,24
- ❑ The numbers that these two lists share in common are the common divisors of 54 and 24: 1,2,3,6
- ❑ The greatest of these is 6. That is, the greatest common divisor of 54 and 24. gcd(54,24)=6

Greatest Common Divisors

- Naive algorithm

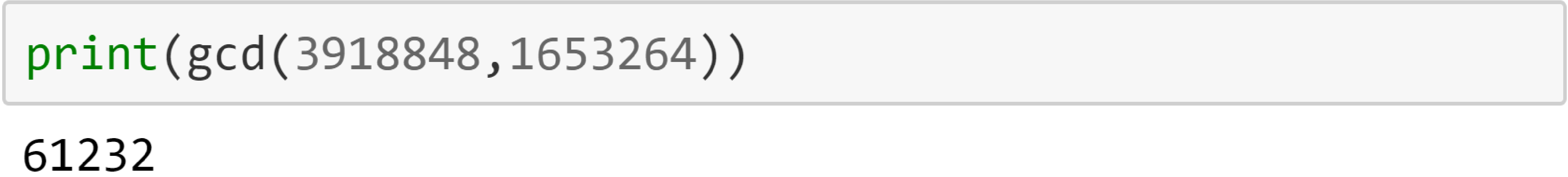
```python
def gcd(a,b):
    if a>b:
        i=a
    else:
        i=b
    while i>=1:
        if a%i==0 and b%i==0:
            break
        i=i-1
    return i
```

Greatest Common Divisors



```
print(gcd(54,24))
```

```
6
```



```
print(gcd(3918848,1653264))
```

```
61232
```

Greatest Common Divisors

- gcd algorithm is slow because of loop
- Time complexity = O(n)
- n depend on a,b

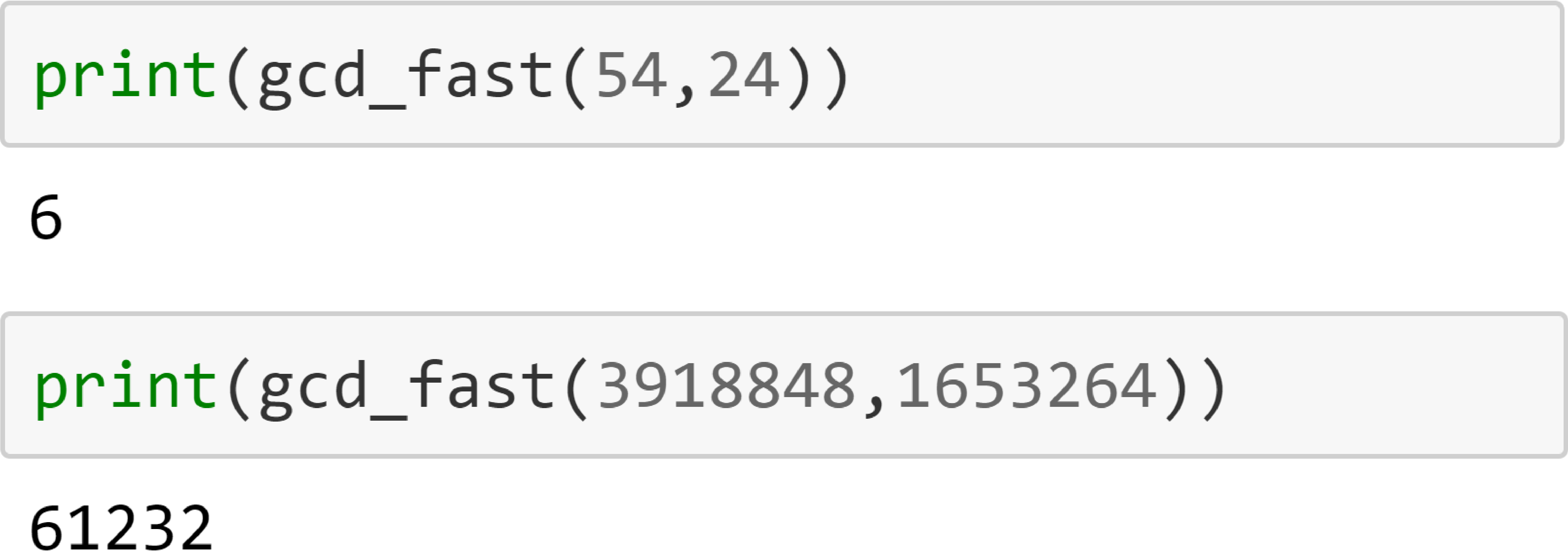
Greatest Common Divisors

- Efficient algorithm

```
def gcd_fast(a, b):
    if a%b == 0:
        return b
    return gcd_fast(b, a%b)
```

Greatest Common Divisors

- Efficient algorithm



```
print(gcd_fast(54,24))
```

```
6
```

```
print(gcd_fast(3918848,1653264))
```

```
61232
```

Greatest Common Divisors

- Efficient algorithm

gcd_fast((3918848, 1653264))

gcd_fast((1653264, 612320))

gcd_fast((612320, 428624))

gcd_fast((428624, 183696))

gcd_fast((183696, 61232))

return 61232

Greatest Common Divisors

- ❑ Efficient algorithm
- ❑ Take 5 steps to solve **gcd_fast((3918848, 1653264))**
- ❑ Time complexity = O(log(n))
- ❑ n depend on a,b

Summary

- ❑Naive algorithm is too slow.
- ❑The Efficient algorithm is much better.
- ❑Finding the correct algorithm requires knowing something interesting about the problem.

Contact Me

facebook.com/mloey

mohamedloey@gmail.com

twitter.com/mloey

linkedin.com/in/mloey

mloey@fci.bu.edu.eg

mloey.github.io

THANKS FOR YOUR TIME