

Fibonacci Numbers

Shusen Wang

Compute Fibonacci Numbers by Recursion

Fibonacci Numbers

Definition

- $F(n)$: the n -th Fibonacci number.
- $F(1) = 1$.
- $F(2) = 1$.
- $F(n) = F(n - 1) + F(n - 2)$, for $n \geq 3$.

Fibonacci Numbers

C/C++ Implementation

```
long F(int n) {  
    if (n <= 2)  
        return 1;  
    else  
        return F(n-1) + F(n-2);  
}
```

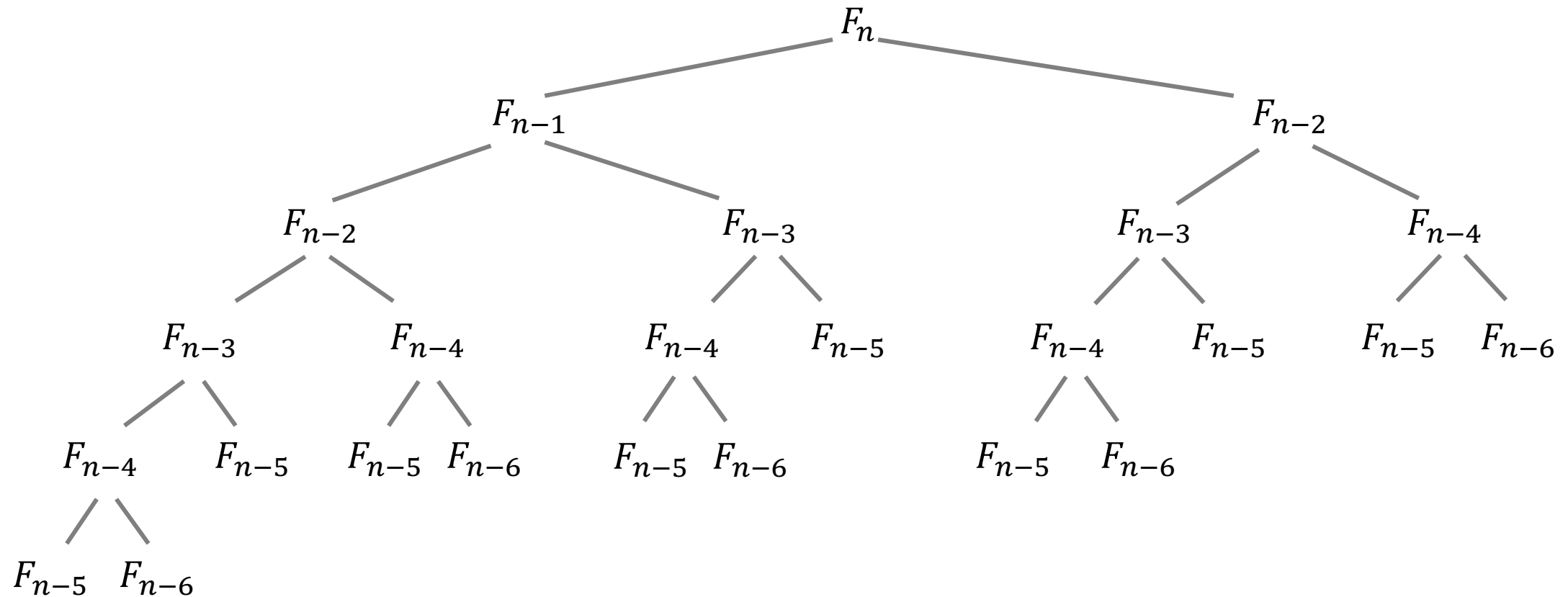
Fibonacci Numbers

- Recursion is slow.
- Time complexity:

$$T(n) = a^n, \text{ where } a = \frac{1+\sqrt{5}}{2}.$$

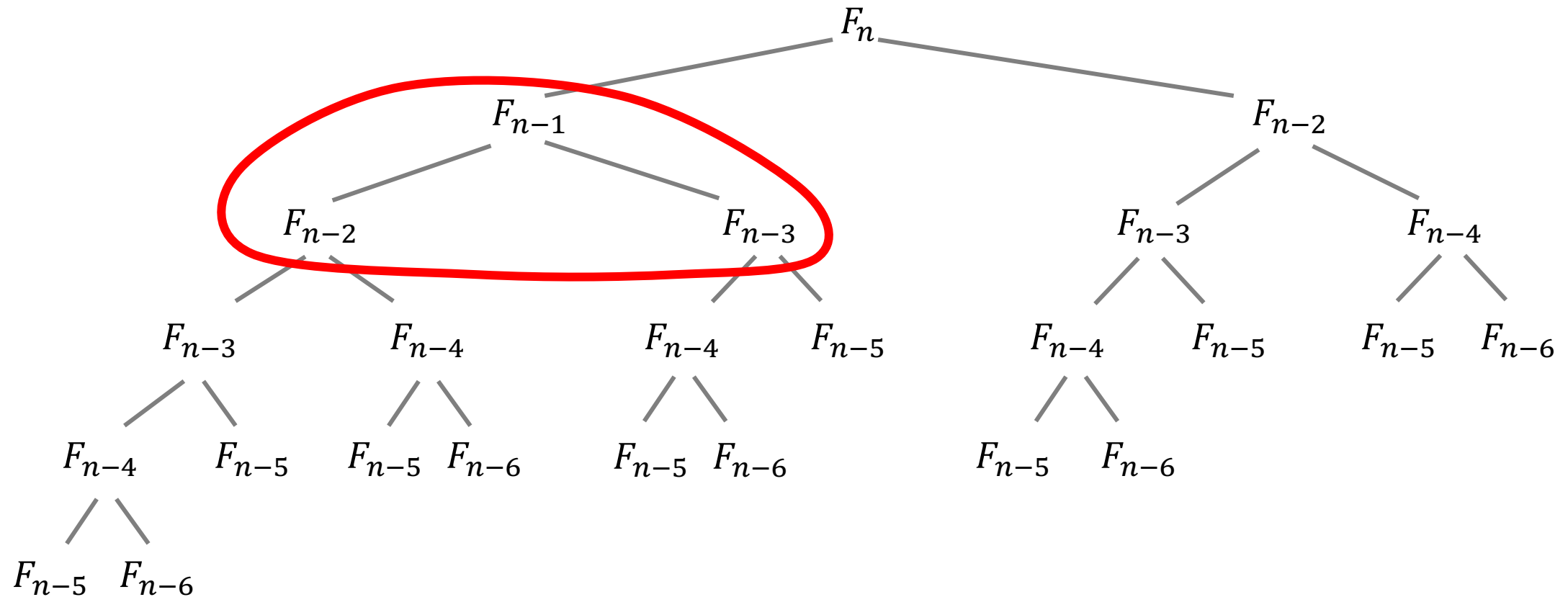
Why is recursion slow?

Trace of the recursive calculation of Fibonacci numbers:



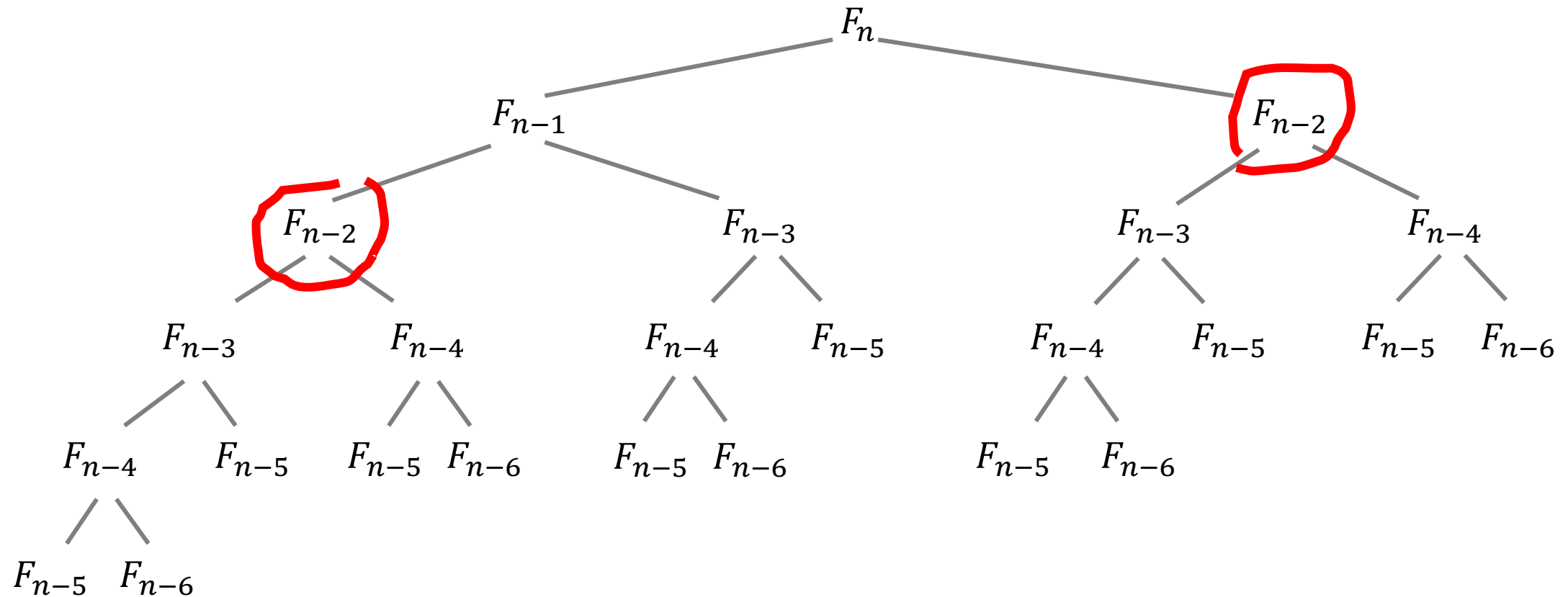
Why is recursion slow?

Trace of the recursive calculation of Fibonacci numbers:



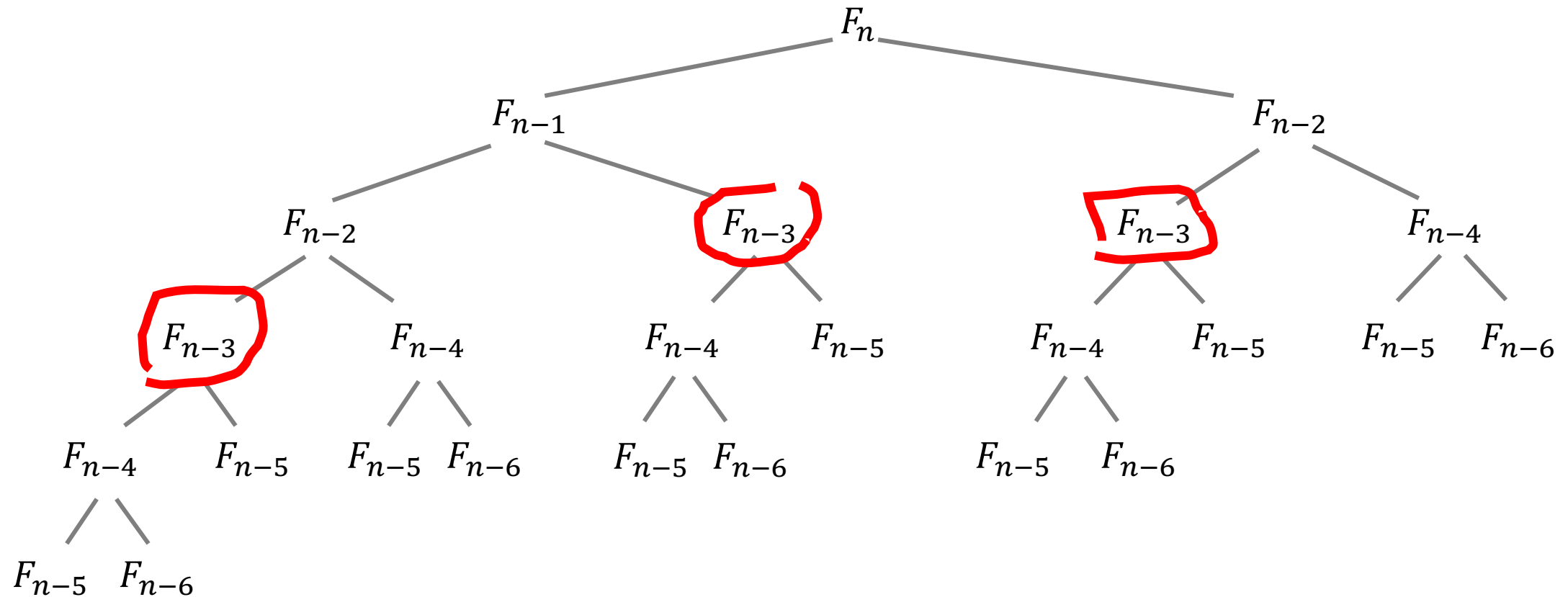
Why is recursion slow?

2 separate calls to compute $F(n - 2)$.



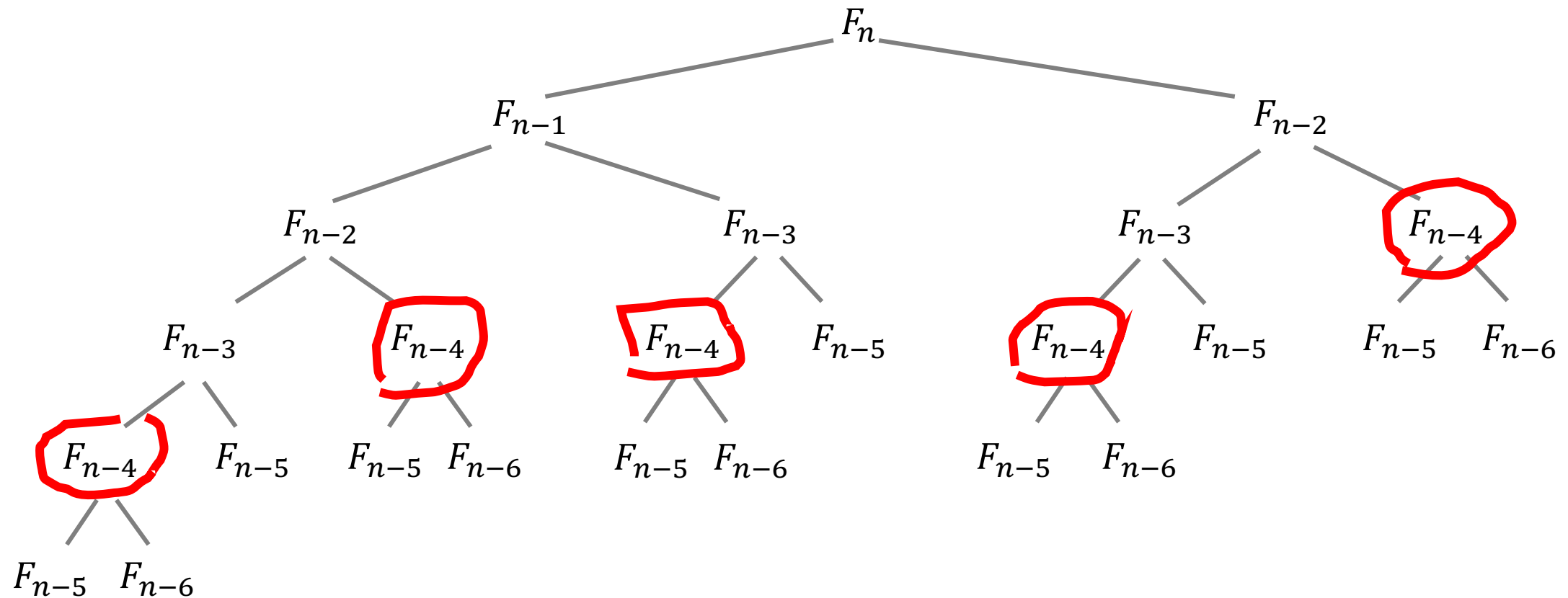
Why is recursion slow?

3 separate calls to compute $F(n - 3)$.



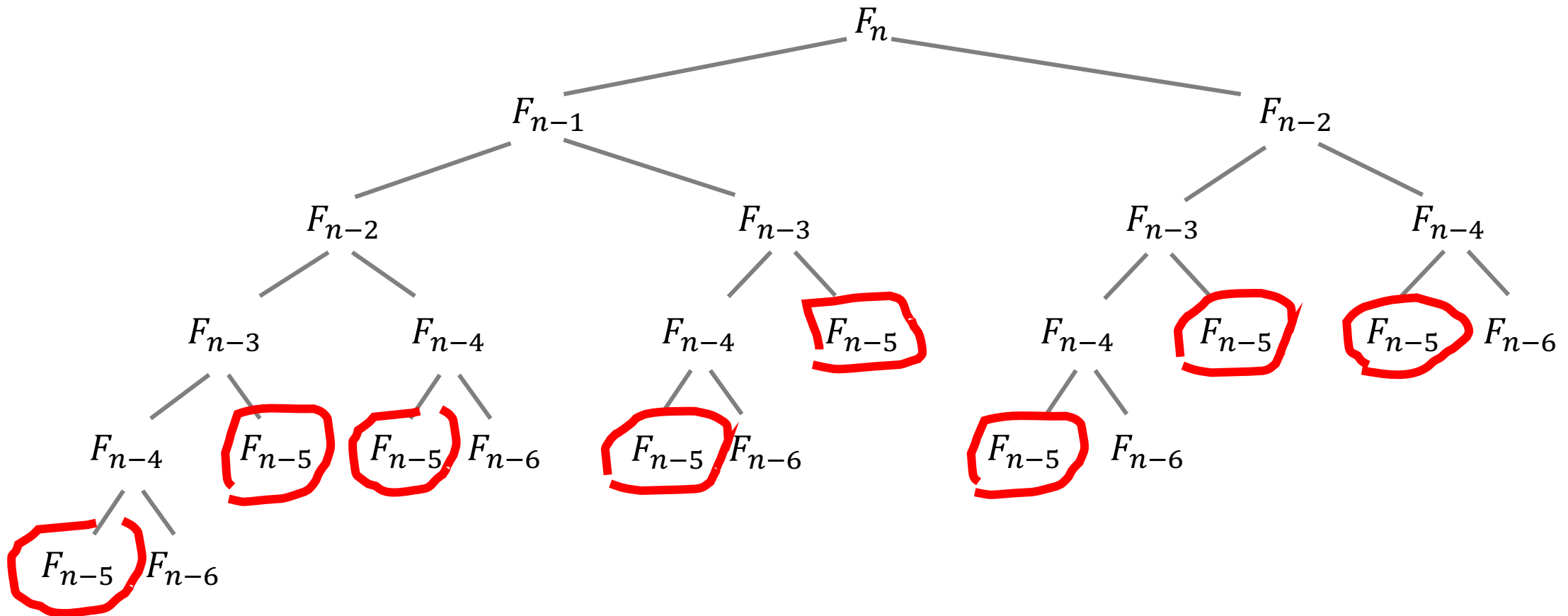
Why is recursion slow?

5 separate calls to compute $F(n - 4)$.



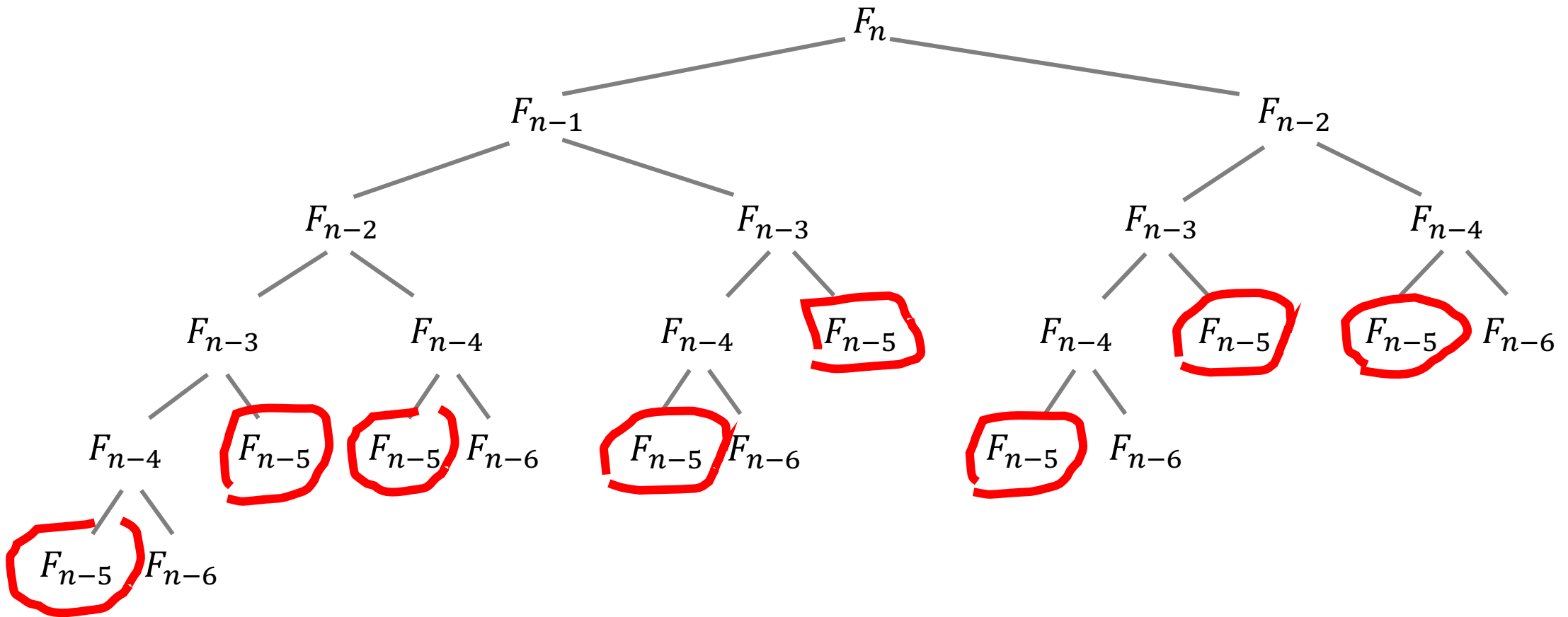
Why is recursion slow?

8 separate calls to compute $F(n - 5)$.



Why is recursion slow?

Repeatedly computing the same function is a waste of computation.



Dynamic Programming

Dynamic Programming (DP)

- Basic idea of DP: Using a table instead of recursion.

[illegible]

Dynamic Programming (DP)

- Basic idea of DP: Using a table instead of recursion.
- DP for Fibonacci numbers:
 - Compute $F(k)$ only once, for $k = 1, 2, \dots, n - 1$.
 - Record it in a table.
 - Use $F(k - 2)$ and $F(k - 1)$ for computing $F(k)$.

[illegible]

Dynamic Programming (DP)

- Basic idea of DP: Using a table instead of recursion.
- DP for Fibonacci numbers:
 - Compute $F(k)$ only once, for $k = 1, 2, \dots, n - 1$.
 - Record it in a table.
 - Use $F(k - 2)$ and $F(k - 1)$ for computing $F(k)$.

k	1	2	3	4	5	6	7	8	9	10
$F(k)$	1	1	2	3	5	?				

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1								

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1								

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2							

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3						

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3	5					

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3	5	8				

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3	5	8	13			

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3	5	8	13	21		

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;  
    F[2] = 1;  
    for (int k=3; k<=n; k++)  
        F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3	5	8	13	21	34	

DP for Fibonacci Numbers

```
long fib(int n) {  
    if (n == 1 || n == 2) return 1;  
    long F[n + 1];  
    F[1] = 1;
```

Time complexity: $O(n)$

```
    F[k] = F[k-1] + F[k-2];  
    return F[n];  
}
```

k	1	2	3	4	5	6	7	8	9	...
$F(k)$	1	1	2	3	5	8	13	21	34	...

Dynamic Programming

Step 1: Break a big problem into smaller sub-problems.

- Write down the recurrence formula.
- E.g., $F(k) = F(k - 2) + F(k - 1)$.

Step 2: Use a table instead of recursion.

- Record the solutions of the small sub-problems.
- Use the recorded solutions for solving bigger sub-problems.

Time Complexity

Input Size

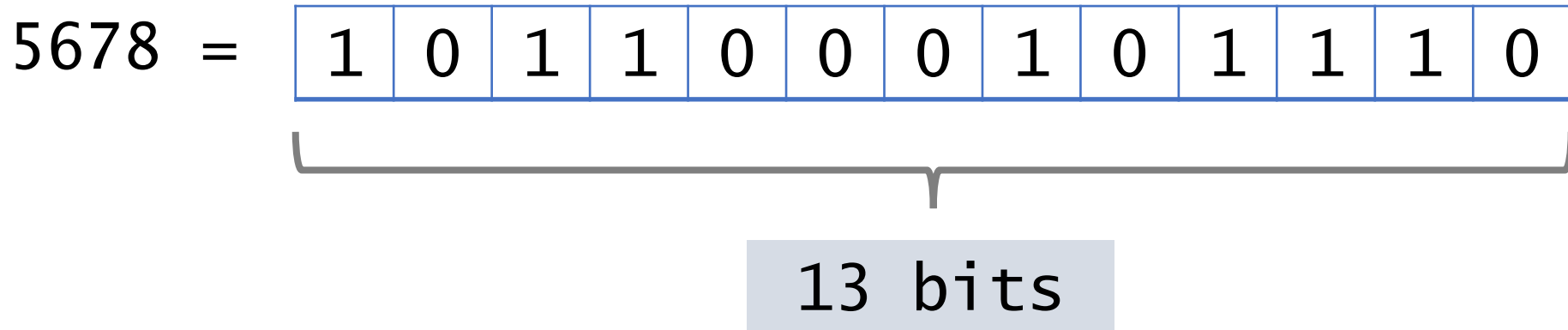
Input size: the number of bits for encoding the inputs.

- Find the n -th Fibonacci number.
- Input size: $b = \log_2 n$ bits.

Input Size

Input size: the number of bits for encoding the inputs.

- Find the n -th Fibonacci number.
- Input size: $b = \log_2 n$ bits.
- **Example 1:** $b = 13$ bits for encoding $n = 5678$.



Input Size

Input size: the number of bits for encoding the inputs.

- Find the n -th Fibonacci number.
- Input size: $b = \log_2 n$ bits.
- **Example 1:** $b = 13$ bits for encoding $n = 5678$.
- **Example 2:** $b = 20$ bits for encoding $n = 1$ million.
- **Example 3:** $b = 30$ bits for encoding $n = 1$ billion.

Time Complexities

Input size: the number of bits for encoding the inputs.

- Find the n -th Fibonacci number.
- Input size: $b = \log_2 n$ bits.

Time complexity of dynamic programming is **exponential**:

$$O(n) = O(2^b).$$

Time Complexities

Input size: the number of bits for encoding the inputs.

- Find the n -th Fibonacci number.
- Input size: $b = \log_2 n$ bits.

Time complexity of recursion is **double exponential**:

$$O(a^n) = O\left(a^{2^b}\right).$$

Thank You!

Time Complexity of Recursion

- **Time complexity:** $T(n) = a^n$, where $a = \frac{1+\sqrt{5}}{2}$.

- **Proof by induction:**

- Recursion: $T(n) = T(n-1) + T(n-2)$.
- Assume $T(n-2) = a^{n-2}$.
- Assume $T(n-1) = a^{n-1} = a \cdot a^{n-2}$.
- Then $T(n) = T(n-1) + T(n-2)$

$$= a^{n-2} \cdot (a + 1).$$

$$= a^{n-2} \cdot a^2$$

$$= a^n.$$

Why? Since $a = \frac{1+\sqrt{5}}{2}$, we have $a^2 = a + 1$.