Linked List, Recursion

王慧妍

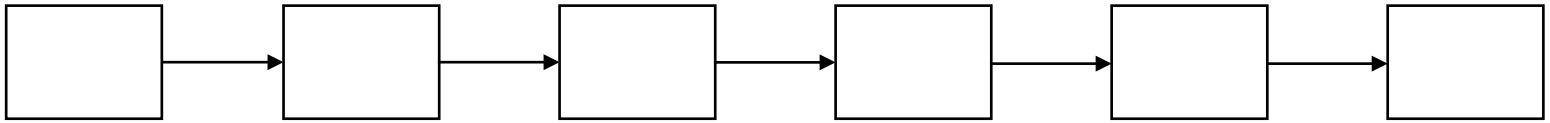
why@nju.edu.cn

南京大学 软件学院 计算机软件研究所

Linked List

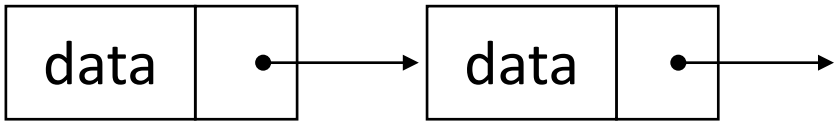
Linked List

- 链表：内存非连续的线性结构



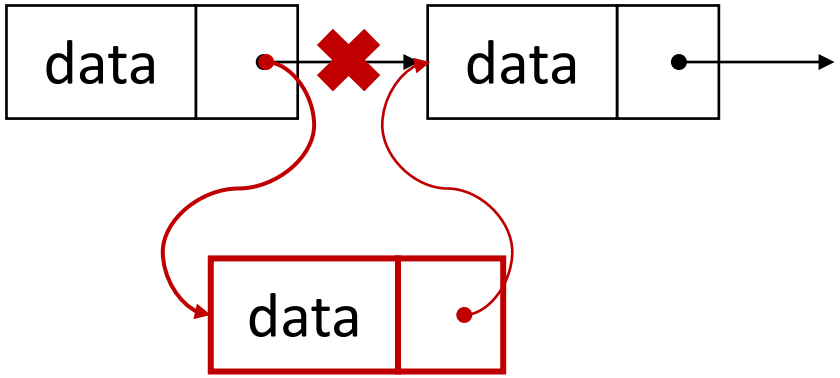
- 链表节点
  - 存储有价值的数据
  - 指向下一个链表节点的指针

  - `struct node {int data; struct node *next;};`

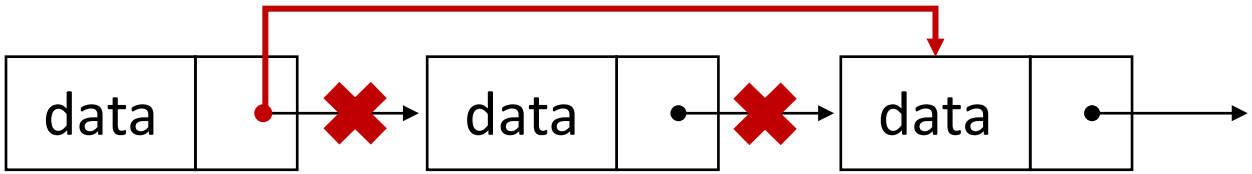


链表的节点插入和删除

- 新链表节点的插入



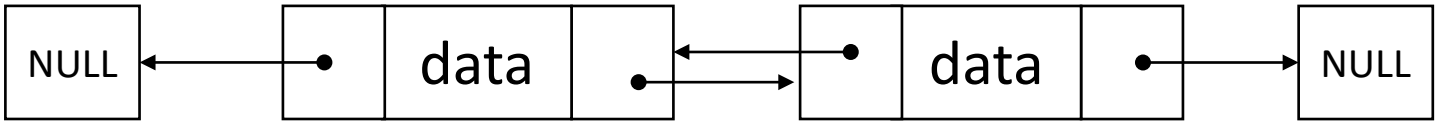
- 已有链表节点的删除



单向链表和双向链表

- 单向链表
  - `struct node {int data; struct node *next;};`

- 双向链表
  -
```
struct node {
  struct node * prev;
  int data;
  struct node *next;
};
```

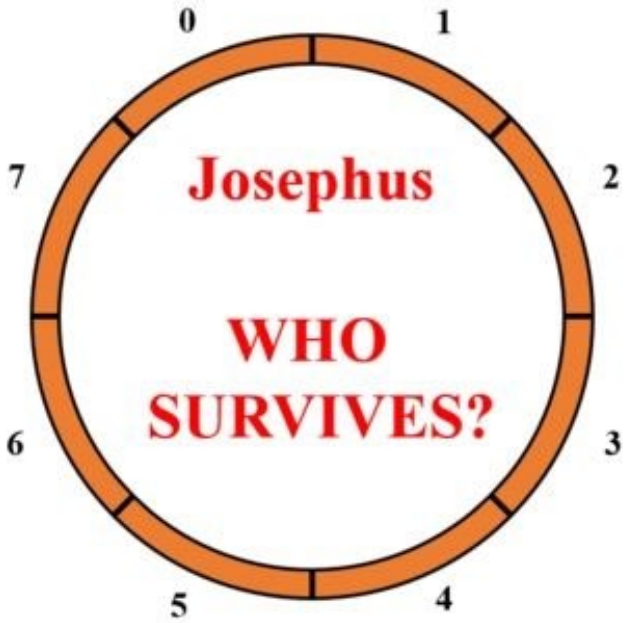


循环链表

- 与单向链表相比，区别在于
  - 尾节点的指针成员不再指向NULL，而是指向首节点

约瑟夫问题

- 由来
  - 在罗马人占领乔塔帕特后，39 个犹太人与Josephus 及他的朋友躲到一个洞中，39个犹太人决定宁愿死 也不要被敌人抓到，于是决定了一个自杀方式。41个人排成一个圆圈，由第1个人开始报数，每报数 到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏 。



约瑟夫问题

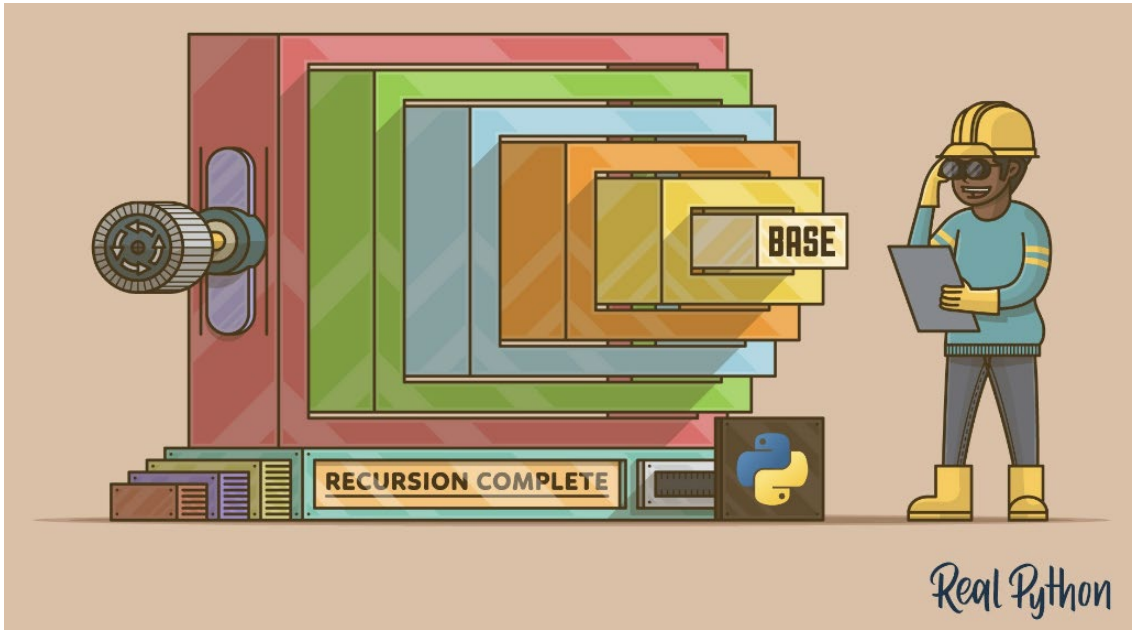
- 用数组实现约瑟夫环?

- 试试用循环链表来实现约瑟夫环?
  - Joseph.c

Recursion

特殊的函数：递归Recursion

- 一个调用本身的函数
  - A function that calls (调用) itself.
- 重点是如何能够递归的思考问题

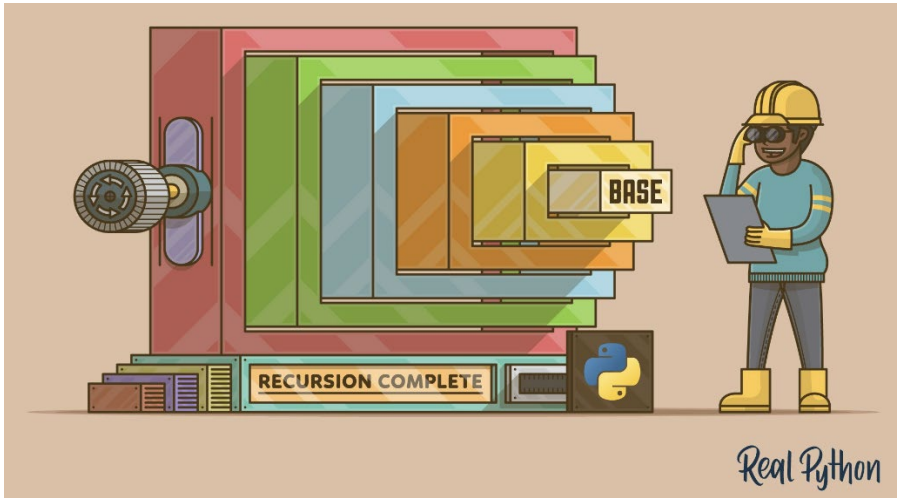


It’s a looooooooooong way to go to master recusion!

Mathematical induction

- 数学归纳法
  - Base Case (基础情况)：n = 0
  - Inductive Step (归纳步骤)：n = k → n = k+1

- 递归函数：自己调用自己的函数
  - 不能变：函数名称，功能，返回类型
  - 唯一能变的部分：参数
  - 通过参数控制问题的解决规模
  - 何时结束?



递归函数的堆栈管理

- Visualization of Function Calls @ C Tutor

求阶乘

- F(n) = n!

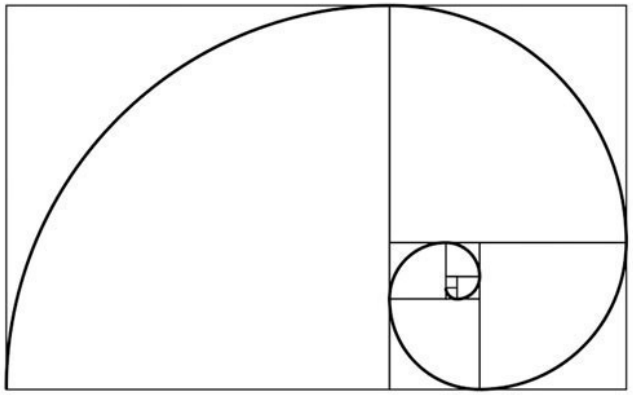
$$n! = \begin{cases} 1 & (n = 0, 1) \\ n * (n-1)! & (n > 1) \end{cases}$$

- 循环 v.s. 递归

斐布拉契数列

- 典型递归函数例子
  - fib.c

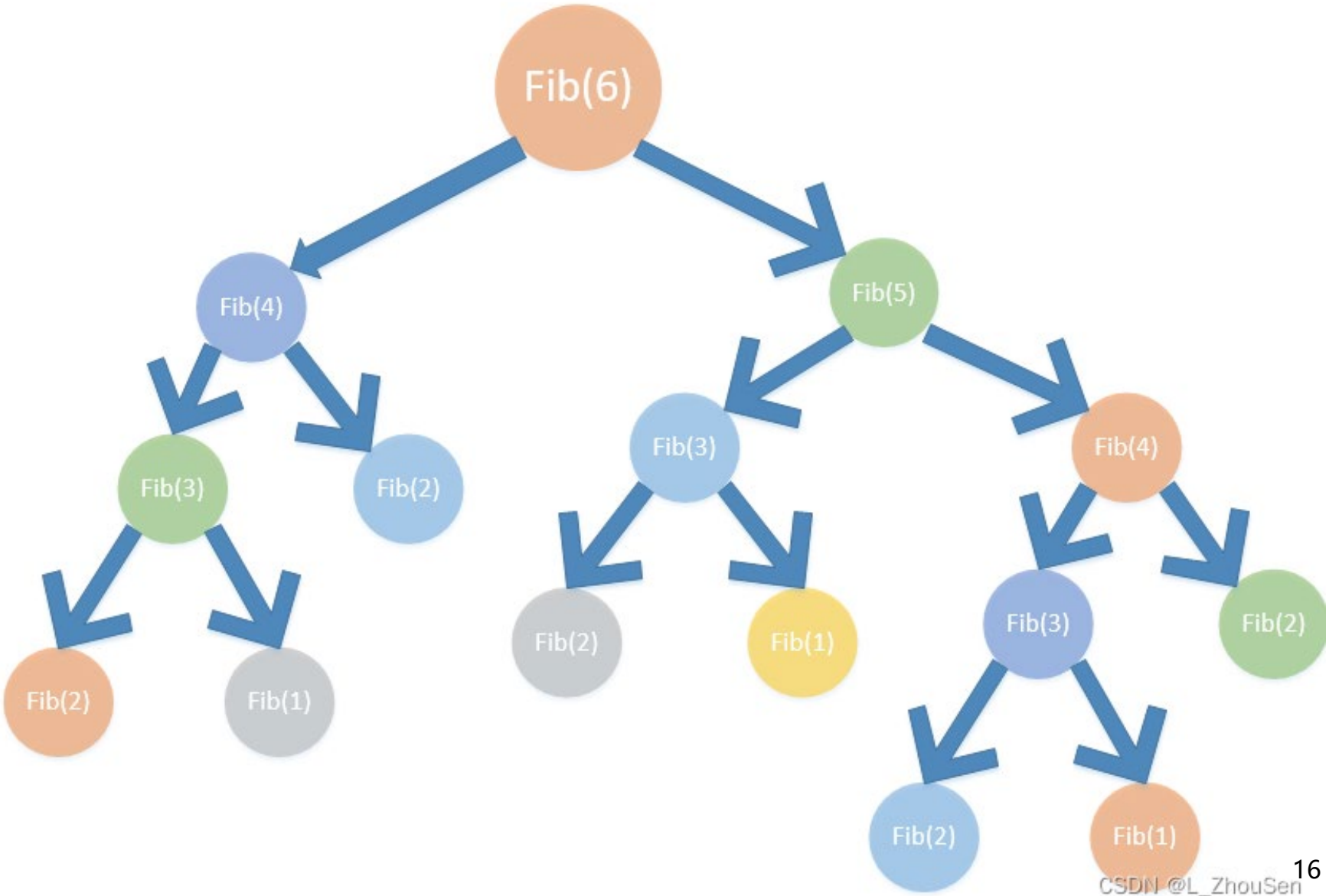
- 黄金分割数列：0,1,1,2,3,5,8,13,21,34,55,89,144,…
  - $F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)$

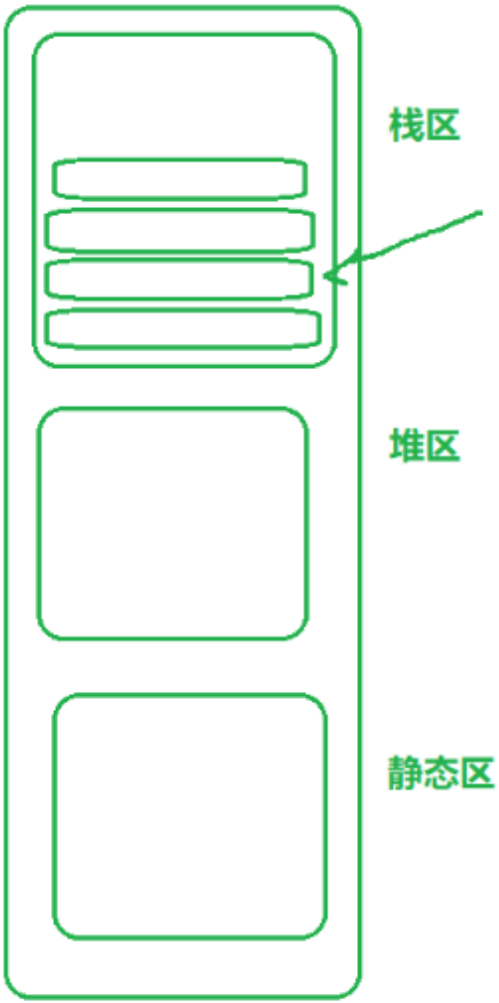


斐布拉契字符串

- f (0) = b, f (1) = a,
- f (2) = f (1) + f (0) = ab,
- f (3) = f (2) + f (1) = aba,
- f (4) = f (3) + f (2) = abaab,
- f(n) = ?

- fib.c
- fib_string.c
- fib_long_iter.c
- fib_long_iter_space.c





每一次函数调用都会在内存的栈区申请一块空间,直到栈区没有空间进行函数调用导致栈溢出

Greatest Common Divisor

- gcd.c

$$a > b \implies \gcd(a, b) = \gcd(a - b, b)$$
$$a < b \implies \gcd(a, b) = \gcd(a, b - a)$$

$$\gcd(a, b) = \gcd(b, a\%b)$$

数组与递归

- 数组求和
  - sum_array.c

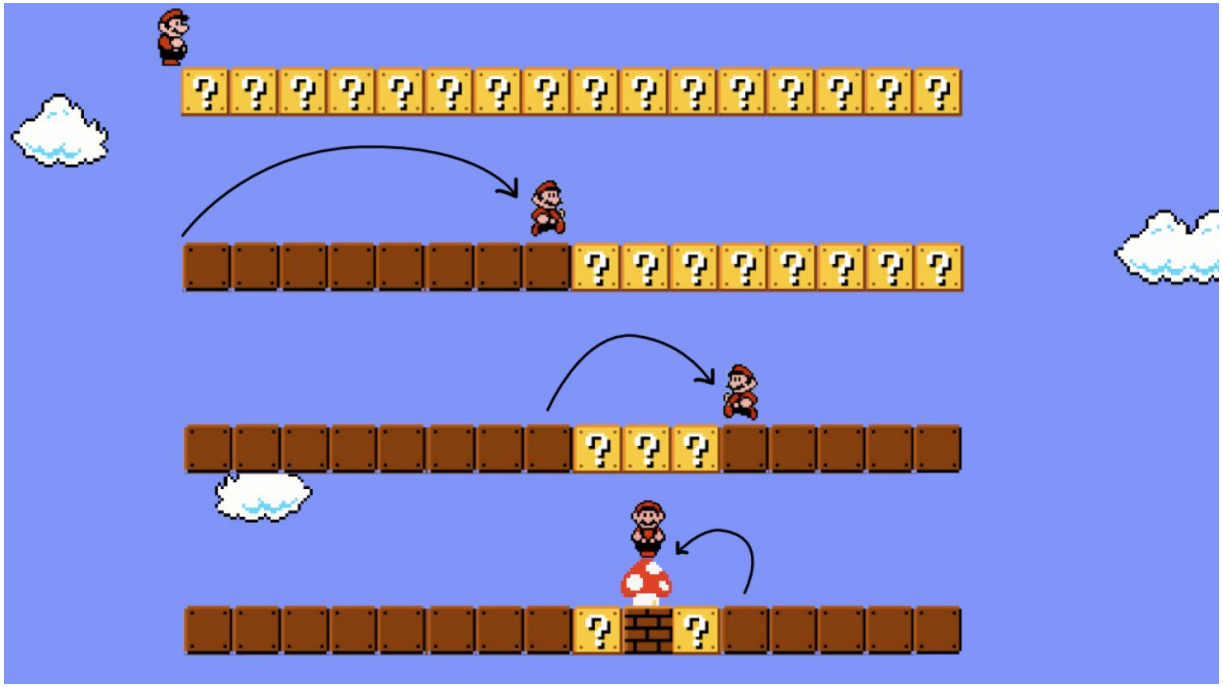
$$
\begin{aligned}
\mathrm{Sum}(1,3,5,7) &= 7 + \mathrm{Sum}(1,3,5) \\
&= 7 + (5 + \mathrm{Sum}(1,3)) \\
&= 7 + (5 + (3 + \mathrm{Sum}(1))) \\
&= 7 + (5 + (3 + 1)) \\
&= 7 + (5 + 4) \\
&= 7 + 9 \\
&= 16
\end{aligned}
$$

- 求最小值
  - min_re.c

$$
\begin{aligned}
\mathrm{Min}(3,5,2,7) &= \min(7, \mathrm{Min}(3,5,2)) \\
&= \min(7, \min(2, \mathrm{Min}(3,5))) \\
&= \min(7, \min(2, \min(5, \mathrm{Min}(3)))) \\
&= \min(7, \min(2, \min(5,3))) \\
&= \min(7, \min(2,3)) \\
&= \min(7,2) \\
&= 2
\end{aligned}
$$

Binary Search

- 典型二分查找算法
  - 斐波那契数列：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ……
  - binarysearch.c
  - binarysearch_re.c



Divide and Conquer 分治法

- Divide the problem into a number of subproblems that are smaller instances of the same problem.

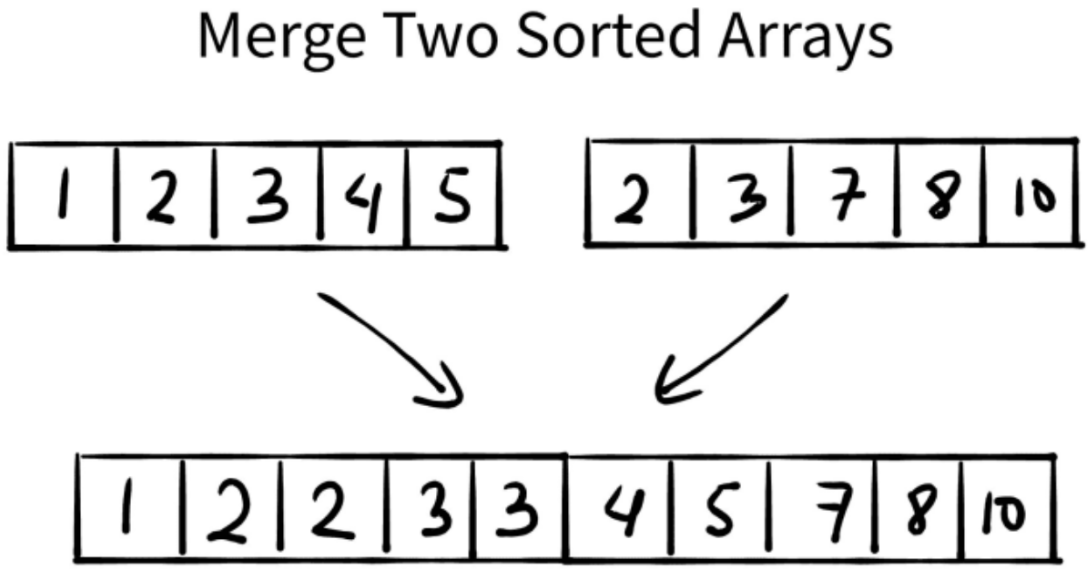
- Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

- Combine the solutions to the subproblems into the solution for the original problem.

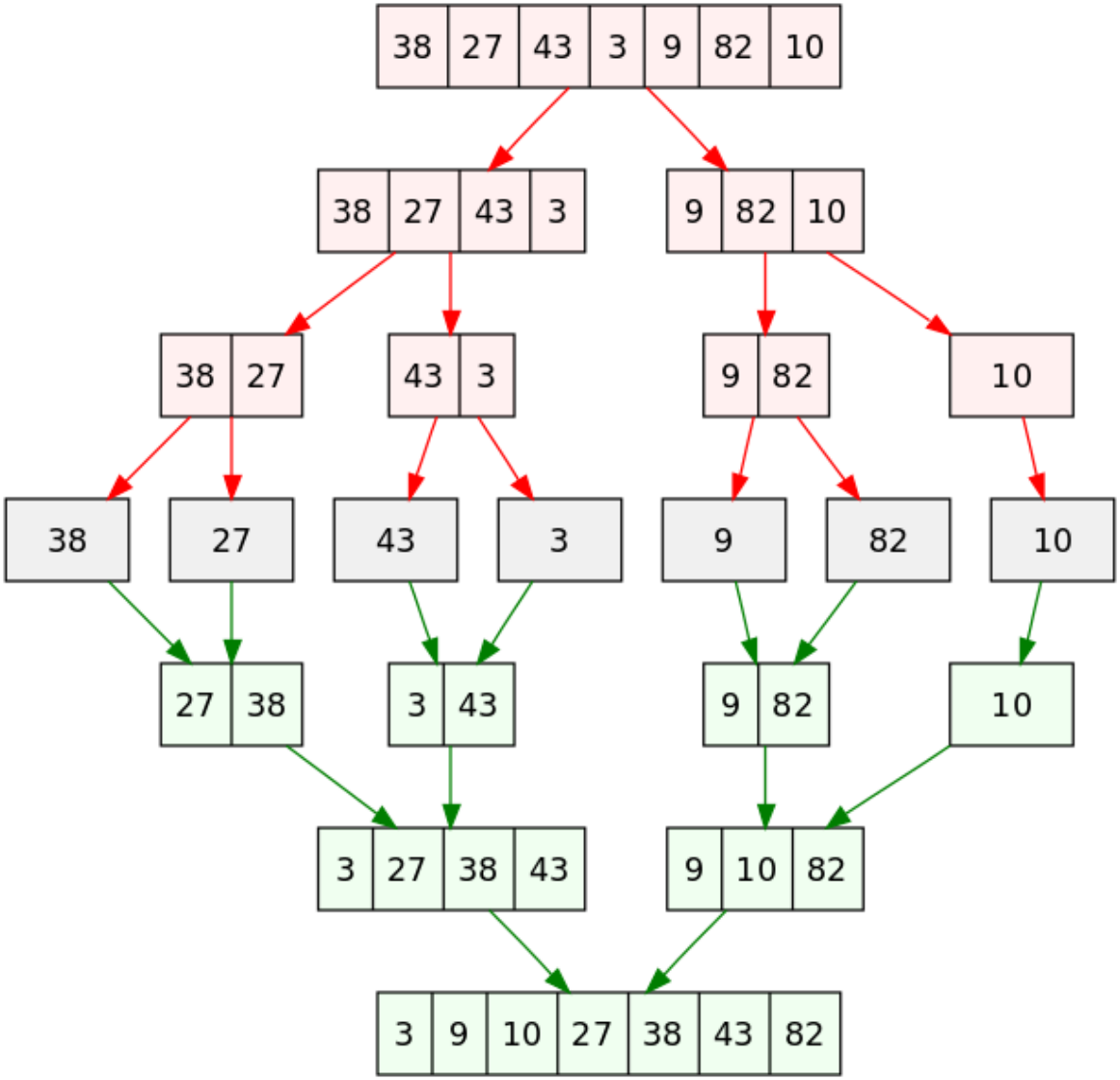
- 重点是：问题分解和基线问题寻找

Merge Sort

- merge.c
- mergesort.c



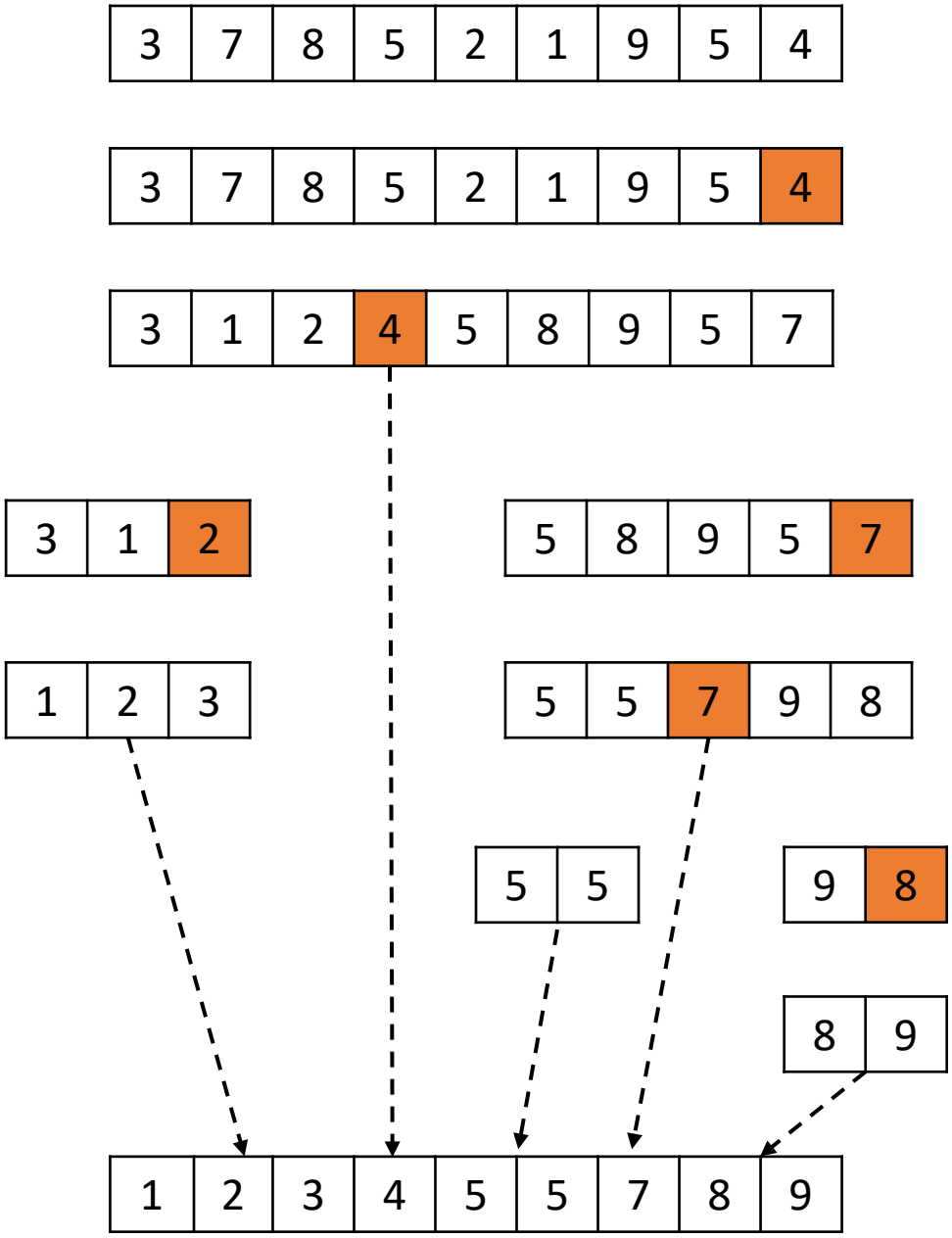
Merge Sort (dance!)

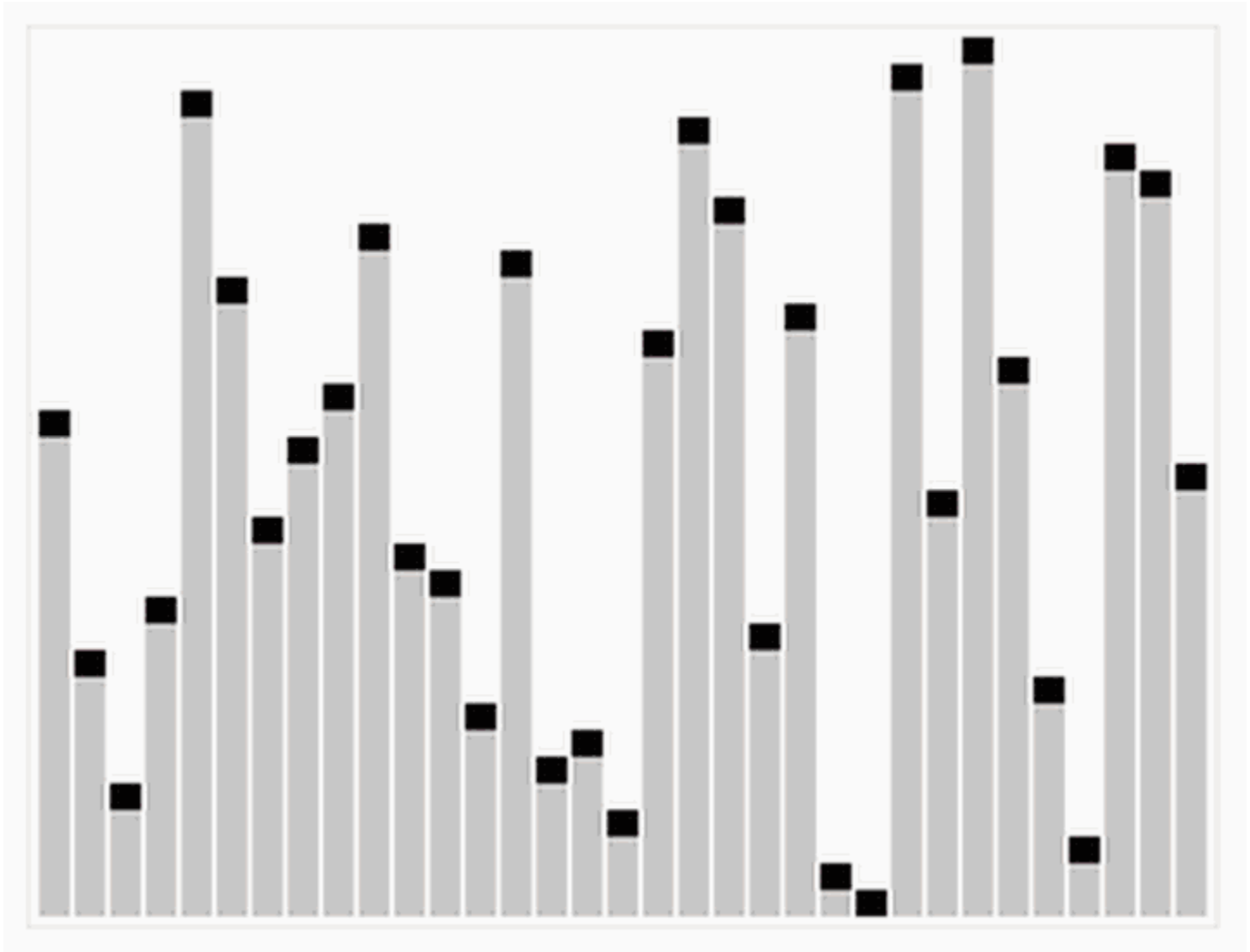


快速排序

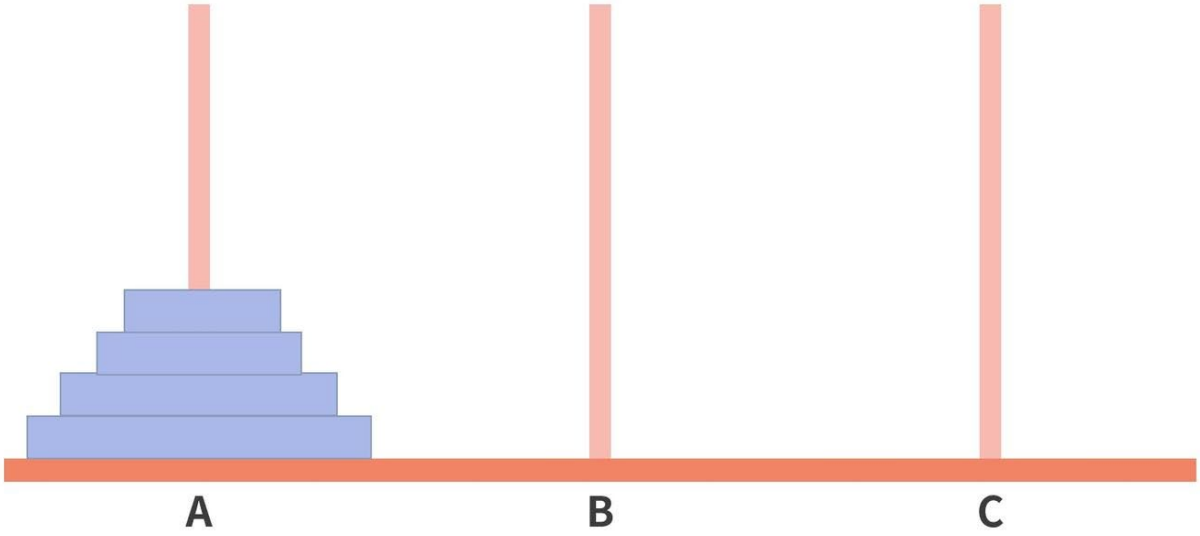
- 基本思想是
  - 通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序
  - 整个排序过程可以递归进行，以此达到整个数据变成有序序列。

- quicksort.c


3 7 8 5 2 1 9 5 4
3 7 8 5 2 1 9 5 4
3 1 2 4 5 8 9 5 7
3 1 2
5 8 9 5 7
1 2 3
5 5 7 9 8
5 5
9 8
8 9
1 2 3 4 5 5 7 8 9

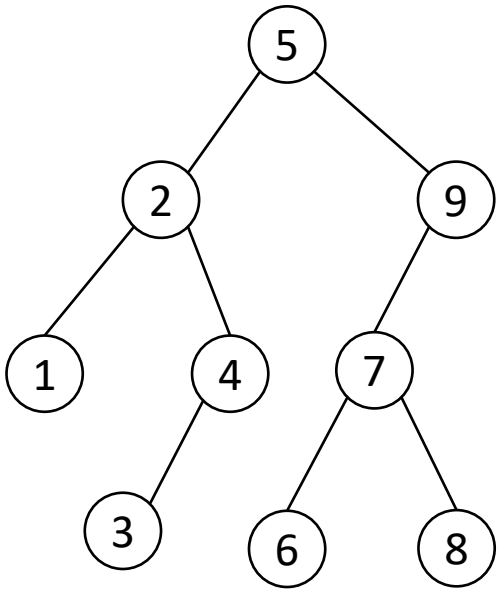


汉诺塔



更复杂点场景：链式结构实现二叉树

- 二叉树的遍历
  - 前序pre-order
    - 521439768
  - 中序in-order
    - 123456789
  - 后序post-order
    - 134268795



- BST.c

End

- 为努力（煎熬）的自己鼓掌！

- 继续加油！