

排序算法

胡船长

初航我带你，远航靠自己

本章题目

1-应试. Leetcode-01: 两数之和

2-应试. Leetcode-148: 排序链表

3-应试. Leetcode-88: 合并两个有序数组

4-应试. Leetcode-21: 合并两个有序链表

5-校招. Leetcode-04: 两个正序数组的中位数

6-校招. Leetcode-219: 存在重复元素 II

7-校招. HZOJ-248: 逆序对个数

8-竞赛. HZOJ-251: 士兵

9-竞赛. HZOJ-256: 国王游戏

本期内容

一. 选择排序

二. 插入排序

三. 希尔排序

四. 冒泡排序

五. 快速排序

六. 归并排序

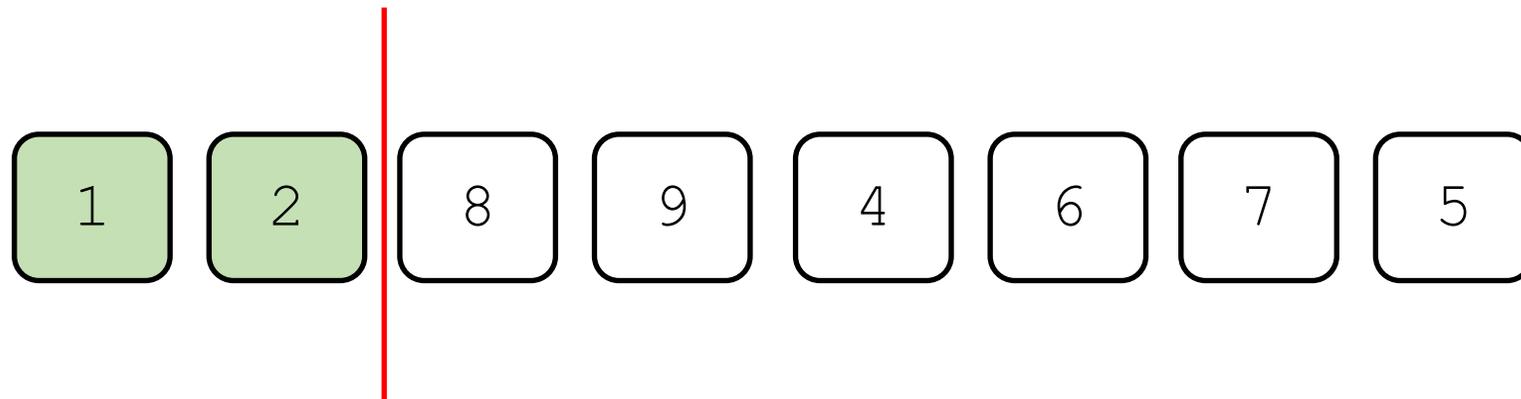
七. 基数排序

八. 排序算法总结

九. C++ sort 使用方法与技巧

一. 选择排序

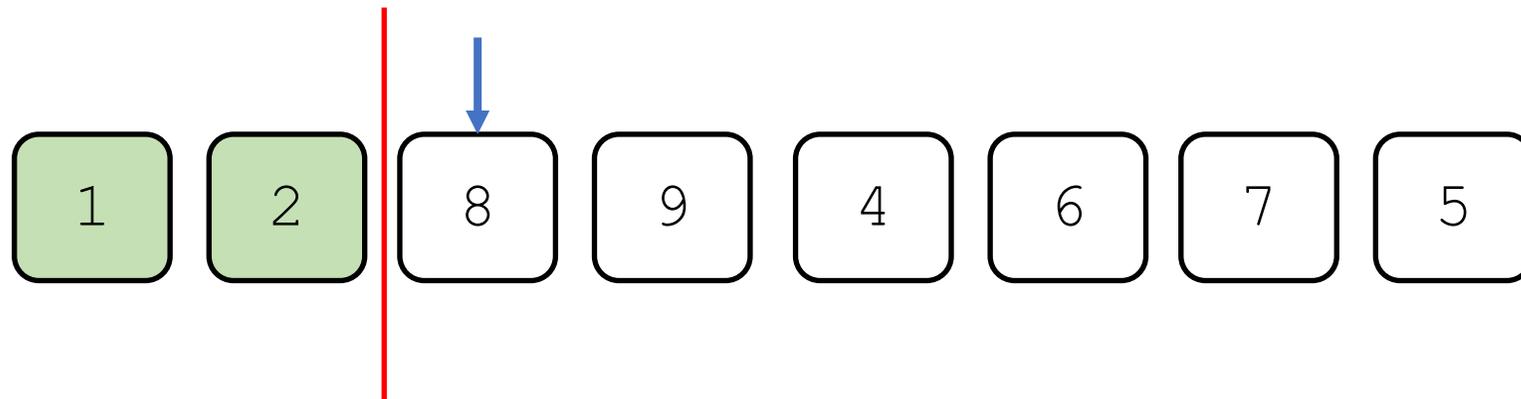
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

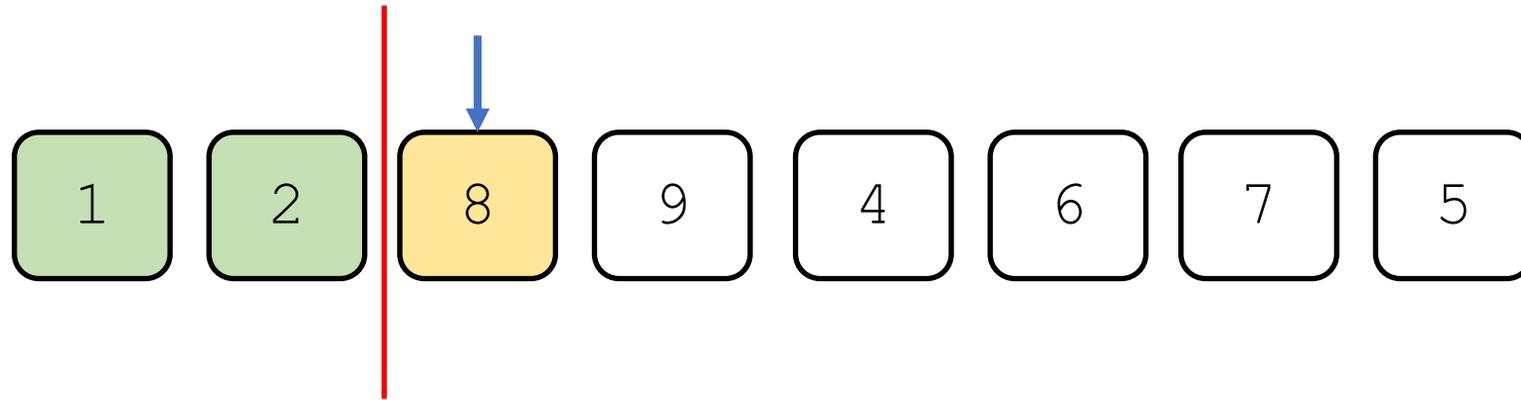
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

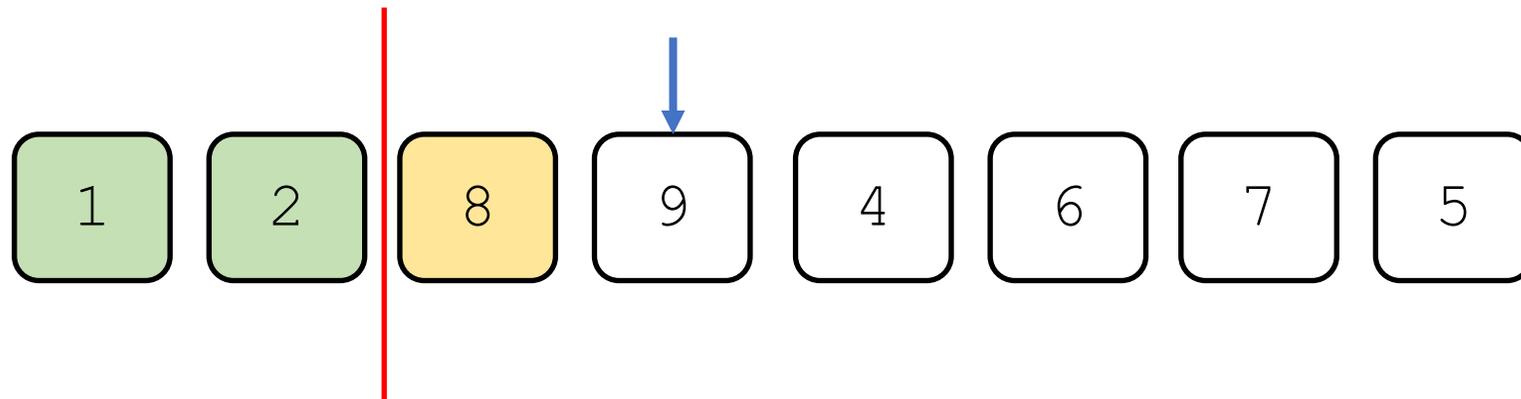
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

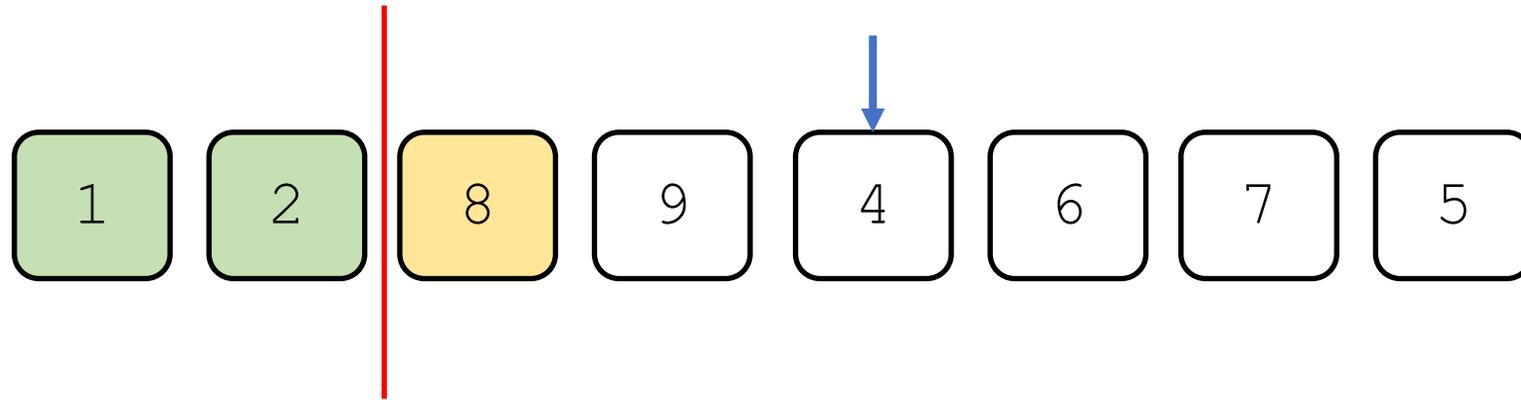
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

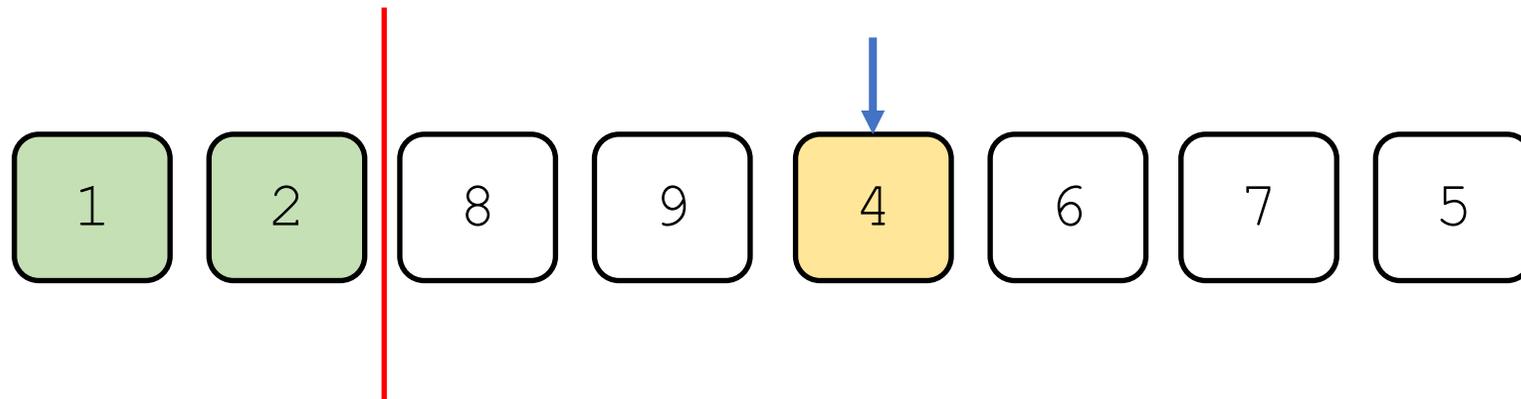
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

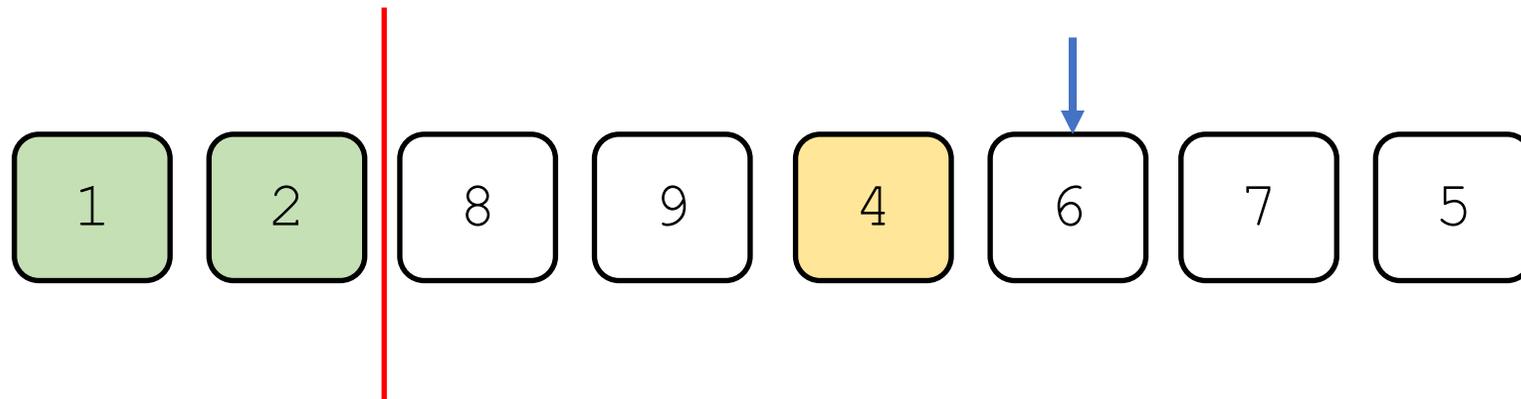
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

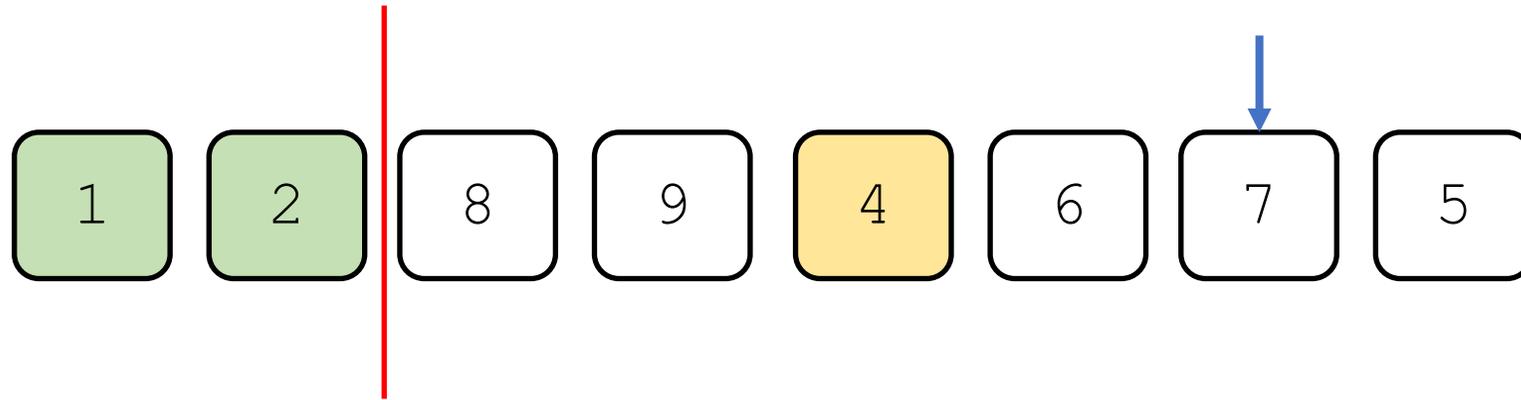
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

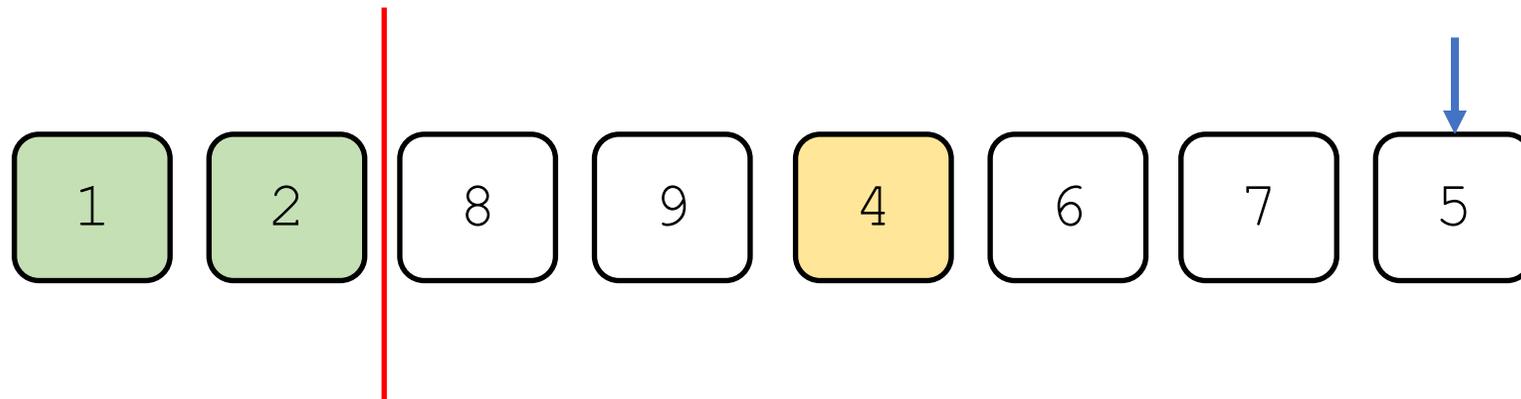
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

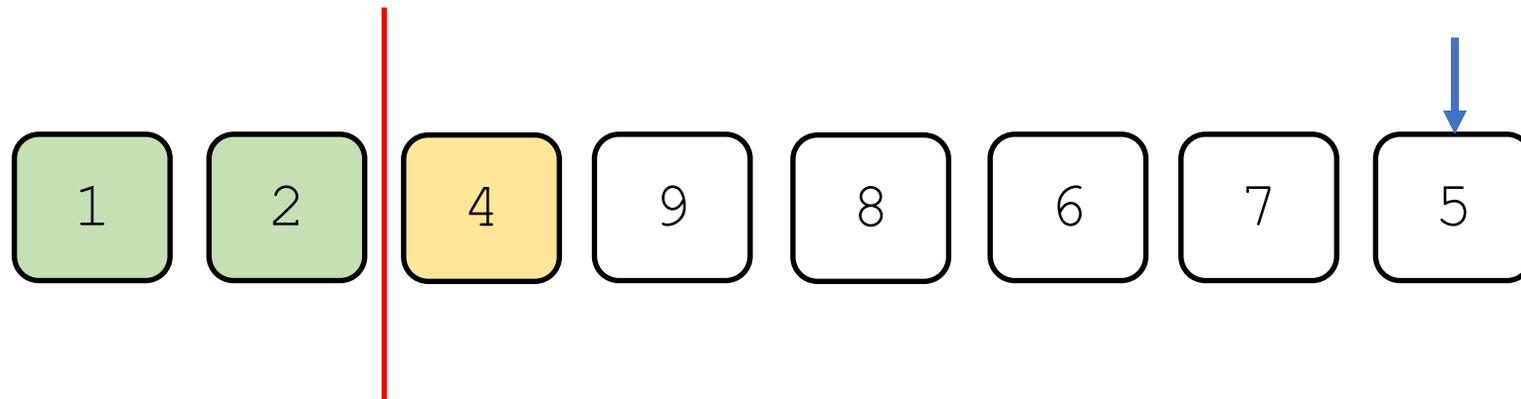
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

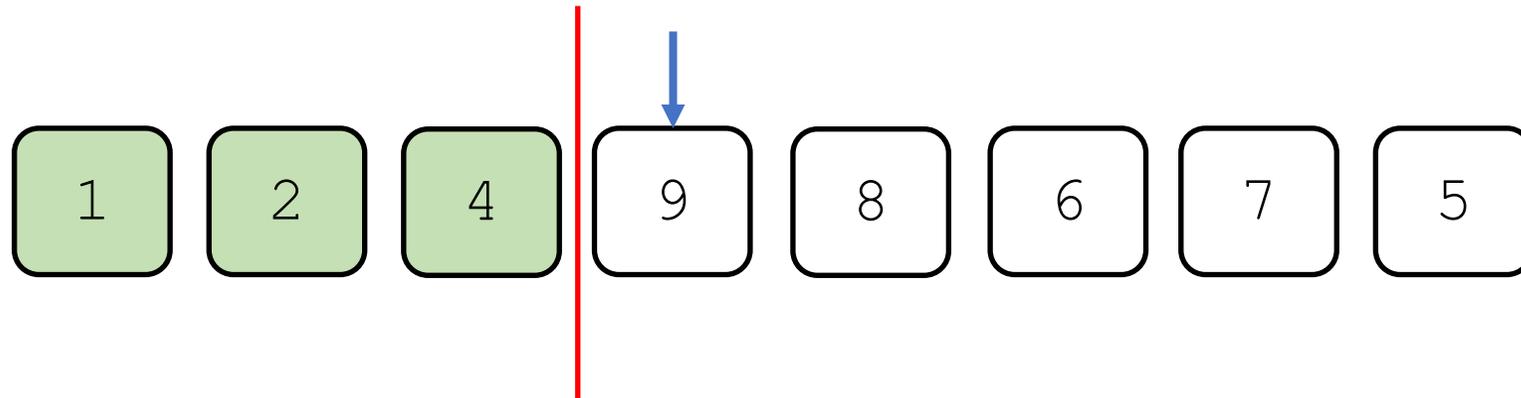
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

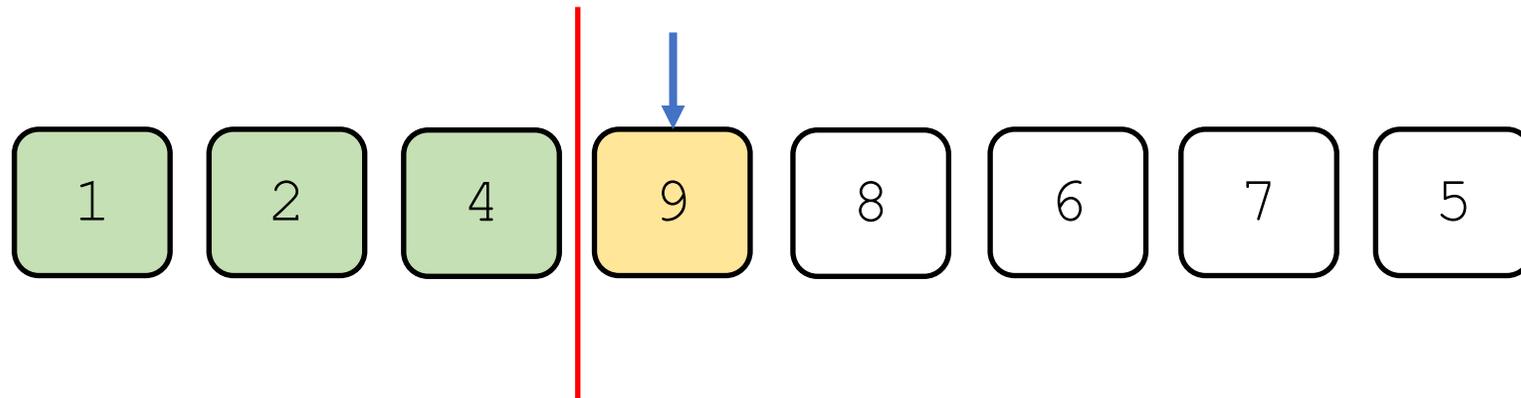
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

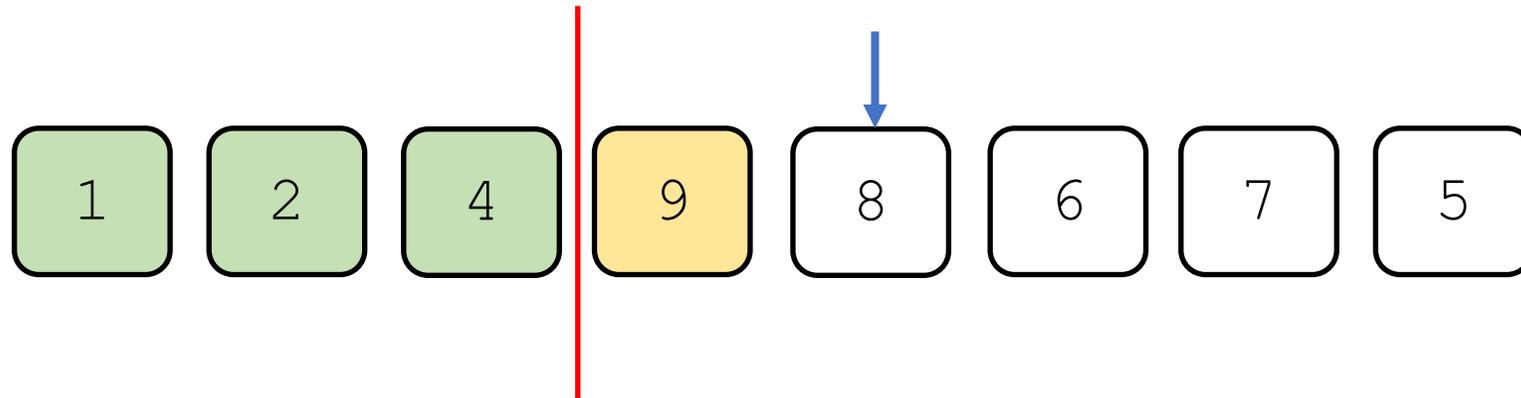
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

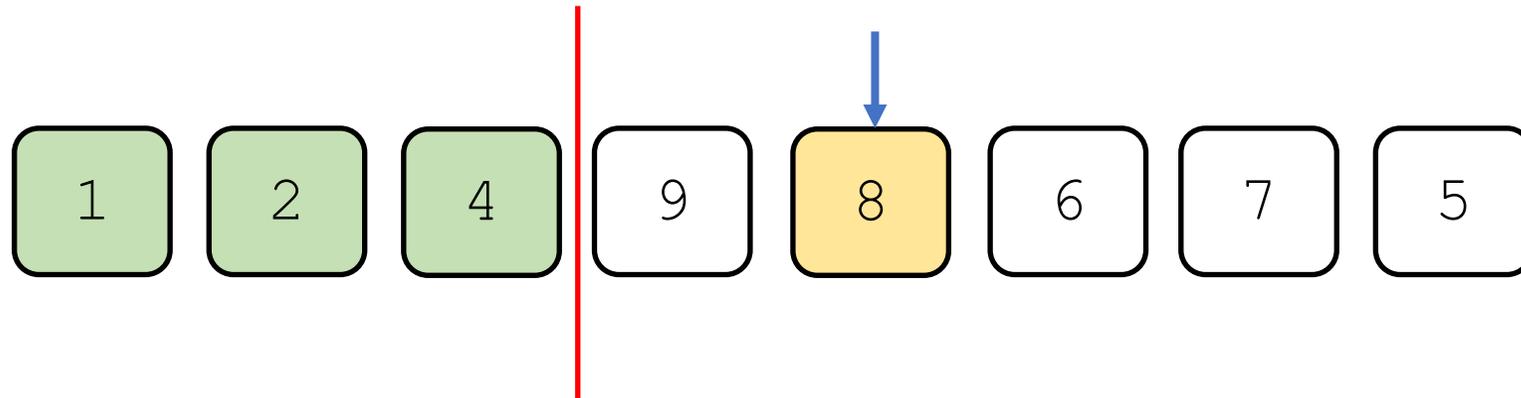
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

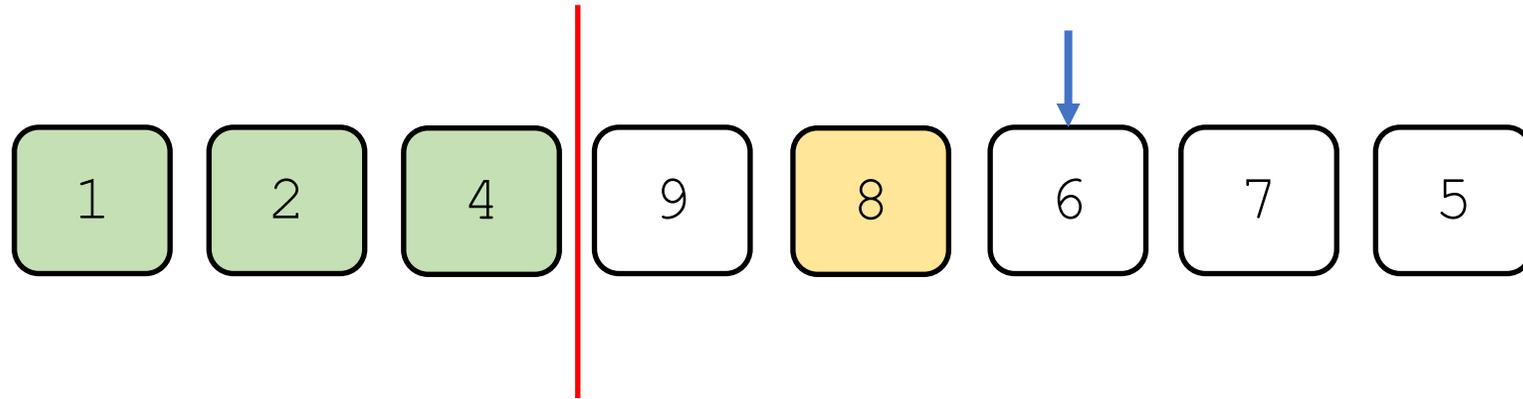
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

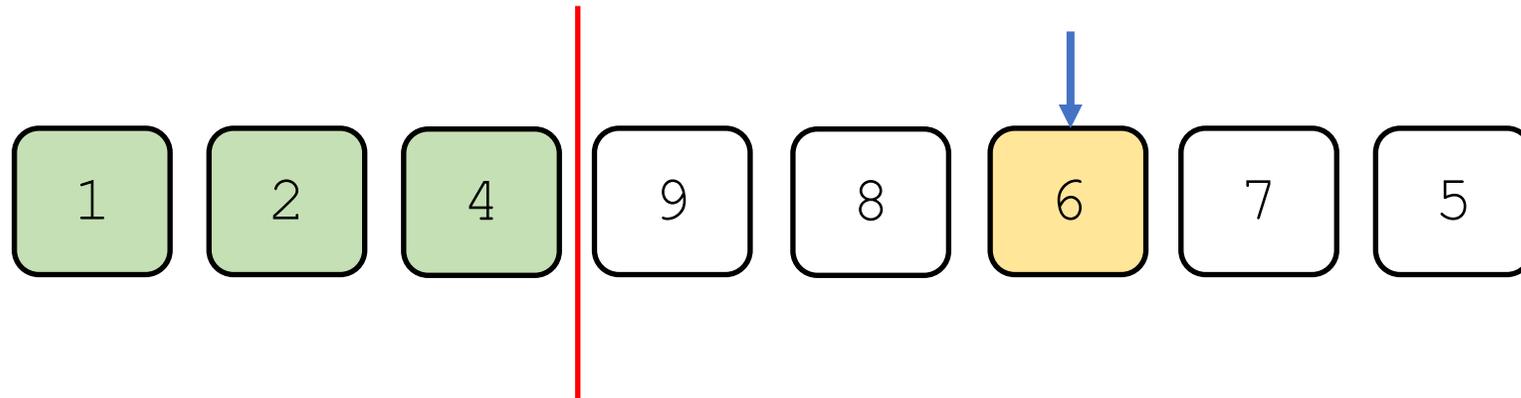
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

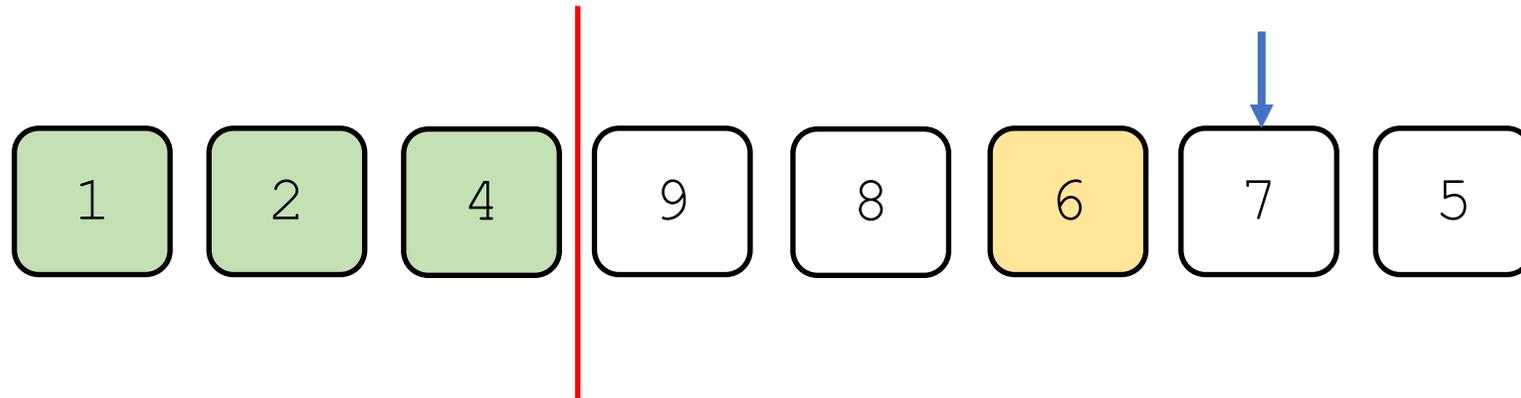
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

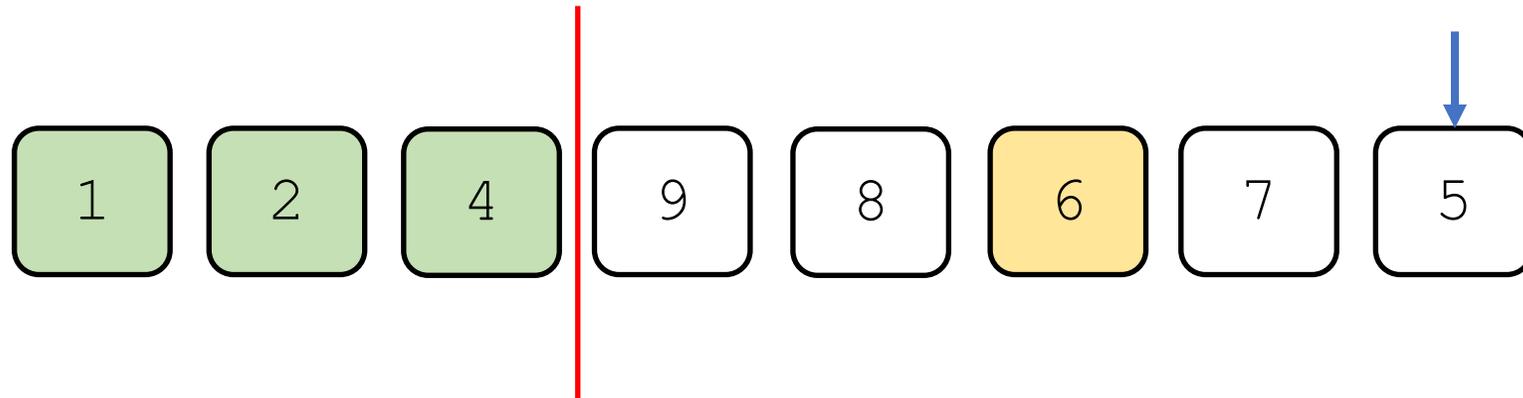
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

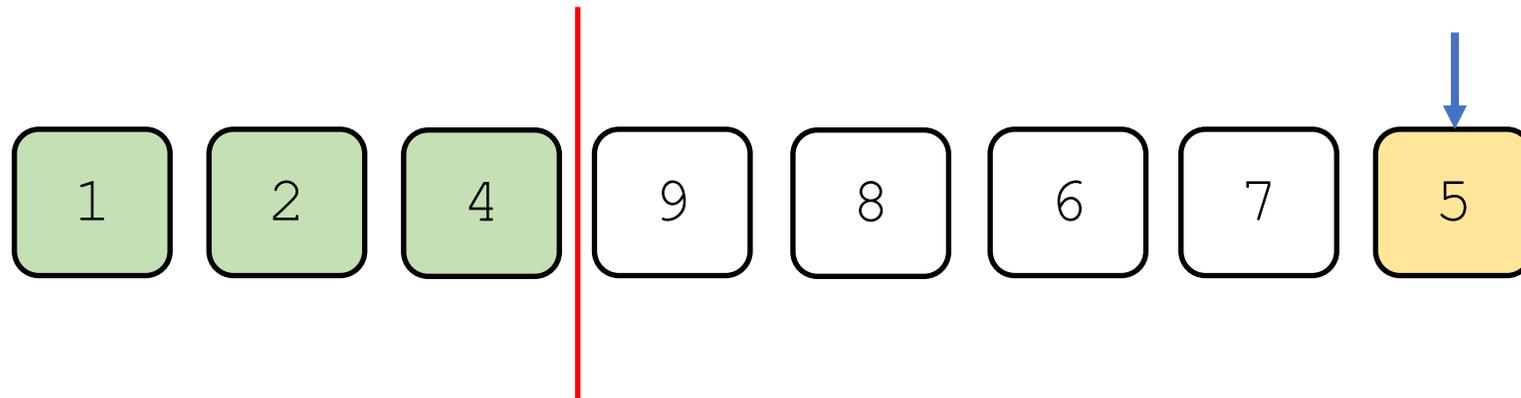
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

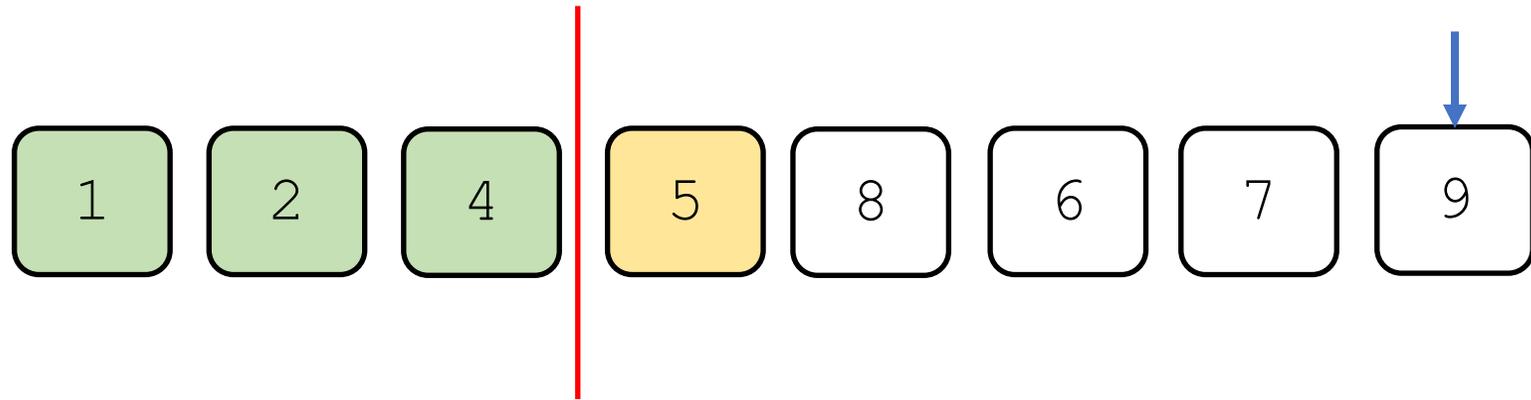
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

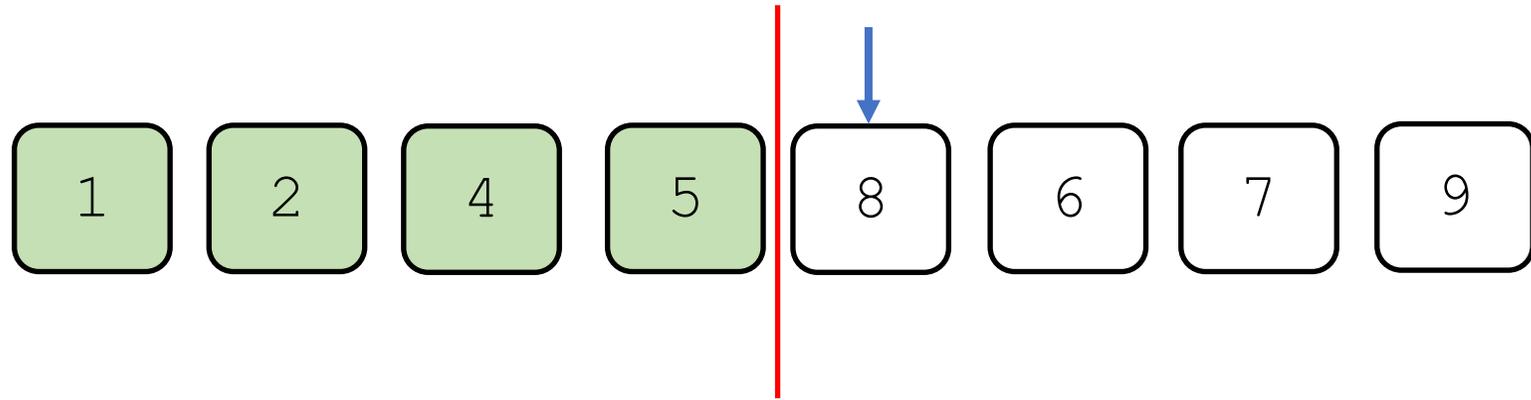
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

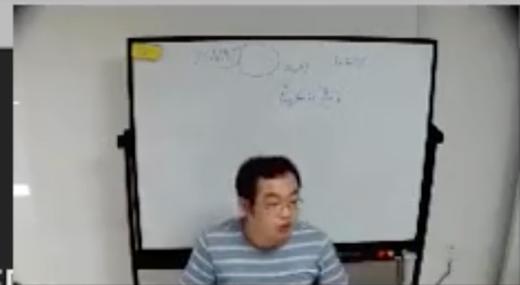
选择排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、每一轮从『待排序区』中选择一个最小的元素放到『已排序区』的尾部
- 3、直到『待排序区』没有元素为止

```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```

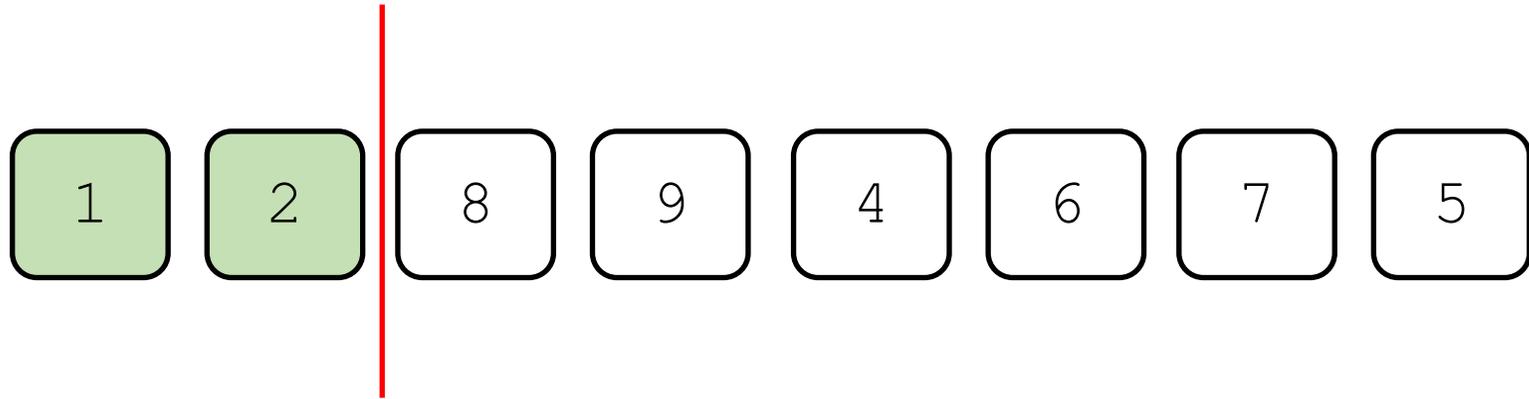


选择排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

二. 插入排序

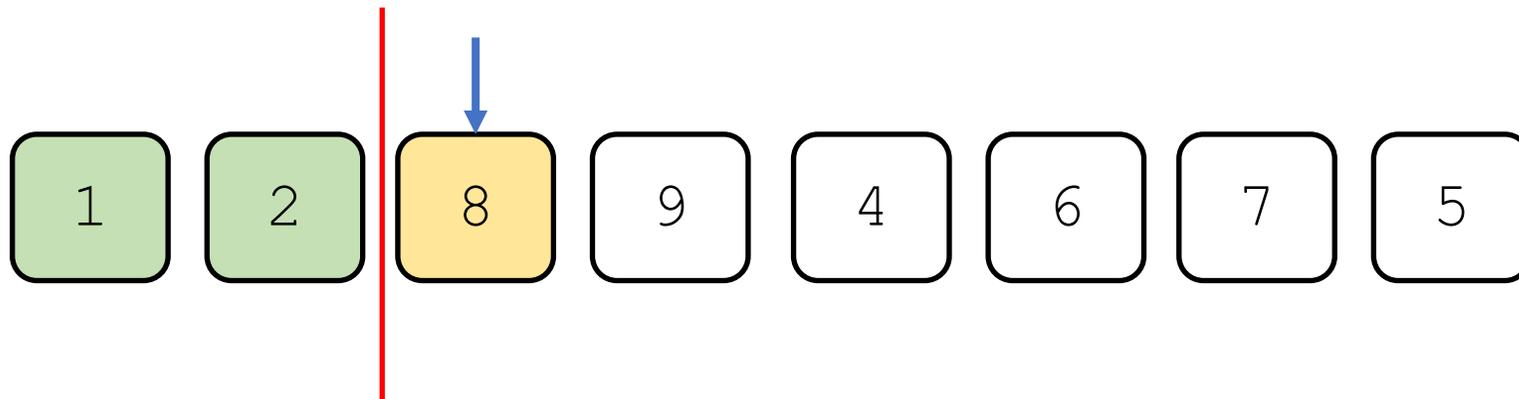
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

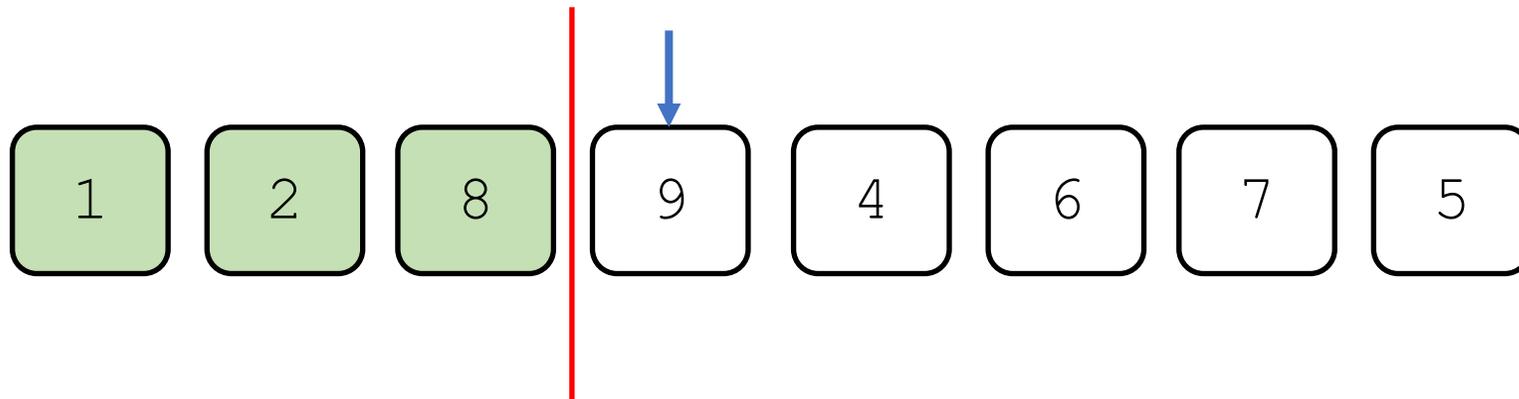
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

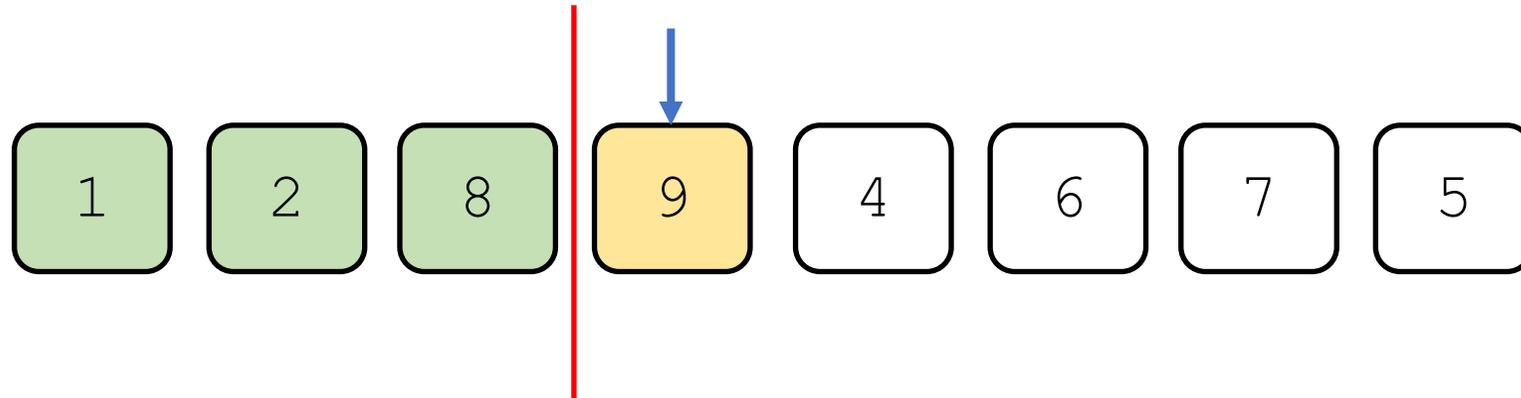
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

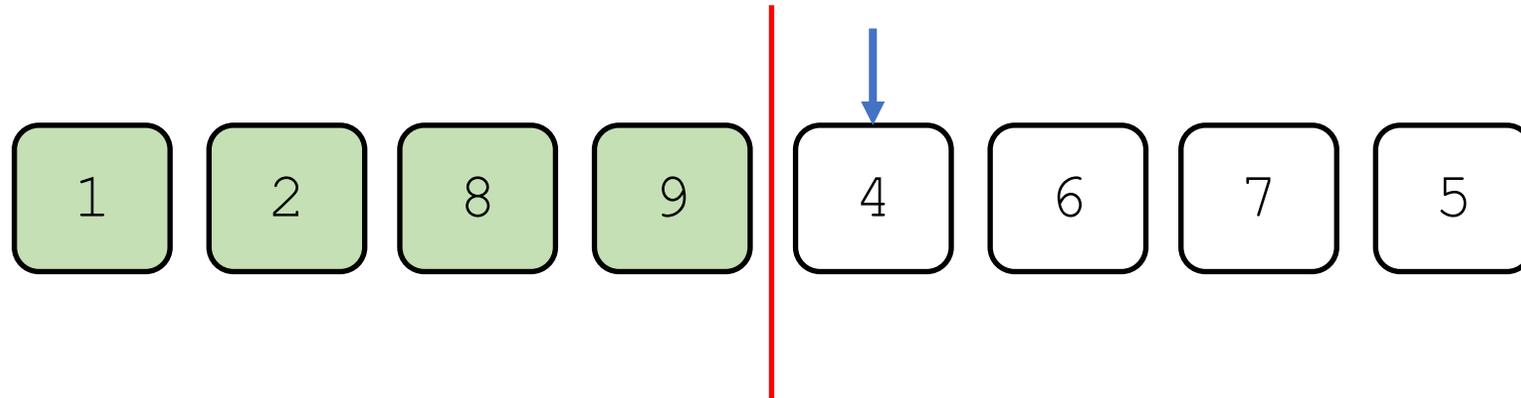
插入排序



口诀：

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

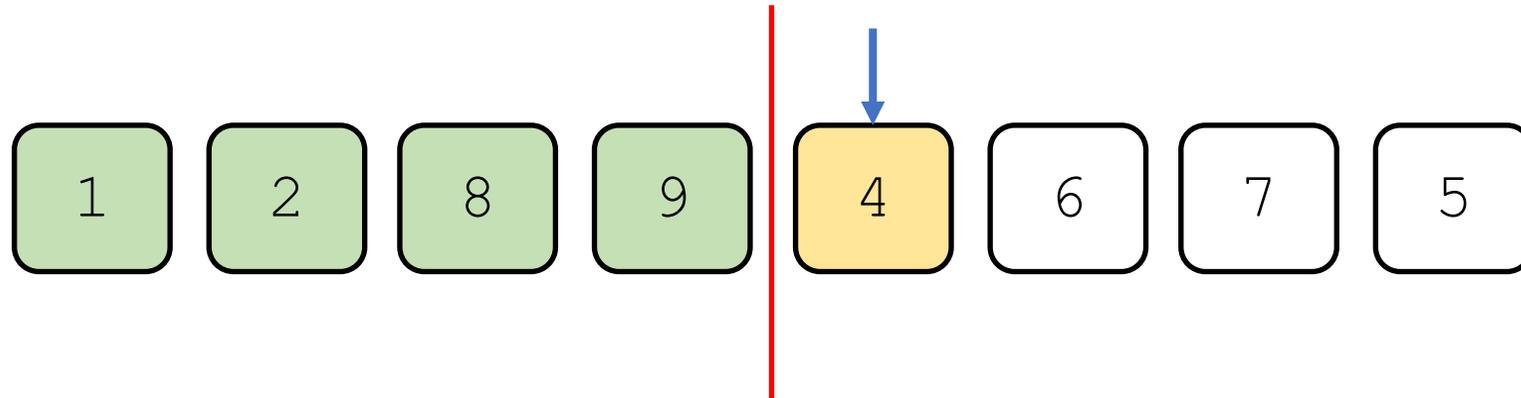
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

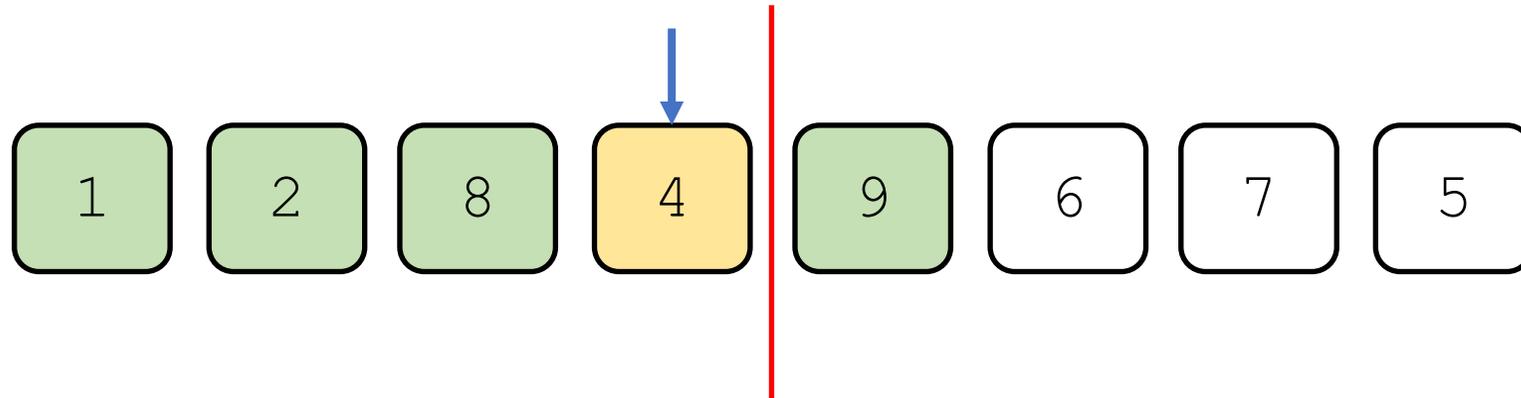
插入排序



口诀：

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

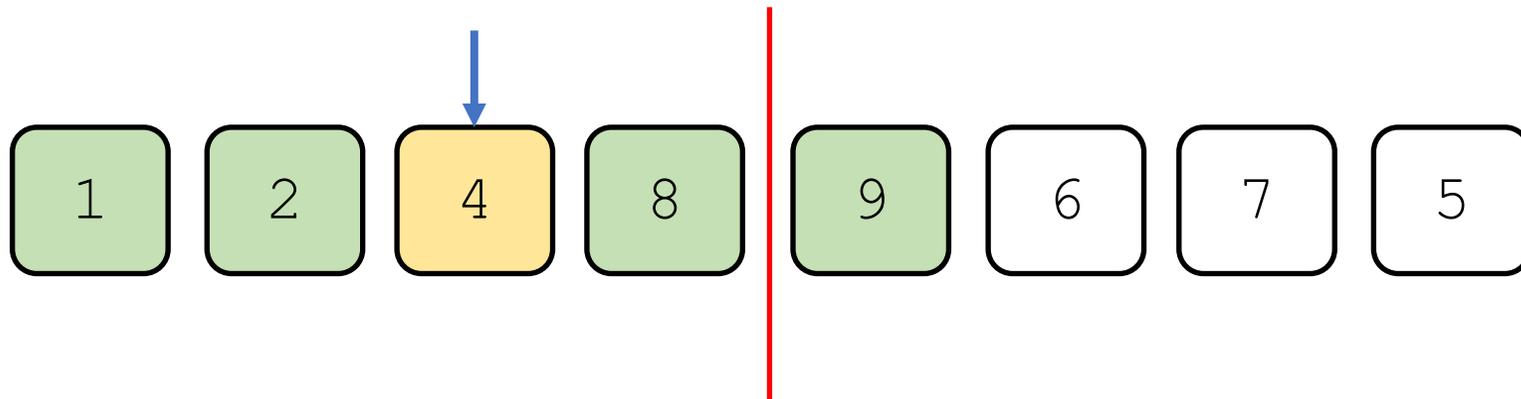
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

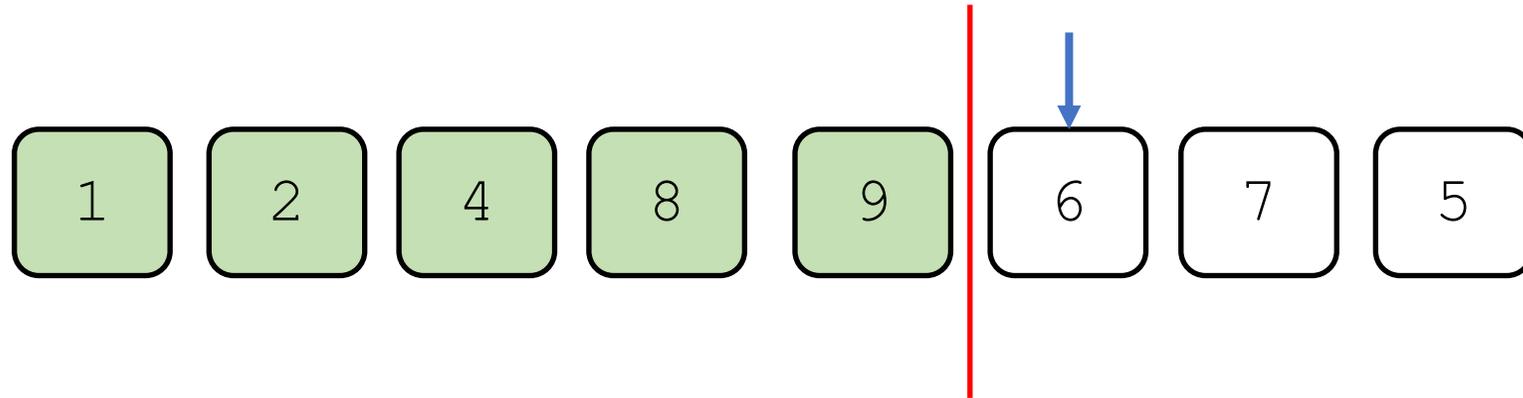
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

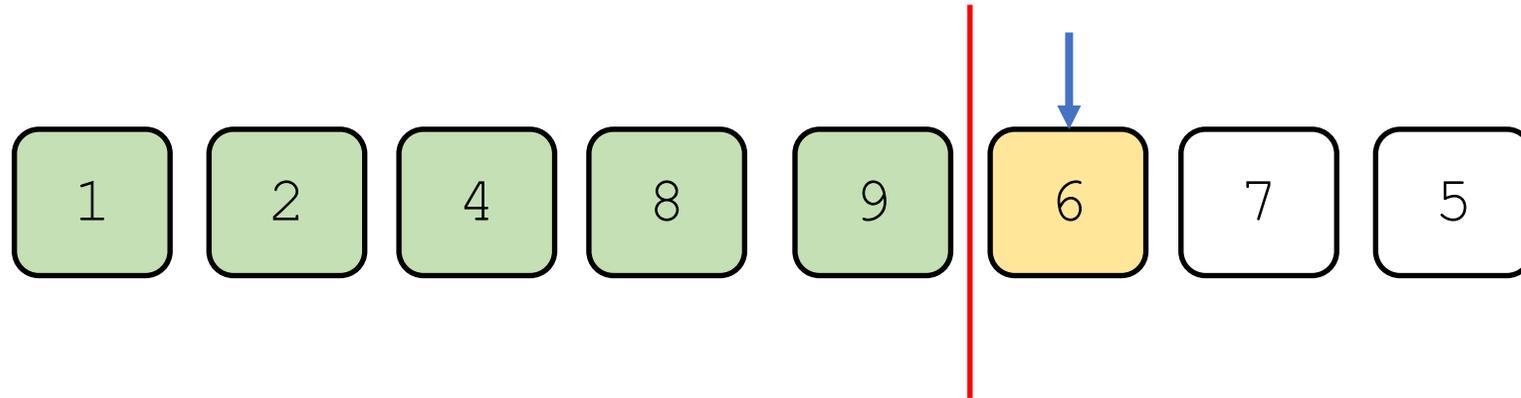
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

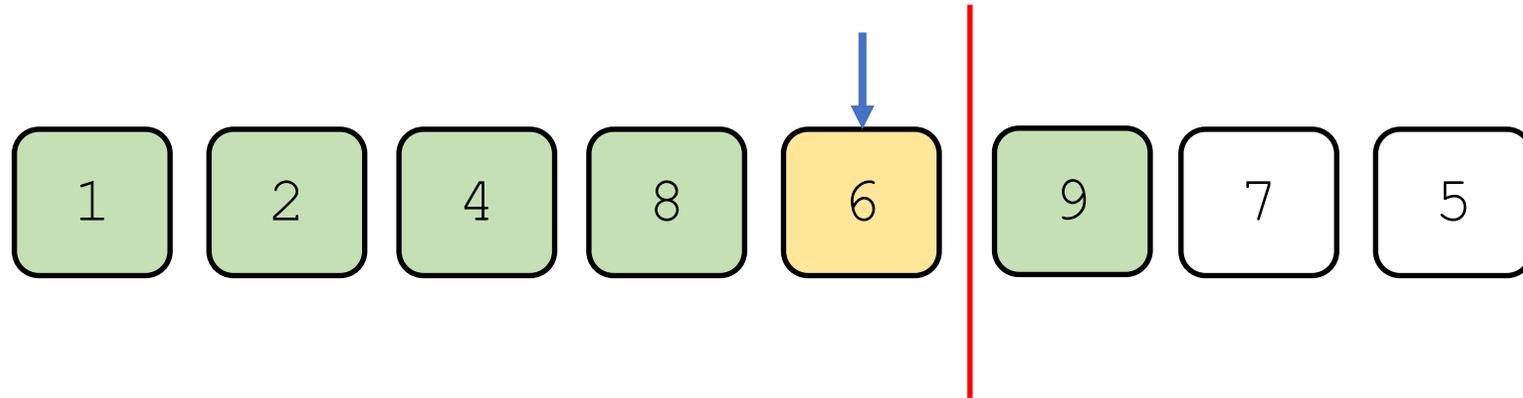
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

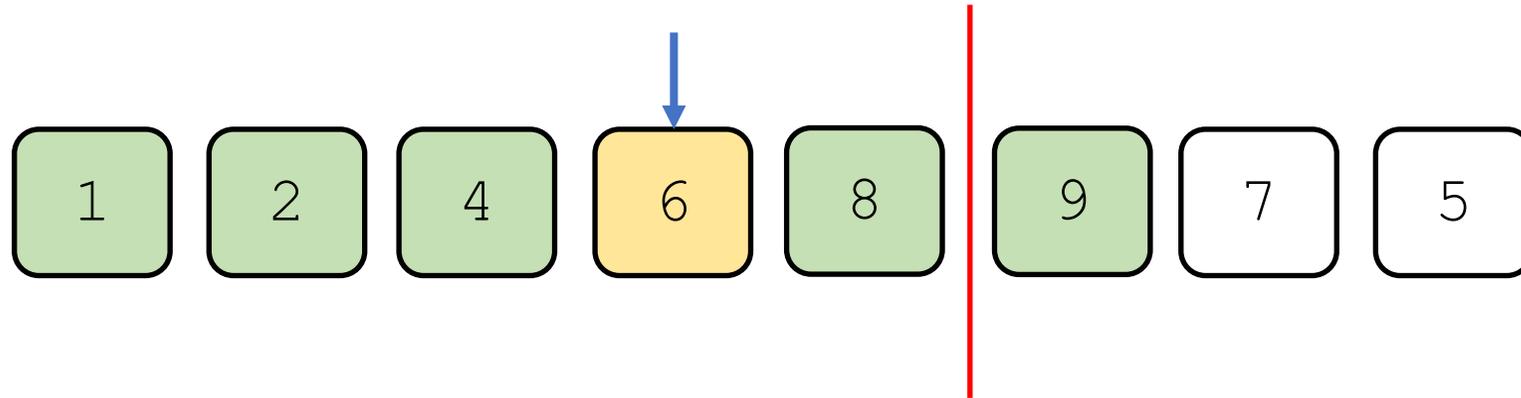
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

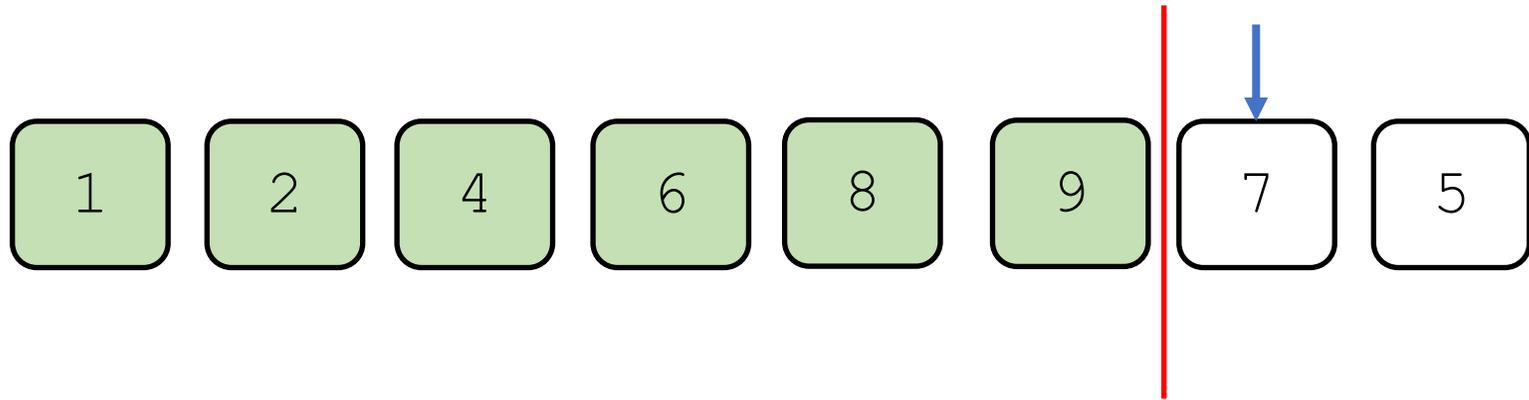
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

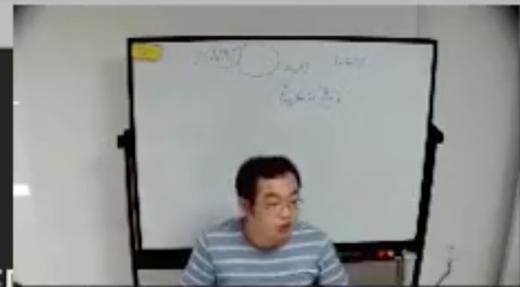
插入排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、将『已排序区』后面一个元素，向前插入到『待排序区』中
- 3、直到『待排序区』没有元素为止

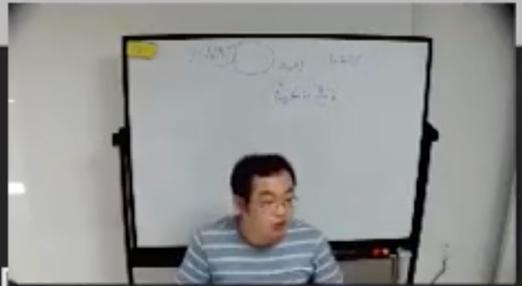
```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```



插入排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

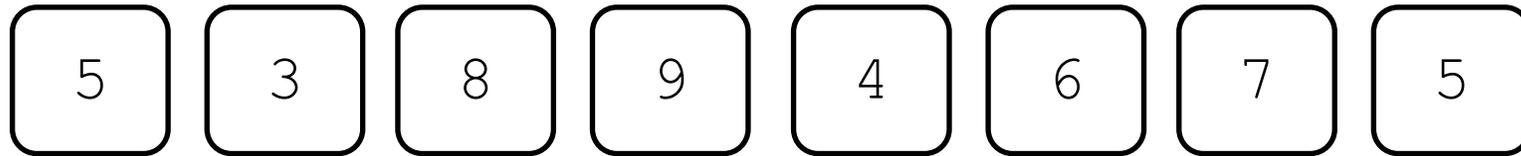
```
1. vim
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```



无监督的插入排序：代码演示

三. 希尔排序

希尔排序（分组插入排序）

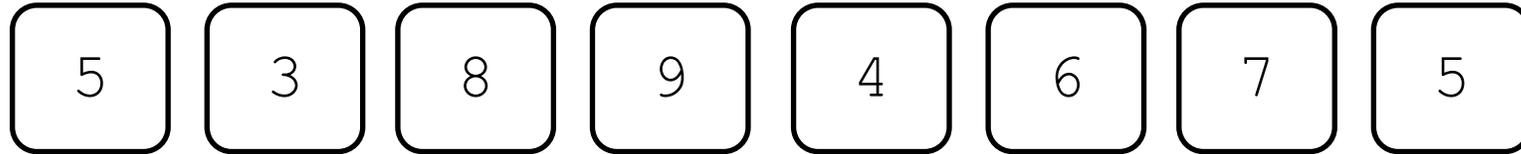


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

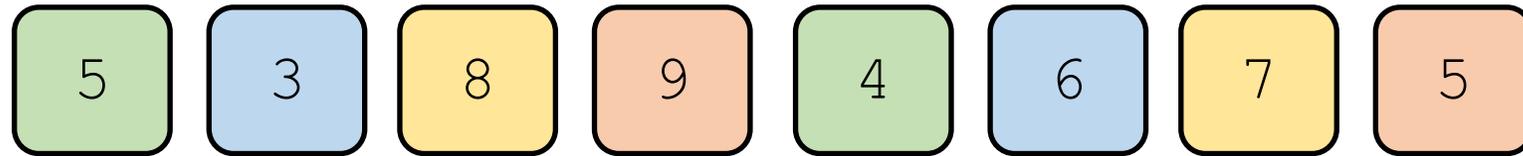


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

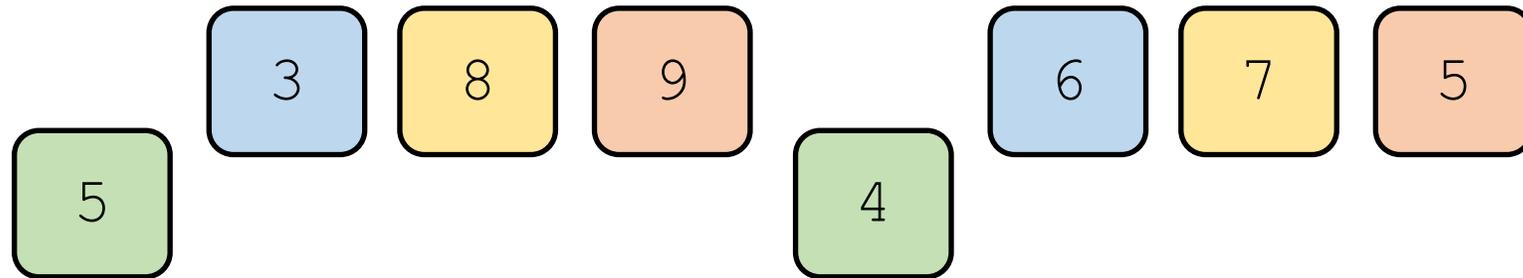


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

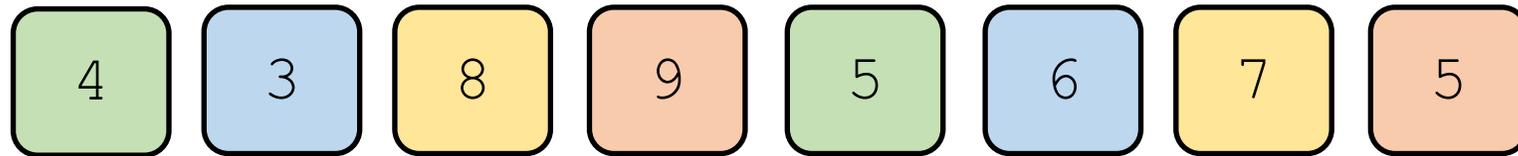


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

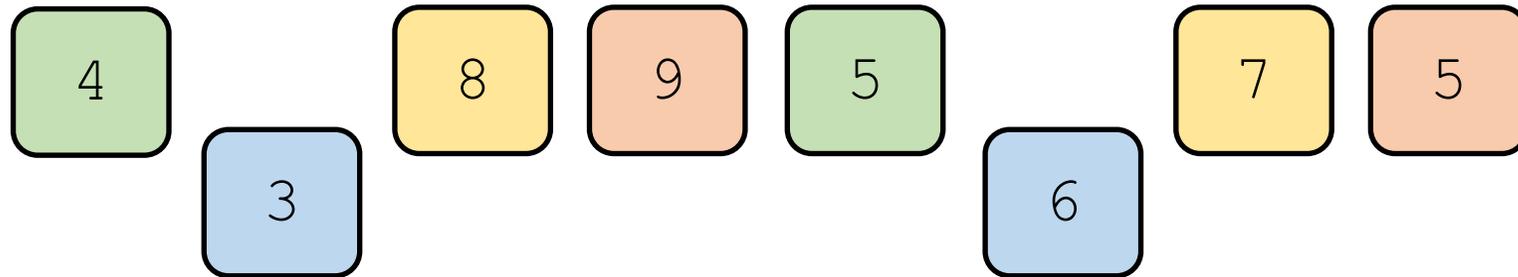


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

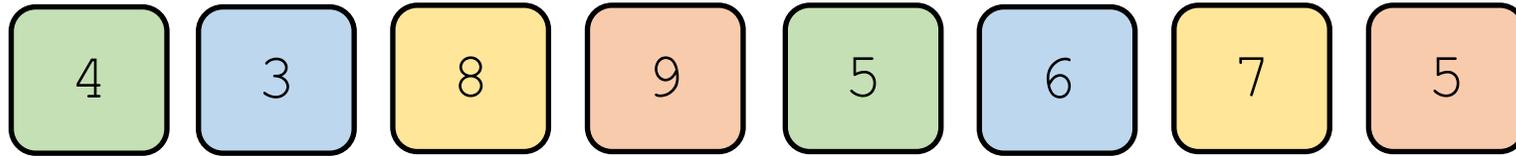


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

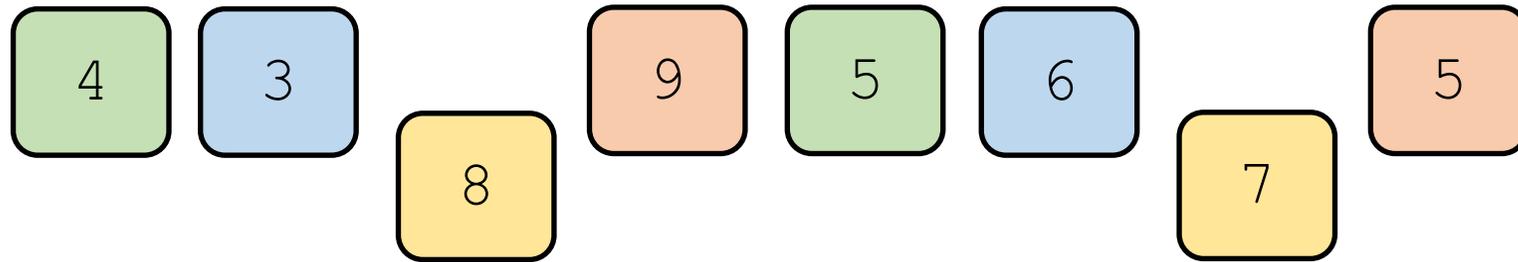


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

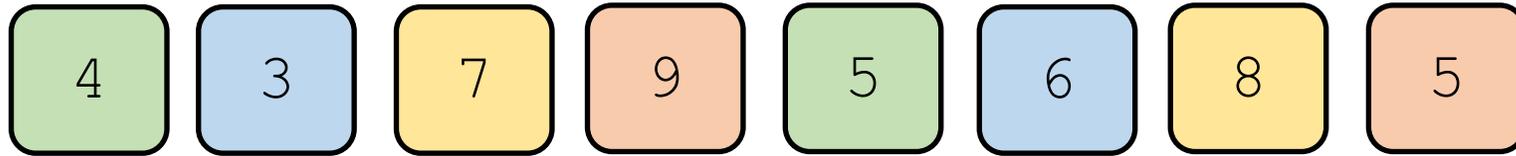


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

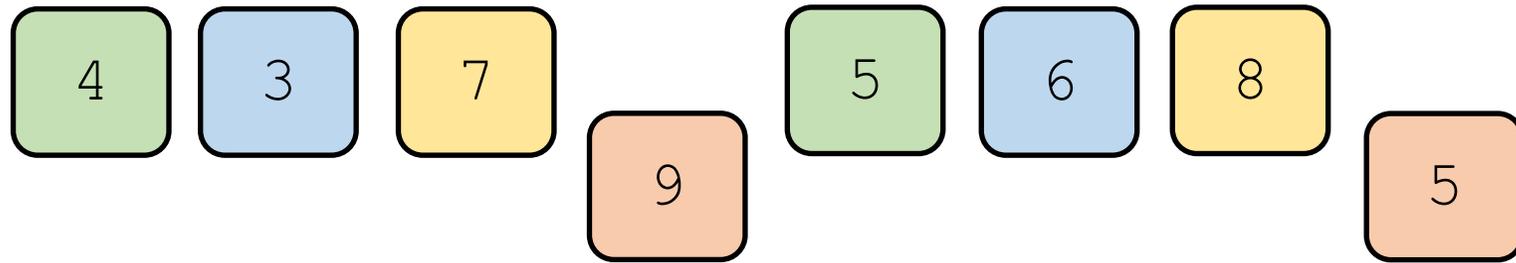


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

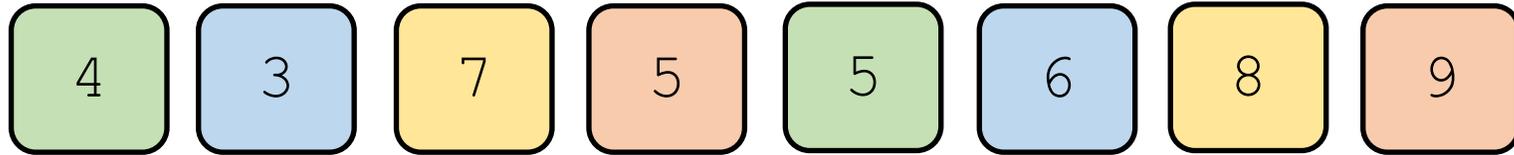


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第一轮步长：4

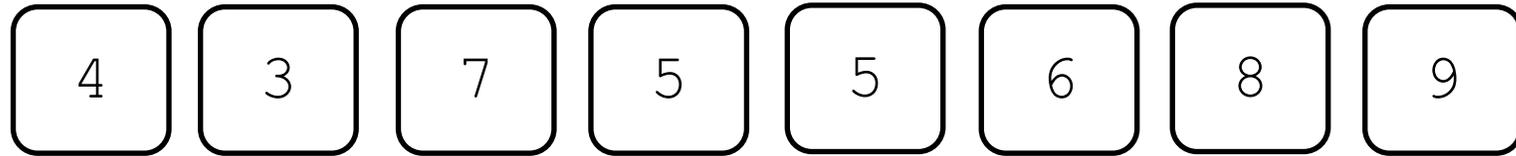


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第二轮步长：2

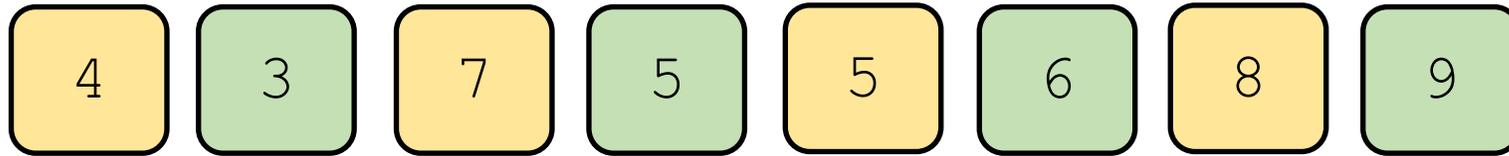


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第二轮步长：2

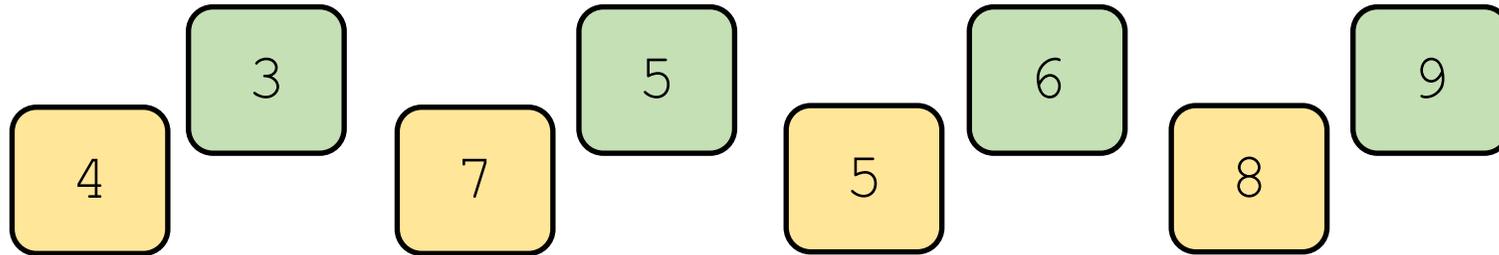


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第二轮步长：2

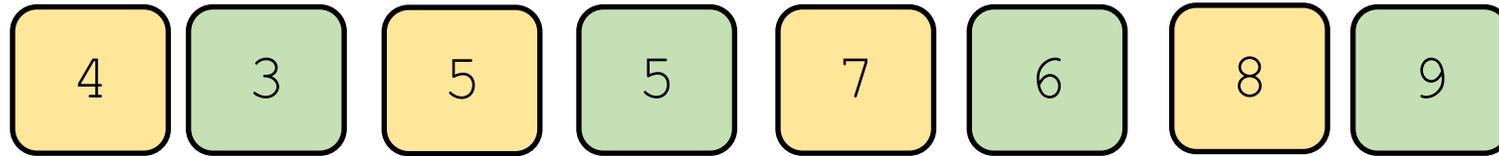


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第二轮步长：2

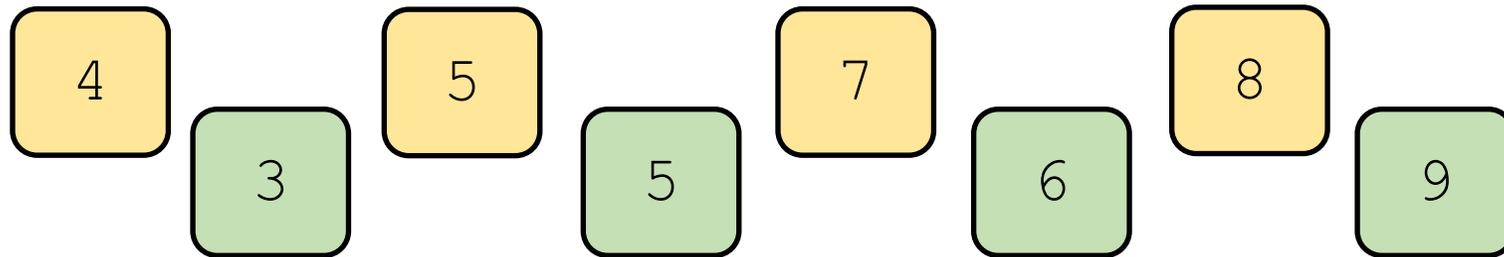


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第二轮步长：2

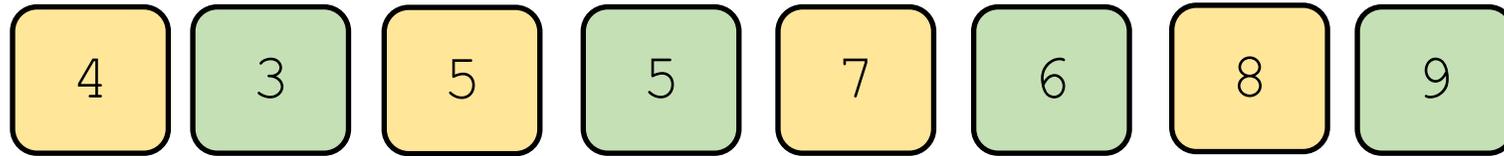


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第二轮步长：2

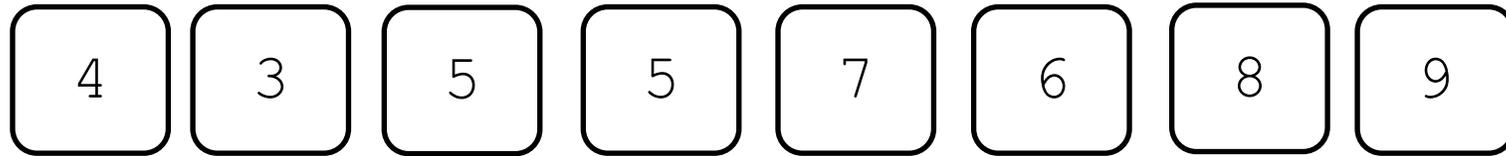


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第三轮步长：1

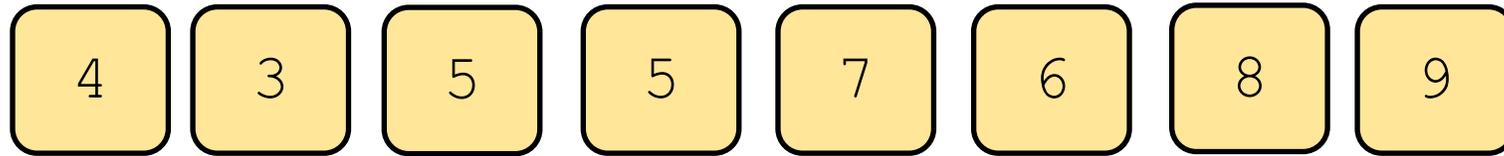


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第三轮步长：1

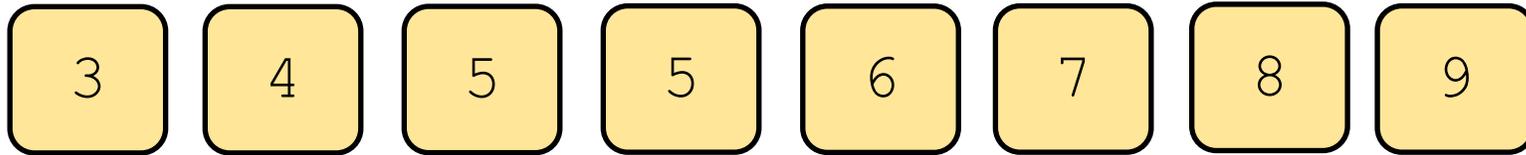


口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

第三轮步长：1



口诀：

- 1、设计一个【步长】序列
- 2、按照步长，对序列进行分组，每组采用插入排序
- 3、直到执行到步长为1为止

希尔排序（分组插入排序）

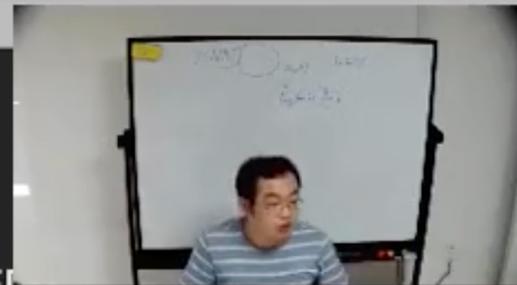
希尔排序的效率和【步长序列】紧密相关

参考时间复杂度： $O(n \log n) \sim O(n^2)$

$O(n^2)$ 希尔增量序列： $n/2$ 、 $n/4$ 、 $n/8$ 、 $n/16$

$O(n^{1.5})$ Hibbard增量序列： 1 、 3 、 7 ... $2^k - 1$

```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```

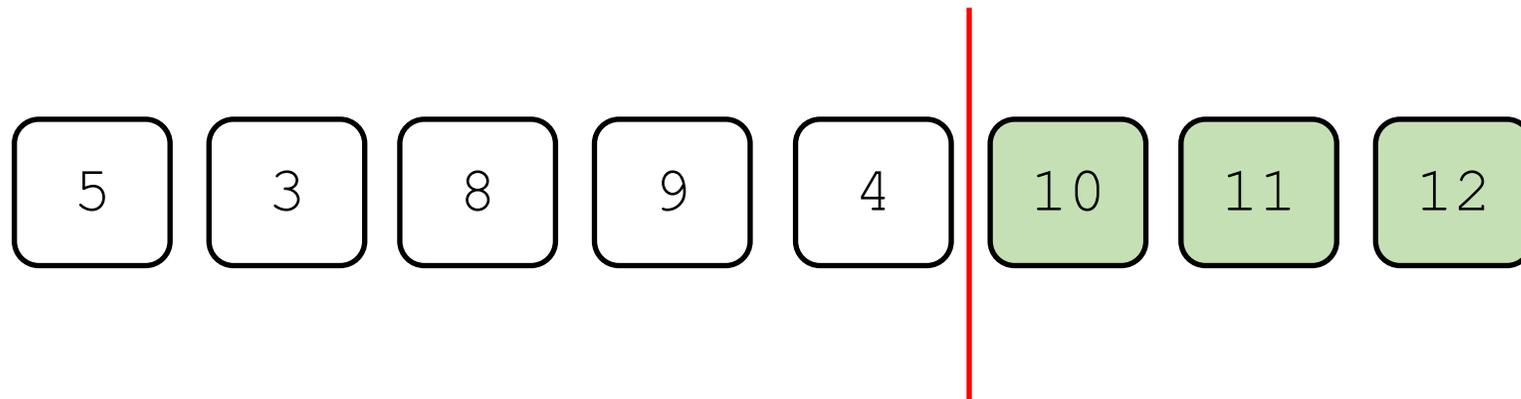


希尔排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

四. 冒泡排序

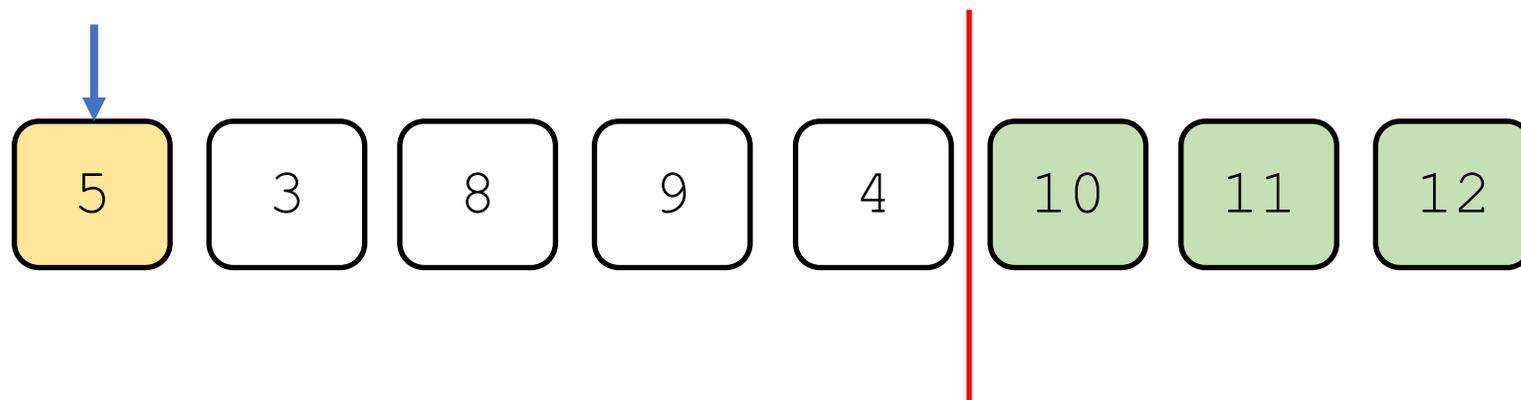
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

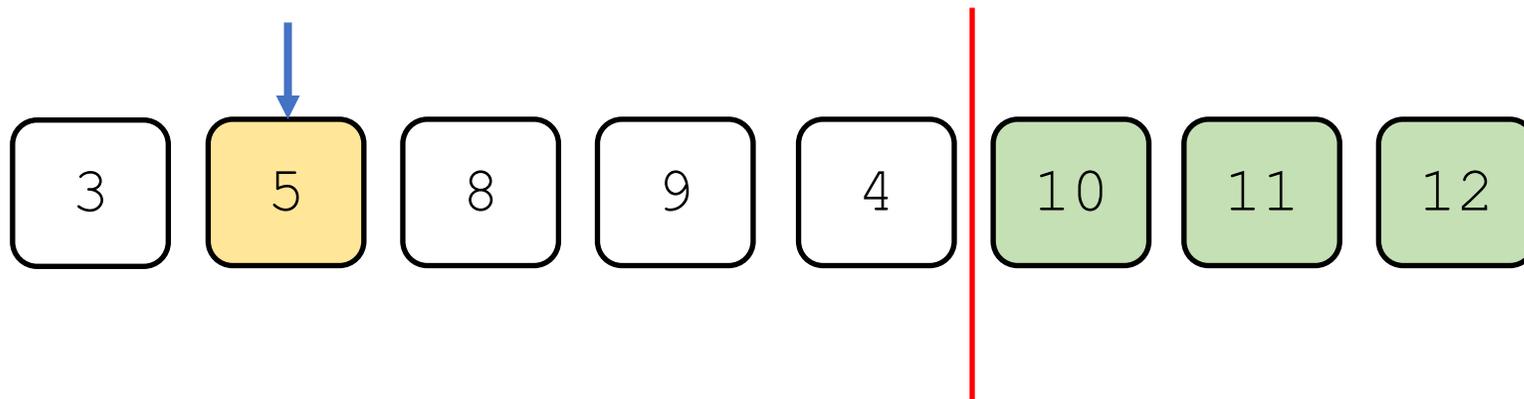
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

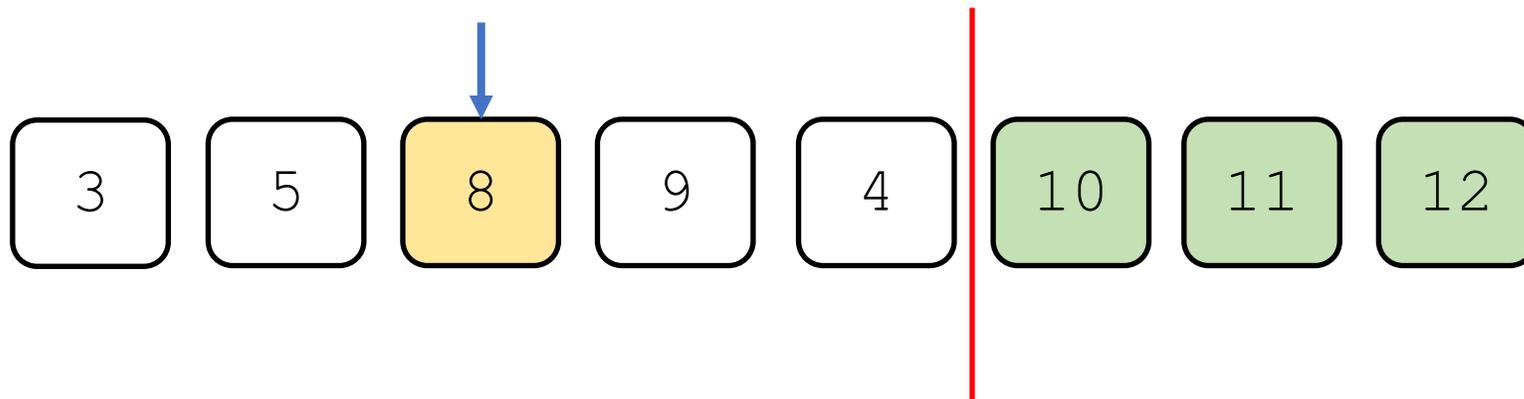
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

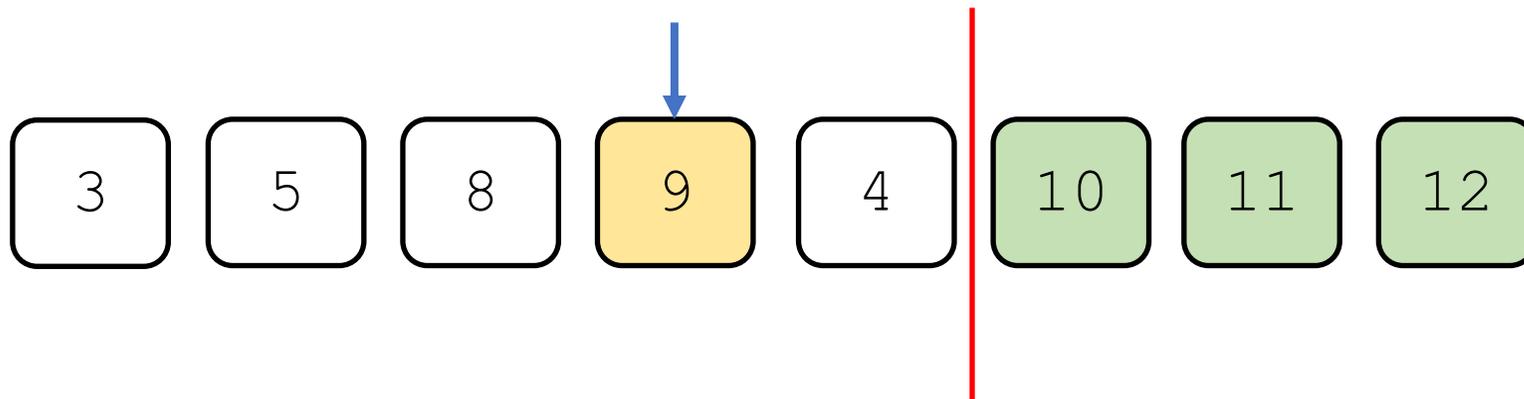
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

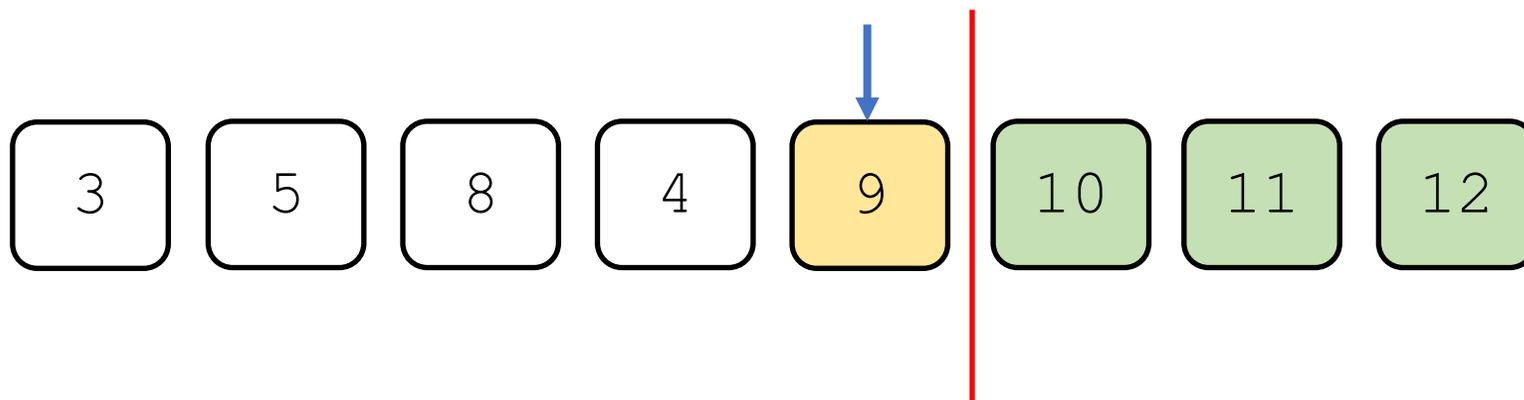
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

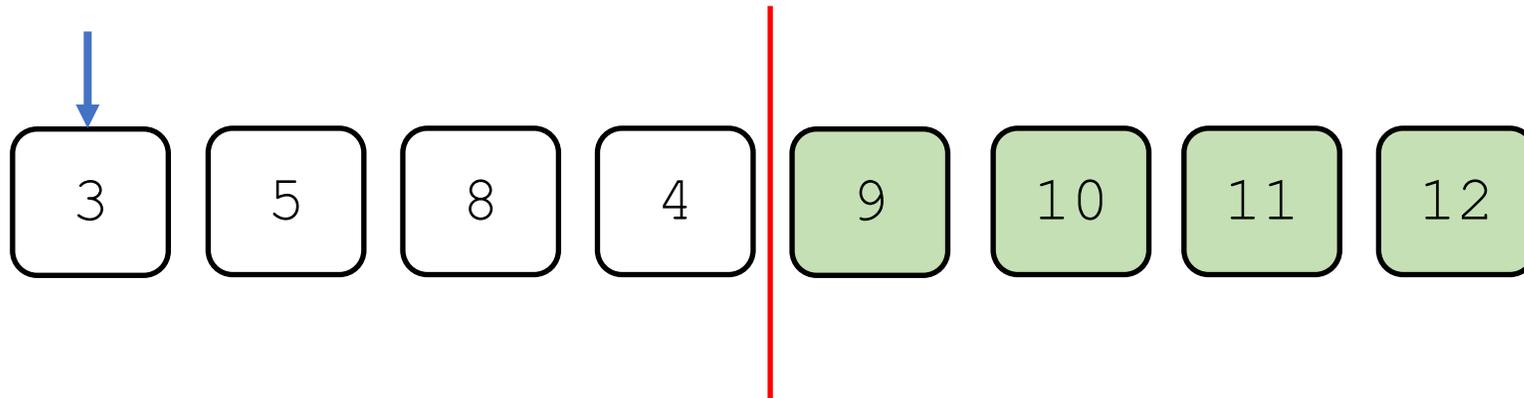
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

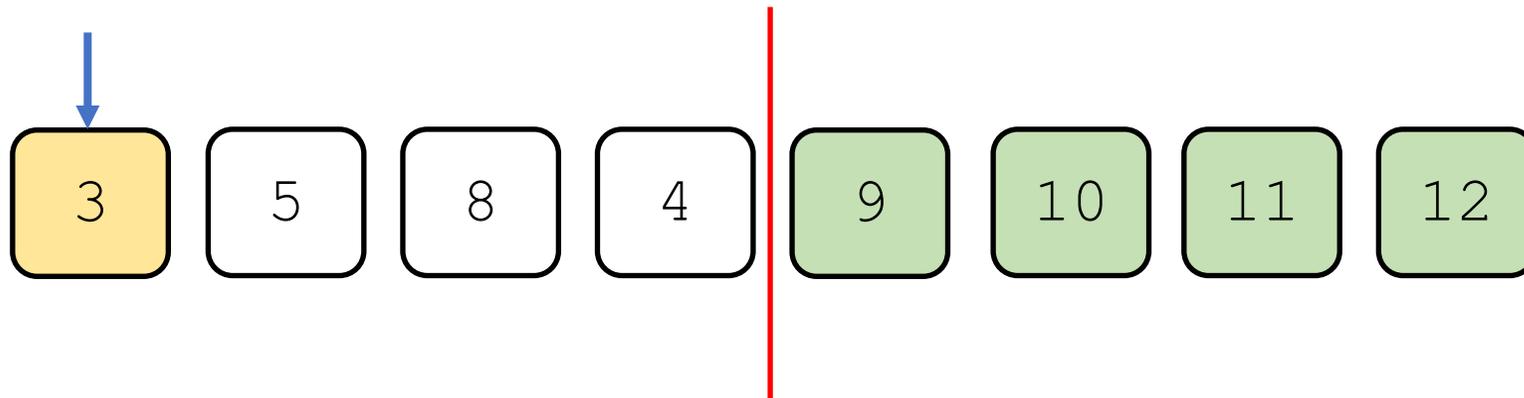
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

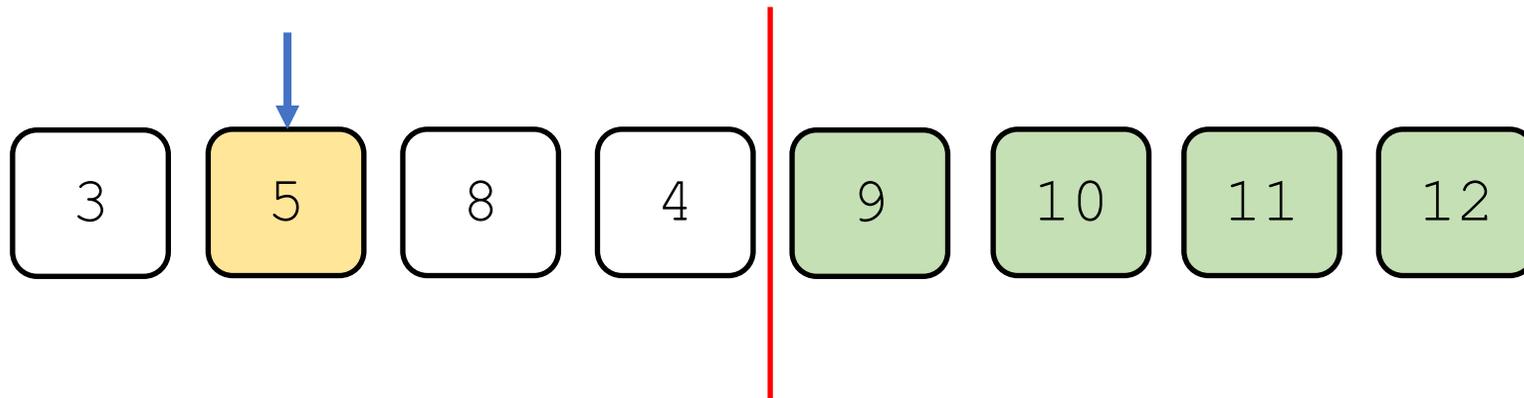
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

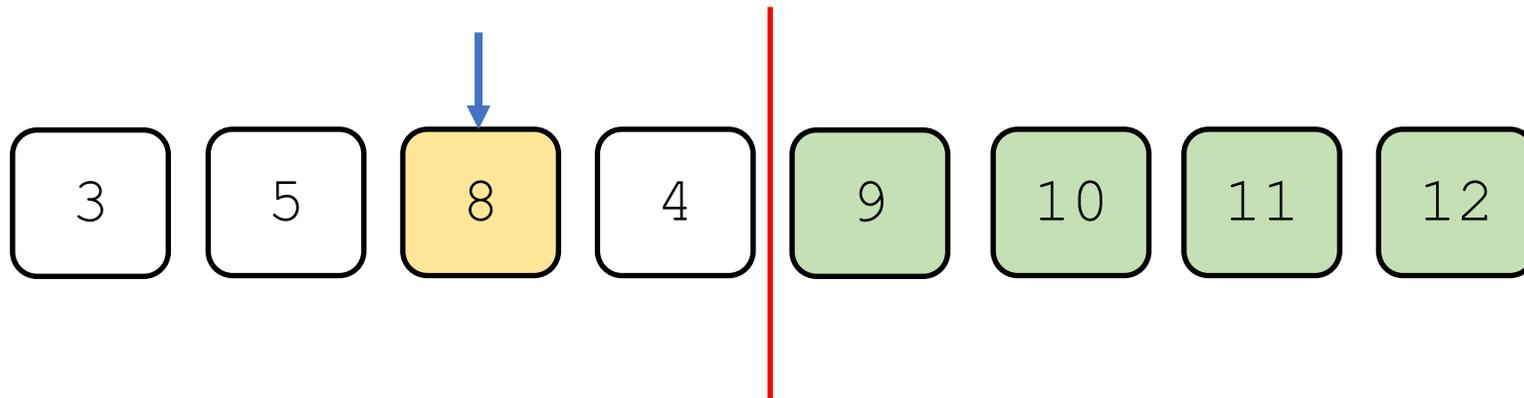
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

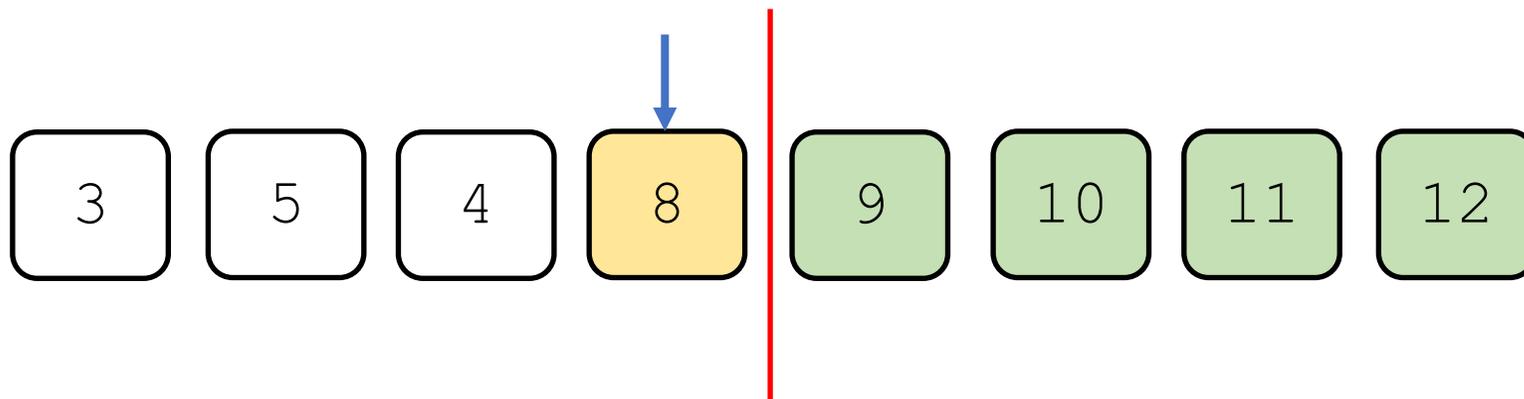
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

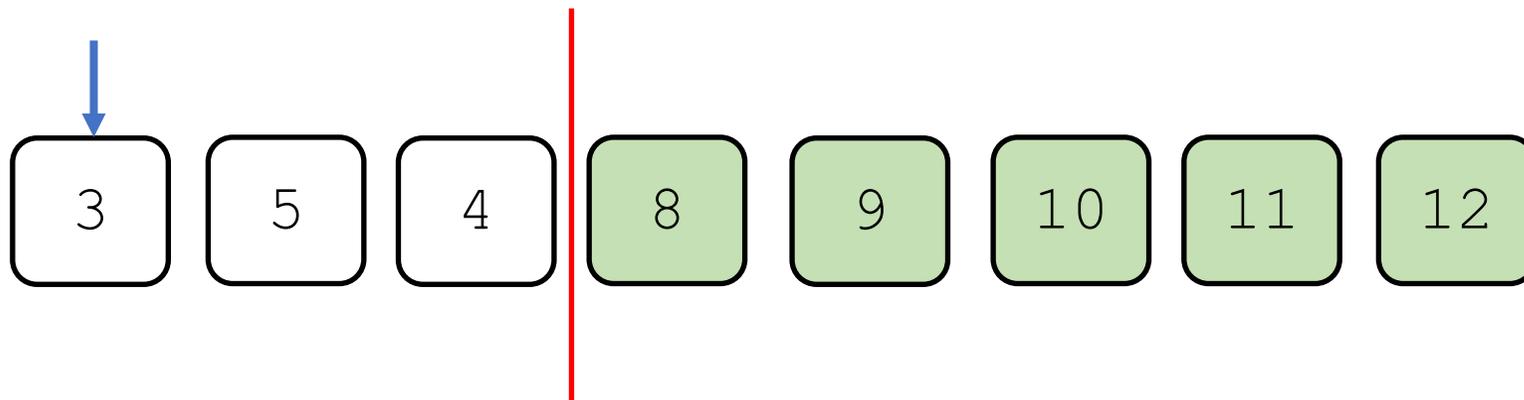
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

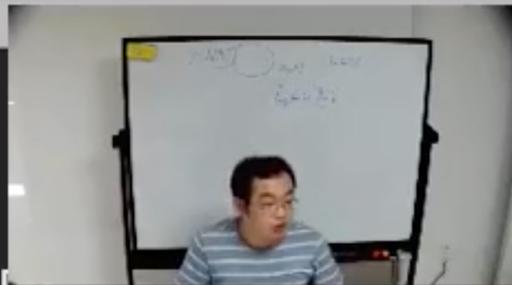
冒泡排序



口诀:

- 1、将数组分成『已排序区』和『待排序区』
- 2、从头到尾扫描『待排序区』，若前面元素比后面元素大，则交换
- 3、每一轮都会将『待排序区』中最大的放到『已排序区』的开头
- 4、直到『待排序区』没有元素为止

```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```

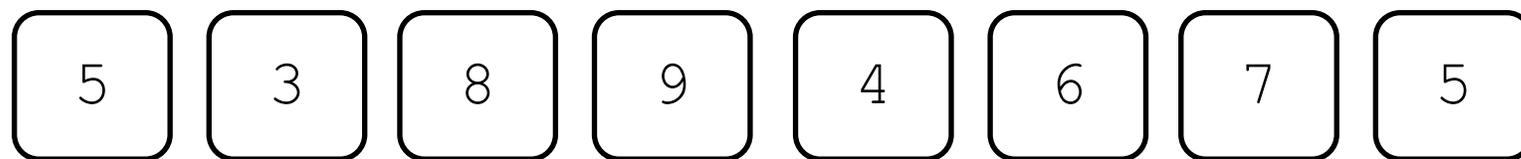


冒泡排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

五. 快速排序

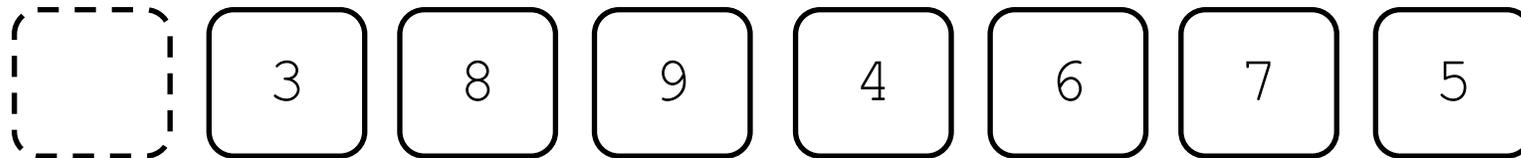
快速排序



快速排序

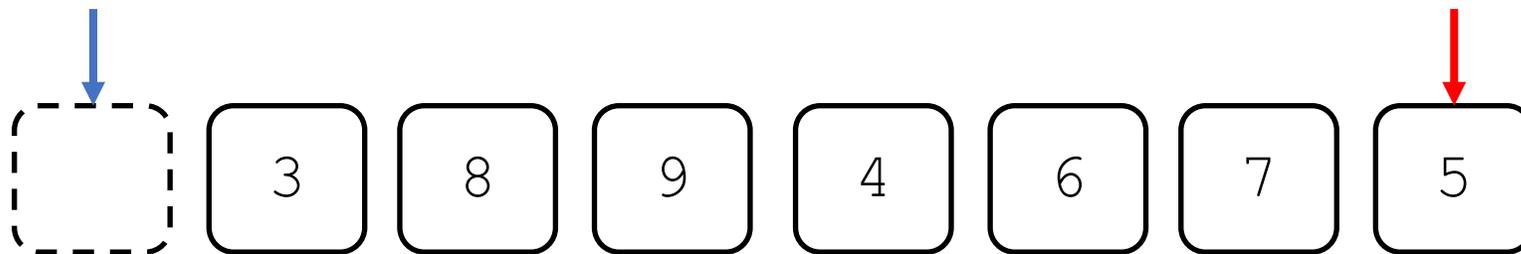
选择基准值

5



快速排序

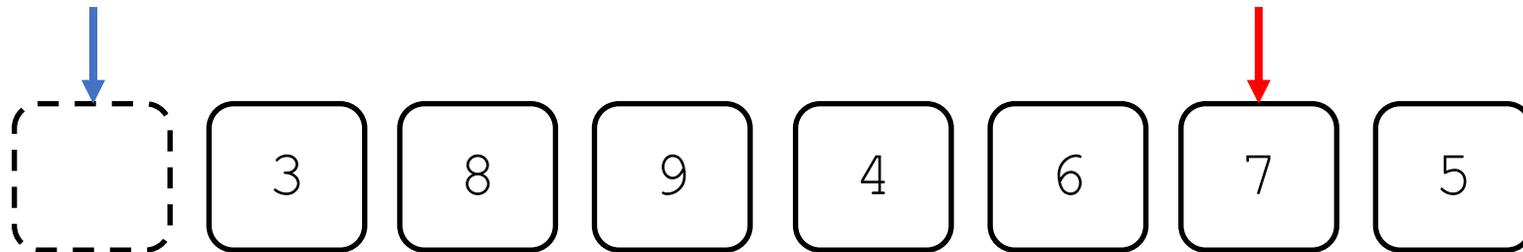
选择基准值 



快速排序

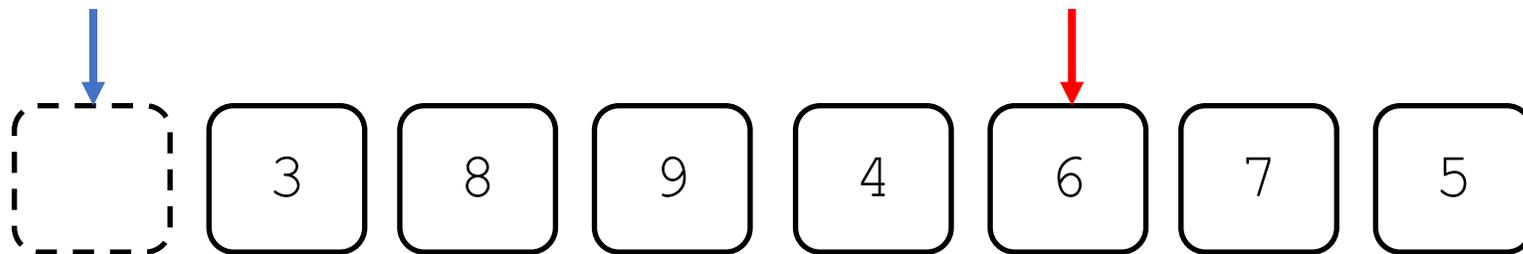
选择基准值

5



快速排序

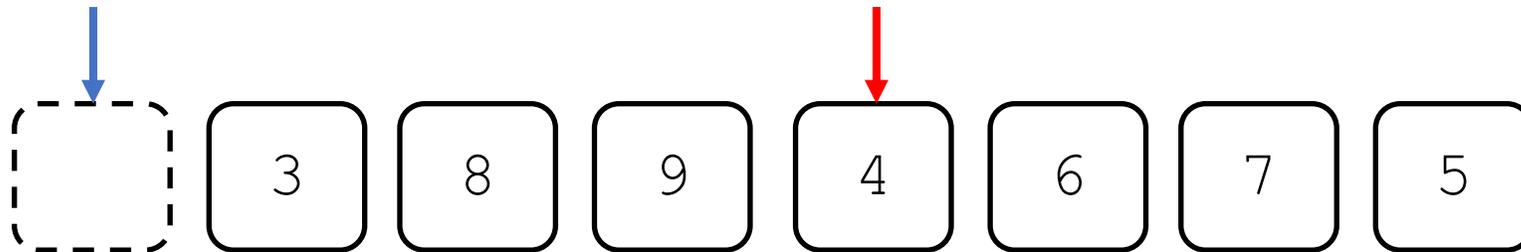
选择基准值 5



快速排序

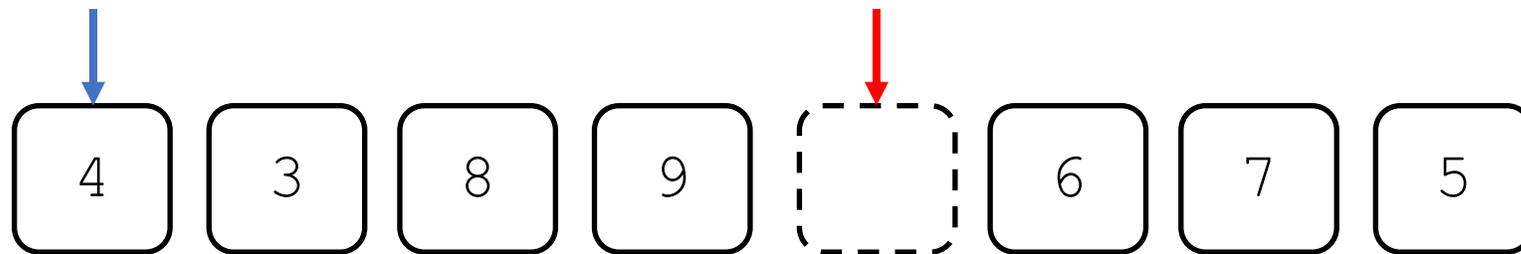
选择基准值

5



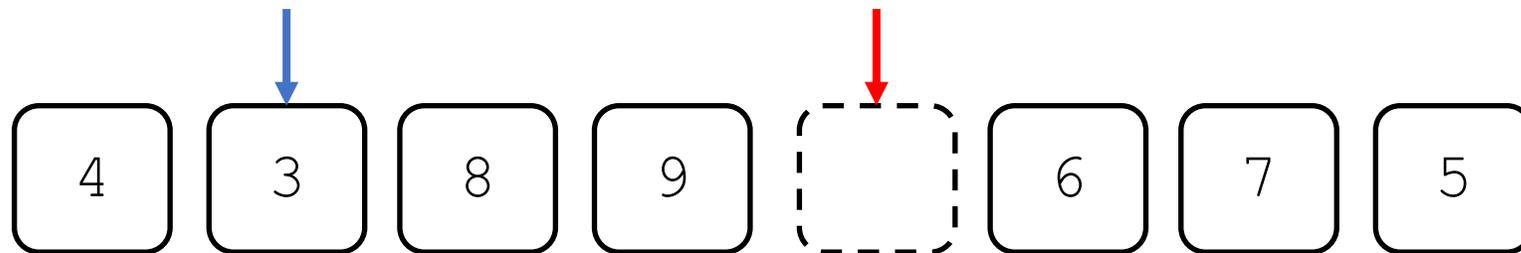
快速排序

选择基准值 5



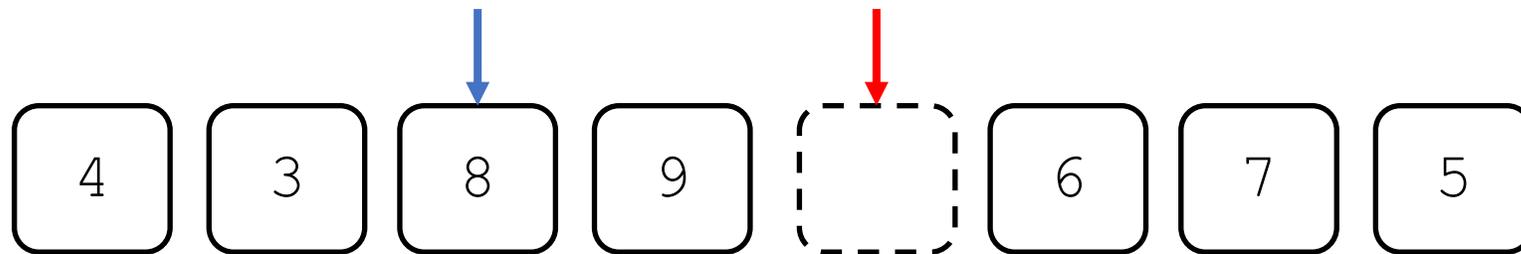
快速排序

选择基准值 5



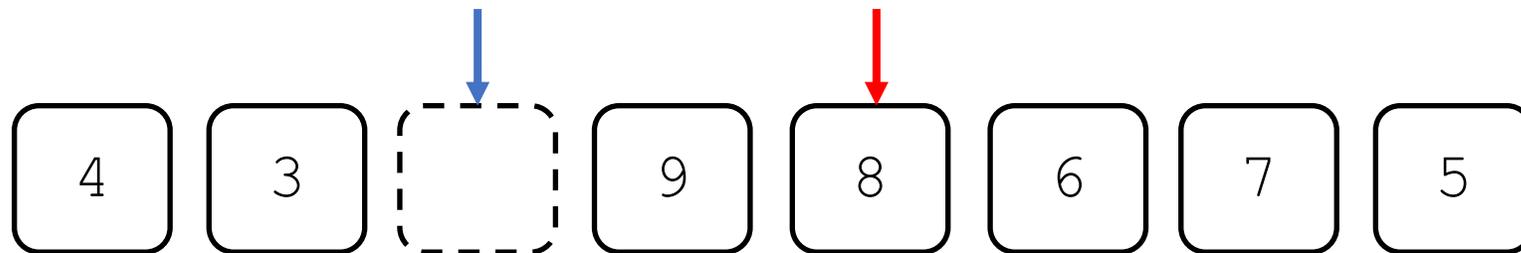
快速排序

选择基准值 5



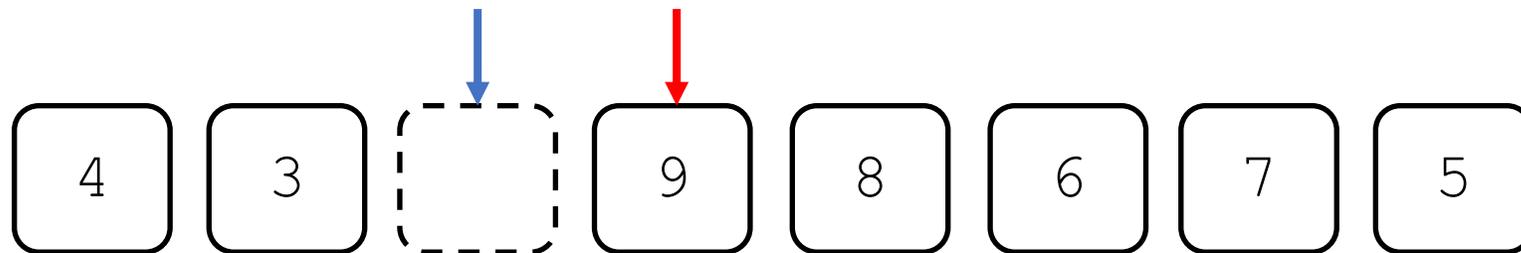
快速排序

选择基准值 5



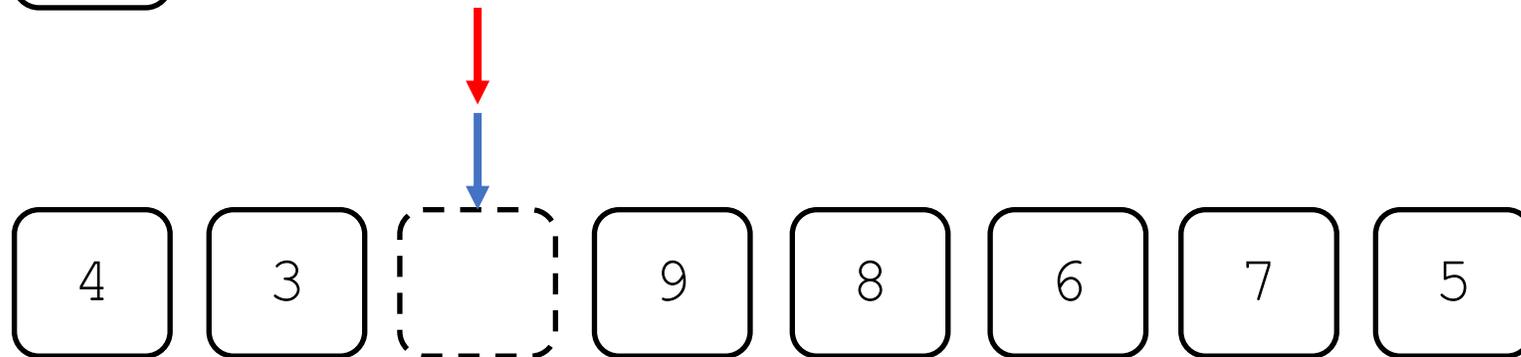
快速排序

选择基准值 5



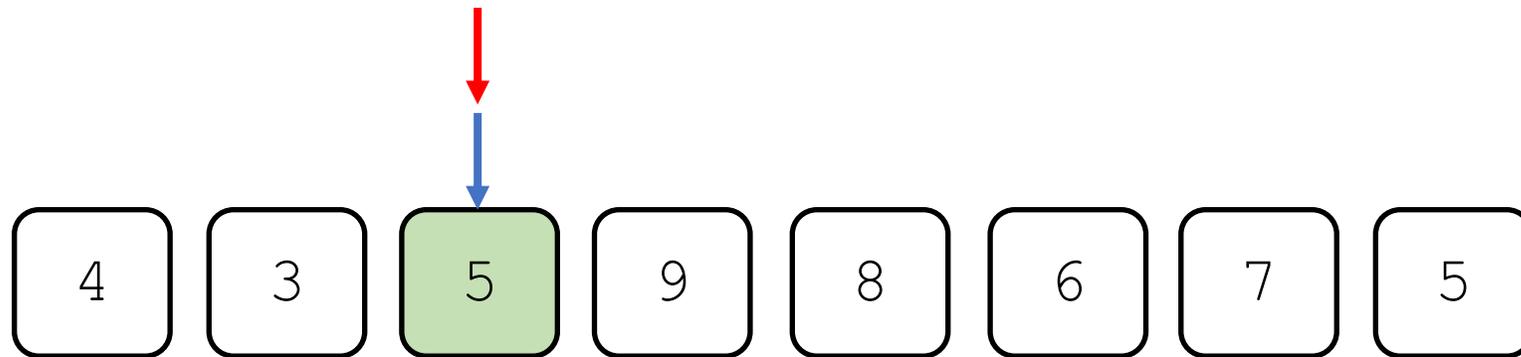
快速排序

选择基准值 5



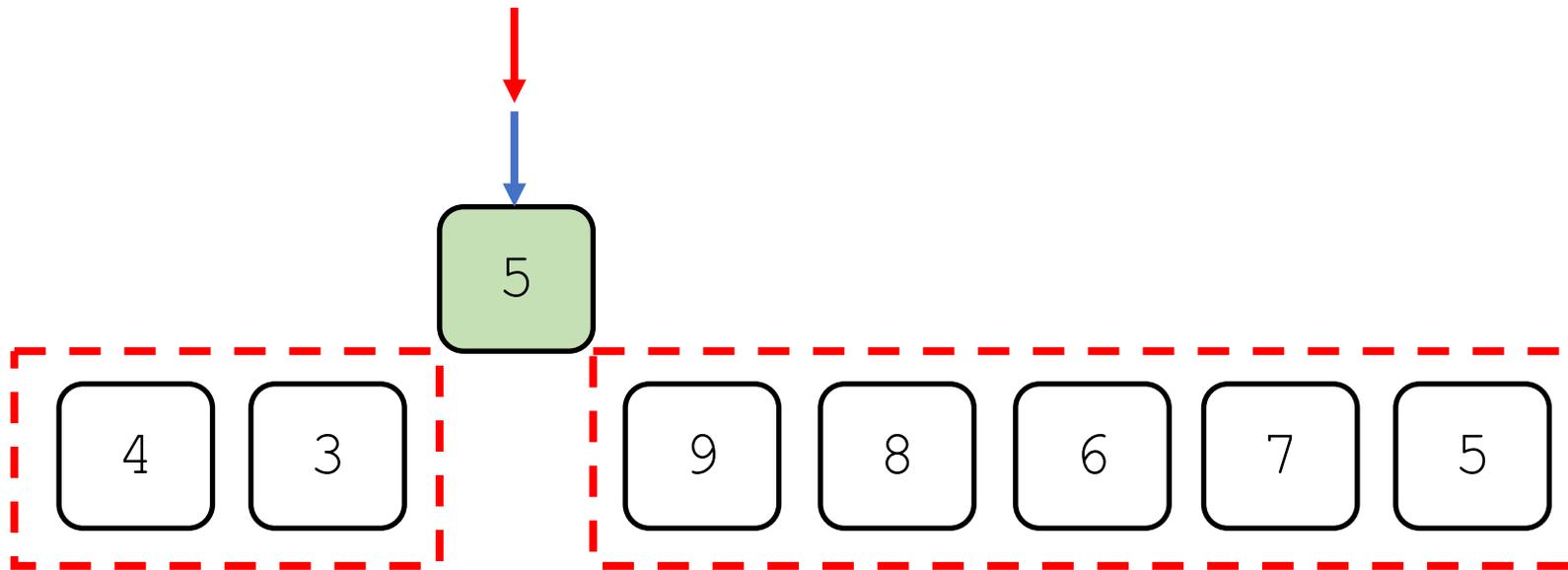
快速排序

选择基准值



快速排序

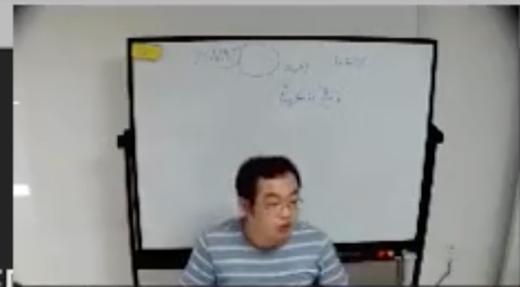
选择基准值



快速排序

时间复杂度： $O(n \log n) \sim O(n^2)$

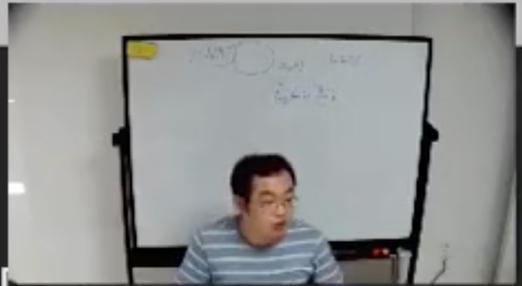
```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```



快速排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

```
1. vim
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```



快速排序的优化：代码演示

六. 归并排序

归并排序

初始

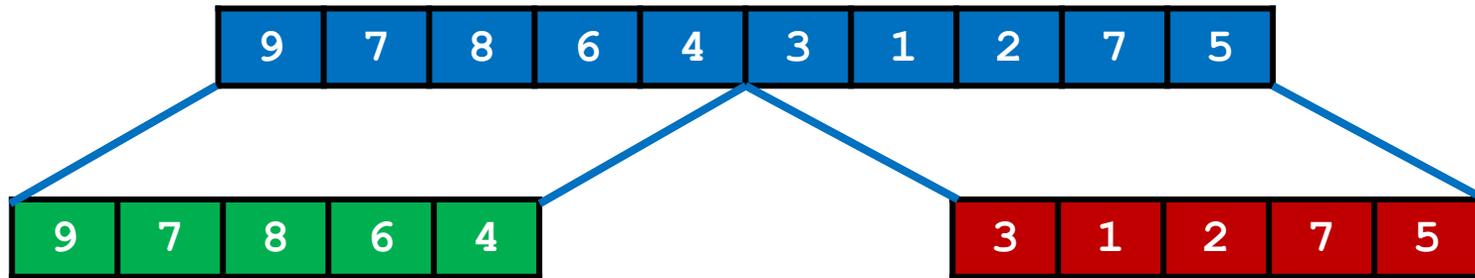


归并排序

归并排序

归并排序

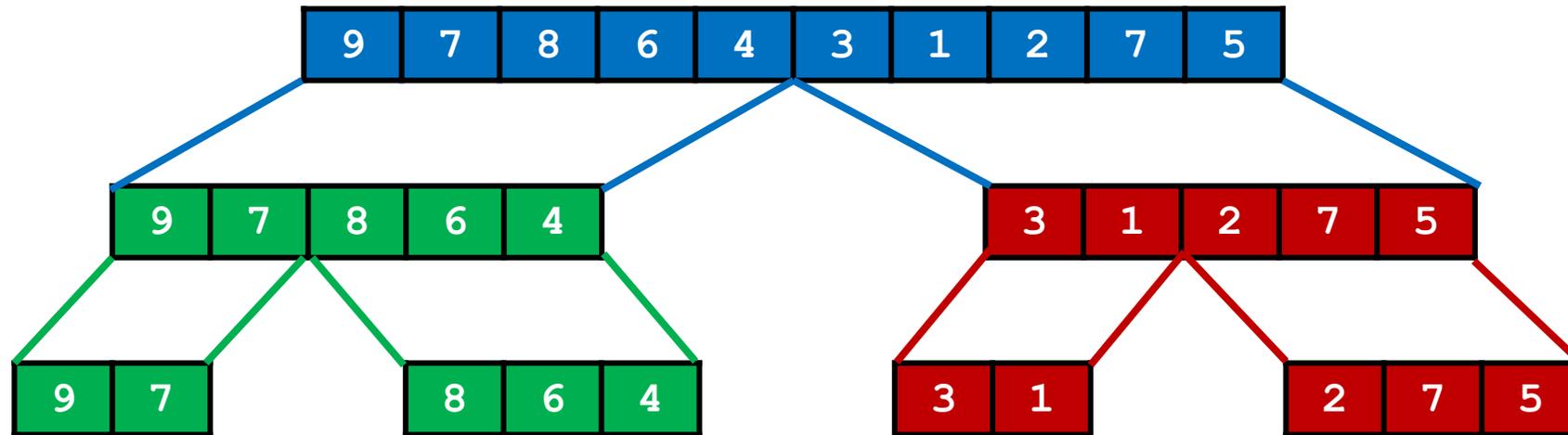
分治



归并排序

归并排序

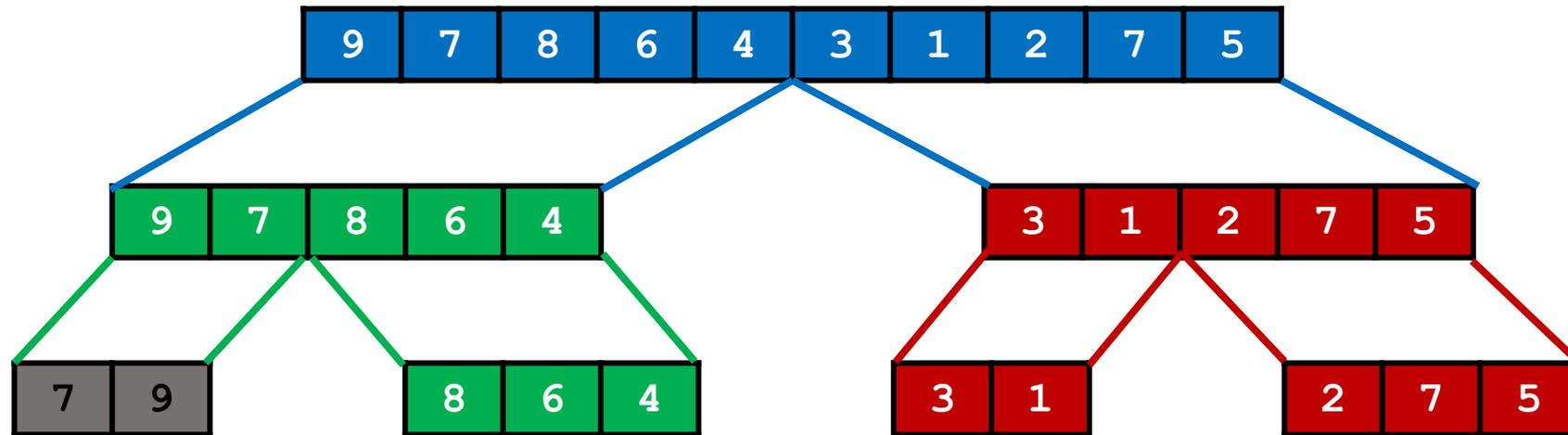
分治



归并排序

归并排序

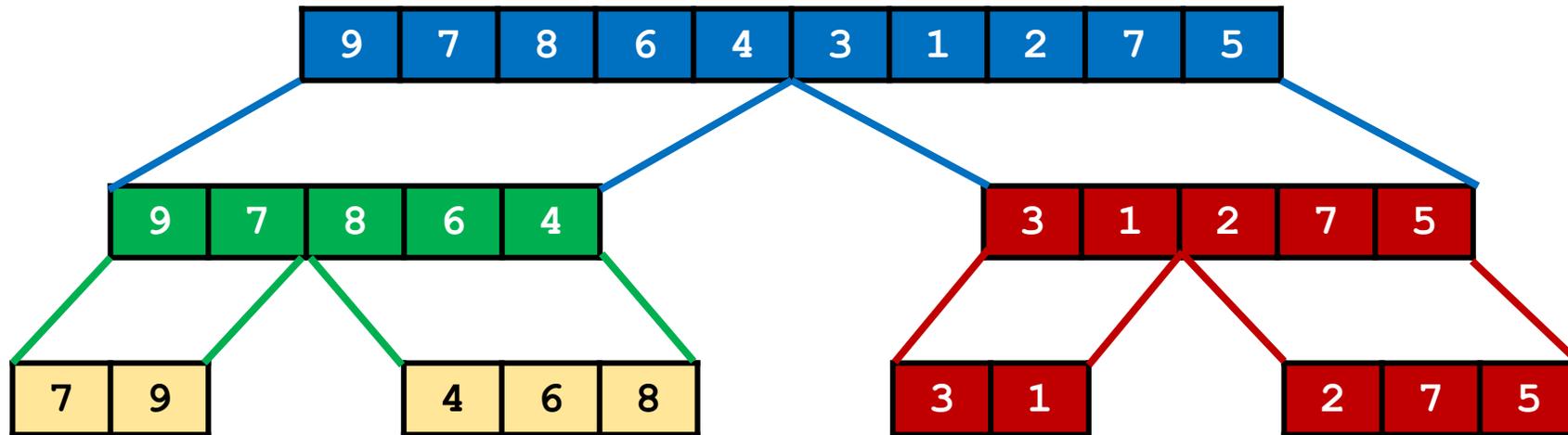
排序



归并排序

归并排序

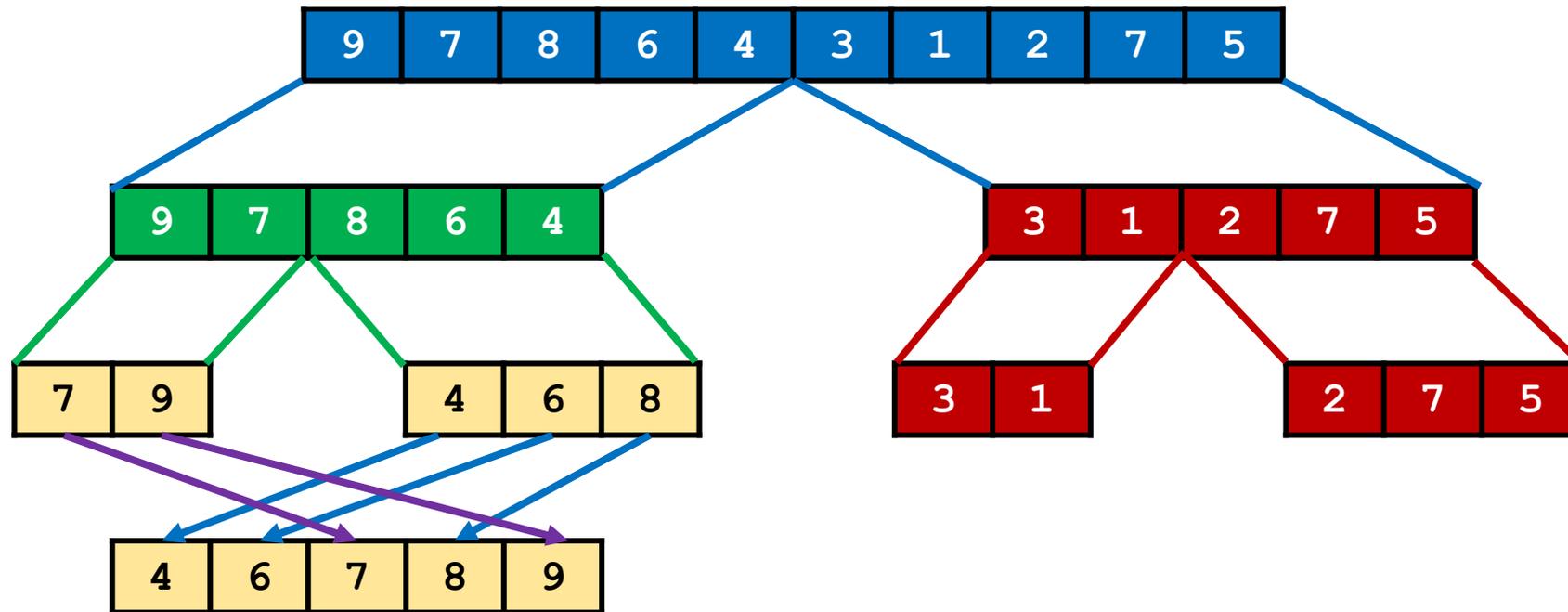
排序



归并排序

归并排序

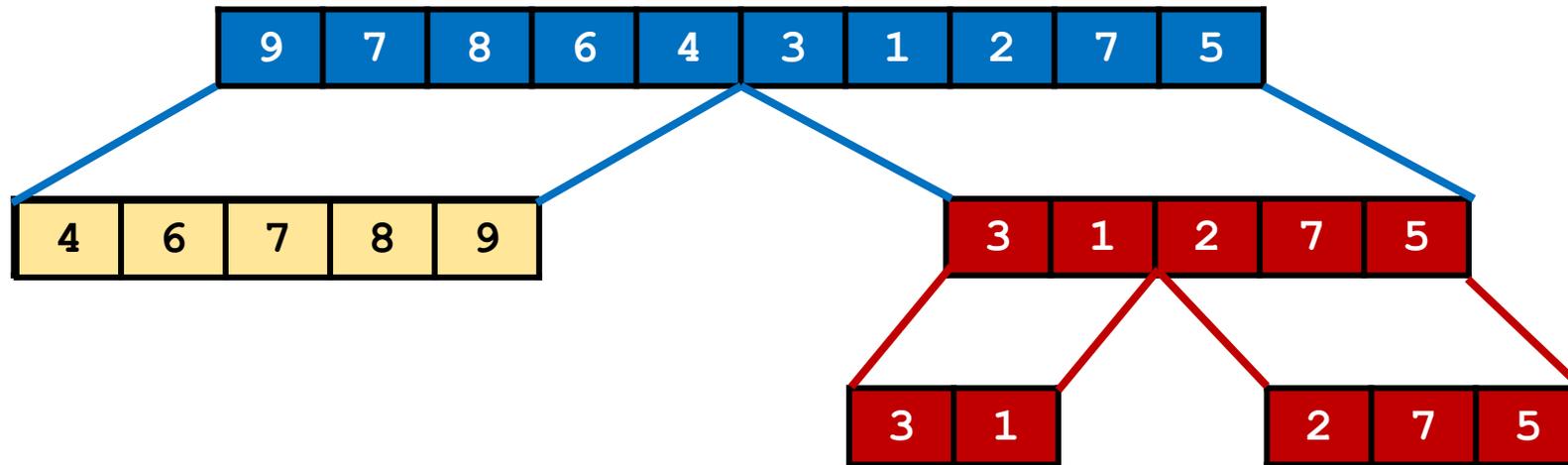
归并



归并排序

归并排序

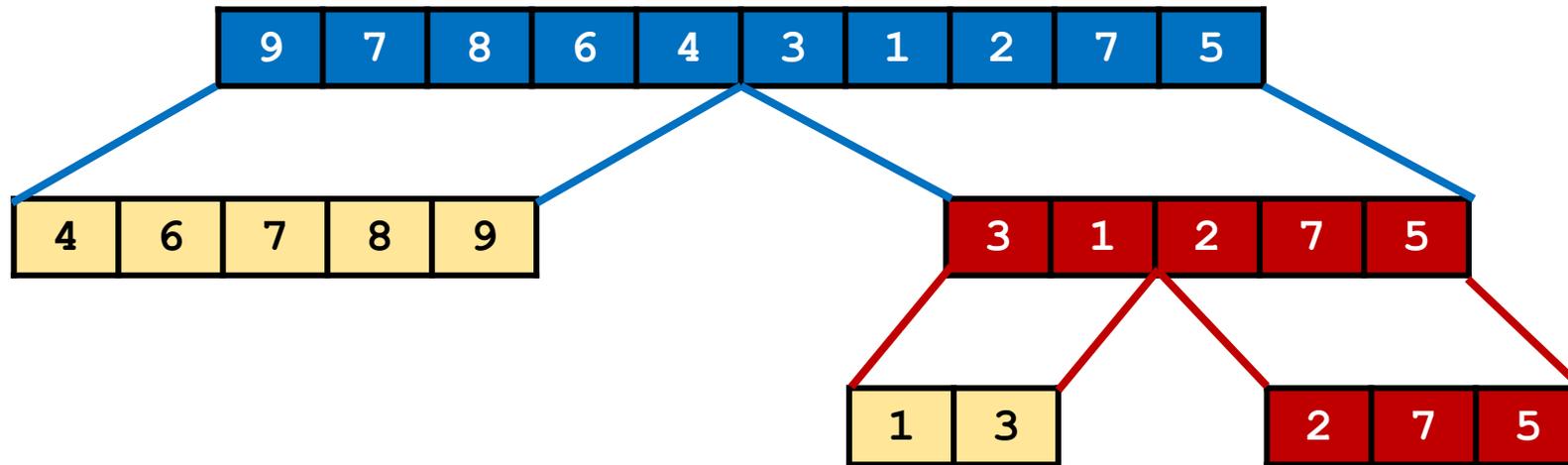
拷贝



归并排序

归并排序

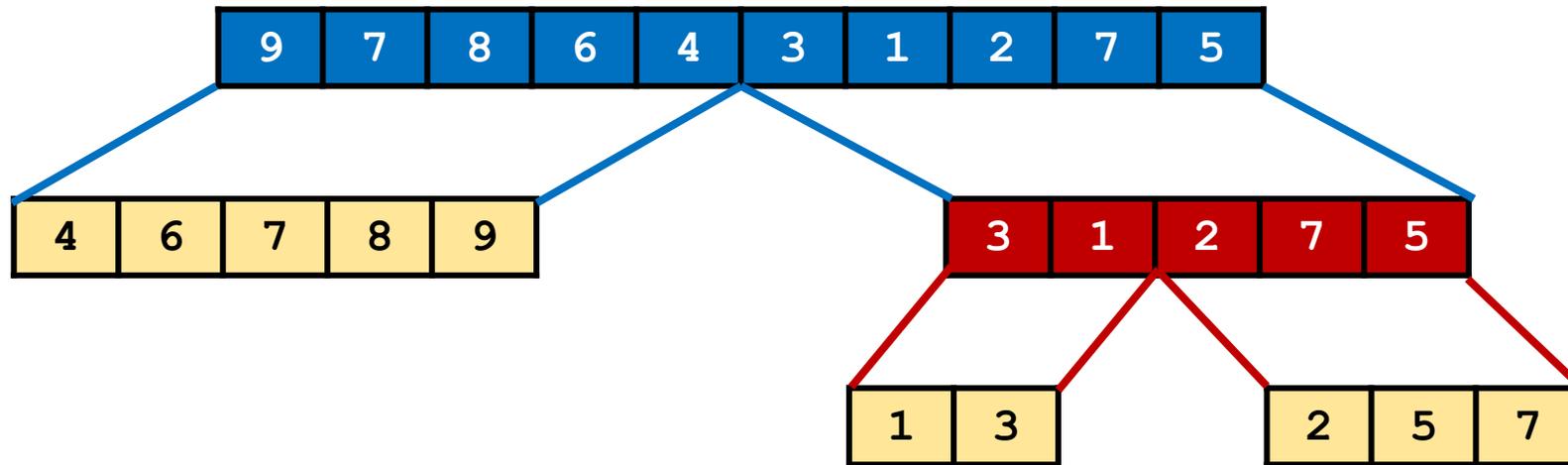
排序



归并排序

归并排序

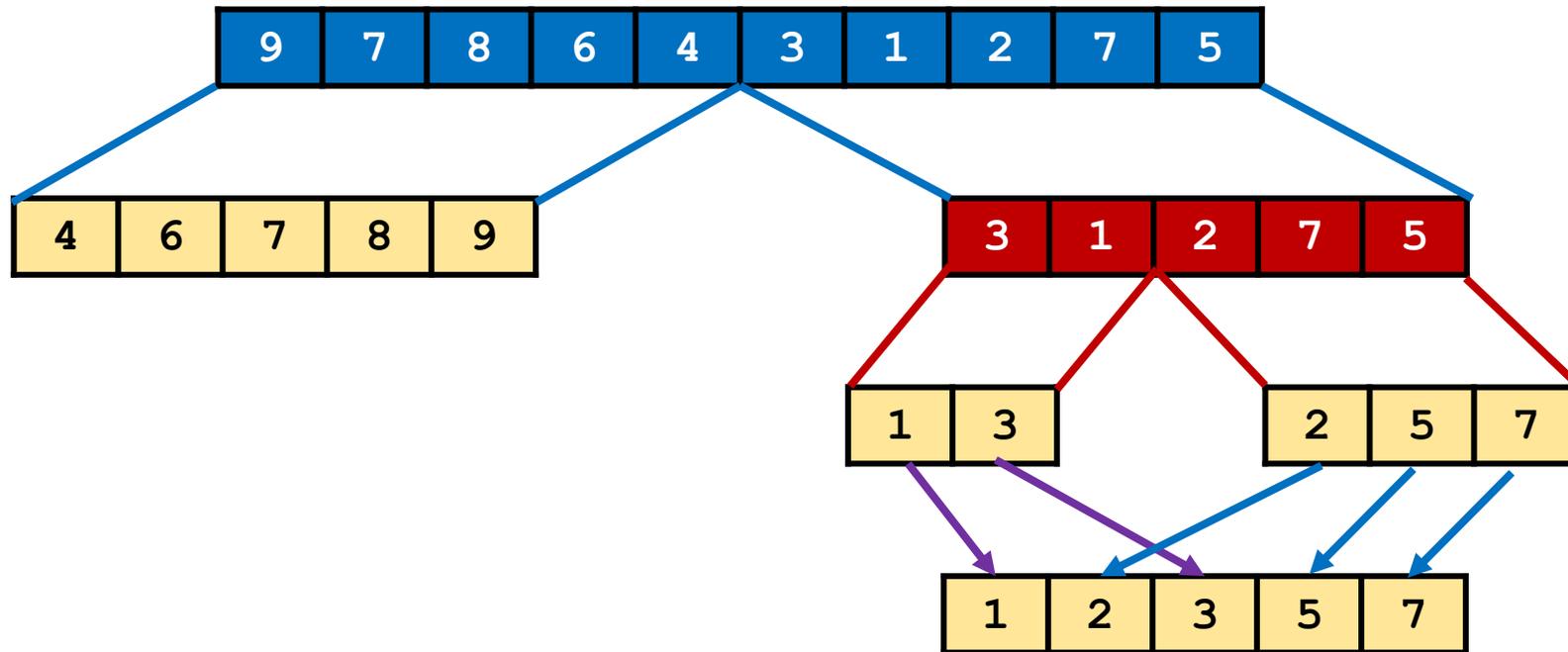
排序



归并排序

归并排序

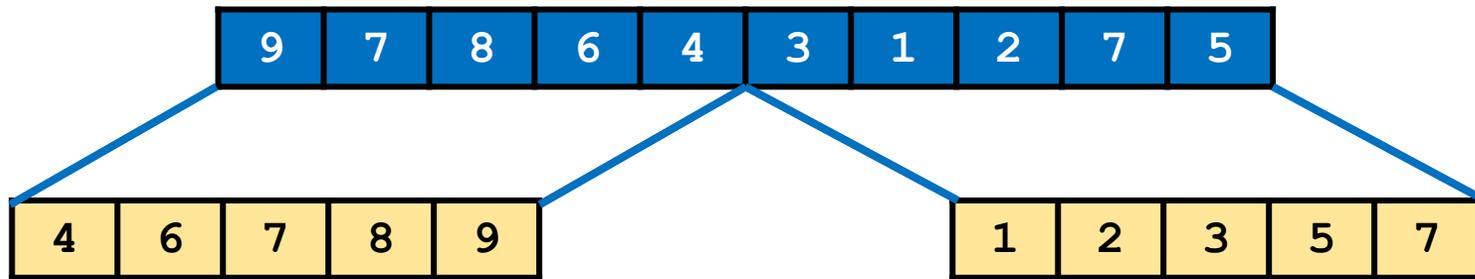
归并



归并排序

归并排序

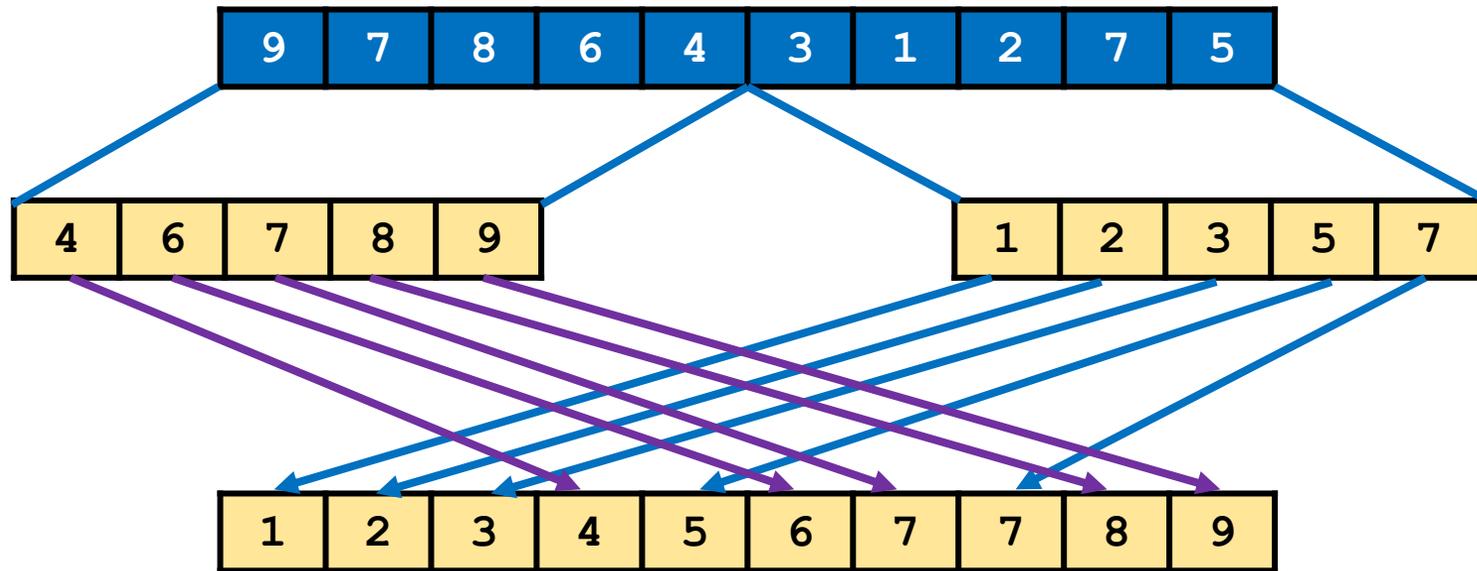
拷贝



归并排序

归并排序

归并



归并排序

归并排序

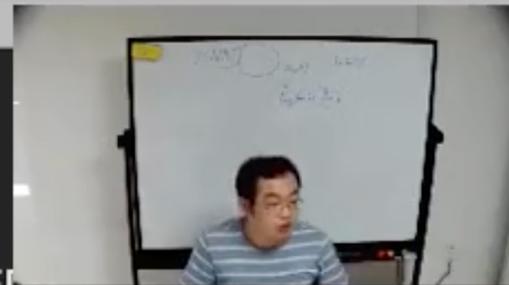
拷贝

1	2	3	4	5	6	7	7	8	9
---	---	---	---	---	---	---	---	---	---

归并排序

时间复杂度： $O(n \log n)$

```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```



归并排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

七. 基数排序

基数排序

13 21 11 32 31 22 21

基数排序

13 21 11 32 31 22 21

基数排序

21 11 31 21 32 22 13

基数排序

21 11 31 21 32 22 13

基数排序

21 11 31 21 32 22 13

基数排序

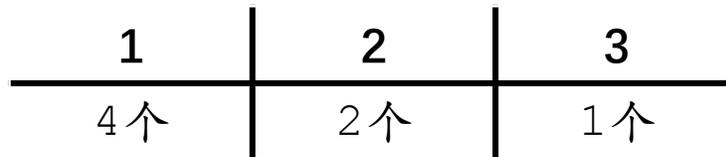
11 13 21 21 22 31 32

基数排序

11 13 21 21 22 31 32

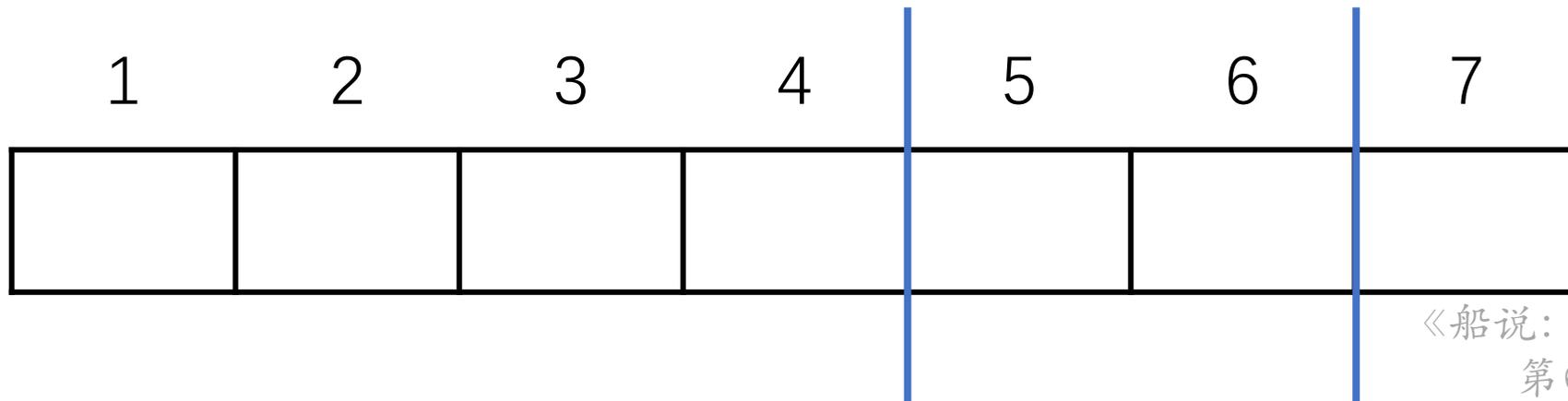
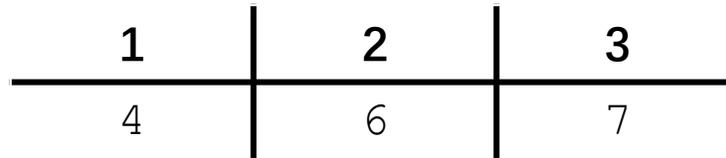
基数排序

对个位计数



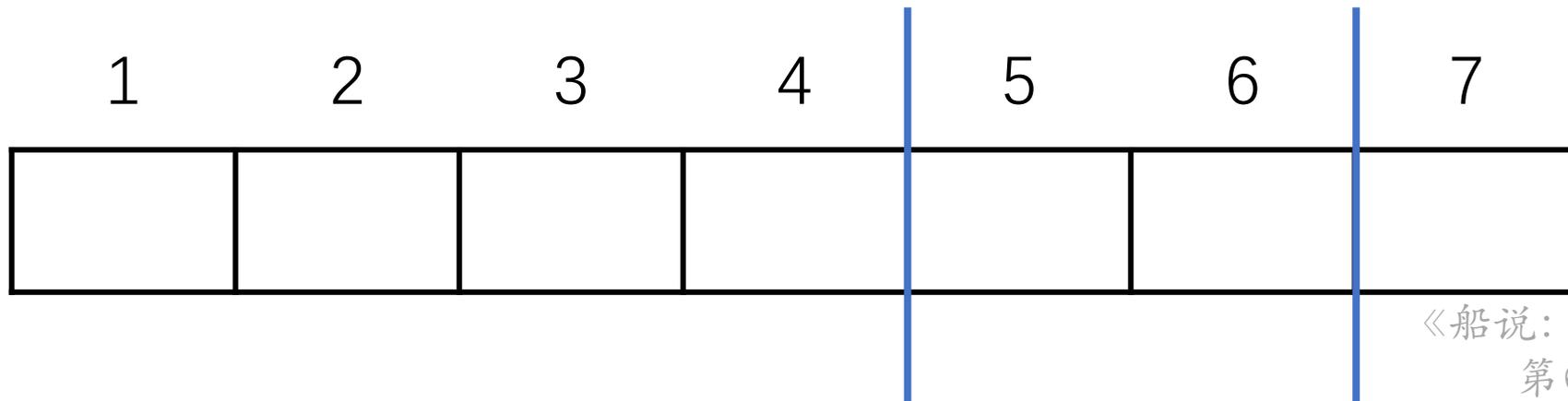
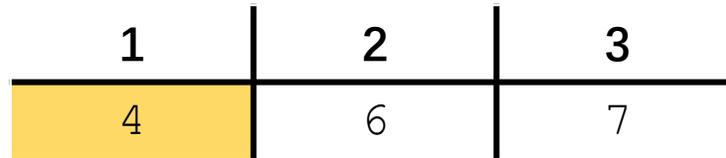
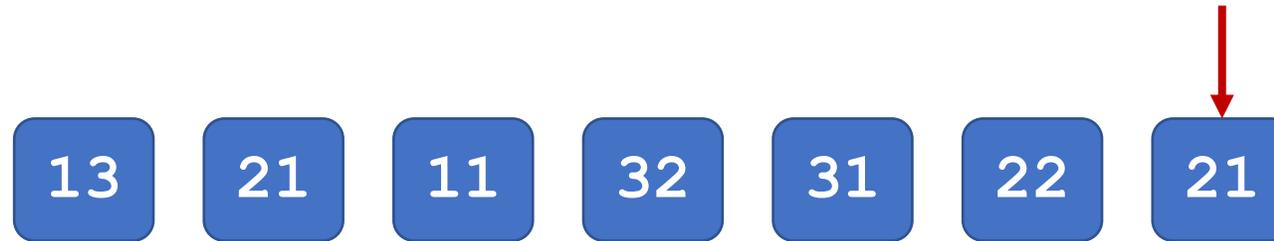
基数排序

求区域尾坐标



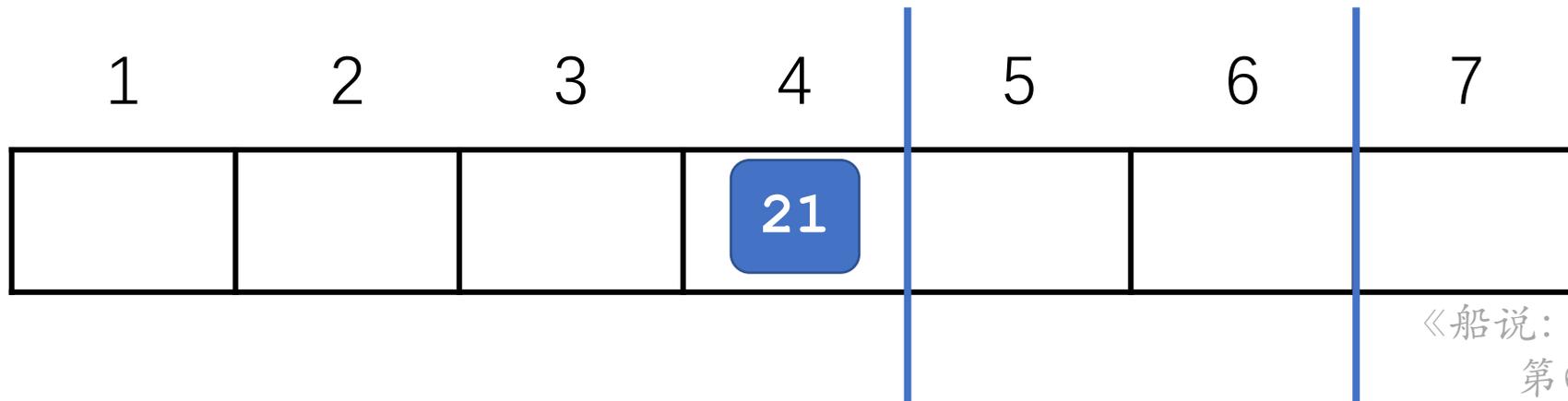
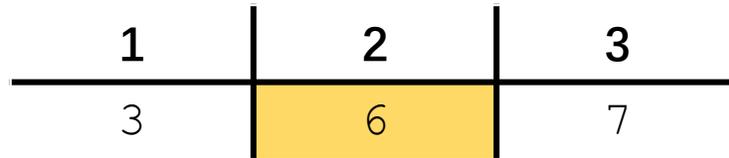
基数排序

归位



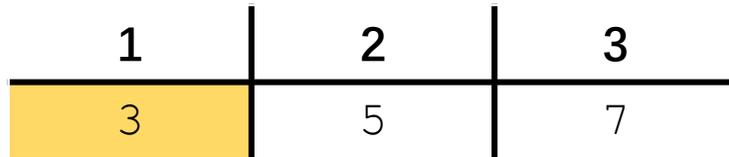
基数排序

归位



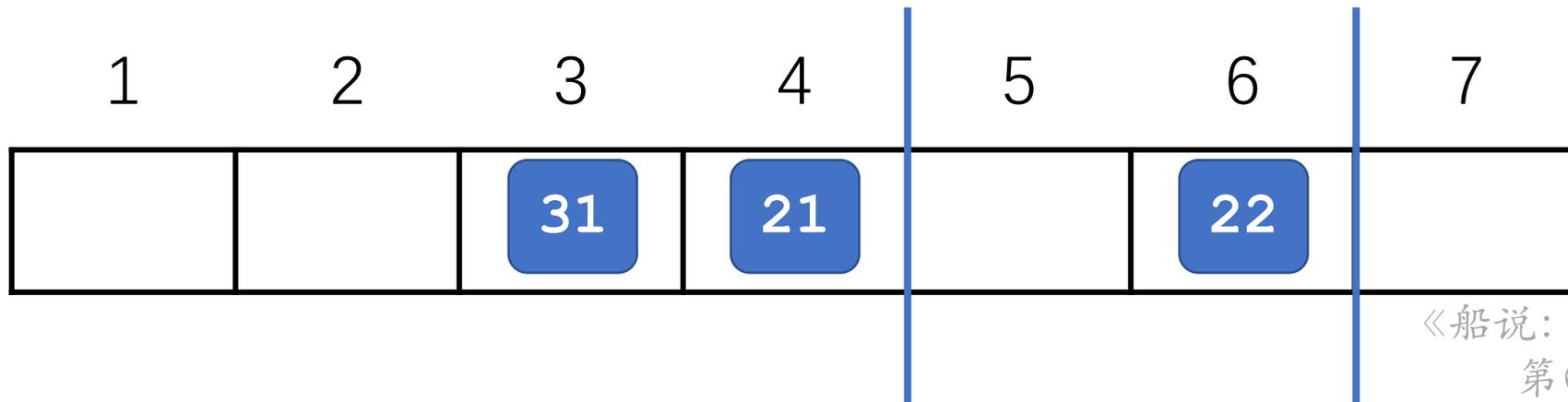
基数排序

归位



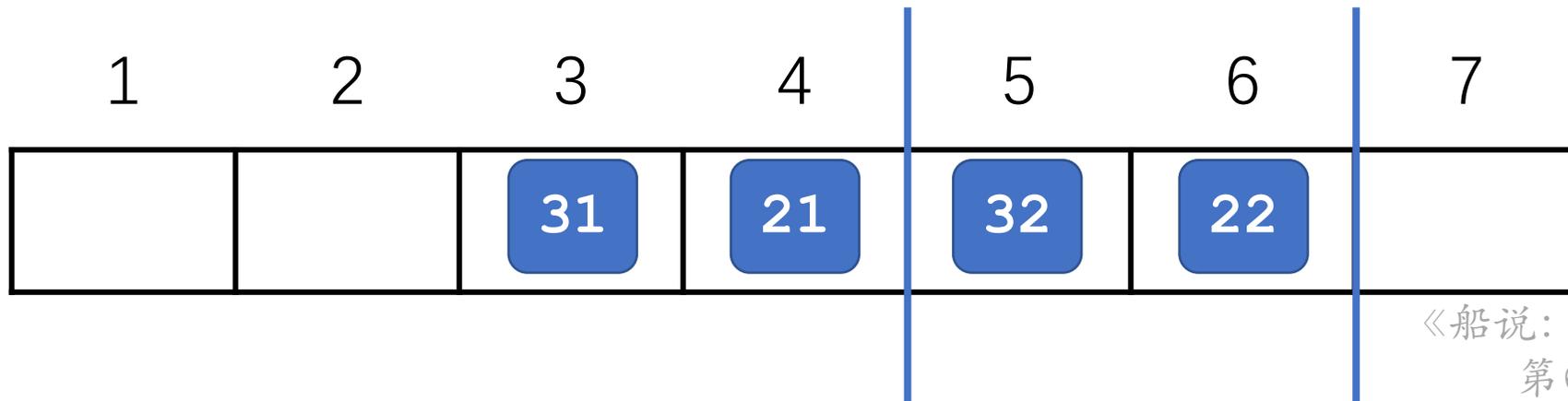
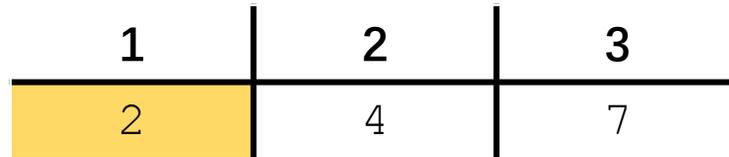
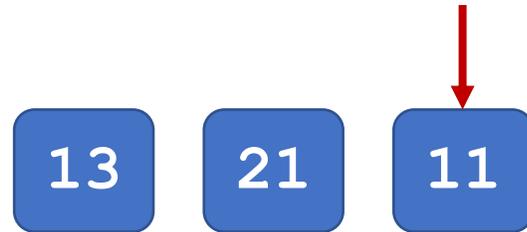
基数排序

归位



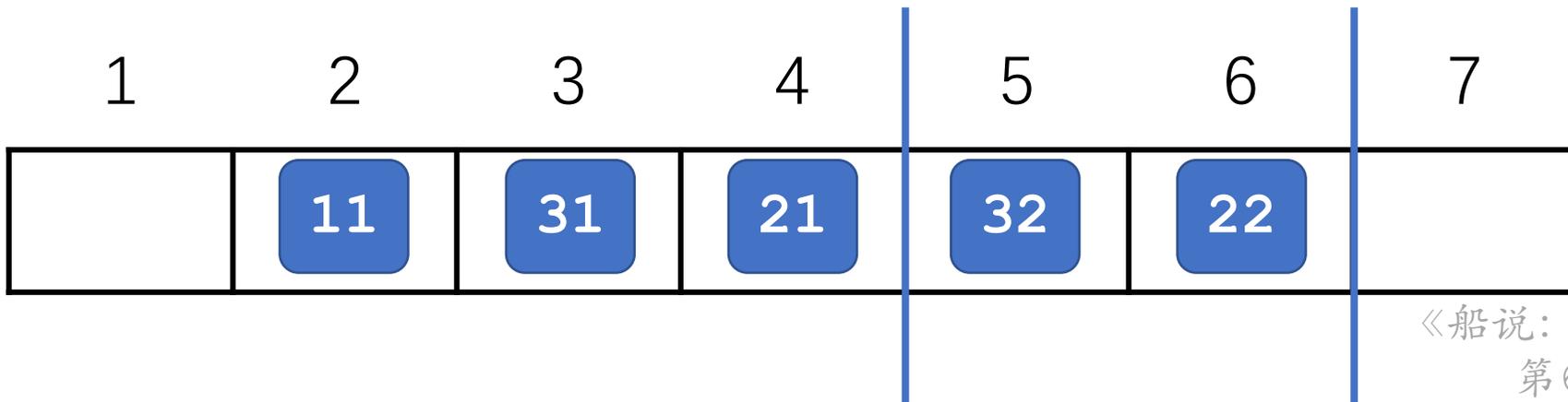
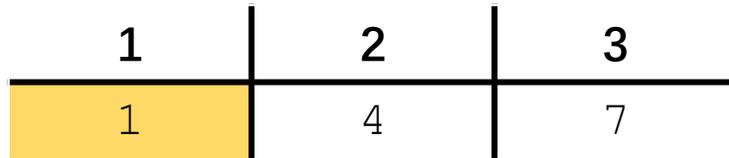
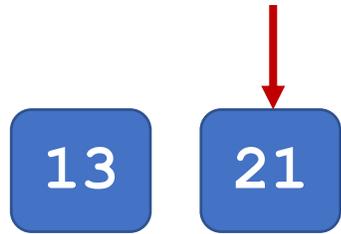
基数排序

归位



基数排序

归位

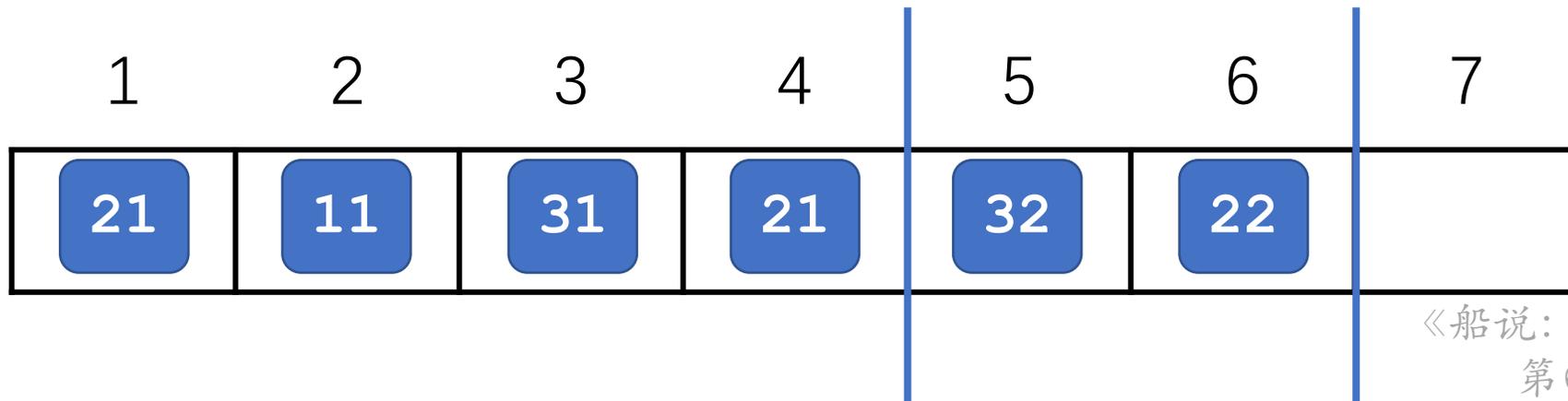


基数排序

归位

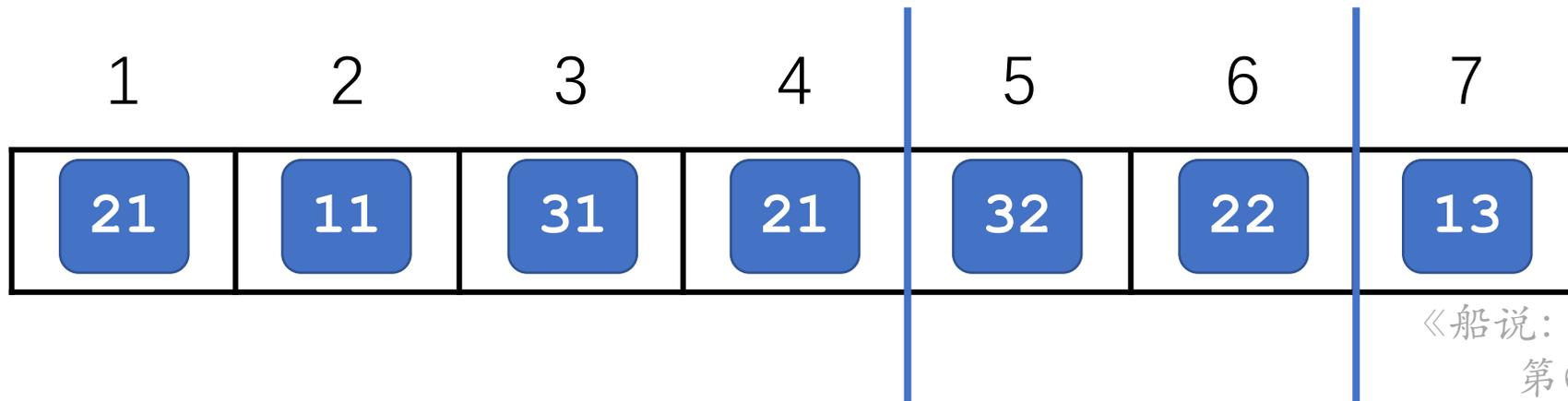
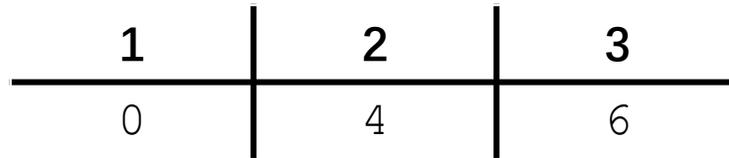
↓
13

1	2	3
0	4	7



基数排序

归位

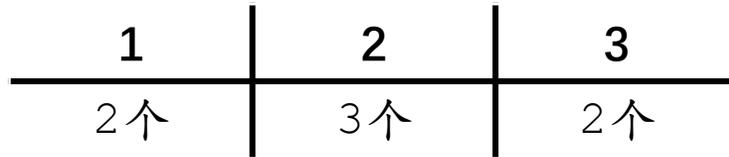


基数排序

21 11 31 21 32 22 13

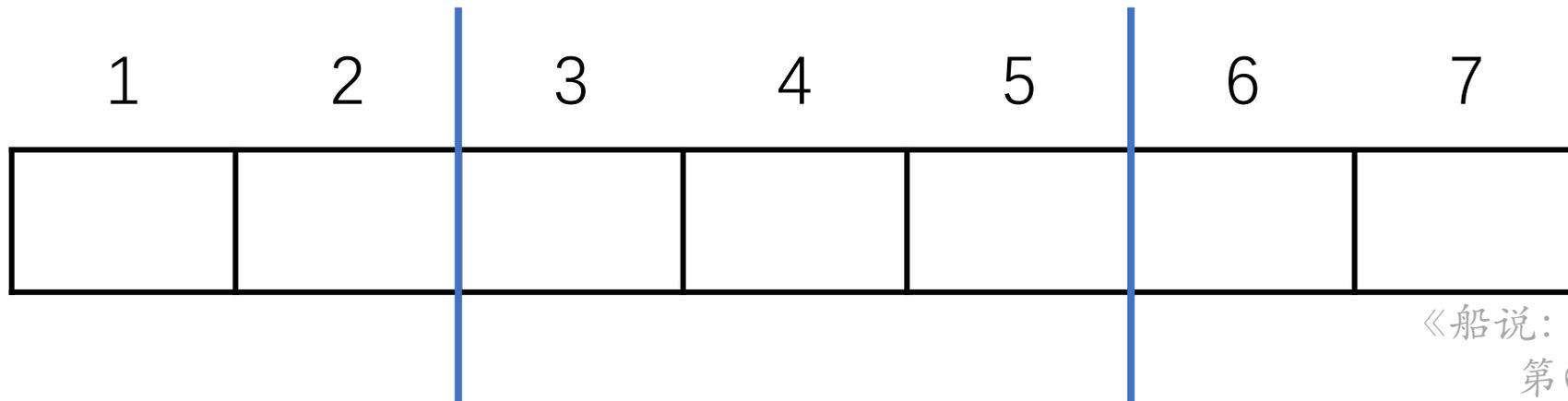
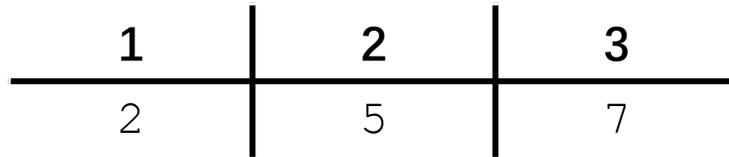
基数排序

对十位计数



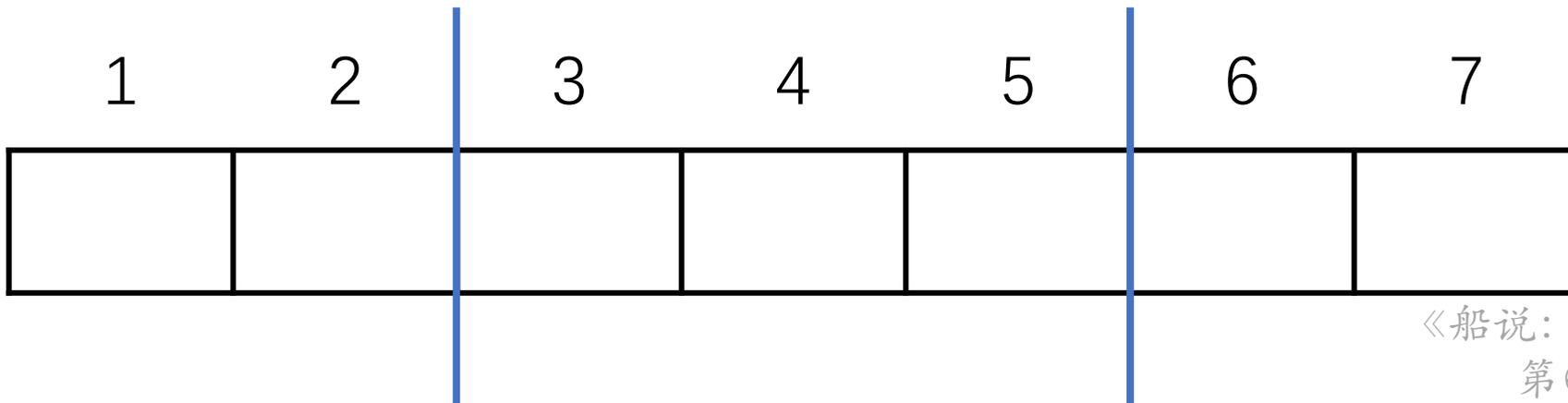
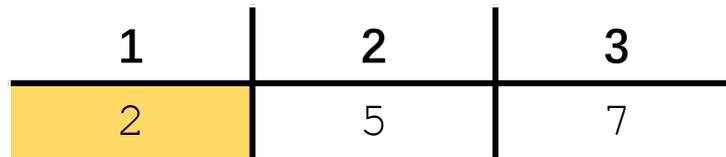
基数排序

求区域尾坐标



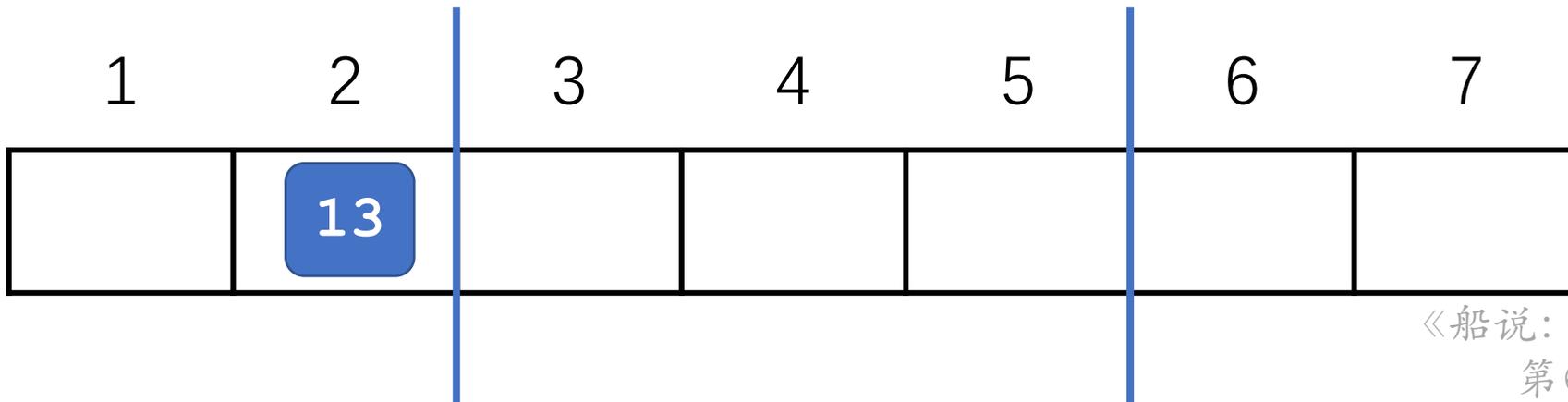
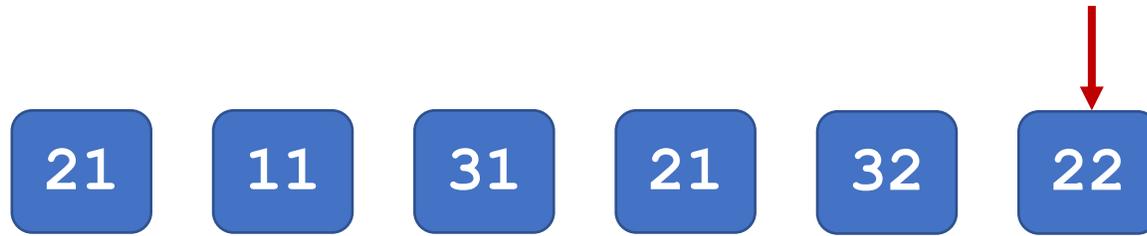
基数排序

归位



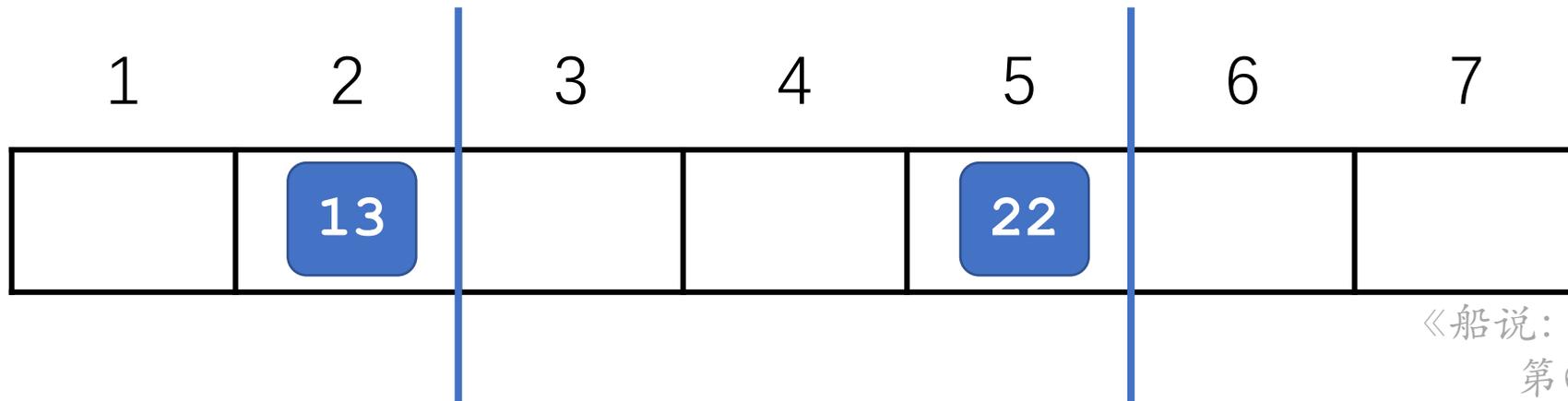
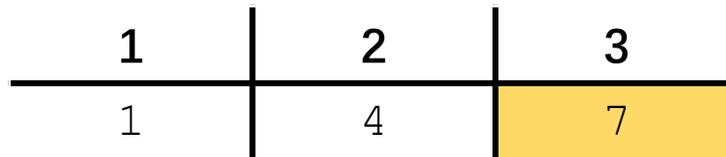
基数排序

归位



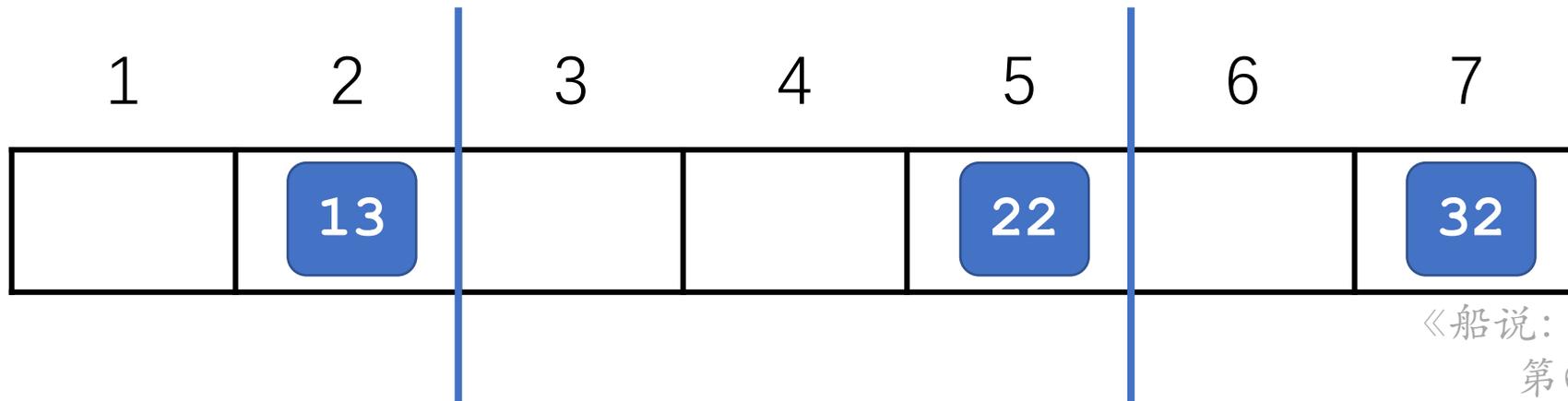
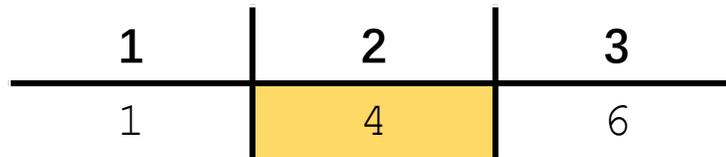
基数排序

归位



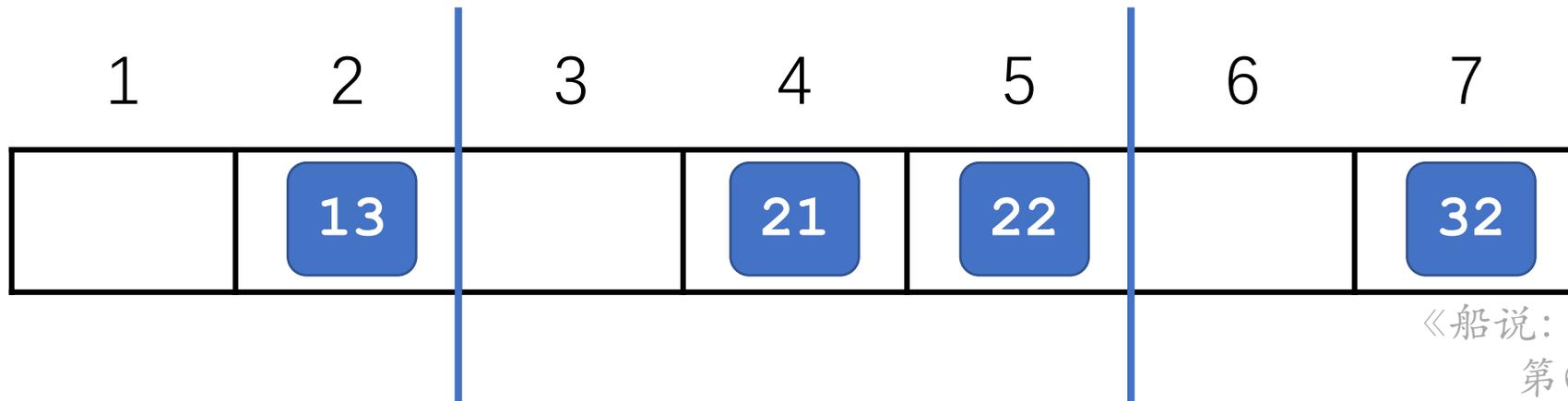
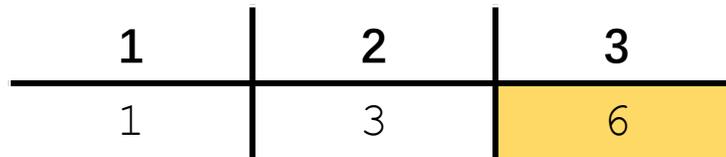
基数排序

归位



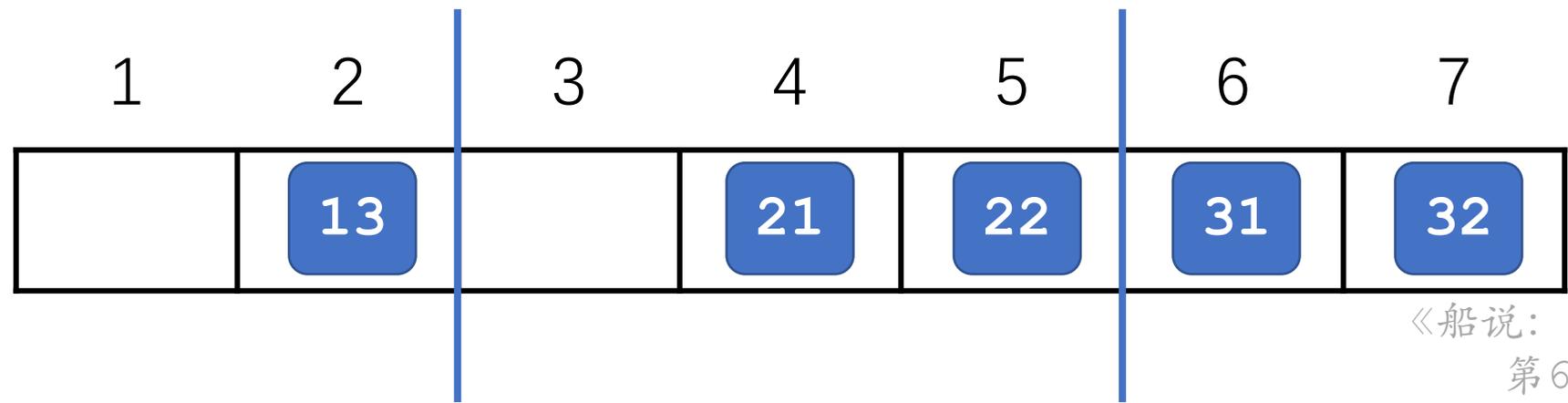
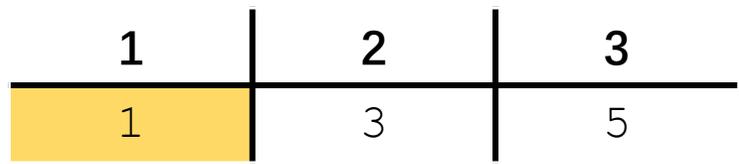
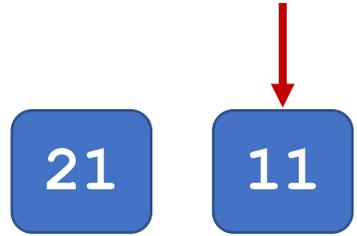
基数排序

归位



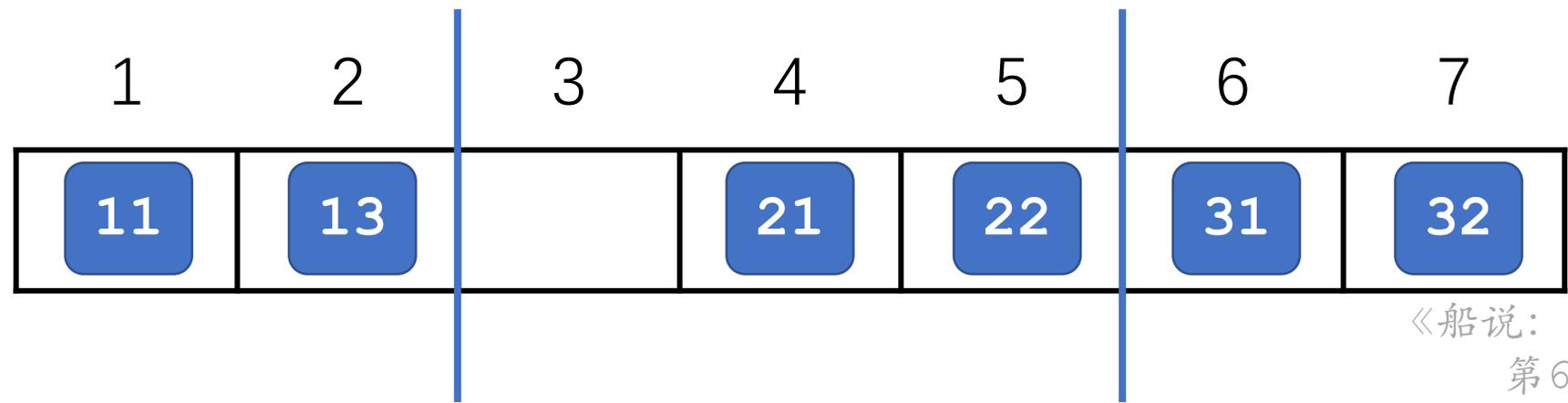
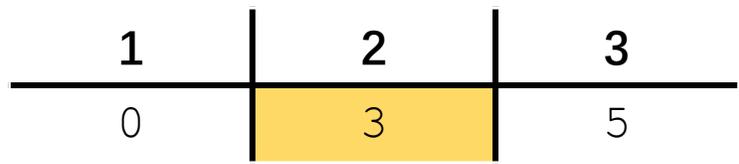
基数排序

归位



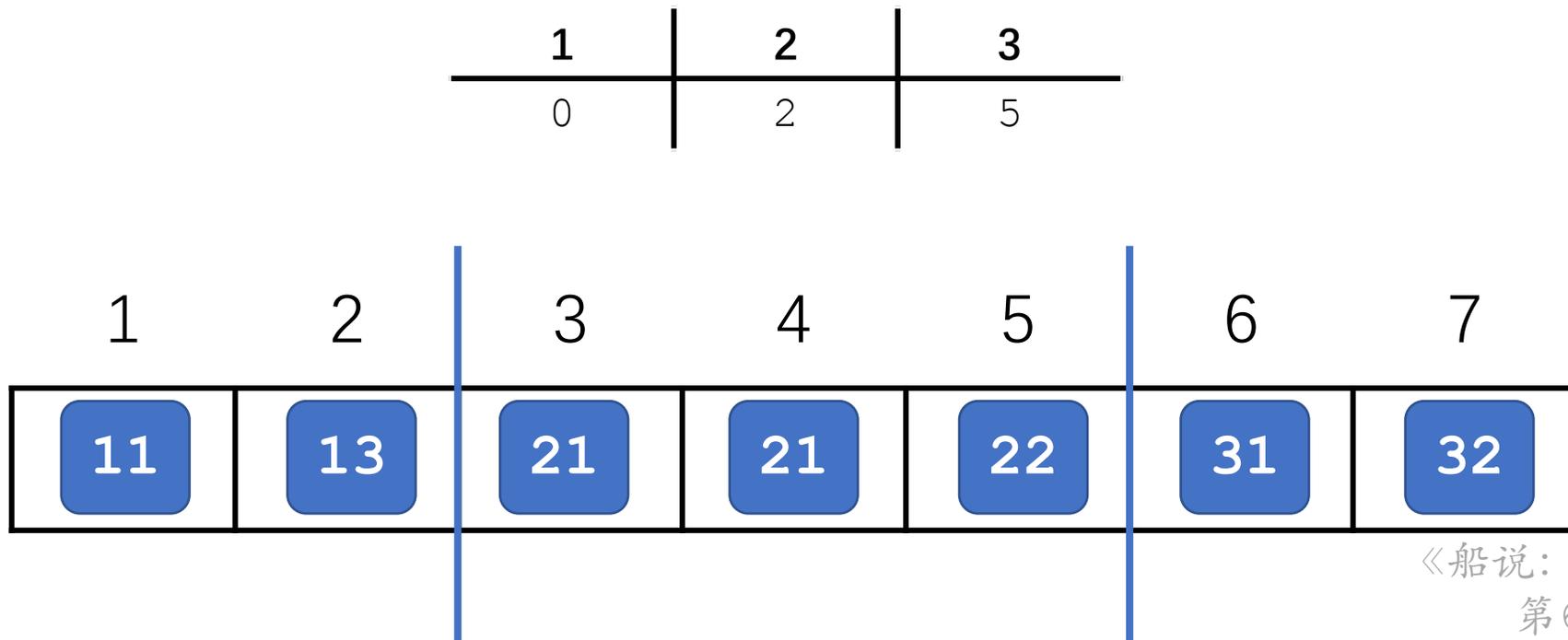
基数排序

归位



基数排序

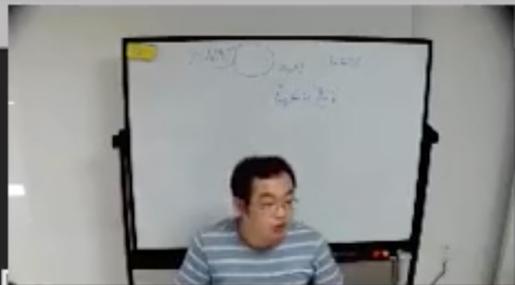
归位



基数排序



```
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
```



基数排序：代码演示

```
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

八. 排序算法总结

稳定排序

插入排序、冒泡排序、归并排序、基数排序

非稳定排序

选择排序、希尔排序、快速排序、堆排序

归并排序在大数据场景下的应用

问题：

电脑内存大小2GB，

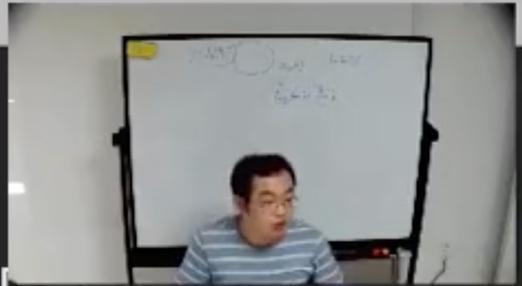
如何对一个 40GB 的文件进行排序？

九. C++ sort 使用方法与技巧

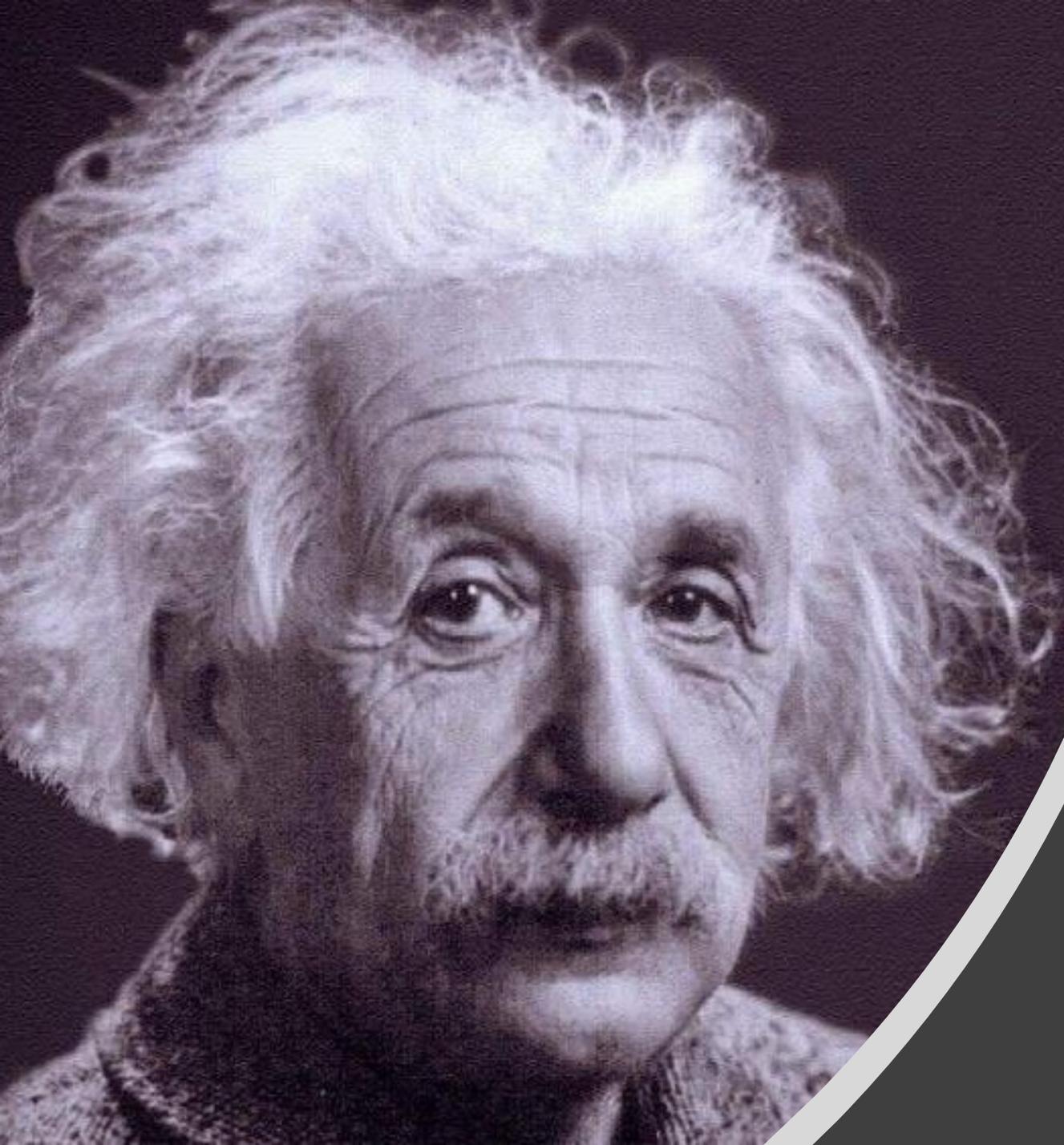
C++ sort 的实现方式

快速排序 + 三点取中法 + 非监督的插入排序 + 堆排序

```
1. vim
vim #1 bash #2 bash #3
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED, {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52     } else {
53         if (!hasRedChild(root->rchild)) return root;
54
55     }
56 }
57
58
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```



C++ sort 使用：代码演示



为什么
会出一样的题目？