力扣 LeetCode × 拉勾 LAGOU.COM

第三课

强化面试中常用的算法 - 排序

拉勾

1.0 回顾 / Review

常用的数据结构

复杂数据结构

它们是学好算法的基石，只有把它们的性质牢牢掌握了，才能在接下来的课程里游刃有余。

2.0 课程内容 / Course Content

力扣 LeetCode × 拉勾 LAGOU.COM

工作中的常用算法

面试中的常用算法

达到巩固知识的效果

通过分析经典的例题

学算法的好处

学习算法的过程其实是一个提高思维能力的过程。

如何应对算法面试

遇到新问题不要慌张，先想出最直观的解法；
直观解法大多数情况下不是最优的，却可以帮你打通思路；
理清解决问题的各个步骤后，根据问题核心来优化某些步骤即可。

后续课程安排

接下来的5节课里将一一讲解面试中最常考的也最核心的算法介绍，本课时分享排序算法。

拉勾

3.0 排序算法 / Sort

力扣 LeetCode × 拉勾 LAGOU.COM

基本的排序算法【简单直接助你迅速写出没有bug的代码】

- 冒泡排序 / Bubble Sort
- 插入排序 / Insertion Sort

常考的排序算法【解决绝大部分涉及排序问题的关键】

- 归并排序 / Merge Sort
- 快速排序 / Quick Sort
- 拓扑排序 / Topological Sort

其他排序算法【掌握好它的解题思想能开阔解题思路】

- 堆排序 / Heap Sort
- 桶排序 / Bucket Sort

拉勾

3.1 冒泡排序 / Bubble Sort

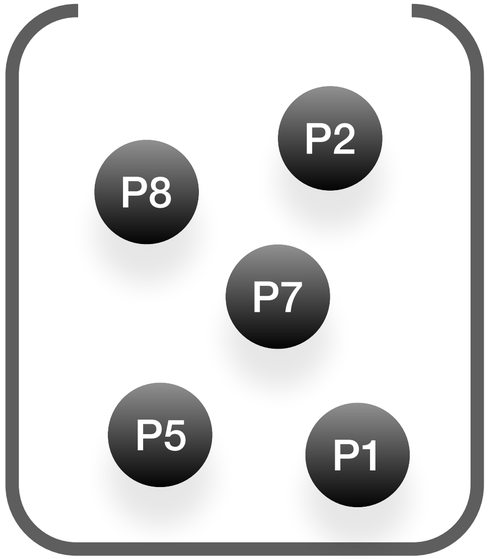
力扣 LeetCode × 拉勾 LAGOU.COM

冒泡排序的算法思想

每一轮，从杂乱无章的数组头部开始，每两个元素比较大小并进行交换；

直到这一轮当中最大或最小的元素被放置在数组的尾部；

然后，不断地重复这个过程，直到所有元素都排好位置。



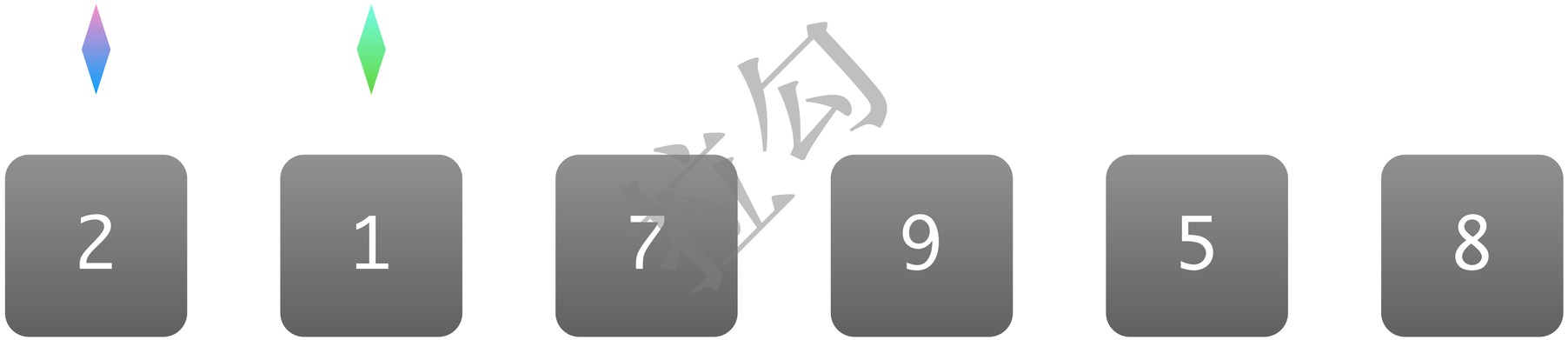
3.1 冒泡排序例题分析 / Example

力扣 LeetCode × 拉勾 LAGOU.COM

拉勾

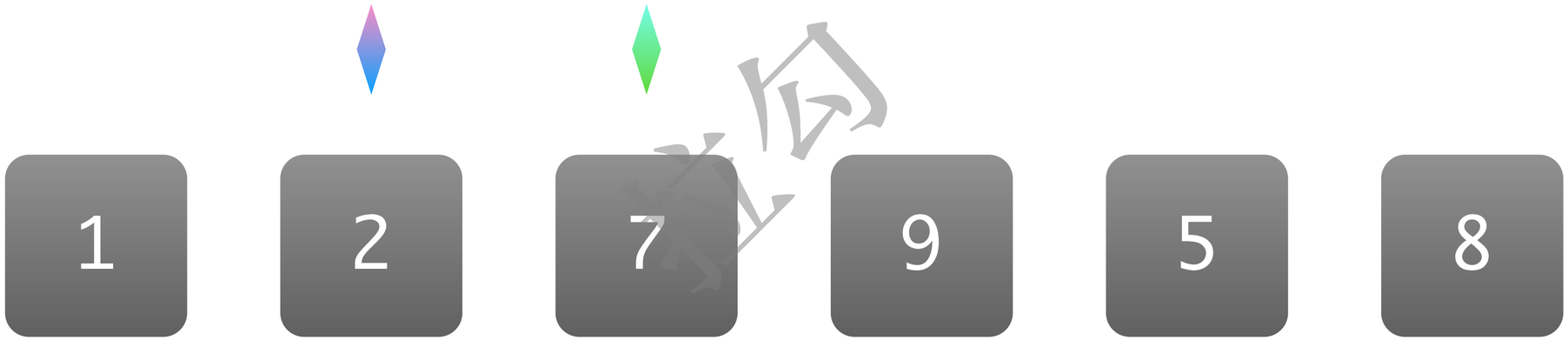
给定数组 [2, 1, 7, 9, 5, 8]，要求按照从左到右、从小到大的顺序进行排序。

3.1 冒泡排序例题分析 / Example



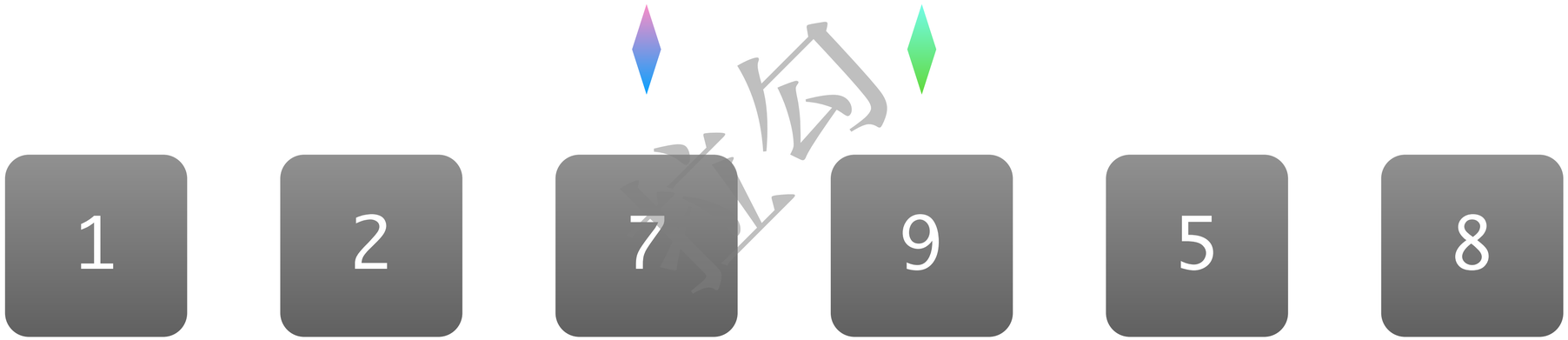
3.1 冒泡排序例题分析 / Example

3.1 冒泡排序例题分析 / Example



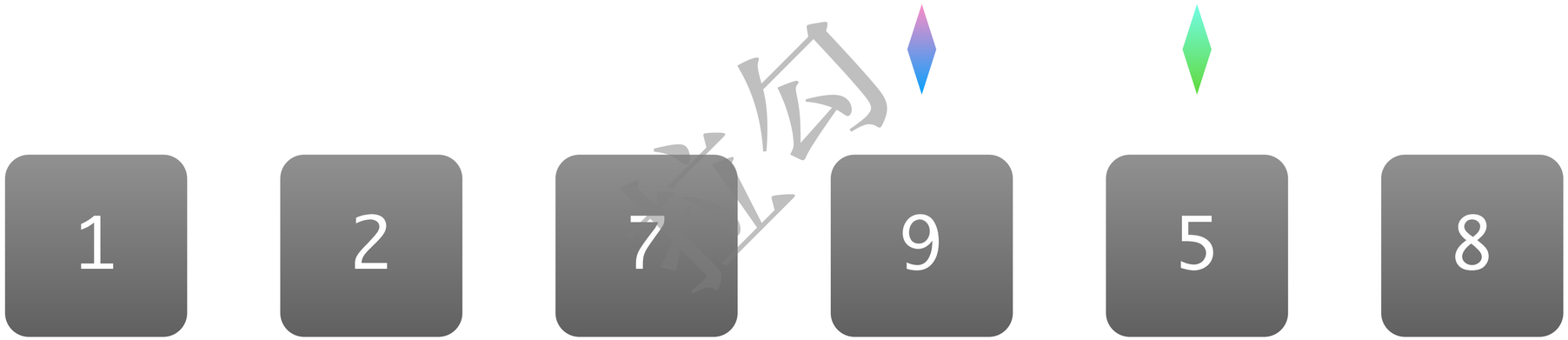
3.1 冒泡排序例题分析 / Example

3.1 冒泡排序例题分析 / Example



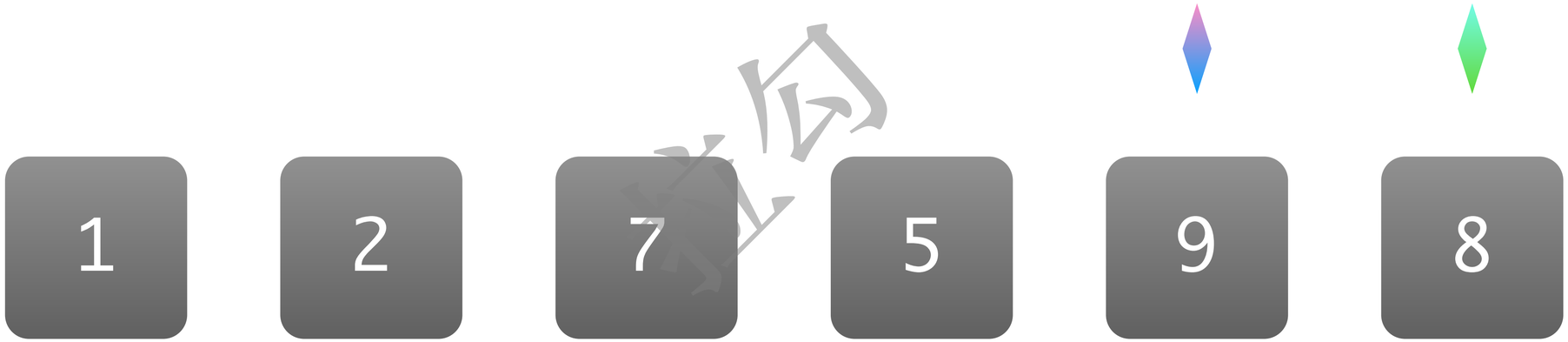
3.1 冒泡排序例题分析 / Example

3.1 冒泡排序例题分析 / Example



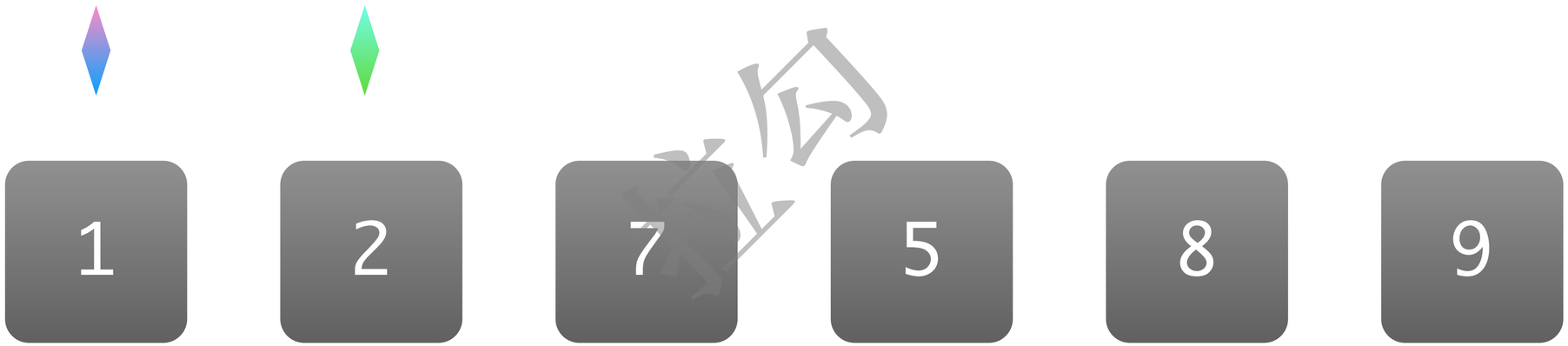
3.1 冒泡排序例题分析 / Example

3.1 冒泡排序例题分析 / Example

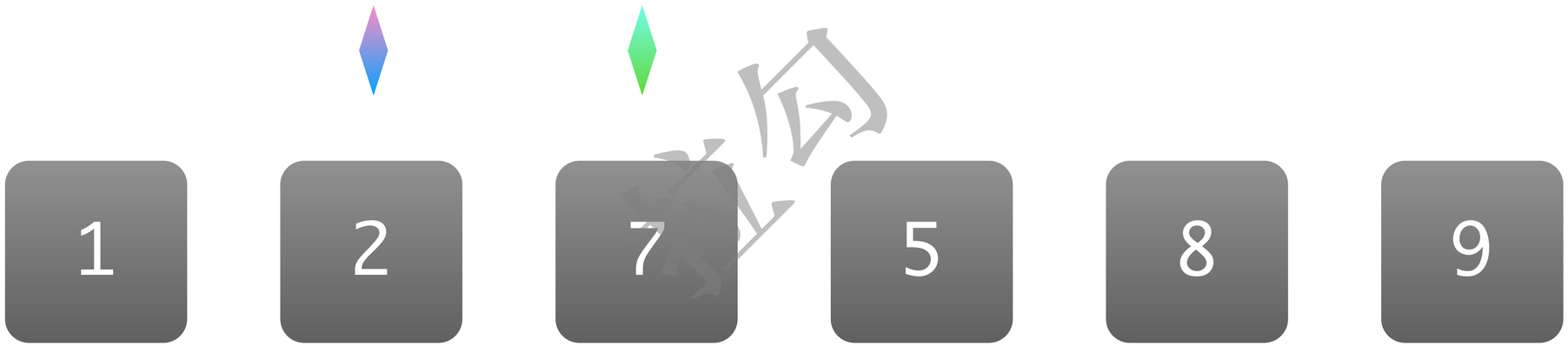


3.1 冒泡排序例题分析 / Example

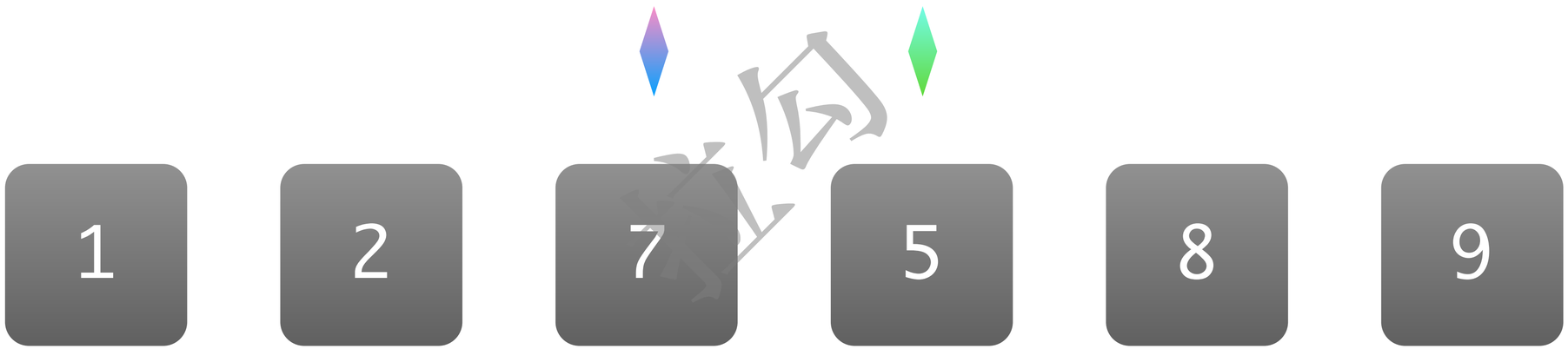
3.1 冒泡排序例题分析 / Example



3.1 冒泡排序例题分析 / Example



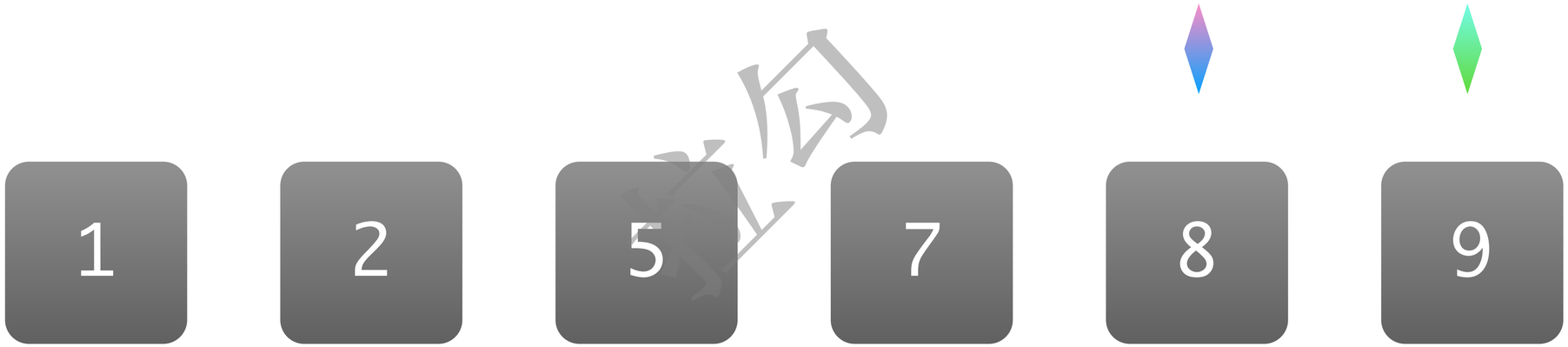
3.1 冒泡排序例题分析 / Example



3.1 冒泡排序例题分析 / Example

3.1 冒泡排序例题分析 / Example

3.1 冒泡排序例题分析 / Example



3.1 冒泡排序例题分析 / Example

3.1 冒泡排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  boolean hasChange = true;

  for (int i = 0; i < nums.length - 1 && hasChange; i++) {
    hasChange = false;

    for (int j = 0; j < nums.length - 1 - i; j++) {
      if (nums[j] > nums[j + 1]) {
        swap(nums, j, j + 1);
        hasChange = true;
      }
    }
  }
}
```

拉勾

3.1 冒泡排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  boolean hasChange = true;

  for (int i = 0; i < nums.length - 1 && hasChange; i++) {
    hasChange = false;

    for (int j = 0; j < nums.length - 1 - i; j++) {
      if (nums[j] > nums[j + 1]) {
        swap(nums, j, j + 1);
        hasChange = true;
      }
    }
  }
}
```

拉勾

3.1 冒泡排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  boolean hasChange = true;

  for (int i = 0; i < nums.length - 1 && hasChange; i++) {
    hasChange = false;

    for (int j = 0; j < nums.length - 1 - i; j++) {
      if (nums[j] > nums[j + 1]) {
        swap(nums, j, j + 1);
        hasChange = true;
      }
    }
  }
}
```

3.1 冒泡排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  boolean hasChange = true;

  for (int i = 0; i < nums.length - 1 && hasChange; i++) {
    hasChange = false;

    for (int j = 0; j < nums.length - 1 - i; j++) {
      if (nums[j] > nums[j + 1]) {
        swap(nums, j, j + 1);
        hasChange = true;
      }
    }
  }
}
```

空间复杂度：O(1)

假设数组的元素个数是n，整个排序的过程中，直接在给定的数组里进行元素的两两交换。

时间复杂度：O(n^2)

- **情景一**：给定的数组按照顺序已经排好

只需要进行n - 1次的比较，两两交换次数为0，**时间复杂度是O(n)**，这是最好的情况。

- **情景二**：给定的数组按照逆序排列

需要进行n(n - 1) / 2次比较，**时间复杂度是O(n^2)**，这是最坏的情况。

- **情景三**：给定的数组杂乱无章

在这种情况下，**平均时间复杂度是O(n^2)**。

拉勾

3.2 插入排序 / Insertion Sort

力扣 LeetCode × 拉勾 LAGOU.COM

与冒泡排序对比

在冒泡排序中，经过每一轮的排序处理后，数组后端的数是排好序的；

在插入排序中，经过每一轮的排序处理后，数组前端的数都是排好序的。

插入排序的算法思想

不断地将尚未排好序的数插入到已经排好序的部分。

3.2 插入排序例题分析 / Example

力扣 LeetCode × 拉勾 LAGOU.COM

拉勾

对数组 [2, 1, 7, 9, 5, 8] 进行插入排序，通过具体的操作来更好地理解它的算法思想。

3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example

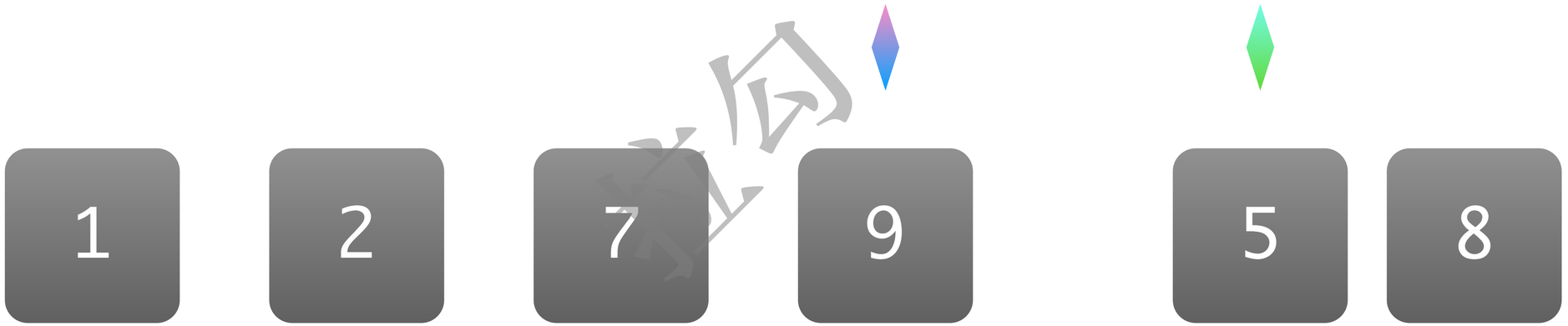
3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example

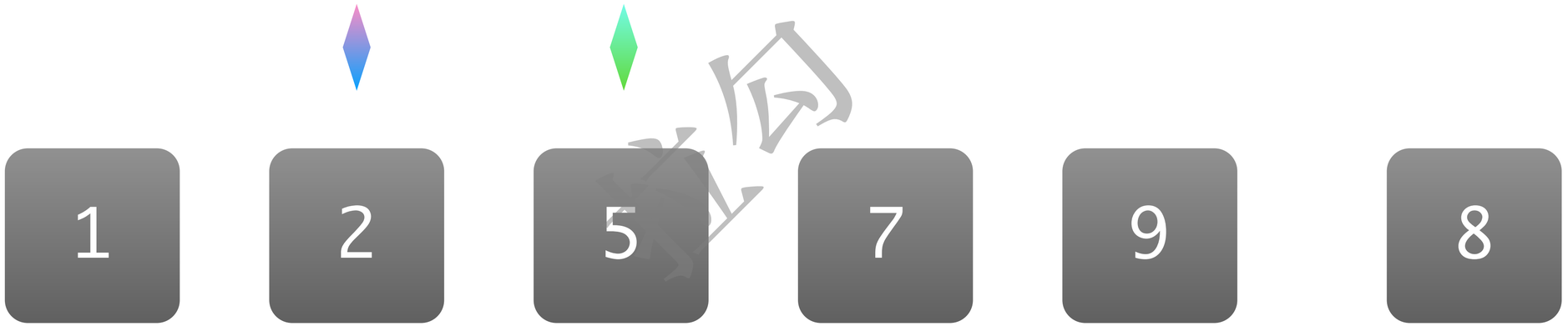


3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example

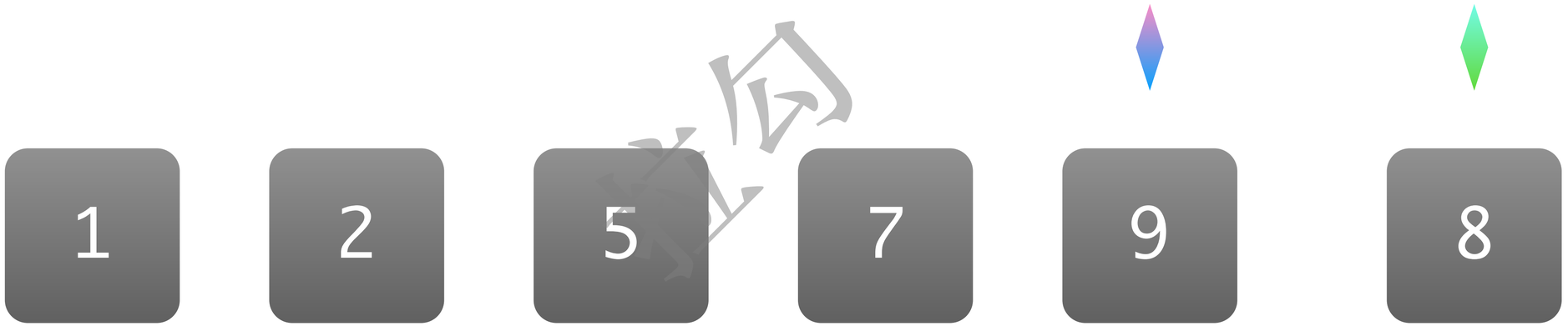
3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example



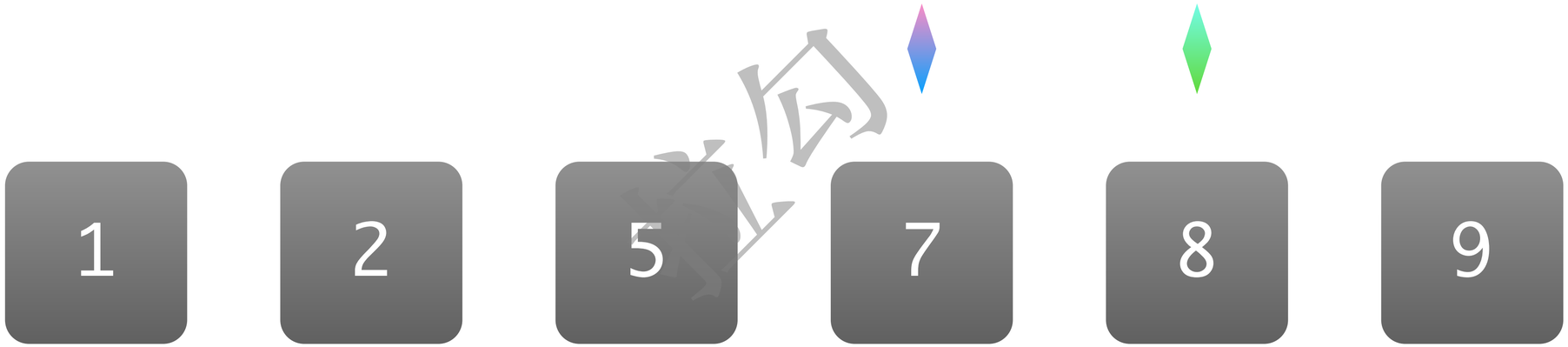
3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example



3.2 插入排序例题分析 / Example

3.2 插入排序例题分析 / Example



3.2 插入排序例题分析 / Example

3.2 插入排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  for (int i = 1, j, current; i < nums.length; i++) {
    current = nums[i];

    for (j = i - 1; j >= 0 && nums[j] > current; j--) {
      nums[j + 1] = nums[j];
    }

    nums[j + 1] = current;
  }
}
```

拉勾

3.2 插入排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  for (int i = 1, j, current; i < nums.length; i++) {
    current = nums[i];

    for (j = i - 1; j >= 0 && nums[j] > current; j--) {
      nums[j + 1] = nums[j];
    }

    nums[j + 1] = current;
  }
}
```

拉勾

3.2 插入排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  for (int i = 1, j, current; i < nums.length; i++) {
    current = nums[i];

    for (j = i - 1; j >= 0 && nums[j] > current; j--) {
      nums[j + 1] = nums[j];
    }

    nums[j + 1] = current;
  }
}
```

拉勾

3.2 插入排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  for (int i = 1, j, current; i < nums.length; i++) {
    current = nums[i];

    for (j = i - 1; j >= 0 && nums[j] > current; j--) {
      nums[j + 1] = nums[j];
    }

    nums[j + 1] = current;
  }
}
```

3.2 插入排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort(int[] nums) {
  for (int i = 1, j, current; i < nums.length; i++) {
    current = nums[i];

    for (j = i - 1; j >= 0 && nums[j] > current; j--) {
      nums[j + 1] = nums[j];
    }

    nums[j + 1] = current;
  }
}
```

拉勾

空间复杂度：O(1)

假设数组的元素个数是n，整个排序的过程中，直接在给定的数组里进行元素的两两交换。

时间复杂度：O(n^2)

- **情景一**：给定的数组按照顺序已经排好

只需要进行n - 1次的比较，两两交换次数为0，**时间复杂度是O(n)**，这是最好的情况。

- **情景二**：给定的数组按照逆序排列

需要进行n(n - 1) / 2次比较，**时间复杂度是O(n^2)**，这是最坏的情况。

- **情景三**：给定的数组杂乱无章

在这种情况下，**平均时间复杂度是O(n^2)**。

拉勾

3.3 归并排序 / Merge Sort

力扣 LeetCode × 拉勾 LAGOU.COM

分治的思想

归并排序的核心思想是分治，把一个复杂问题拆分成若干个子问题来求解。

拉勾

归并排序的算法思想

把数组从中间划分成两个子数组；

一直递归地把子数组划分成更小的子数组，直到子数组里面只有一个元素；

依次按照递归的返回顺序，不断地合并排好序的子数组，直到最后把整个数组的顺序排好。

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 归并排序的主体函数 */

void sort(int[] A, int lo, int hi) {
  if (lo >= hi) return;

  int mid = lo + (hi - lo) / 2;

  sort(A, lo, mid);
  sort(A, mid + 1, hi);

  merge(A, lo, mid, hi);
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 归并排序的主体函数 */

void sort(int[] A, int lo, int hi) {
  if (lo >= hi) return;

  int mid = lo + (hi - lo) / 2;

  sort(A, lo, mid);
  sort(A, mid + 1, hi);

  merge(A, lo, mid, hi);
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 归并排序的主体函数 */

void sort(int[] A, int lo, int hi) {
  if (lo >= hi) return;

  int mid = lo + (hi - lo) / 2;

  sort(A, lo, mid);
  sort(A, mid + 1, hi);

  merge(A, lo, mid, hi);
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 归并排序的主体函数 */

void sort(int[] A, int lo, int hi) {
  if (lo >= hi) return;

  int mid = lo + (hi - lo) / 2;

  sort(A, lo, mid);
  sort(A, mid + 1, hi);

  merge(A, lo, mid, hi);
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 归并排序的主体函数 */

void sort(int[] A, int lo, int hi) {
  if (lo >= hi) return;

  int mid = lo + (hi - lo) / 2;

  sort(A, lo, mid);
  sort(A, mid + 1, hi);

  merge(A, lo, mid, hi);
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

拉勾

3.3 归并排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void merge(int[] nums, int lo, int mid, int hi) {
  int[] copy = nums.clone();

  int k = lo, i = lo, j = mid + 1;

  while (k <= hi) {
    if (i > mid) {
      nums[k++] = copy[j++];
    } else if (j > hi) {
      nums[k++] = copy[i++];
    } else if (copy[j] < copy[i]) {
      nums[k++] = copy[j++];
    } else {
      nums[k++] = copy[i++];
    }
  }
}
```

拉勾

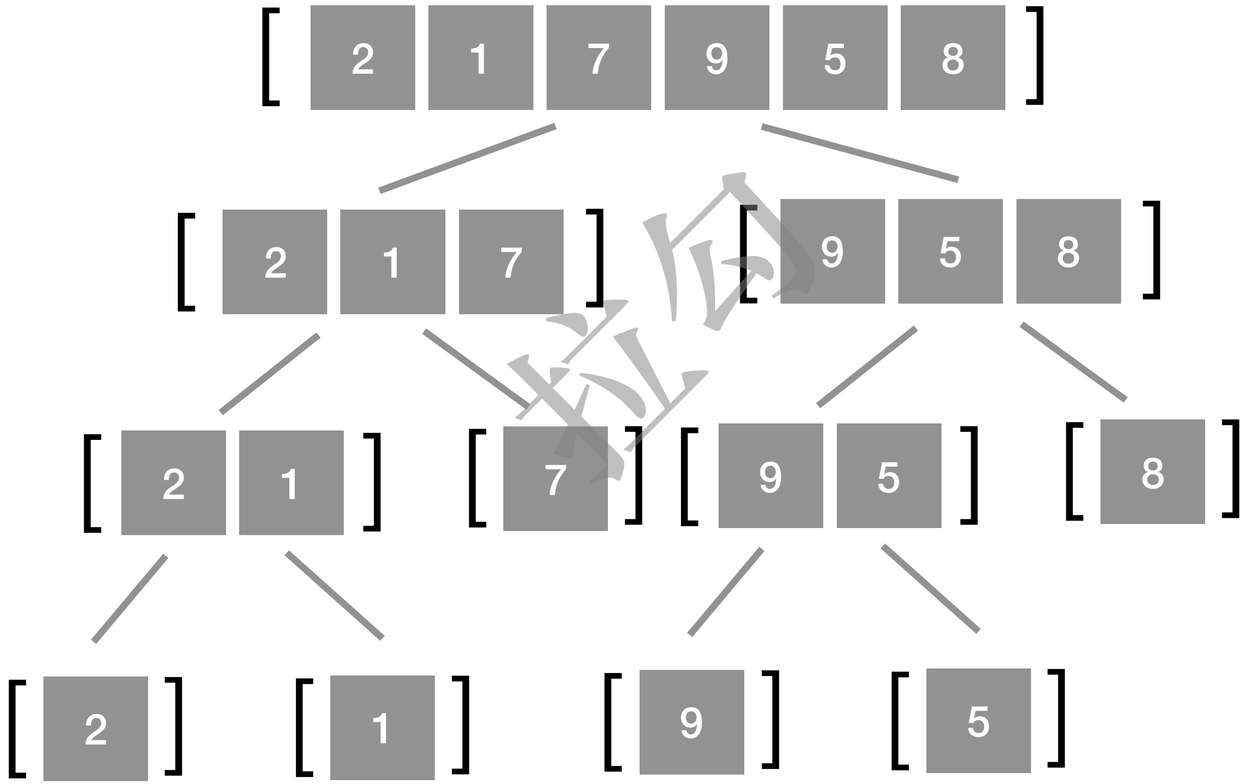
3.3 归并排序例题分析 / Example

力扣 LeetCode × 拉勾 LAGOU.COM

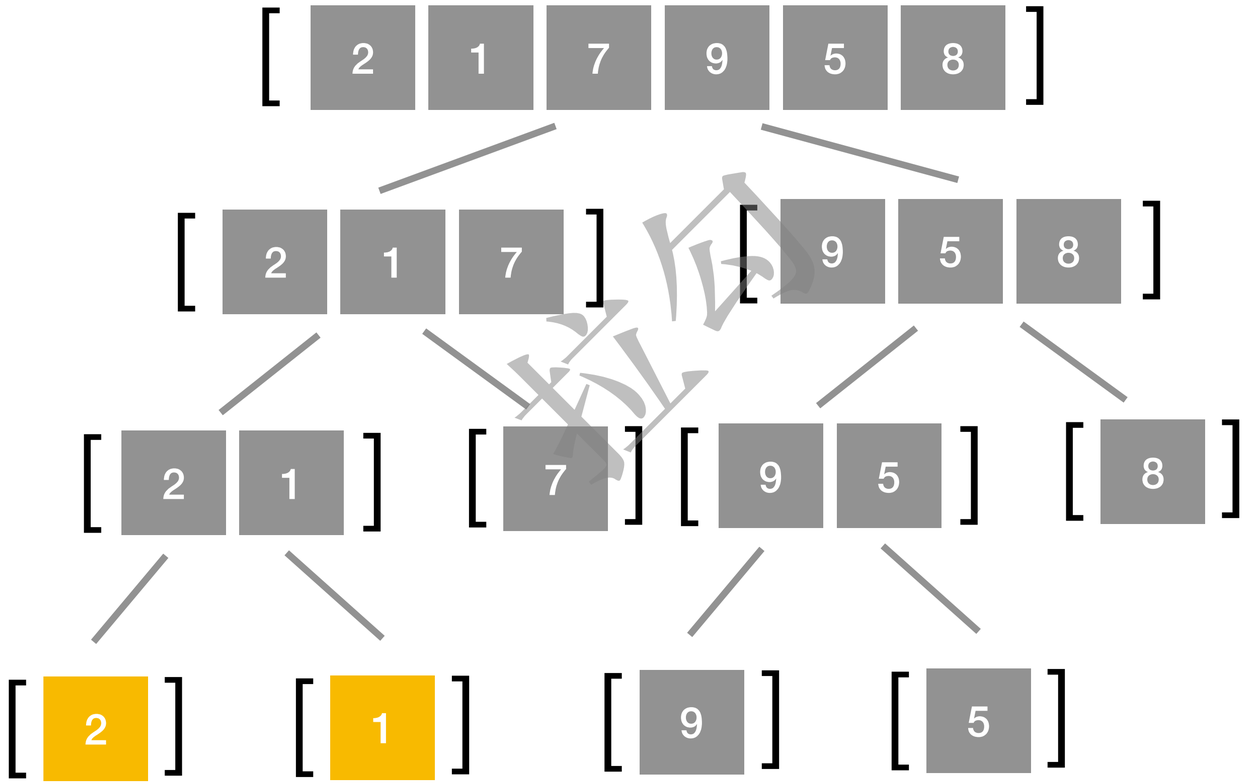
拉勾

如何利用归并排序算法对数组[2, 1, 7, 9, 5, 8]进行排序?

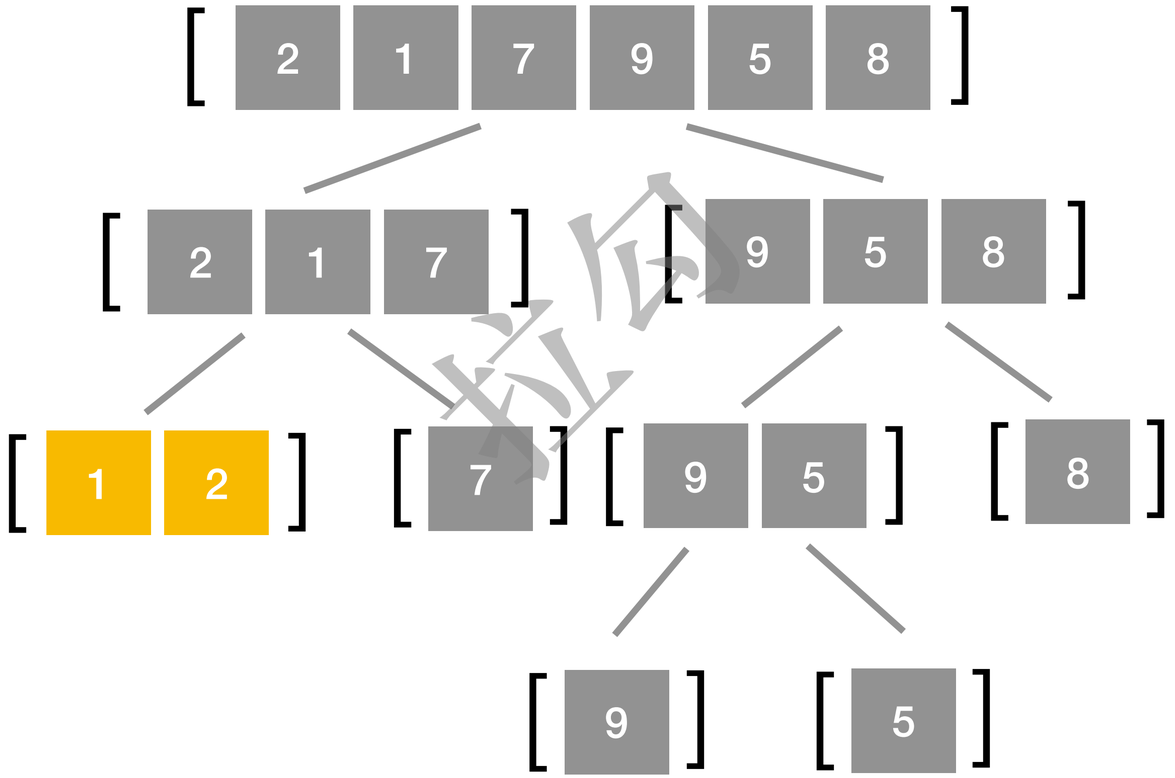
3.3 归并排序例题分析 / Example



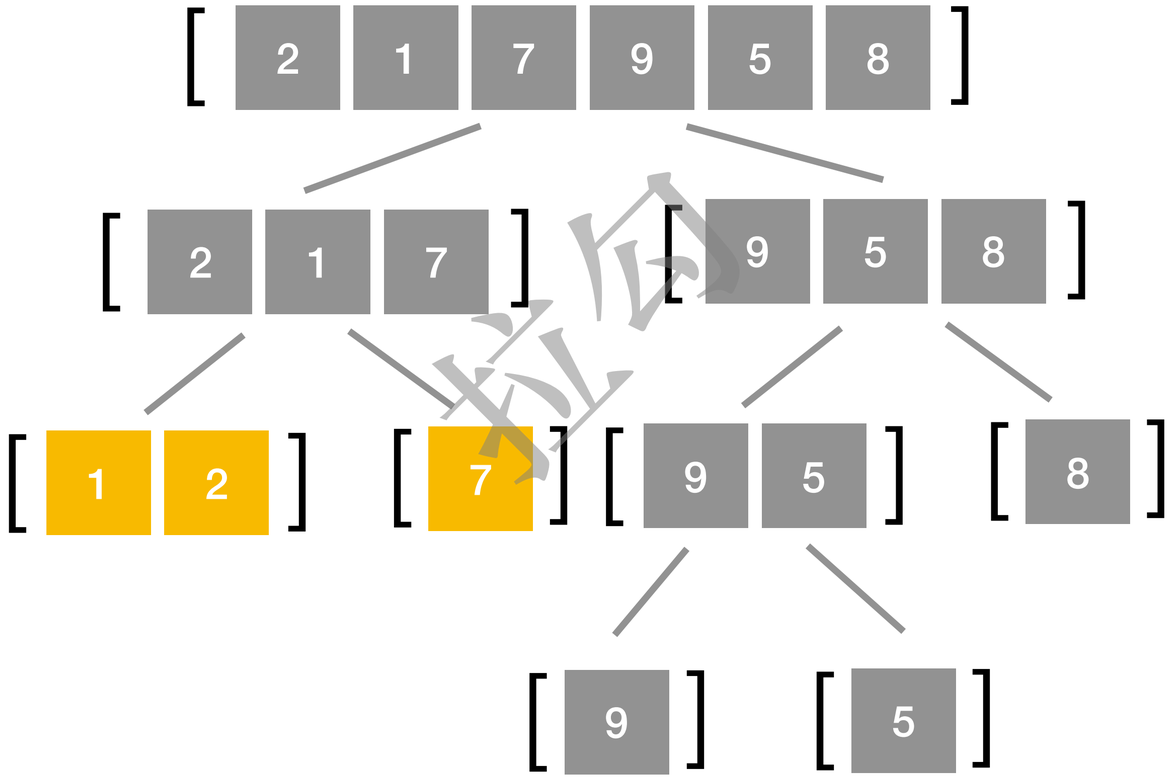
3.3 归并排序例题分析 / Example



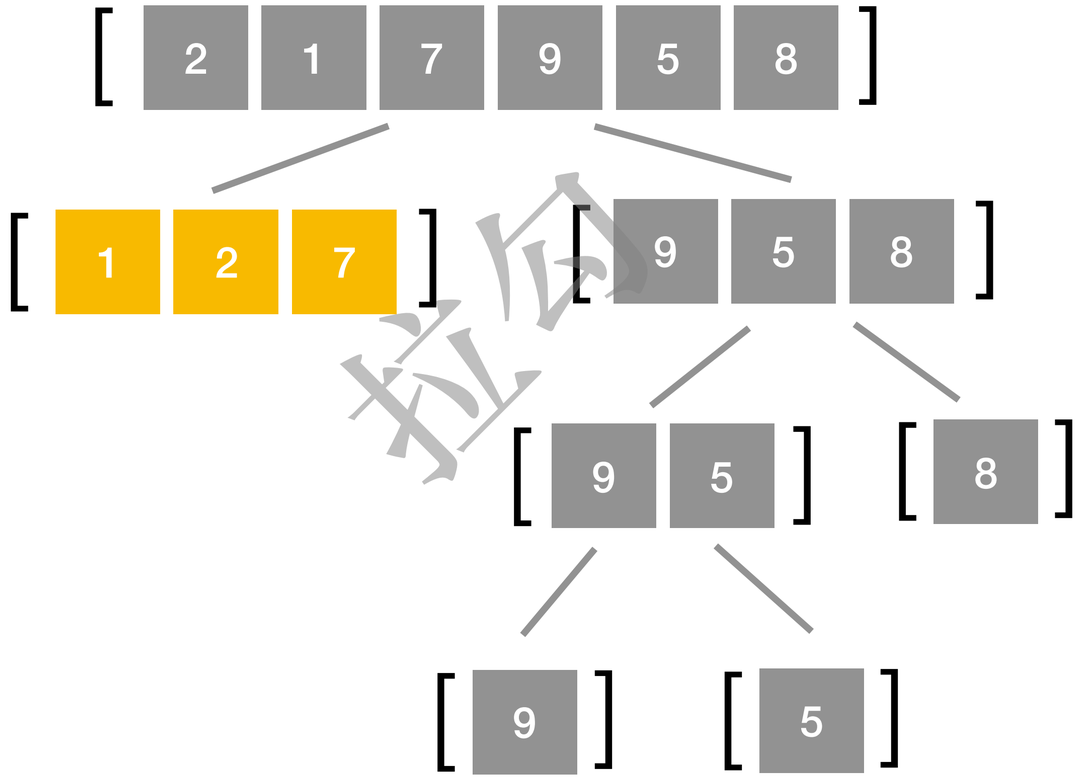
3.3 归并排序例题分析 / Example



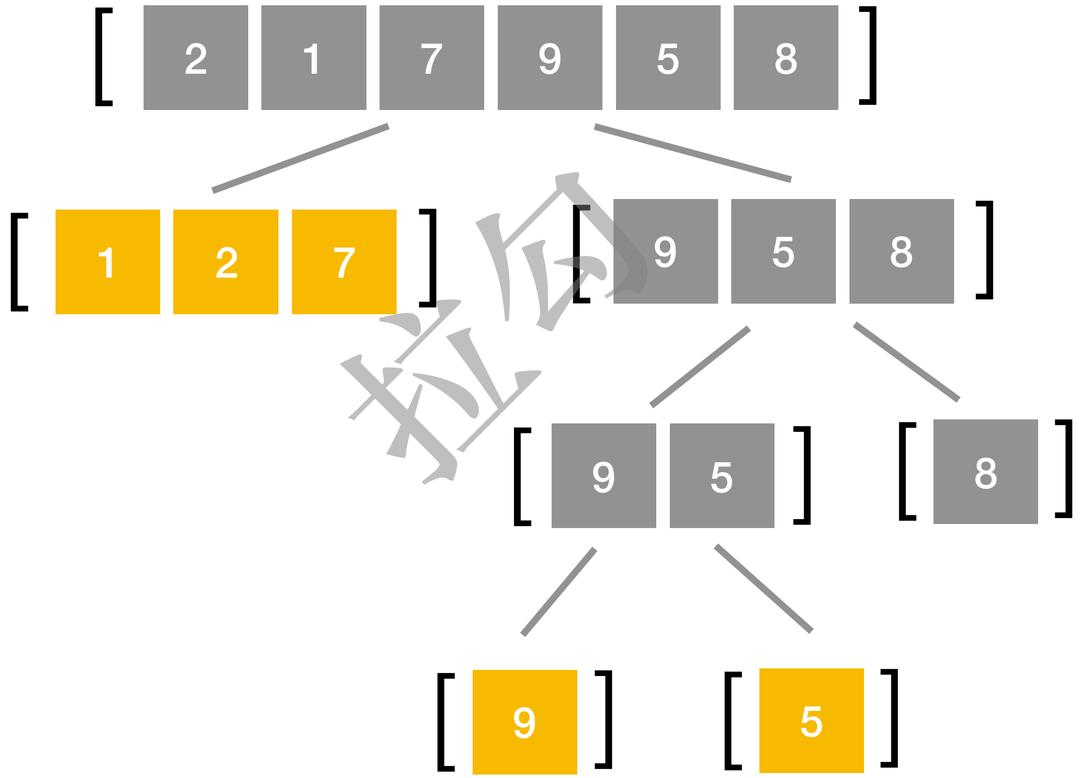
3.3 归并排序例题分析 / Example



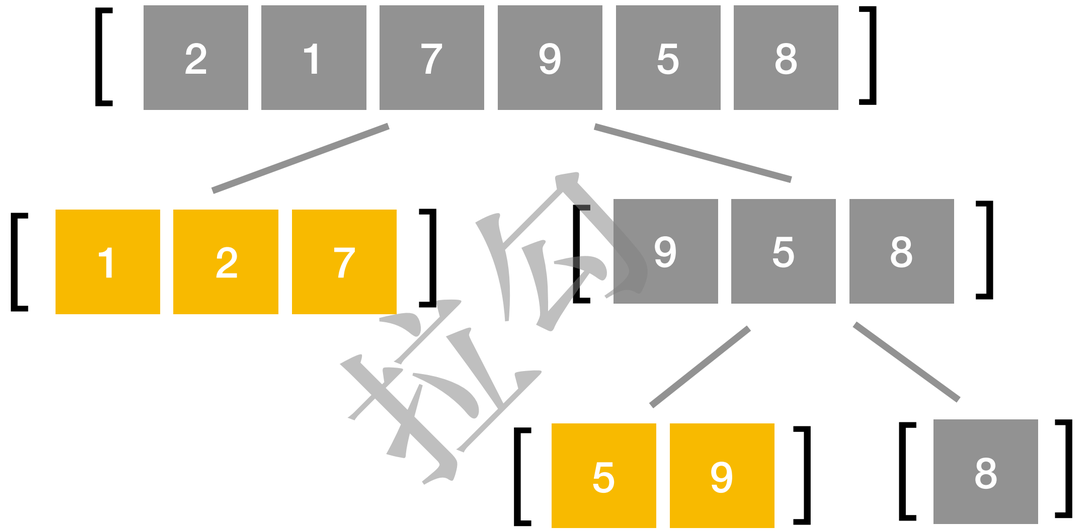
3.3 归并排序例题分析 / Example



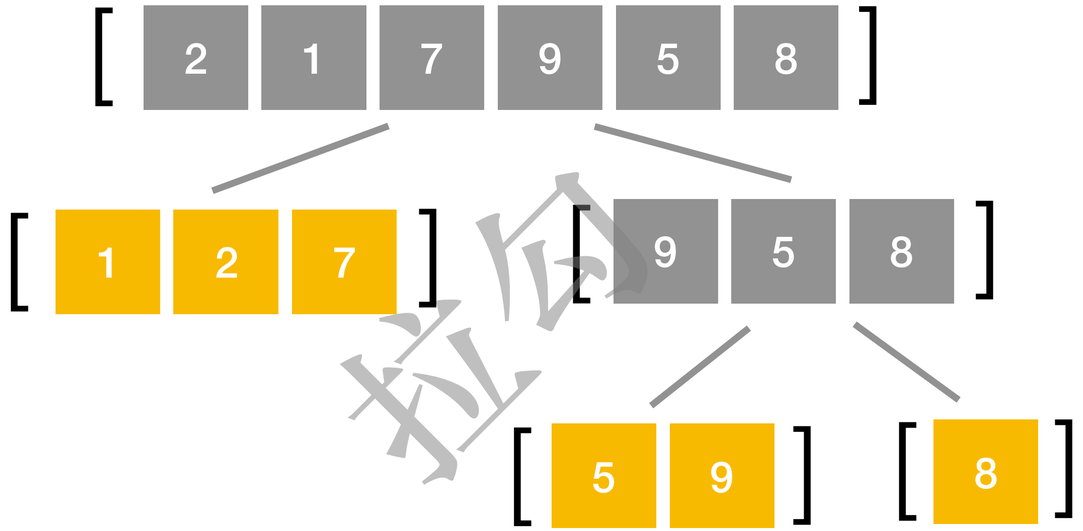
3.3 归并排序例题分析 / Example



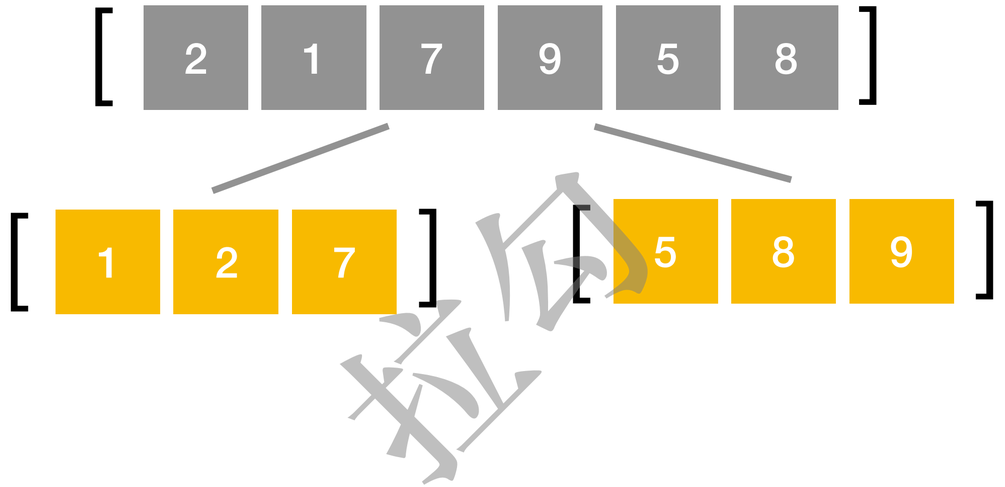
3.3 归并排序例题分析 / Example



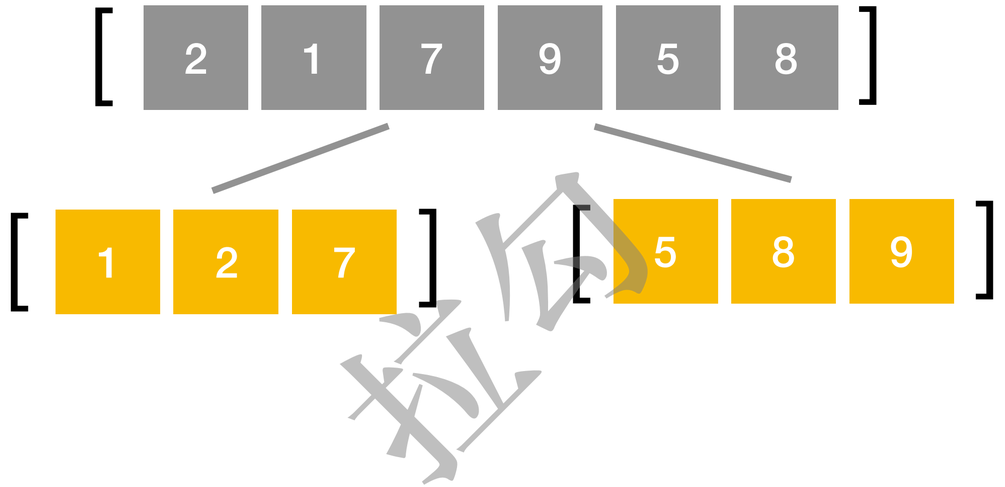
3.3 归并排序例题分析 / Example



3.3 归并排序例题分析 / Example



3.3 归并排序例题分析 / Example



3.3 归并排序例题分析 / Example

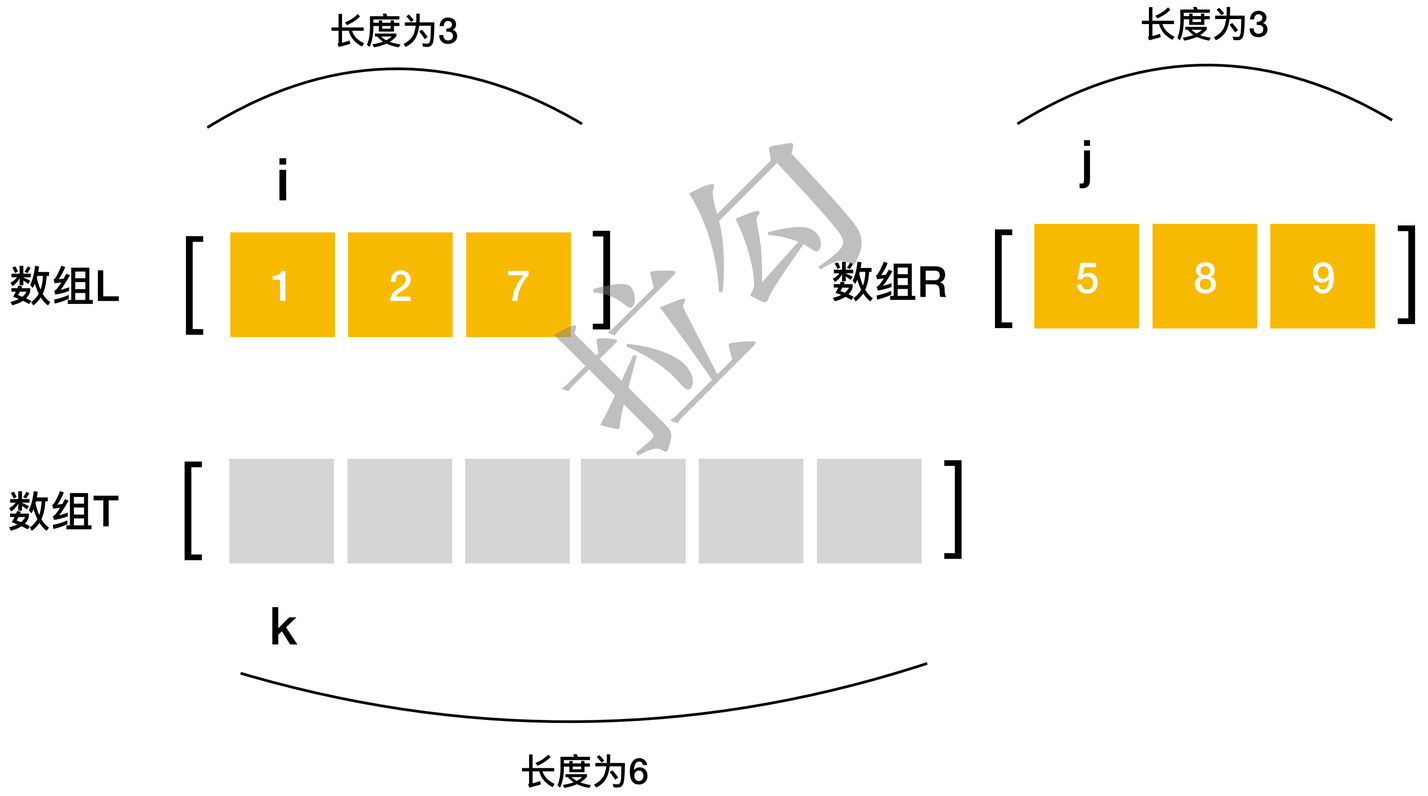
力扣 LeetCode × 拉勾 LAGOU.COM

3.3 归并排序例题分析 / Example

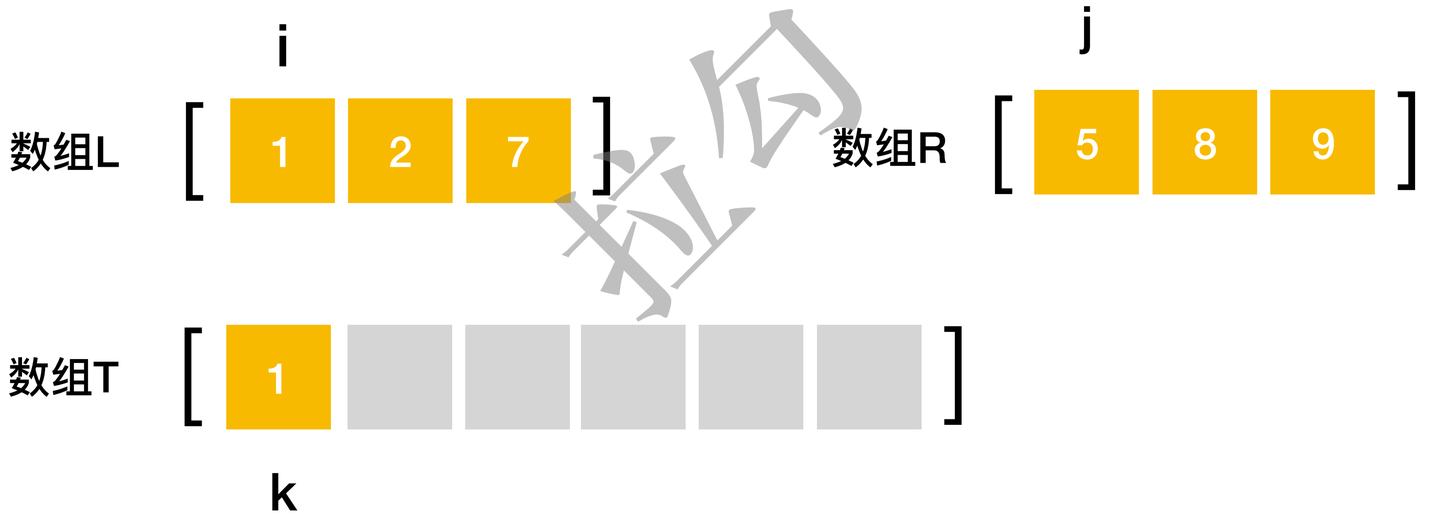
拉勾

合并是如何操作的呢?

3.3 归并排序例题分析 / Example

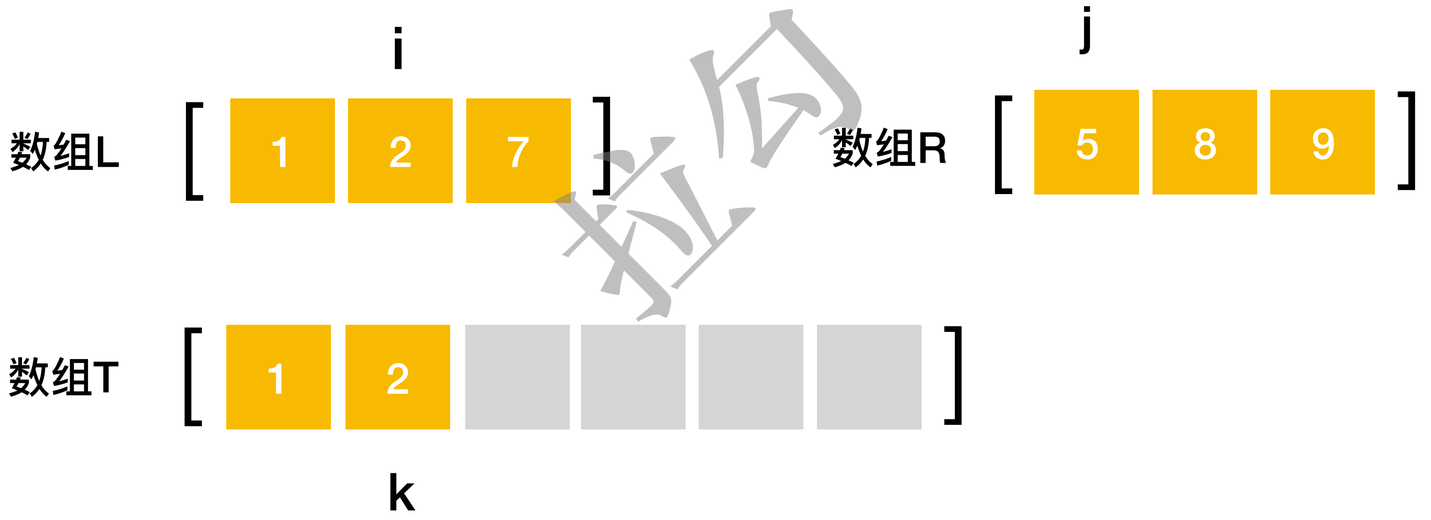


3.3 归并排序例题分析 / Example

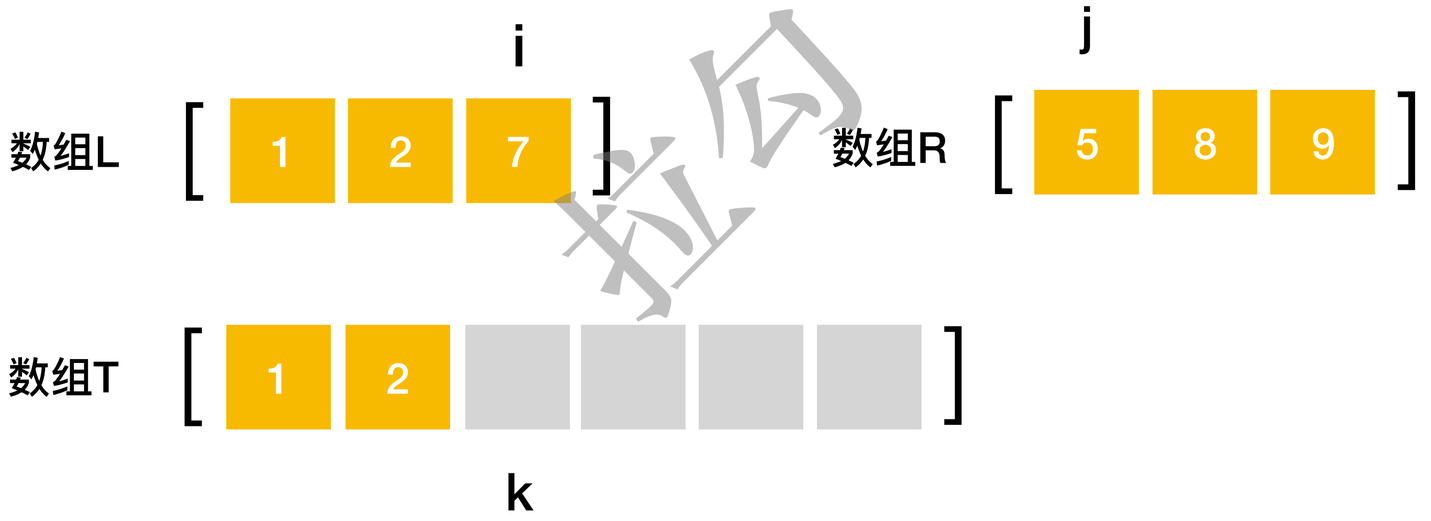


3.3 归并排序例题分析 / Example

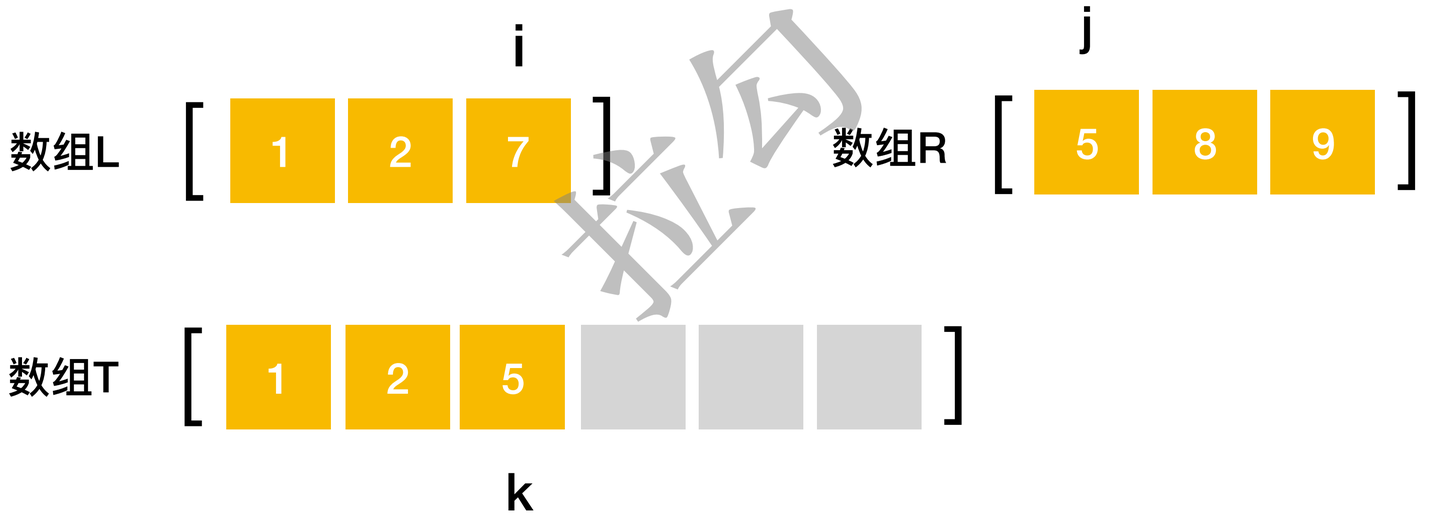
3.3 归并排序例题分析 / Example



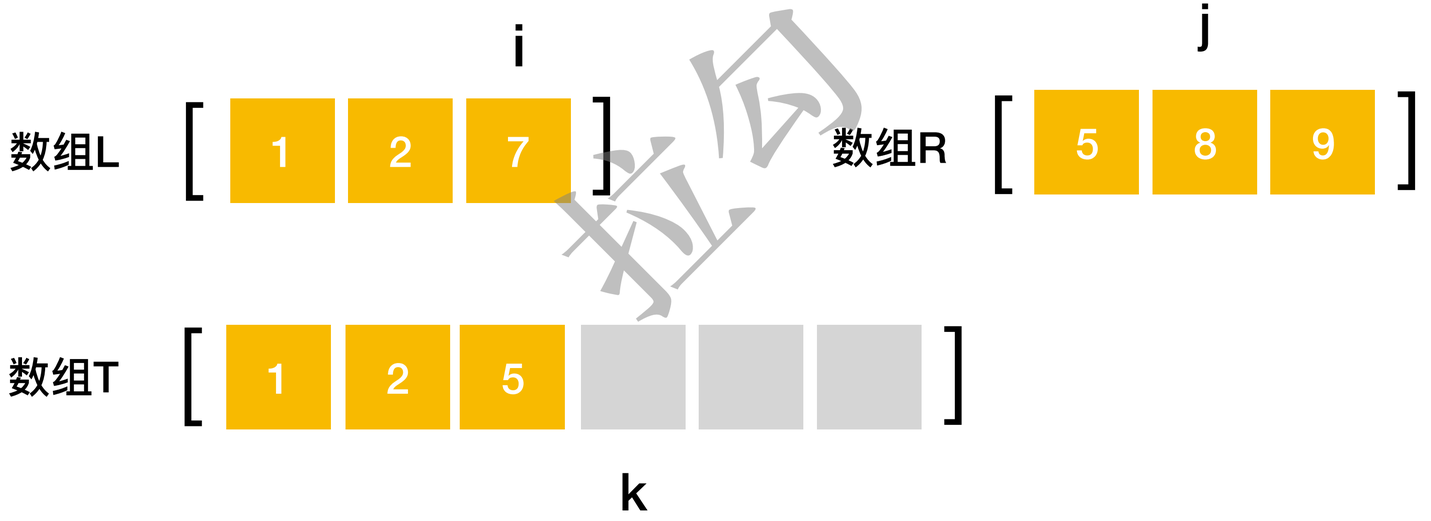
3.3 归并排序例题分析 / Example



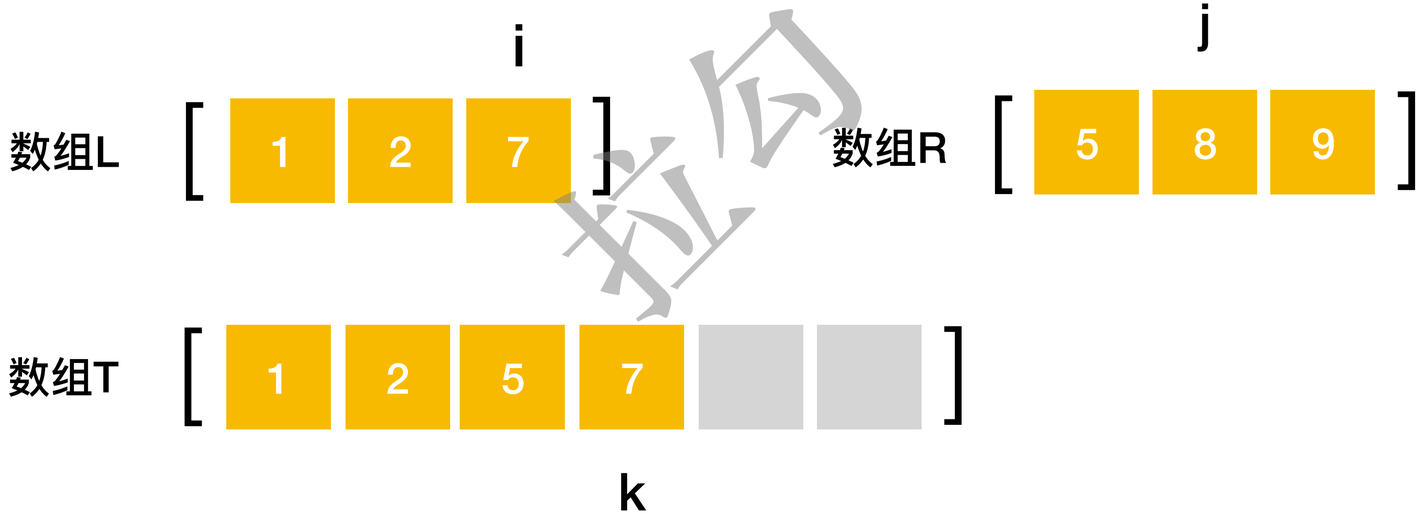
3.3 归并排序例题分析 / Example



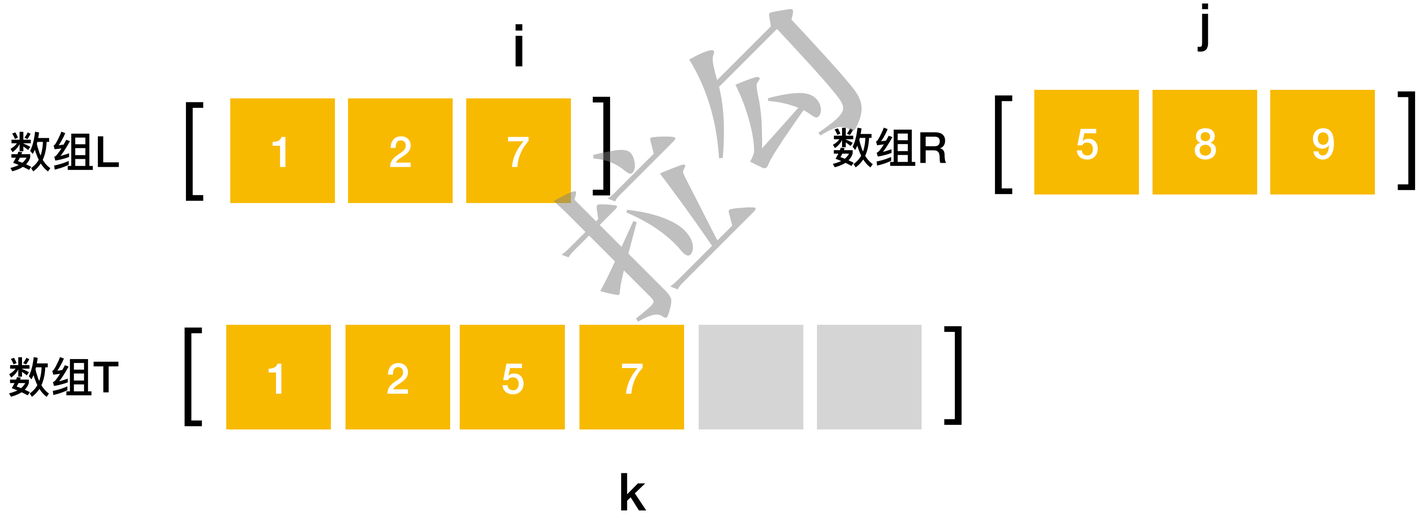
3.3 归并排序例题分析 / Example



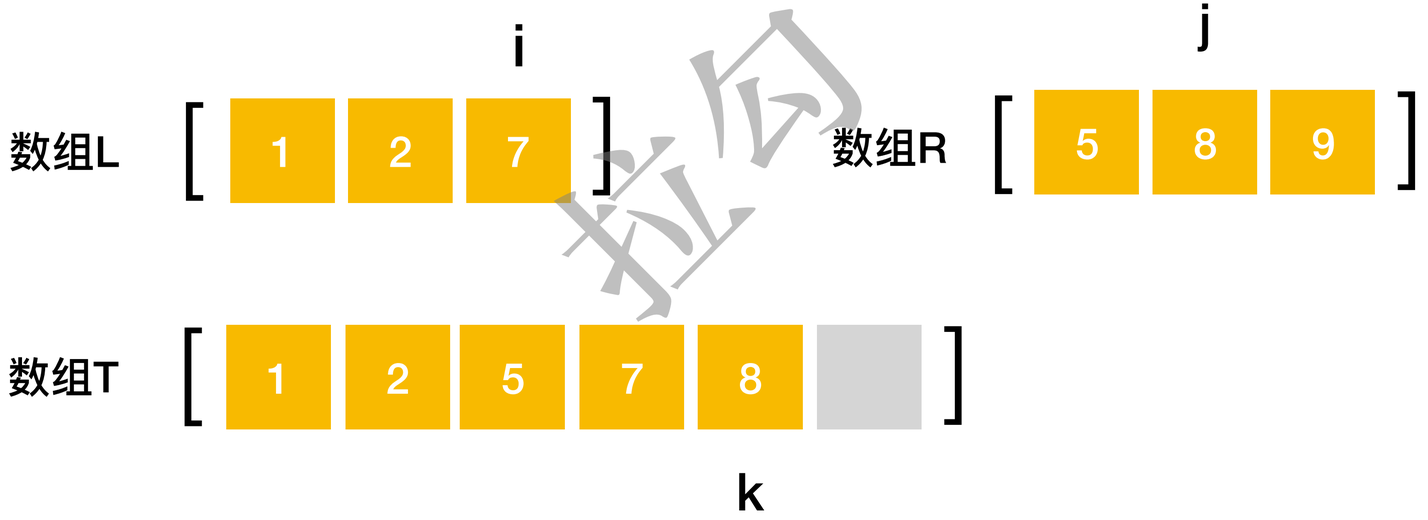
3.3 归并排序例题分析 / Example



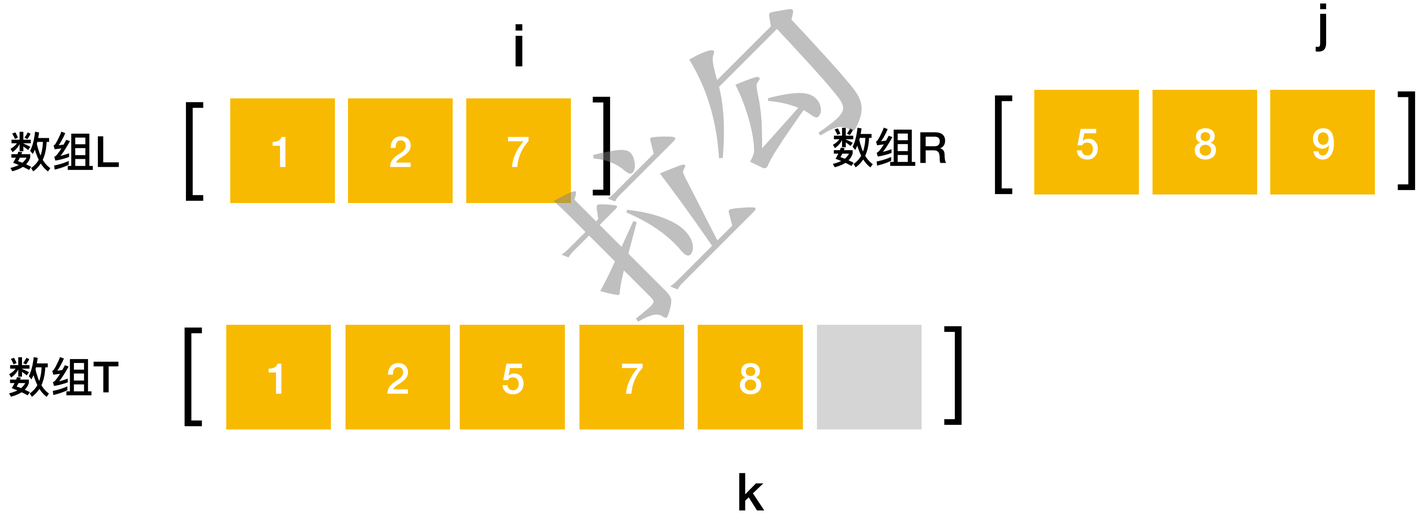
3.3 归并排序例题分析 / Example



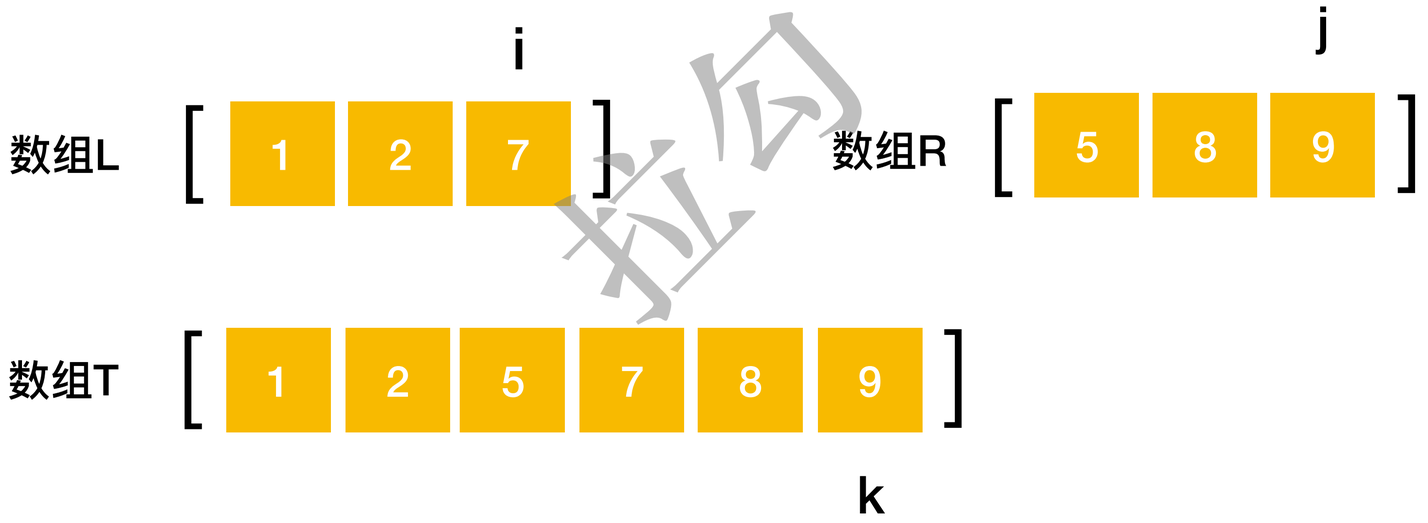
3.3 归并排序例题分析 / Example



3.3 归并排序例题分析 / Example



3.3 归并排序例题分析 / Example



3.3 归并排序算法分析 / Algorithm Analysis

力扣 LeetCode × 拉勾 LAGOU.COM

时间复杂度：T(n)

归并算法是一个不断递归的过程，假设数组的元素个数是n。

时间复杂度是T(n)的函数：T(n) = 2*T(n/2) + O(n)

怎么解这个公式呢?

对于规模为n的问题，一共要进行log(n)层的大小切分；

每一层的合并复杂度都是O(n)；

所以整体的复杂度就是O(nlogn)。

空间复杂度：O(n)

由于合并n个元素需要分配一个大小为n的额外数组，合并完成之后，这个数组的空间就会被释放。

拉勾

3.4 快速排序 / Quick Sort

力扣 LeetCode × 拉勾 LAGOU.COM

快速排序的算法思想

快速排序也采用了分治的思想；

把原始的数组筛选成较小和较大的两个子数组，然后递归地排序两个子数组；

在分成较小和较大的两个子数组过程中，如何选定一个基准值尤为关键。

拉勾

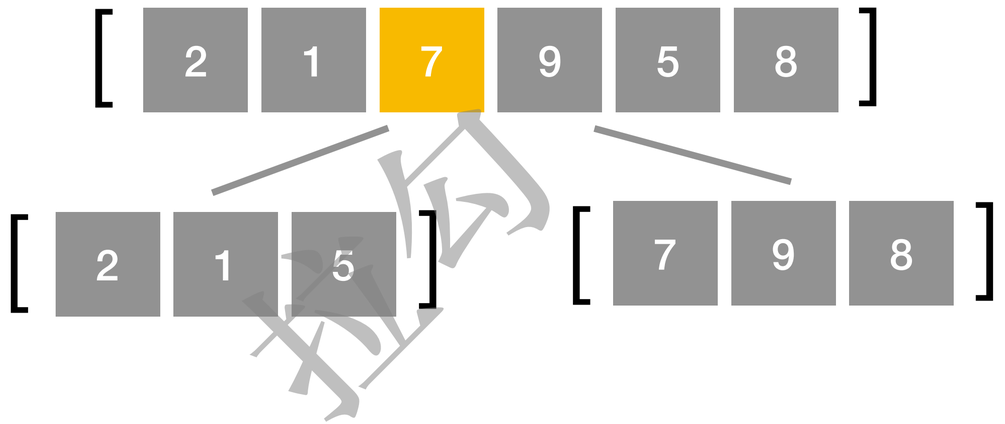
3.4 快速排序例题分析 / Example

力扣 LeetCode × 拉勾 LAGOU.COM

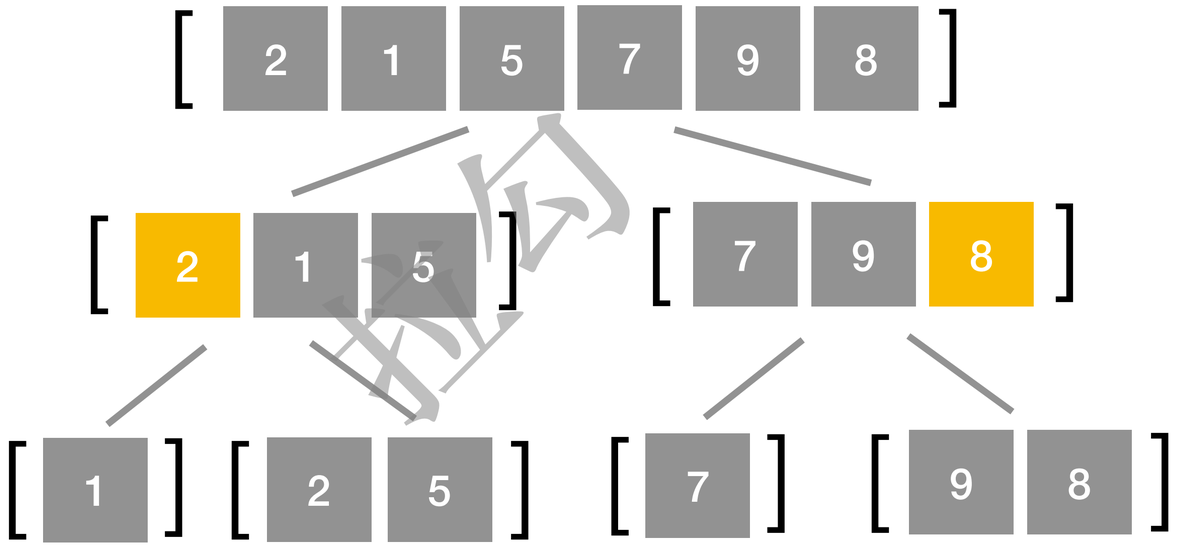
拉勾

如何利用快速排序算法对数组[2, 1, 7, 9, 5, 8]进行排序?

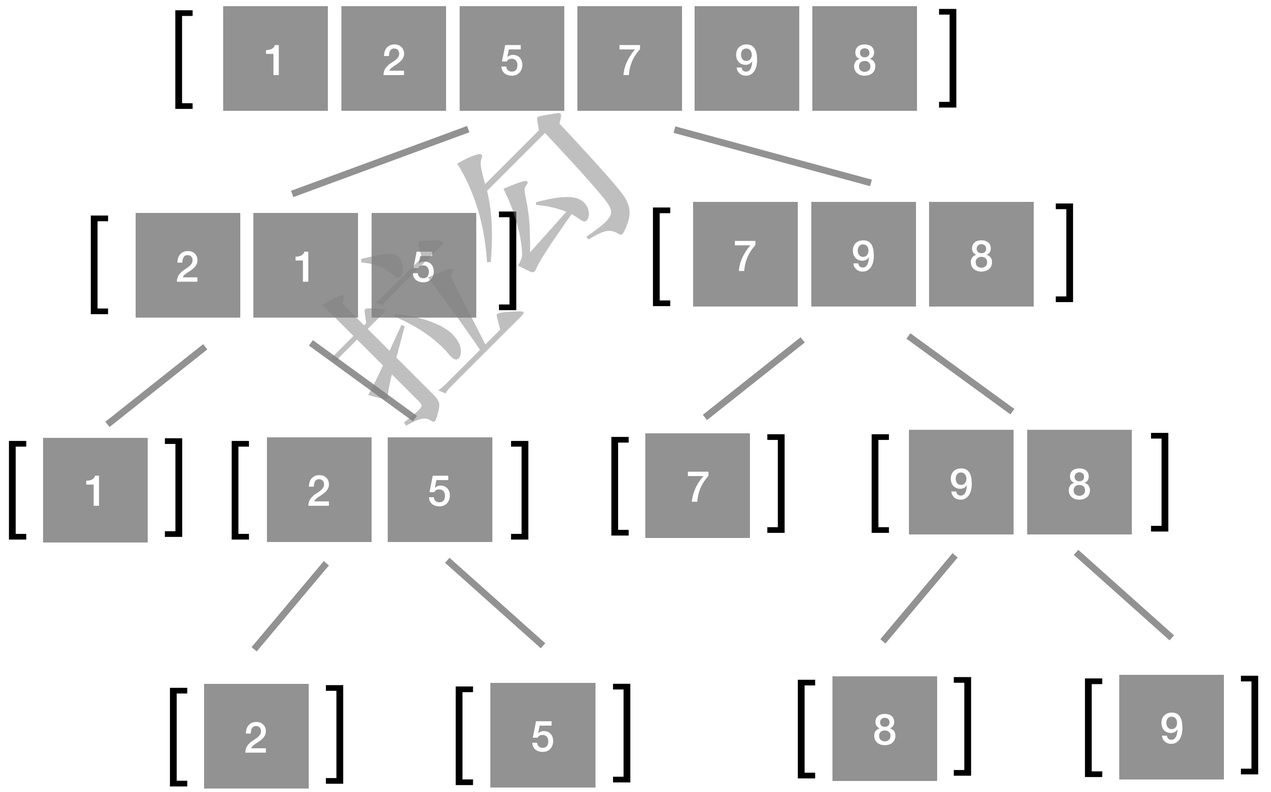
3.4 快速排序例题分析 / Example



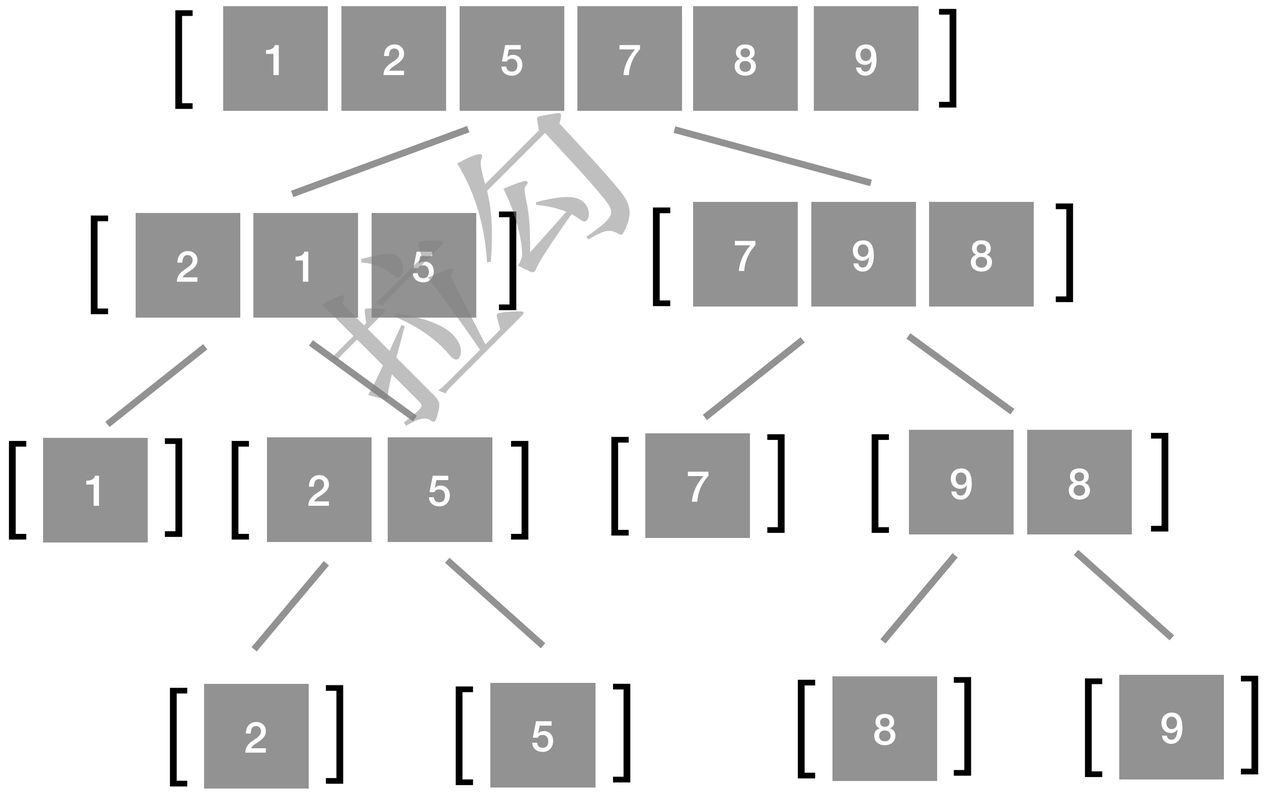
3.4 快速排序例题分析 / Example



3.4 快速排序例题分析 / Example



3.4 快速排序例题分析 / Example



3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 快速排序的主体函数 */

void sort(int[] nums, int lo, int hi) {
  if (lo >= hi) return;

  int p = partition(nums, lo, hi);

  sort(nums, lo, p - 1);
  sort(nums, p + 1, hi);
}
```

拉勾

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 快速排序的主体函数 */

void sort(int[] nums, int lo, int hi) {
  if (lo >= hi) return;

  int p = partition(nums, lo, hi);

  sort(nums, lo, p - 1);
  sort(nums, p + 1, hi);
}
```

拉勾

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 快速排序的主体函数 */

void sort(int[] nums, int lo, int hi) {
  if (lo >= hi) return;

  int p = partition(nums, lo, hi);

  sort(nums, lo, p - 1);
  sort(nums, p + 1, hi);
}
```

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
/** 快速排序的主体函数 */

void sort(int[] nums, int lo, int hi) {
  if (lo >= hi) return;

  int p = partition(nums, lo, hi);

  sort(nums, lo, p - 1);
  sort(nums, p + 1, hi);
}
```

拉勾

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
int partition(int[] nums, int lo, int hi) {
  swap(nums, randRange(lo, hi), hi);

  int i, j;

  for (i = lo, j = lo; j < hi; j++) {
    if (nums[j] <= nums[hi]) {
      swap(nums, i++, j);
    }
  }

  swap(nums, i, j);

  return i;
}
```

拉勾

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
int partition(int[] nums, int lo, int hi) {
  swap(nums, randRange(lo, hi), hi);

  int i, j;

  for (i = lo, j = lo; j < hi; j++) {
    if (nums[j] <= nums[hi]) {
      swap(nums, i++, j);
    }
  }

  swap(nums, i, j);

  return i;
}
```

拉勾

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
int partition(int[] nums, int lo, int hi) {
  swap(nums, randRange(lo, hi), hi);

  int i, j;

  for (i = lo, j = lo; j < hi; j++) {
    if (nums[j] <= nums[hi]) {
      swap(nums, i++, j);
    }
  }

  swap(nums, i, j);

  return i;
}
```

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
int partition(int[] nums, int lo, int hi) {
  swap(nums, randRange(lo, hi), hi);

  int i, j;

  for (i = lo, j = lo; j < hi; j++) {
    if (nums[j] <= nums[hi]) {
      swap(nums, i++, j);
    }
  }

  swap(nums, i, j);

  return i;
}
```

拉勾

3.4 快速排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
int partition(int[] nums, int lo, int hi) {
  swap(nums, randRange(lo, hi), hi);

  int i, j;

  for (i = lo, j = lo; j < hi; j++) {
    if (nums[j] <= nums[hi]) {
      swap(nums, i++, j);
    }
  }

  swap(nums, i, j);

  return i;
}
```

拉勾

最优情况下的时间复杂度

$$T(n) = 2*T(n/2) + O(n)$$

拉勾

O(n)是怎么得出来的呢?

把规模大小为n的问题分解成n/2的两个子问题；

和基准值进行n-1次比较，n-1次比较的复杂度就是O(n)；

快速排序的复杂度也是O(nlogn)。

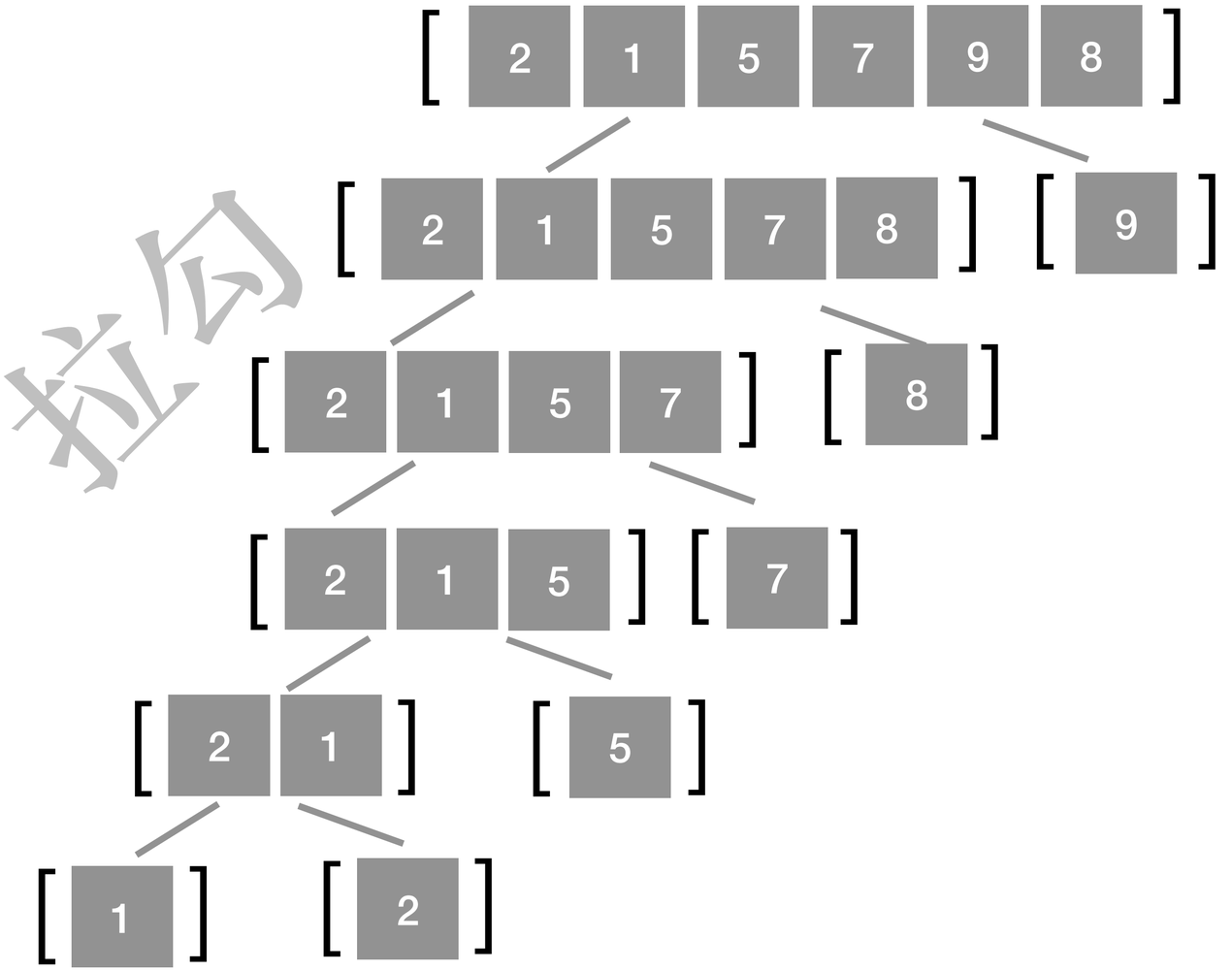
最复杂的情况

每次在选择基准值的时候；

都不幸选择了子数组里的最大或最小值；

其中一个子数组长度为1；

另一个长度只比父数组少1。



拉勾

空间复杂度：O(logn)

和归并排序不同，快速排序在每次递归的过程中；

只需要开辟O(1)的存储空间来完成交换操作实现直接对数组的修改；

而递归次数为logn，所以它的整体空间复杂度完全取决于压堆栈的次数。

拉勾

3.5 拓扑排序 / Topological Sort

力扣 LeetCode × 拉勾 LAGOU.COM

应用场合

拓扑排序就是要将图论里的顶点按照相连的性质进行排序

拉勾

前提

必须是有向图

图里没有环

算法思想

用图的数据结构去描述问题，然后利用广度优先搜索或深度优先搜索进行拓扑排序。

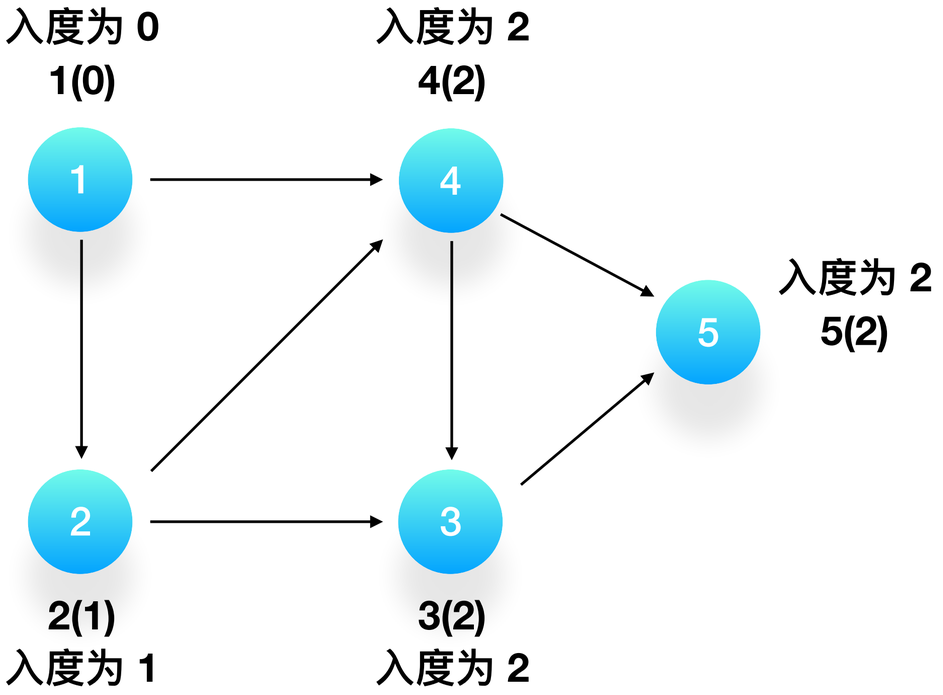
3.5 拓扑排序例题分析 / Example

有一个学生想要修完5门课程的学分，这5门课程分别用1、2、3、4、5来表示，现在已知学习这些课程有如下的要求：

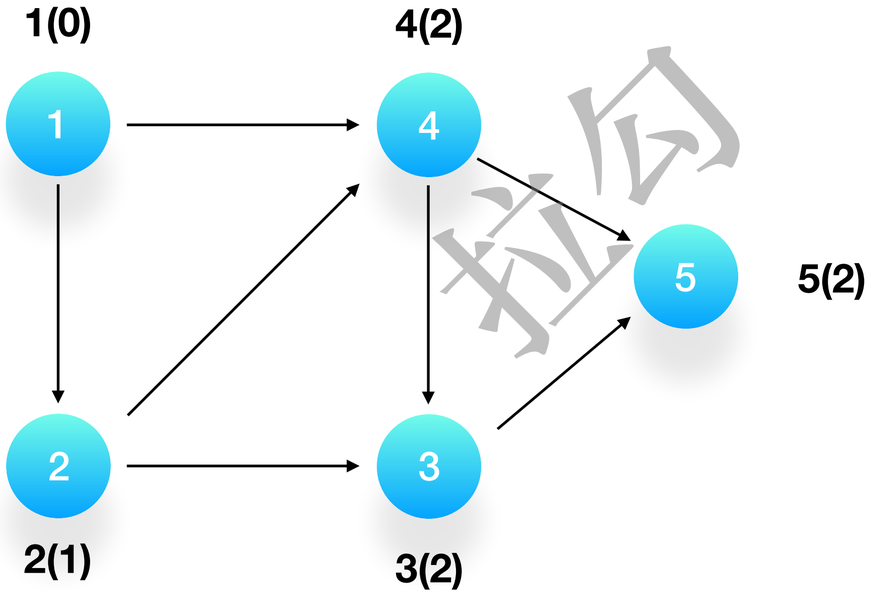
- 课程2和4依赖于课程1
- 课程3依赖于课程2和4
- 课程4依赖于课程1和2
- 课程5依赖于课程3和4

那么

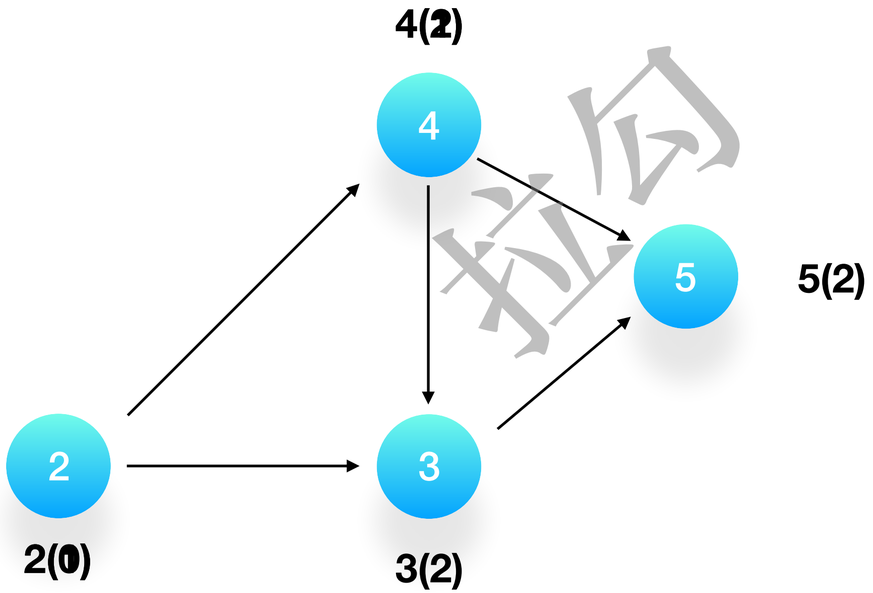
这个学生应该按照怎样的顺序来学习这5门课程呢?



3.5 拓扑排序例题分析 / Example

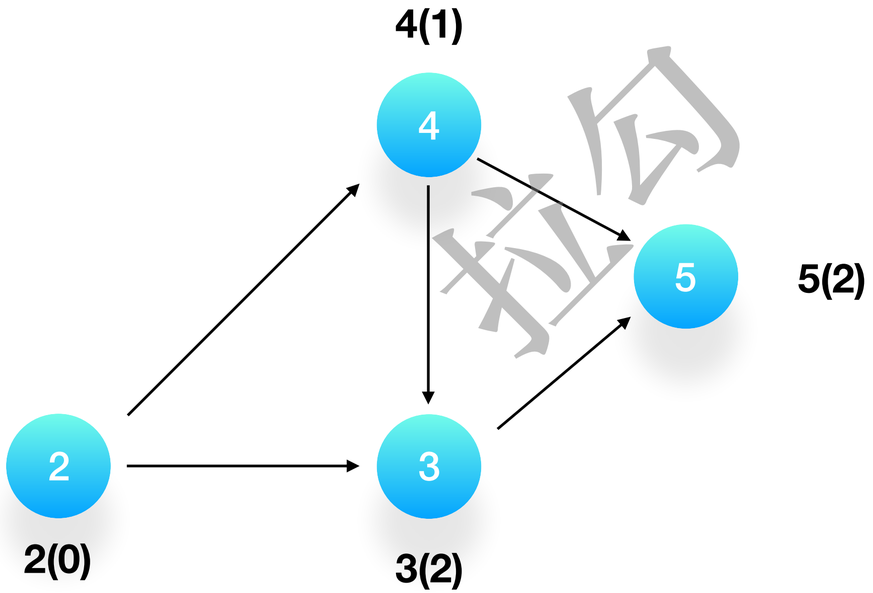


3.5 拓扑排序例题分析 / Example



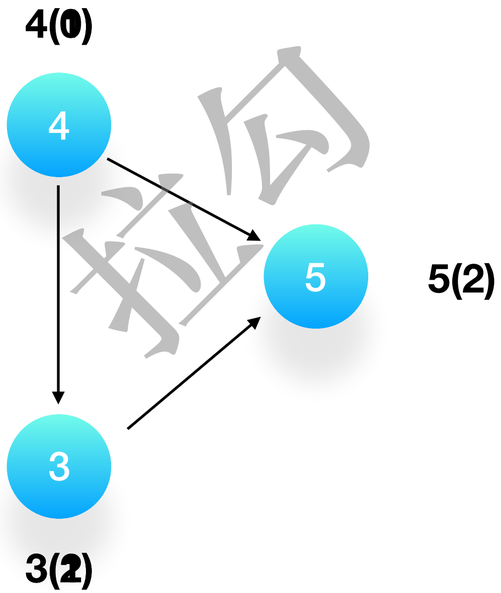
输出结果：

3.5 拓扑排序例题分析 / Example



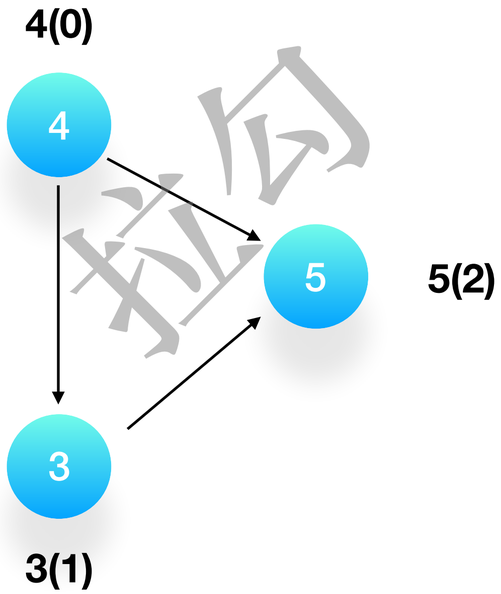
输出结果：

3.5 拓扑排序例题分析 / Example



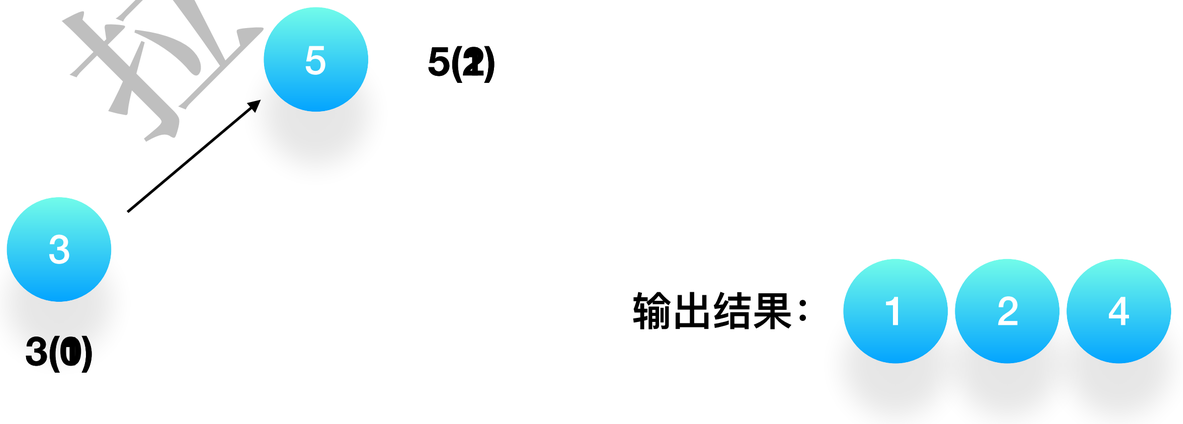
输出结果：

3.5 拓扑排序例题分析 / Example

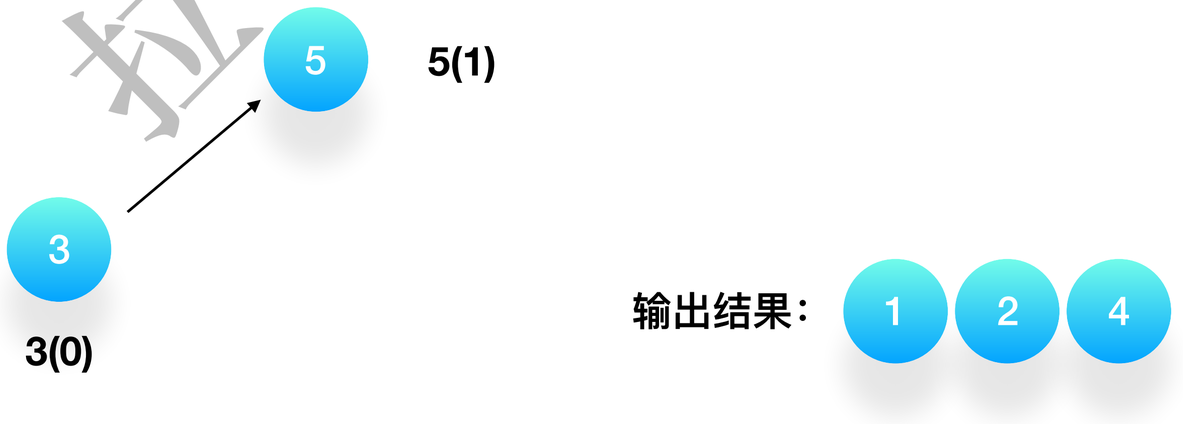


输出结果：

3.5 拓扑排序例题分析 / Example



3.5 拓扑排序例题分析 / Example



3.5 拓扑排序例题分析 / Example

力扣 LeetCode × 拉勾 LAGOU.COM

5(0)

输出结果：

3.5 拓扑排序例题分析 / Example

力扣 LeetCode × 拉勾 LAGOU.COM

输出结果：

3.5 拓扑排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort() {
  Queue<Integer> q = new LinkedList();

  for (int v = 0; v < V; v++) {
    if (indegree[v] == 0) q.add(v);
  }

  while (!q.isEmpty()) {
    int v = q.poll();
    print(v);

    for (int u = 0; u < adj[v].length; u++) {
      if (--indegree[u] == 0) {
        q.add(u);
      }
    }
  }
}
```

拉勾

3.5 拓扑排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort() {
  Queue<Integer> q = new LinkedList();

  for (int v = 0; v < V; v++) {
    if (indegree[v] == 0) q.add(v);
  }

  while (!q.isEmpty()) {
    int v = q.poll();
    print(v);

    for (int u = 0; u < adj[v].length; u++) {
      if (--indegree[u] == 0) {
        q.add(u);
      }
    }
  }
}
```

拉勾

3.5 拓扑排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort() {
  Queue<Integer> q = new LinkedList();

  for (int v = 0; v < V; v++) {
    if (indegree[v] == 0) q.add(v);
  }

  while (!q.isEmpty()) {
    int v = q.poll();
    print(v);

    for (int u = 0; u < adj[v].length; u++) {
      if (--indegree[u] == 0) {
        q.add(u);
      }
    }
  }
}
```

拉勾

3.5 拓扑排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort() {
  Queue<Integer> q = new LinkedList();

  for (int v = 0; v < V; v++) {
    if (indegree[v] == 0) q.add(v);
  }

  while (!q.isEmpty()) {
    int v = q.poll();
    print(v);

    for (int u = 0; u < adj[v].length; u++) {
      if (--indegree[u] == 0) {
        q.add(u);
      }
    }
  }
}
```

拉勾

3.5 拓扑排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort() {
  Queue<Integer> q = new LinkedList();

  for (int v = 0; v < V; v++) {
    if (indegree[v] == 0) q.add(v);
  }

  while (!q.isEmpty()) {
    int v = q.poll();
    print(v);

    for (int u = 0; u < adj[v].length; u++) {
      if (--indegree[u] == 0) {
        q.add(u);
      }
    }
  }
}
```

拉勾

3.5 拓扑排序代码示例 / Code Sample

力扣 LeetCode × 拉勾 LAGOU.COM

```
void sort() {
  Queue<Integer> q = new LinkedList();

  for (int v = 0; v < V; v++) {
    if (indegree[v] == 0) q.add(v);
  }

  while (!q.isEmpty()) {
    int v = q.poll();
    print(v);

    for (int u = 0; u < adj[v].length; u++) {
      if (--indegree[u] == 0) {
        q.add(u);
      }
    }
  }
}
```

拉勾

时间复杂度：O(n)

统计顶点的入度需要O(n)的时间；

接下来每个顶点被遍历一次，同样需要O(n)的时间。

拉勾

力扣 LeetCode × 拉勾 LAGOU.COM

Next：课时 4《强化面试中常用的算法 - 递归、回溯》

记得多加练习，才能更好地巩固知识点。

拉勾

关注 “拉勾教育”
学习技术干货

关注 “LeetCode力扣”
获得算法技术干货