浅谈递归

Kaizing Wong

UICHCC技术沙龙

2019.4.21

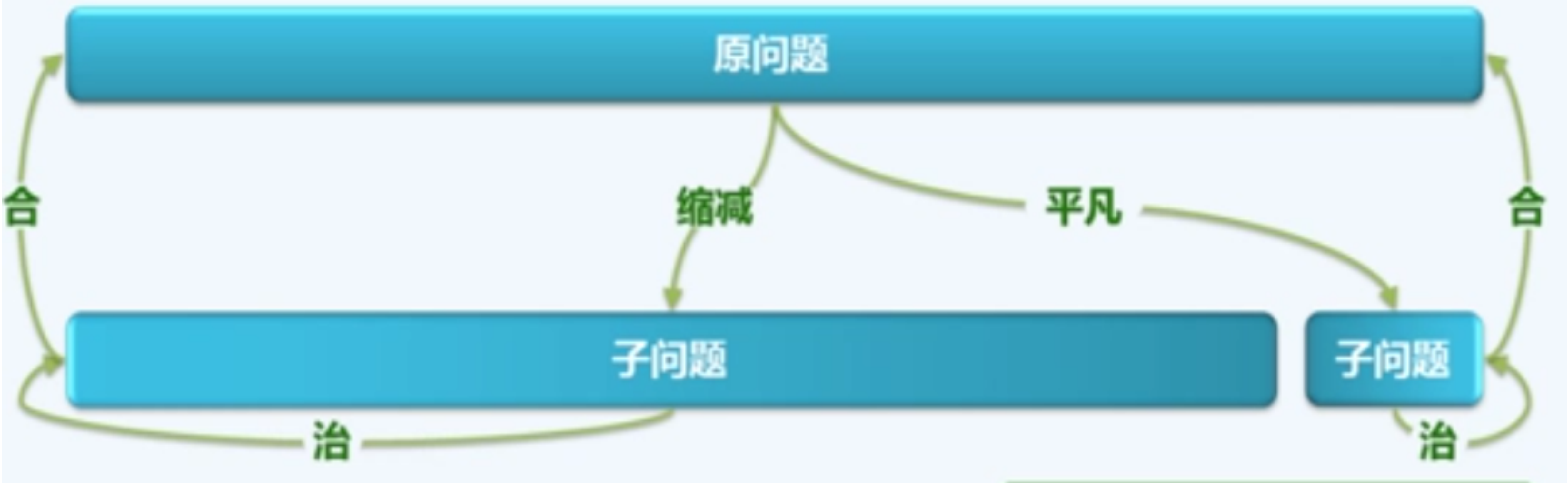
递归

简而治之

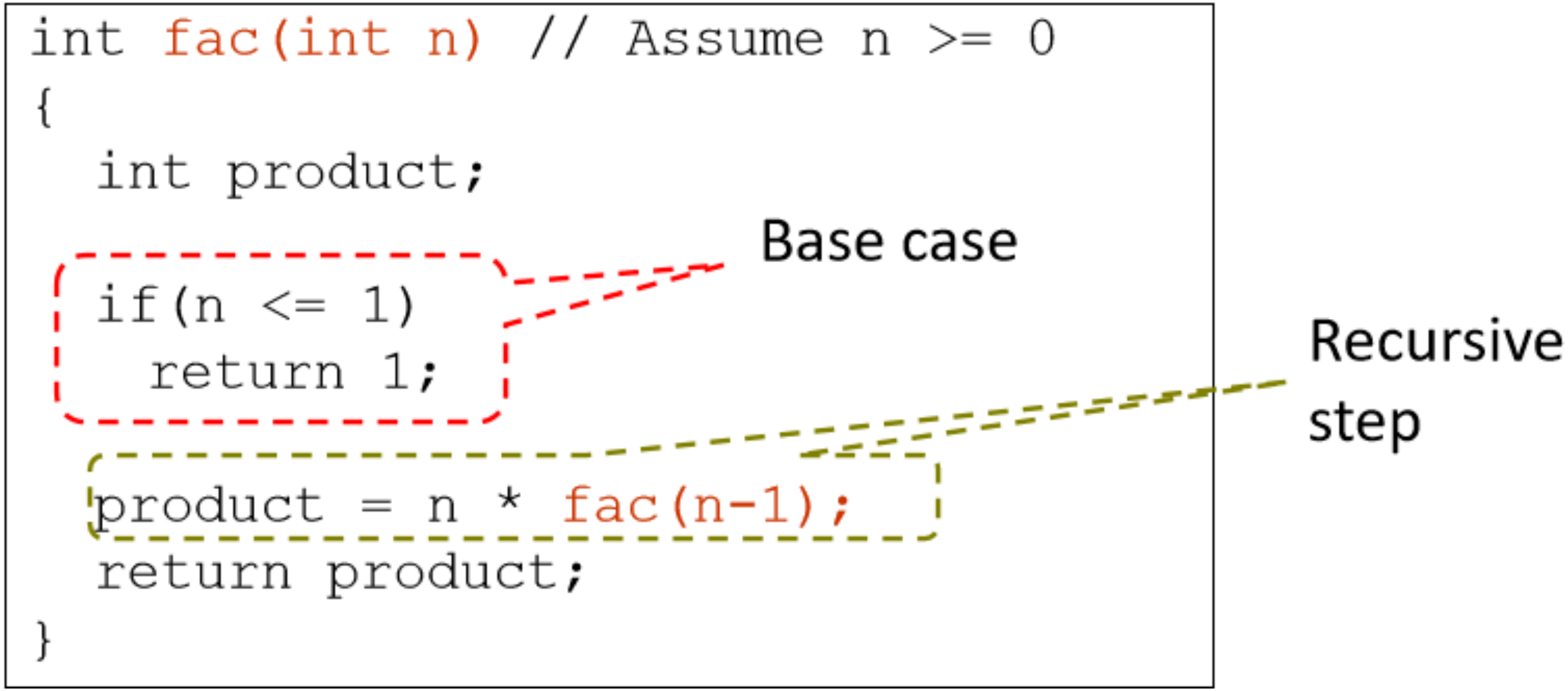
分而治之

查找（扩展）

简而治之



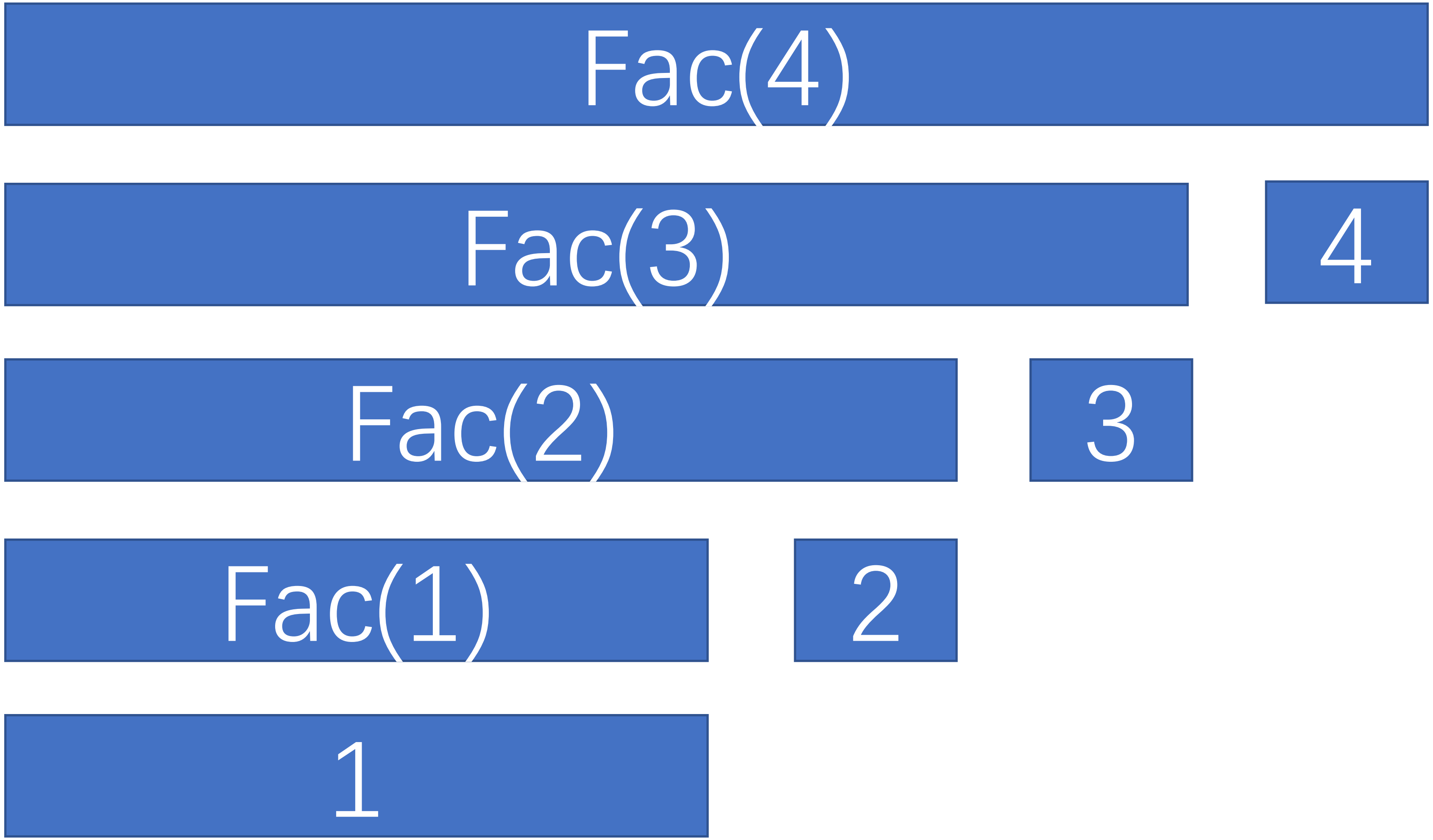
简而治之



```
int fac(int n) // Assume n >= 0
{
  int product;

  if(n <= 1)
    return 1;

  product = n * fac(n-1);
  return product;
}
```

Base case

Recursive step



Fac(4)

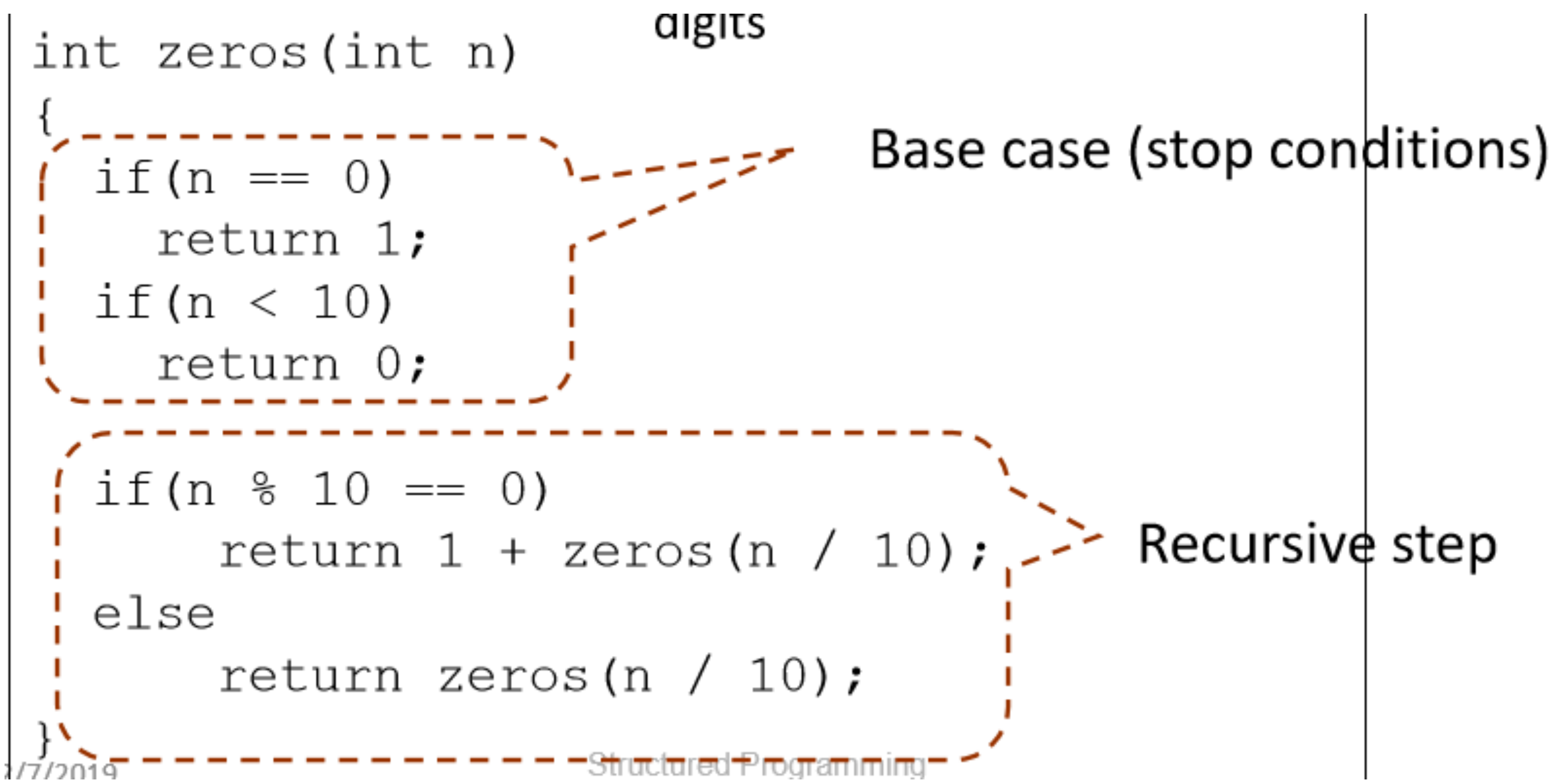
Fac(3) 4

Fac(2) 3

Fac(1) 2

1

Lecture例题



digits

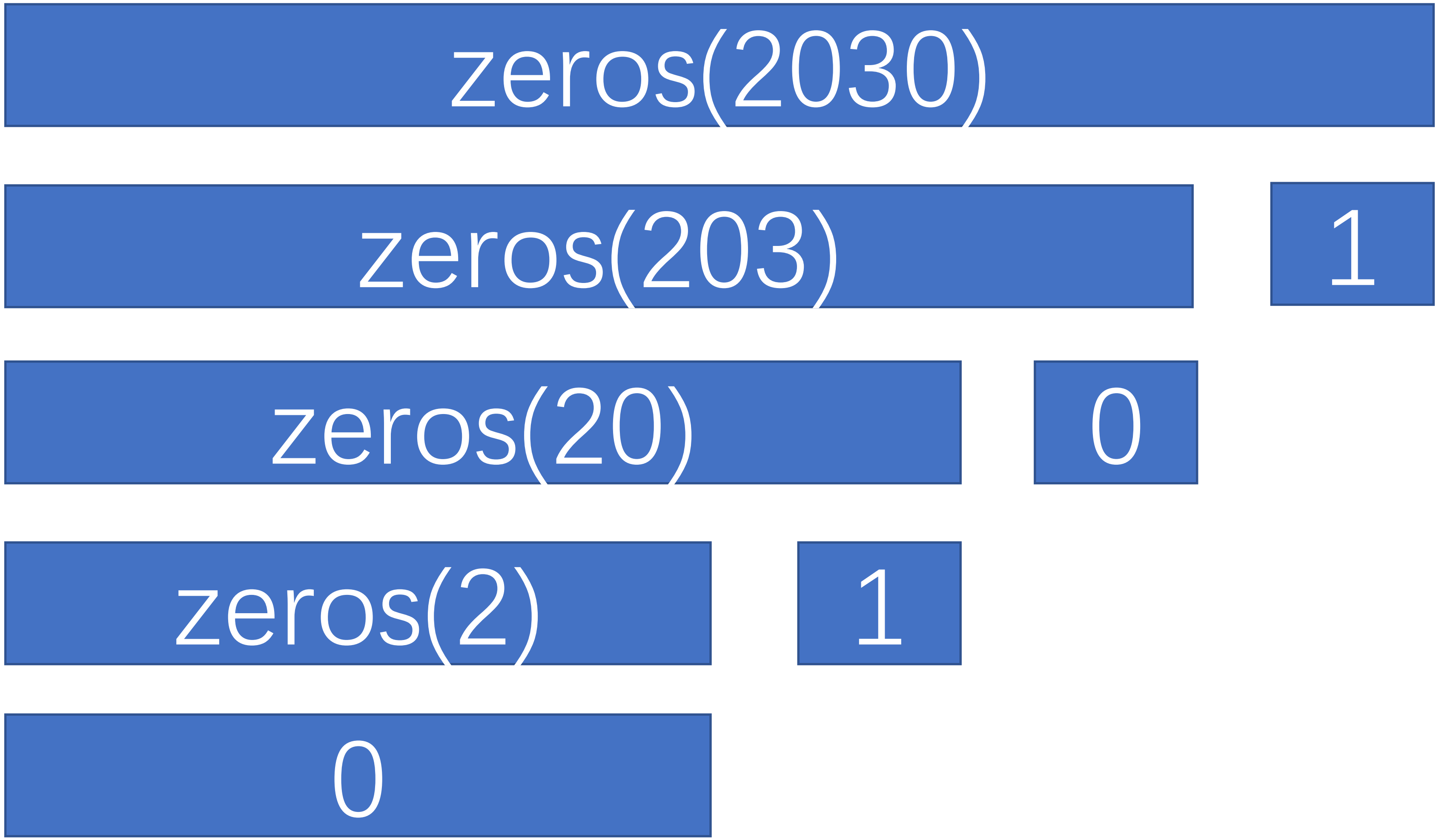
```
int zeros(int n)
{
  if(n == 0)
    return 1;
  if(n < 10)
    return 0;

  if(n % 10 == 0)
      return 1 + zeros(n / 10);
  else
      return zeros(n / 10);
}
```

Base case (stop conditions)

Recursive step

/7/2019

Structured Programming



zeros(2030)

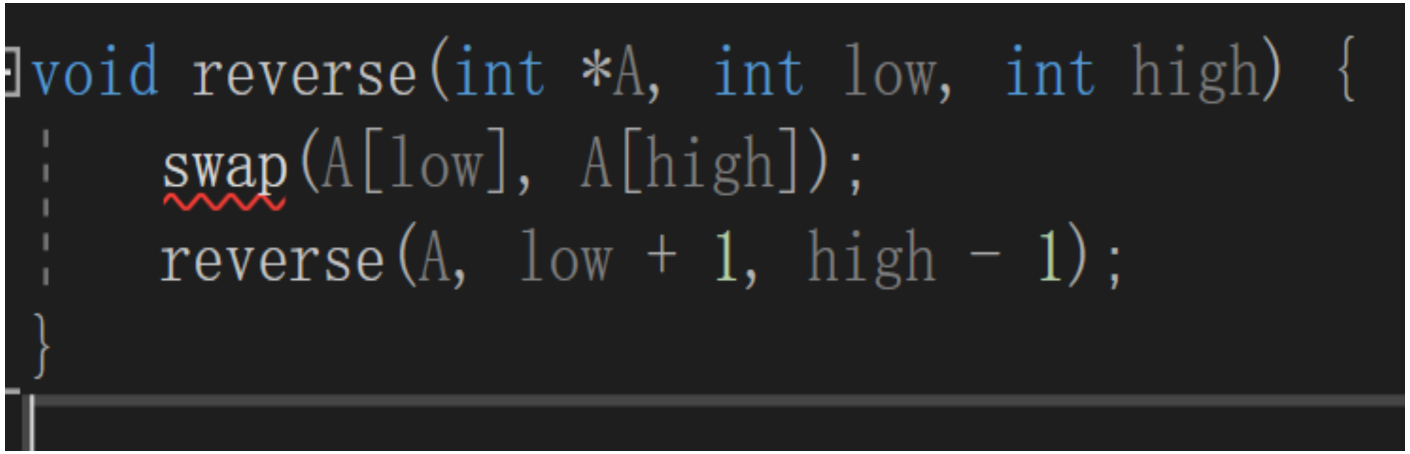
zeros(203) 1

zeros(20) 0

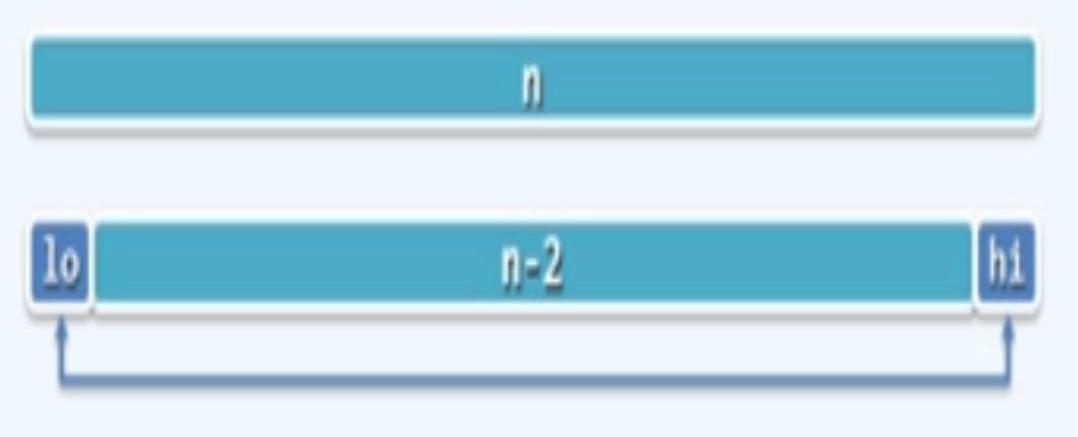
zeros(2) 1

0

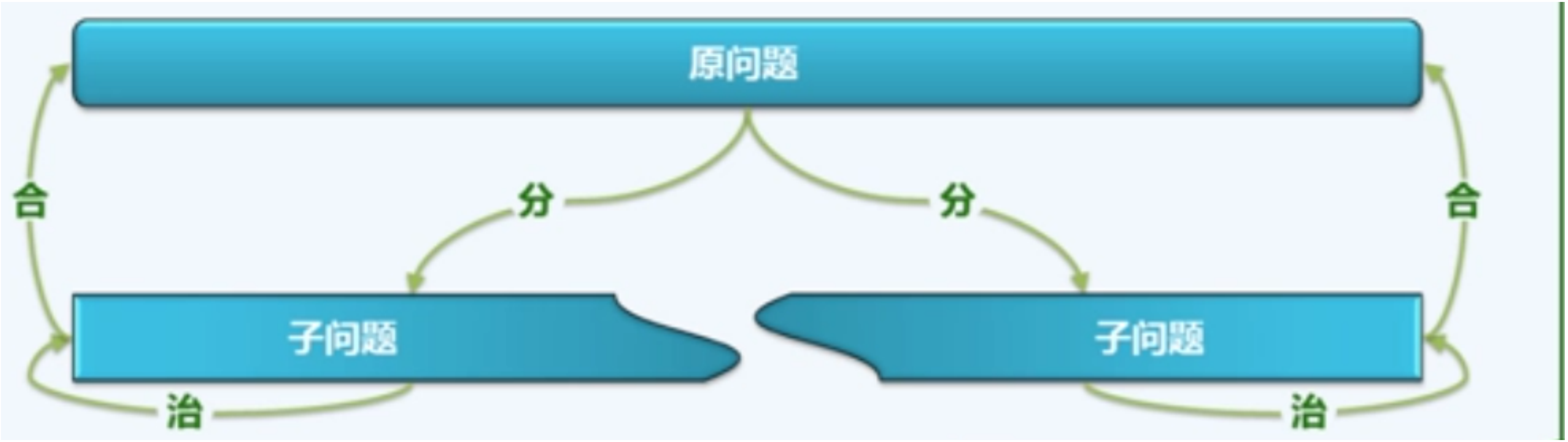
数组颠倒



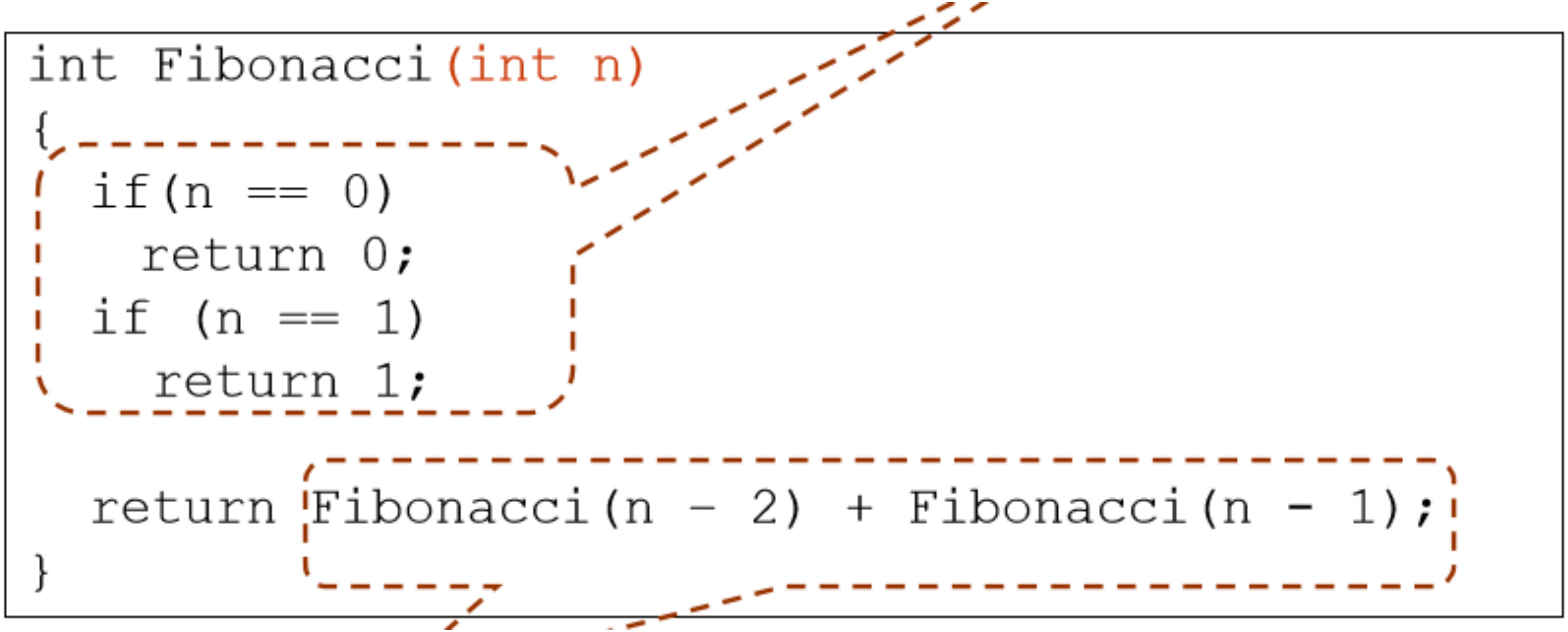
```
void reverse(int *A, int low, int high) {
    swap(A[low], A[high]);
    reverse(A, low + 1, high - 1);
}
```



分而治之

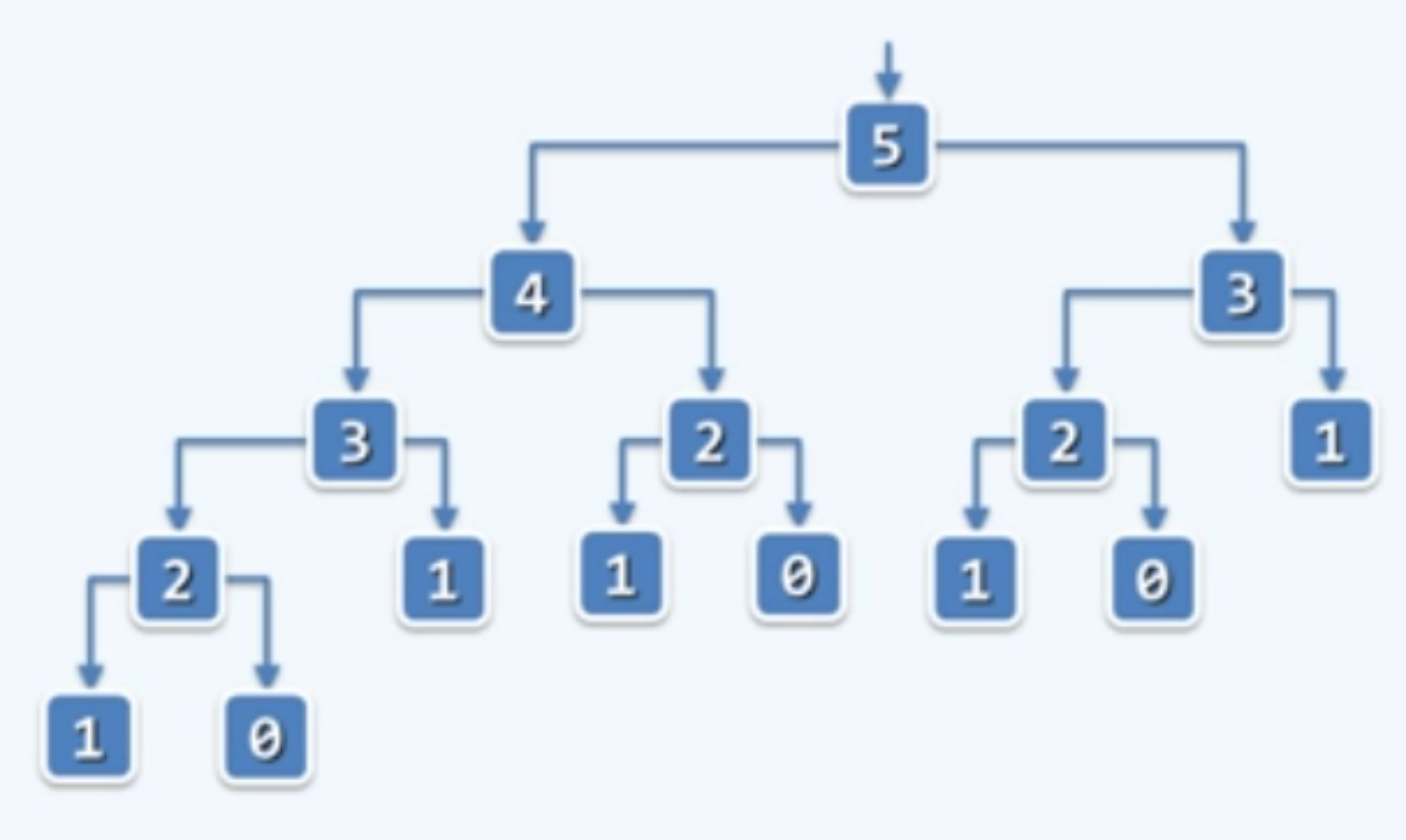


斐波那契数列



```
int Fibonacci(int n)
{
  if(n == 0)
    return 0;
  if (n == 1)
    return 1;

  return Fibonacci(n - 2) + Fibonacci(n - 1);
}
```

So slow, but why ?

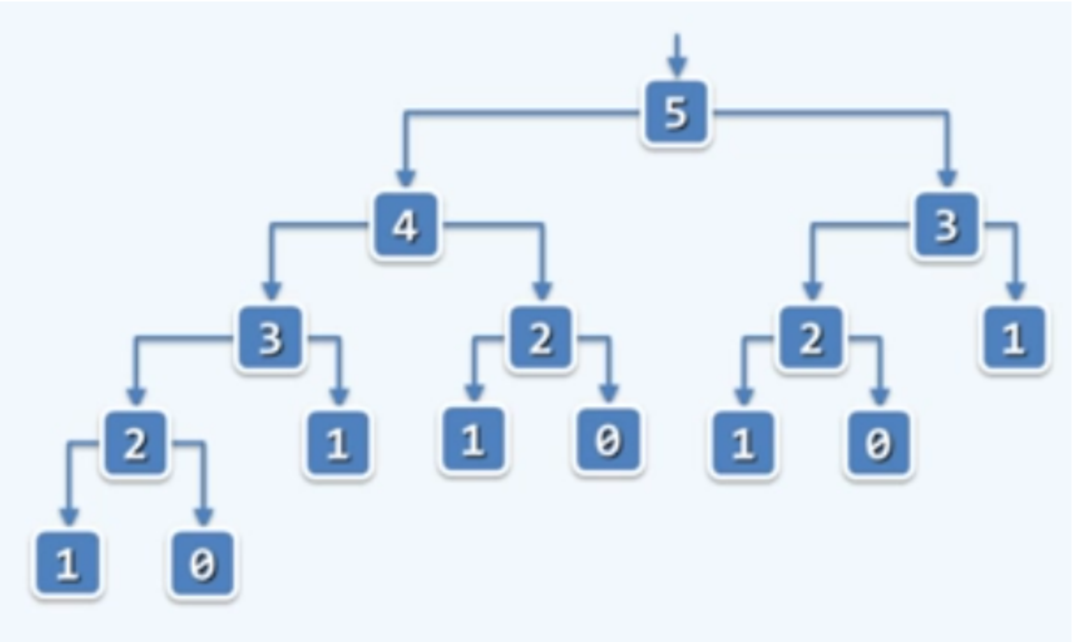


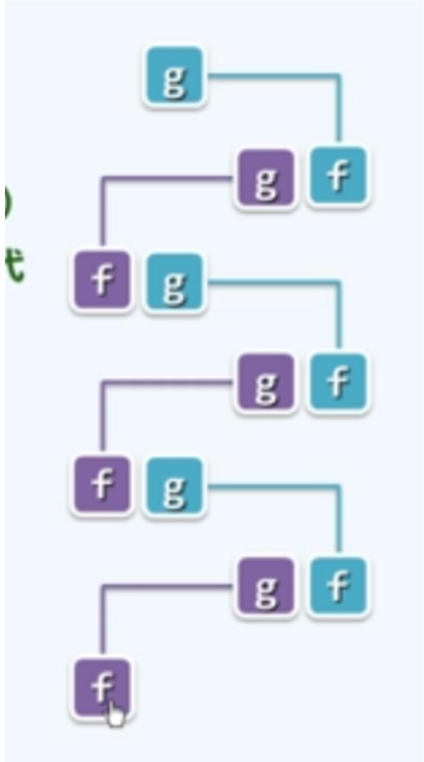
不够直观。。。

$T(0) = T(1)=1$

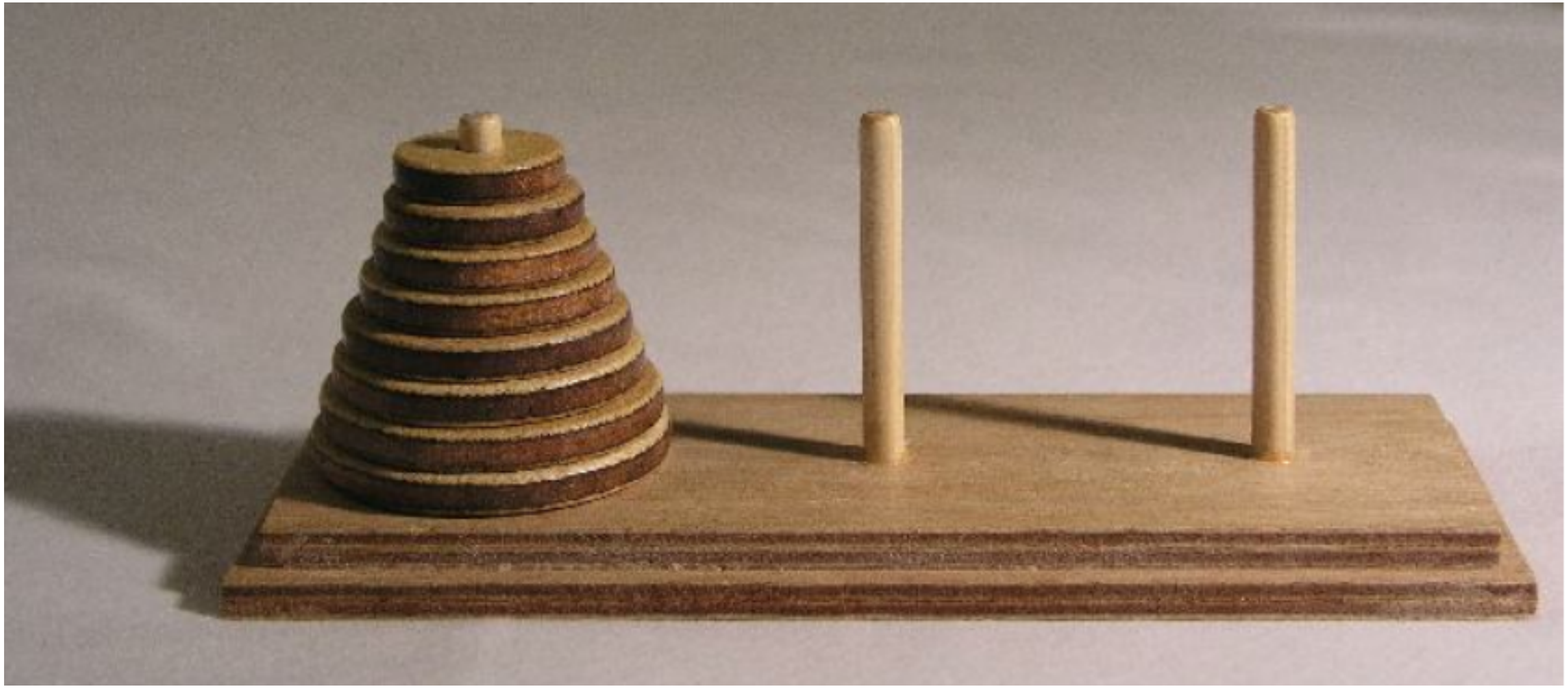
$T(N) = T(N-1)+T(N-2)$

改进斐波那契





Van游戏



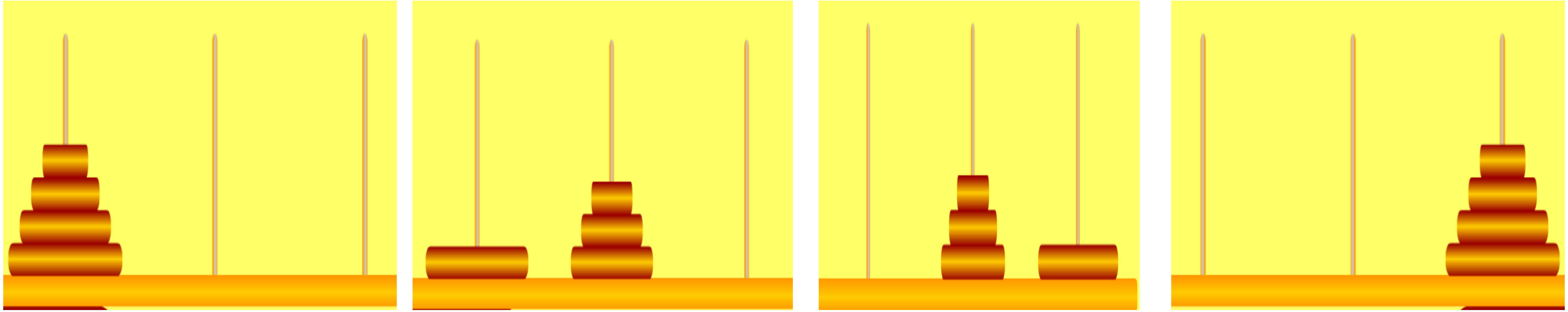
- 1 -> y
- 2 -> z
- 1 -> z
- 3 -> y
- 1 -> x
- 2 -> y
- 1 -> y
- 4 -> z
- 1 -> z
- 2 -> x
- 1 -> x
- 3 -> z
- 1 -> y
- 2 -> z
- 1 -> z

- 1 -> y
- 2 -> z
- 1 -> z
- 3 -> y
- 1 -> x
- 2 -> y
- 1 -> y

- 4-> z

- 1 -> z
- 2 -> x
- 1 -> x
- 3 -> z
- 1 -> y
- 2 -> z
- 1 -> z

实际运行步骤

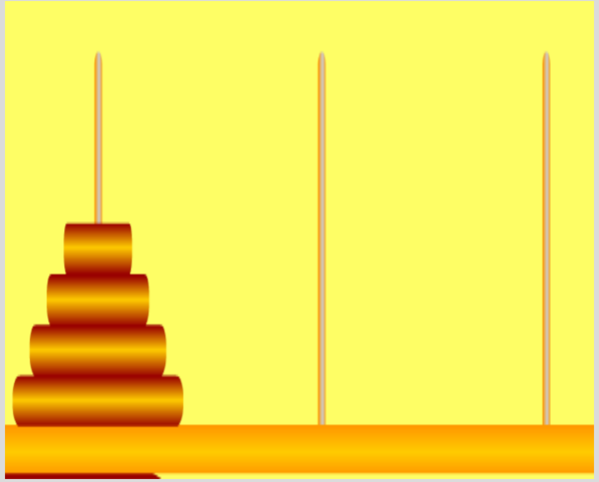


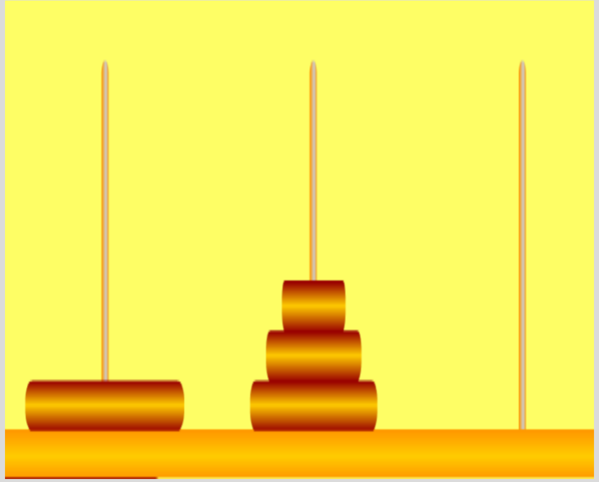
- 1 -> y
- 2 -> z
- 1 -> z
- 3 -> y
- 1 -> x
- 2 -> y
- 1 -> y

- 4-> z

- 1 -> z
- 2 -> x
- 1 -> x
- 3 -> z
- 1 -> y
- 2 -> z
- 1 -> z

- 1 -> y
- 2 -> z
- 1 -> z
- 3 -> y
- 1 -> x
- 2 -> y
- 1 -> y





Hanoi(4)

- 1 -> y
- 2 -> z
- 1 -> z
- 3 -> y
- 1 -> x
- 2 -> y
- 1 -> y
- 4 -> z
- 1 -> z
- 2 -> x
- 1 -> x
- 3 -> z
- 1 -> y
- 2 -> z
- 1 -> z

Hanoi(3)

Hanoi(3)

Hanoi(2)

Hanoi(2)

Hanoi(2)

Hanoi(2)

Hanoi(1)

Hanoi(1)

Hanoi(1)

Hanoi(1)

Hanoi(1)

Hanoi(1)

Hanoi(1)

Hanoi(1)

Coding

- 定义接口
- Hanoi(int n, char x,char y,chay z);

  将n个圆盘从x移动到y

算法分析

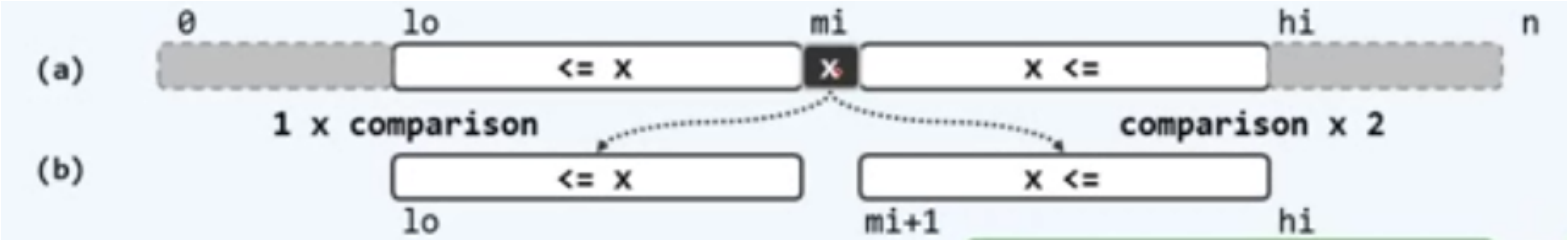
$T(1)=1$

$T(N)=2T(N-1)+1$

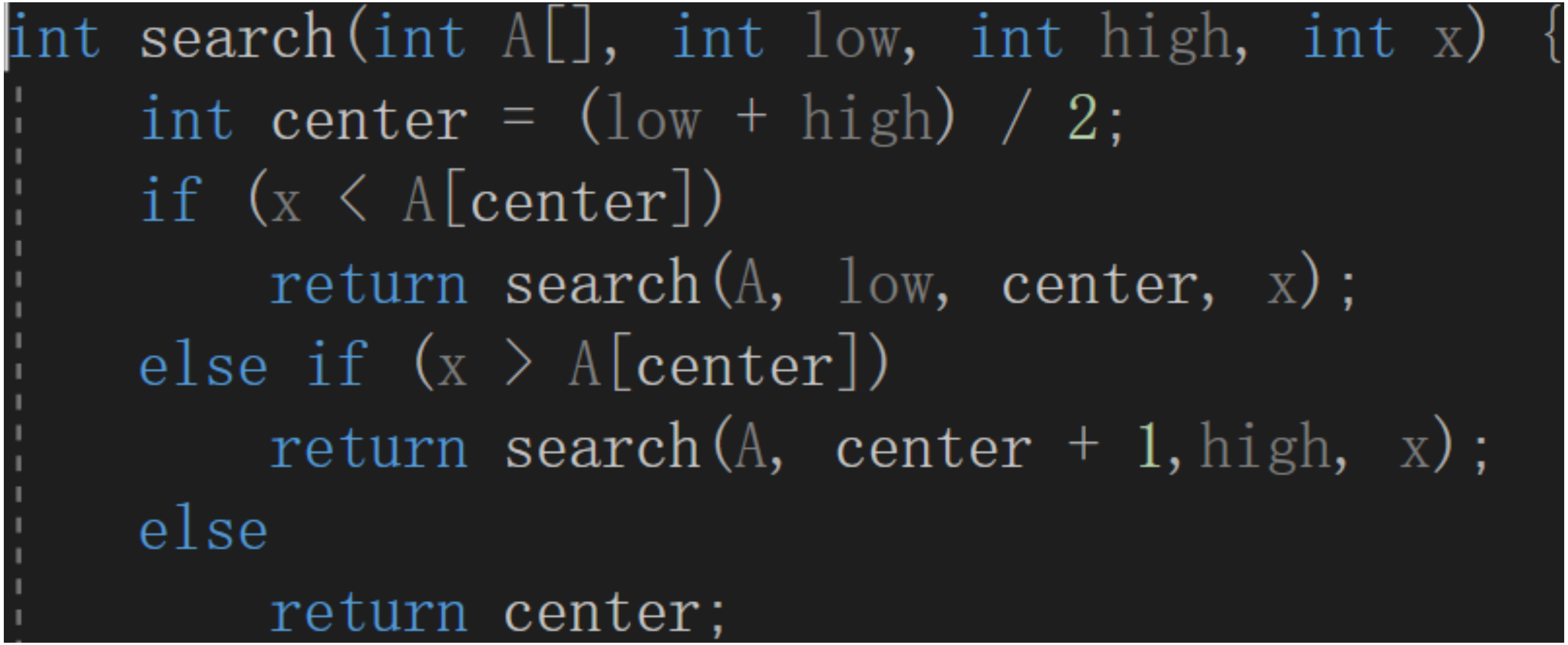
有序数组查找

- 遍历

二分查找（版本A）

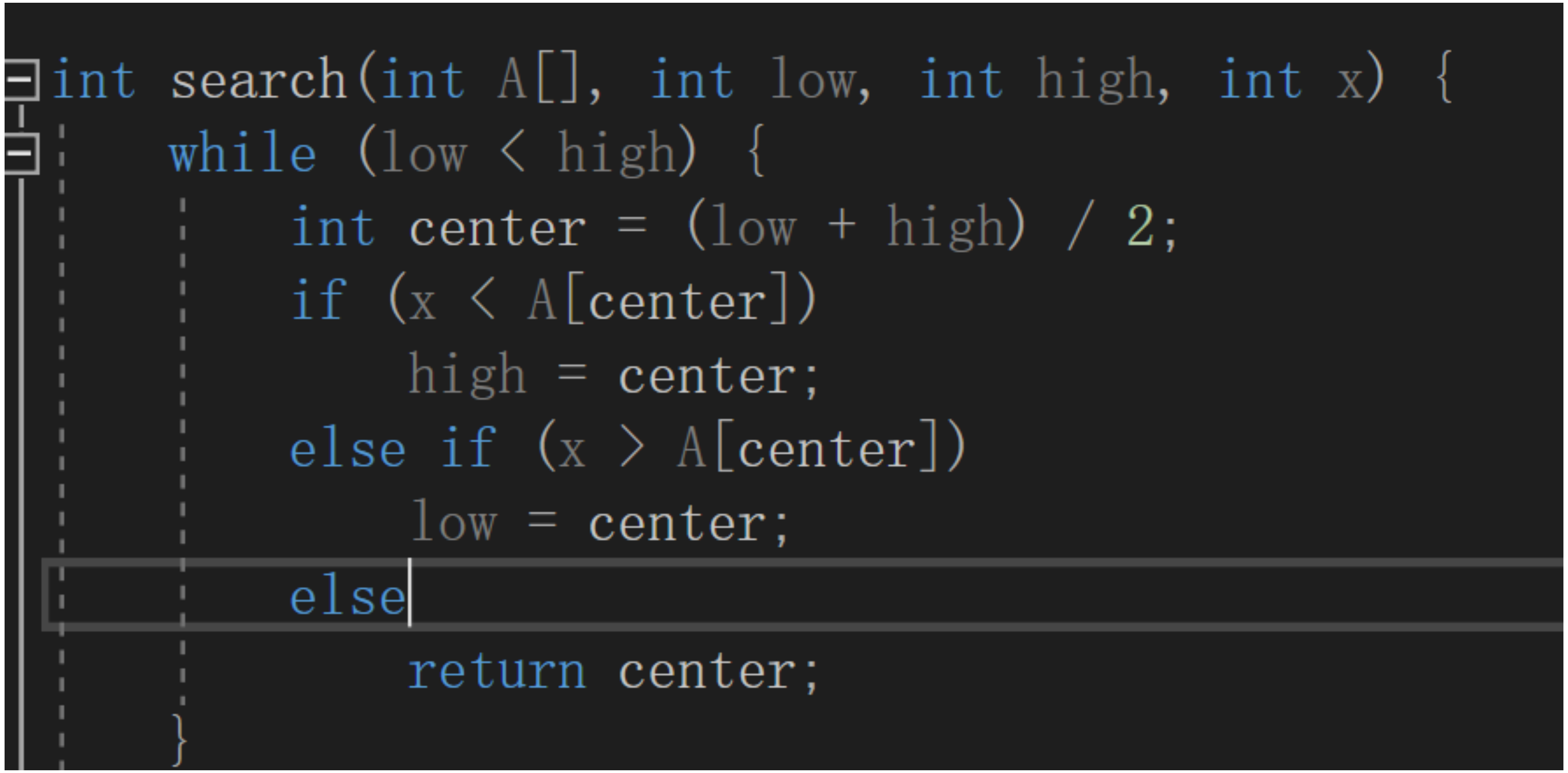


主体代码



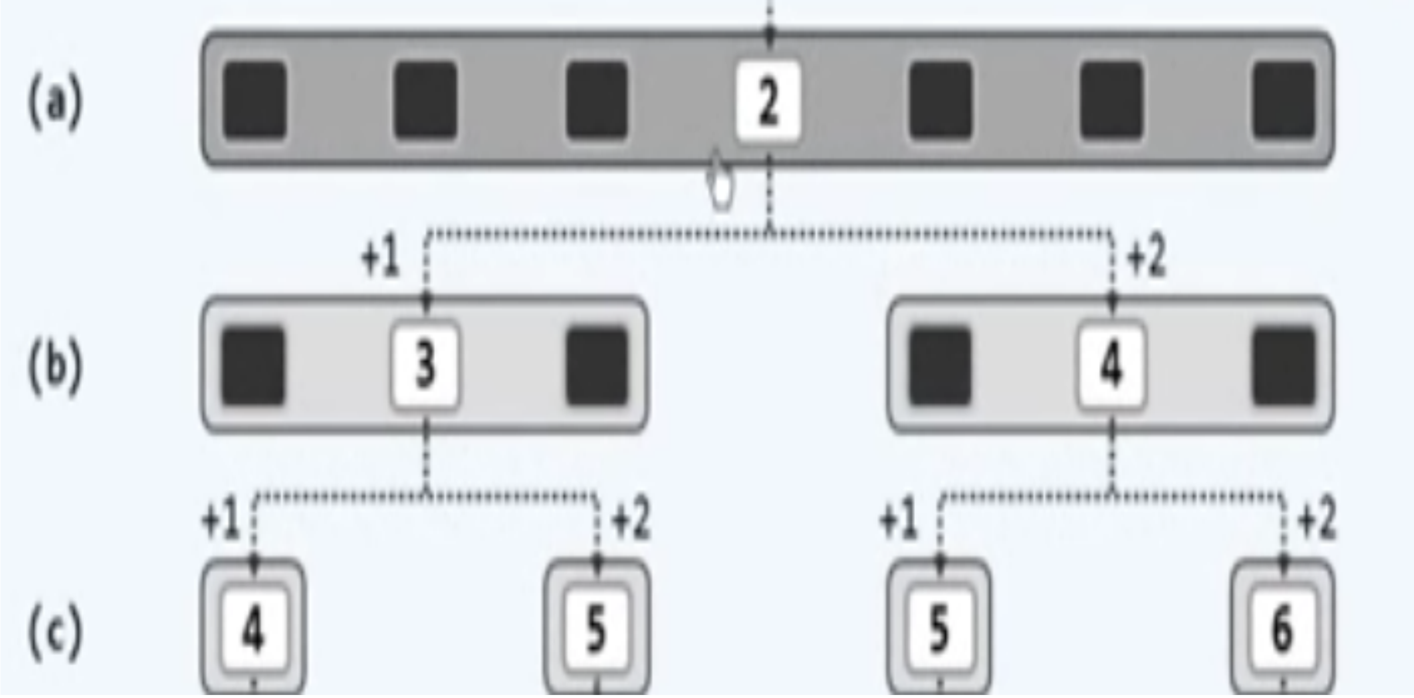
```
int search(int A[], int low, int high, int x) {
    int center = (low + high) / 2;
    if (x < A[center])
        return search(A, low, center, x);
    else if (x > A[center])
        return search(A, center + 1,high, x);
    else
        return center;
```

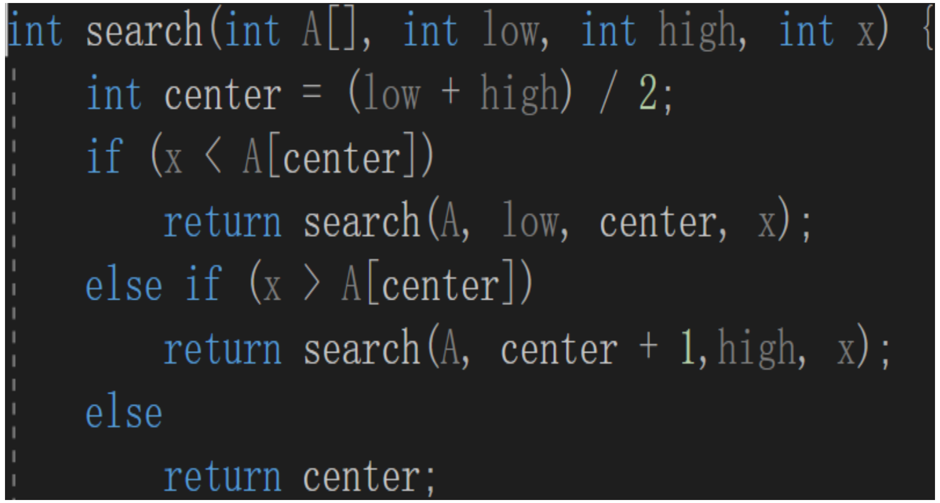
迭代版本



```
int search(int A[], int low, int high, int x) {
    while (low < high) {
        int center = (low + high) / 2;
        if (x < A[center])
            high = center;
        else if (x > A[center])
            low = center;
        else
            return center;
    }
```

一点小问题

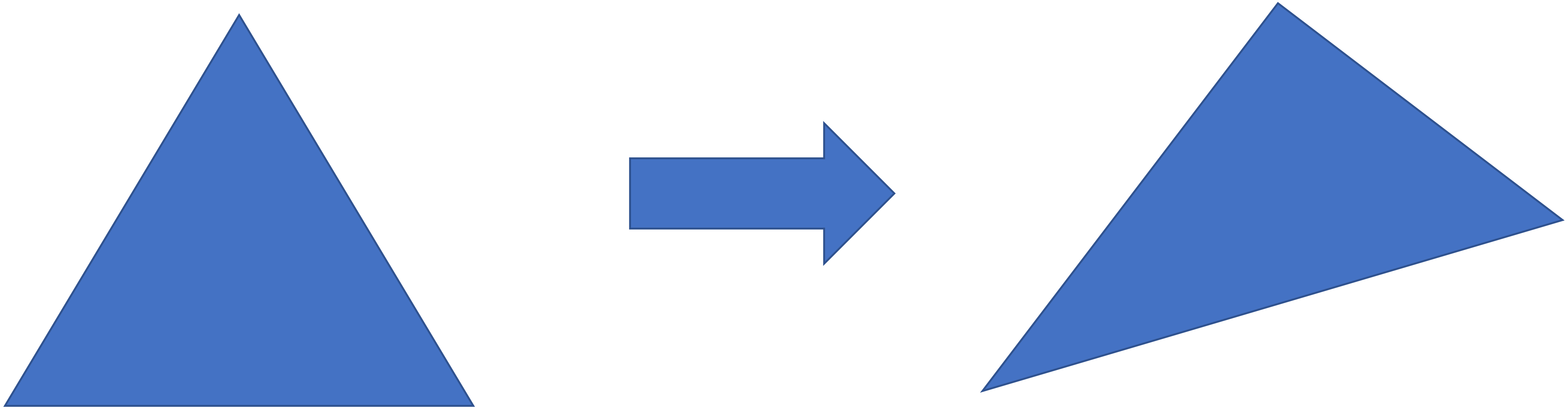




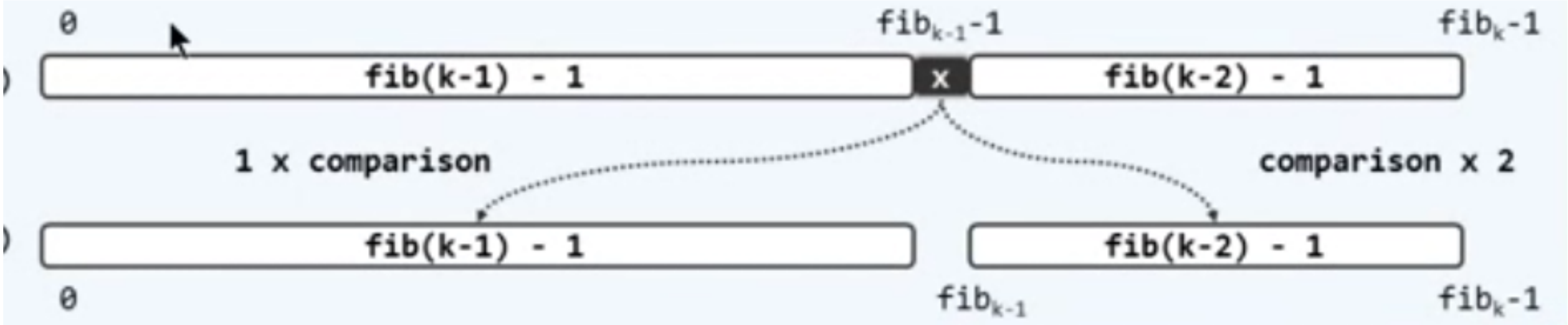
```
int search(int A[], int low, int high, int x) {
    int center = (low + high) / 2;
    if (x < A[center])
        return search(A, low, center, x);
    else if (x > A[center])
        return search(A, center + 1,high, x);
    else
        return center;
```

二分查找改进

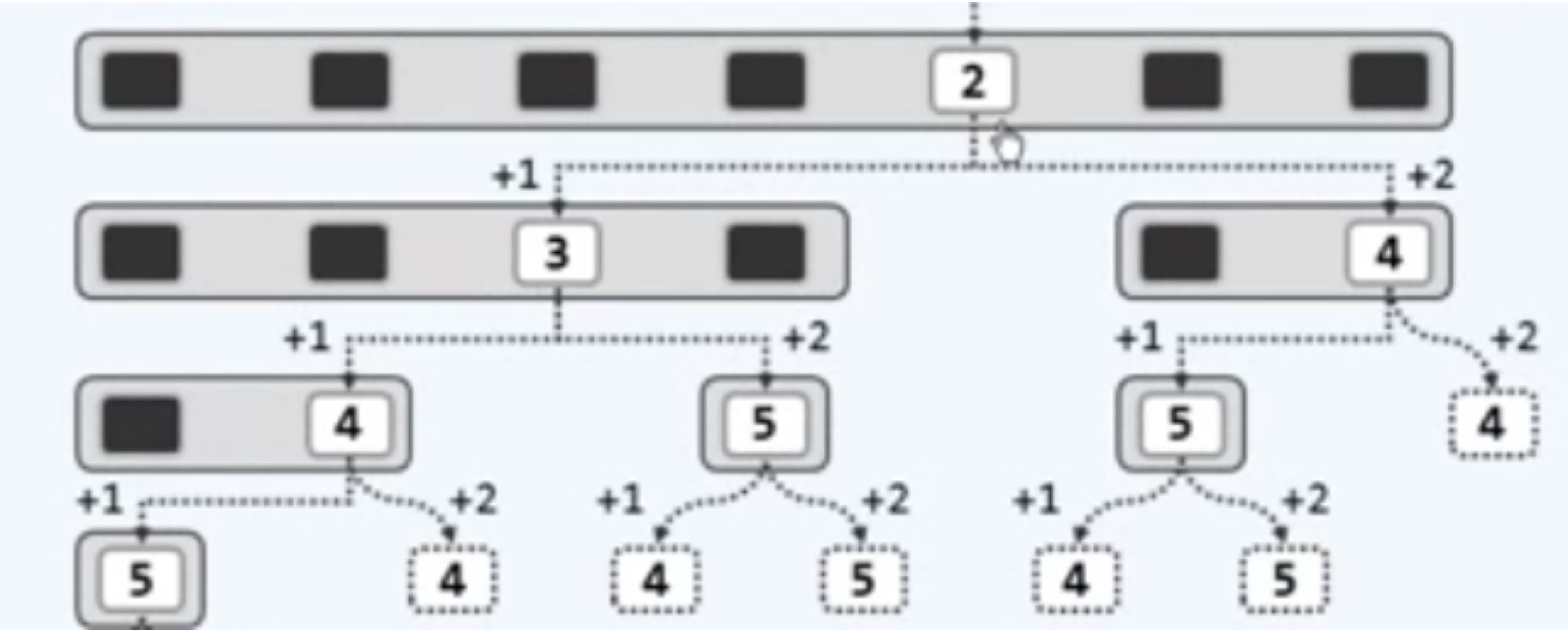
问题：比较次数不等，递归深度相同



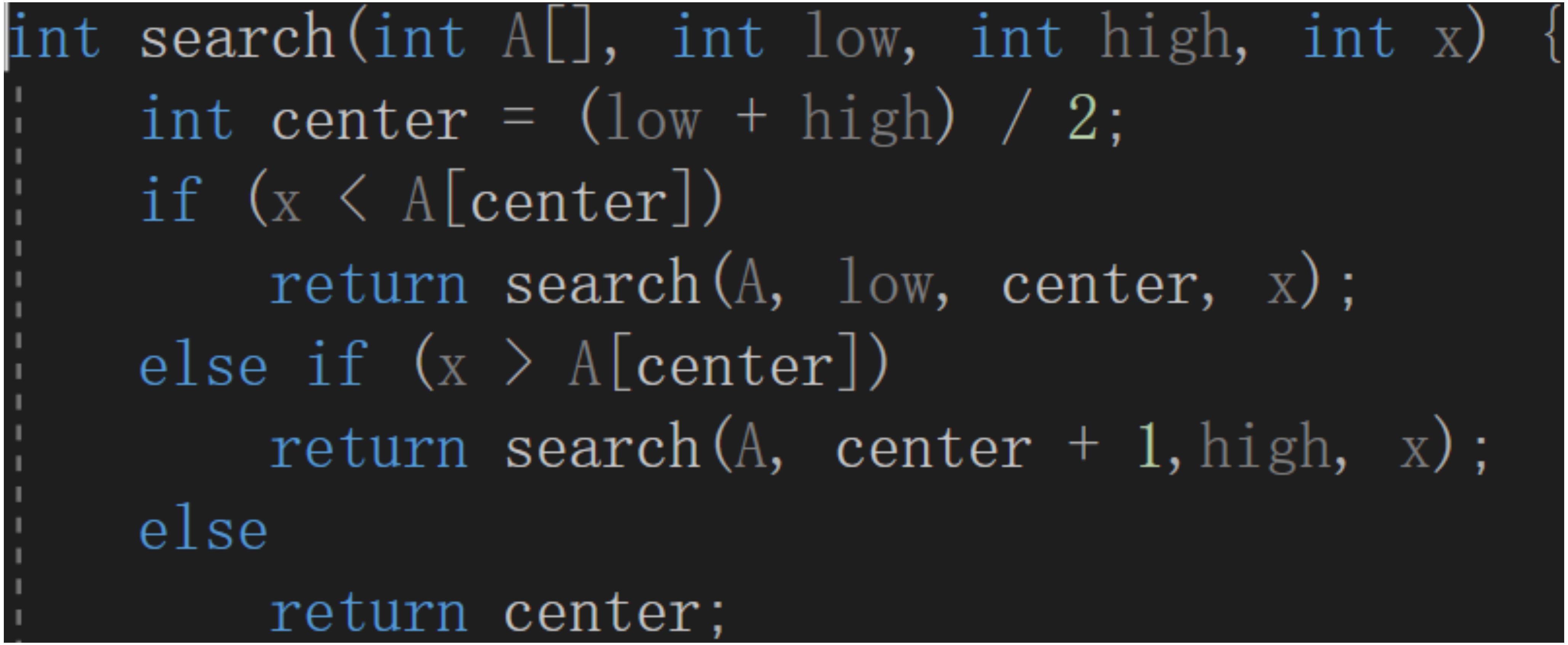
怎么切


0
fib_{k-1}-1
fib_k-1
fib(k-1) - 1
x
fib(k-2) - 1
1 x comparison
comparison x 2
fib(k-1) - 1
fib(k-2) - 1
0
fib_{k-1}
fib_k-1

查找树

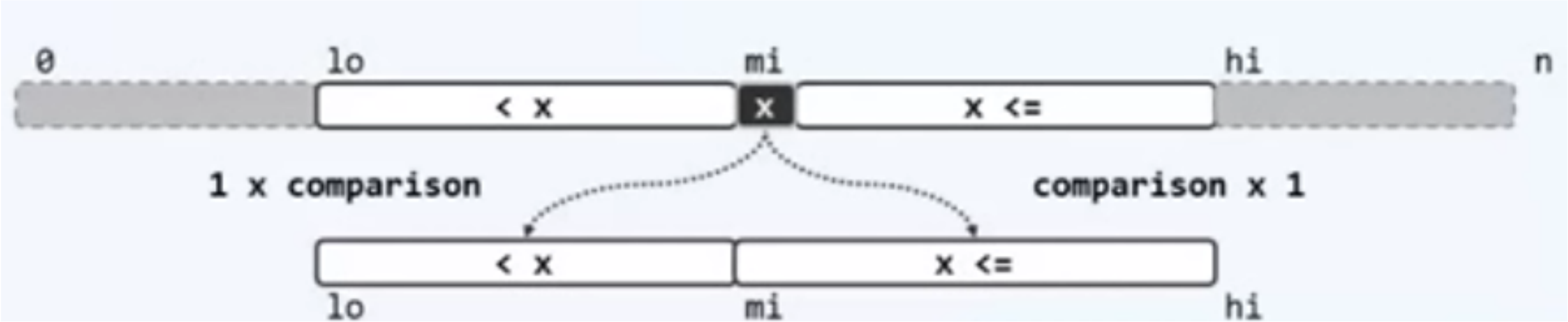


二分查找版本(B)



```
int search(int A[], int low, int high, int x) {
    int center = (low + high) / 2;
    if (x < A[center])
        return search(A, low, center, x);
    else if (x > A[center])
        return search(A, center + 1,high, x);
    else
        return center;
```

消除左右不平衡


0
lo
mi
hi
n
< x
x
x <=
1 x comparison
comparison x 1
< x
x <=
lo
mi
hi