数据结构

第 13 周 – 排序 II：归并排序和快速排序

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **简单排序**（第 5 周）：冒泡、选择、插入 – **$O(n^2)$**
- **归并**（第 3、6 周）：由两个有序序列得到一个有序序列 – **$O(n + m)$**
- **递归**（第 12 周）：把问题归结为较小的实例

今天的问题

- 怎样在 O(n log n) 时间内排序?
- 什么是“分而治之”的原则?
- 为什么快速排序是实践中的首选 – 它什么时候会变慢?

分而治之

分治法（divide and conquer） – 三个步骤：

1. **分解：**把问题分成若干个同类型的较小子问题
2. **解决：****递归地**求解各个子问题（非常小的直接求解）
3. **合并：**由各个子问题的解构造出整个问题的解

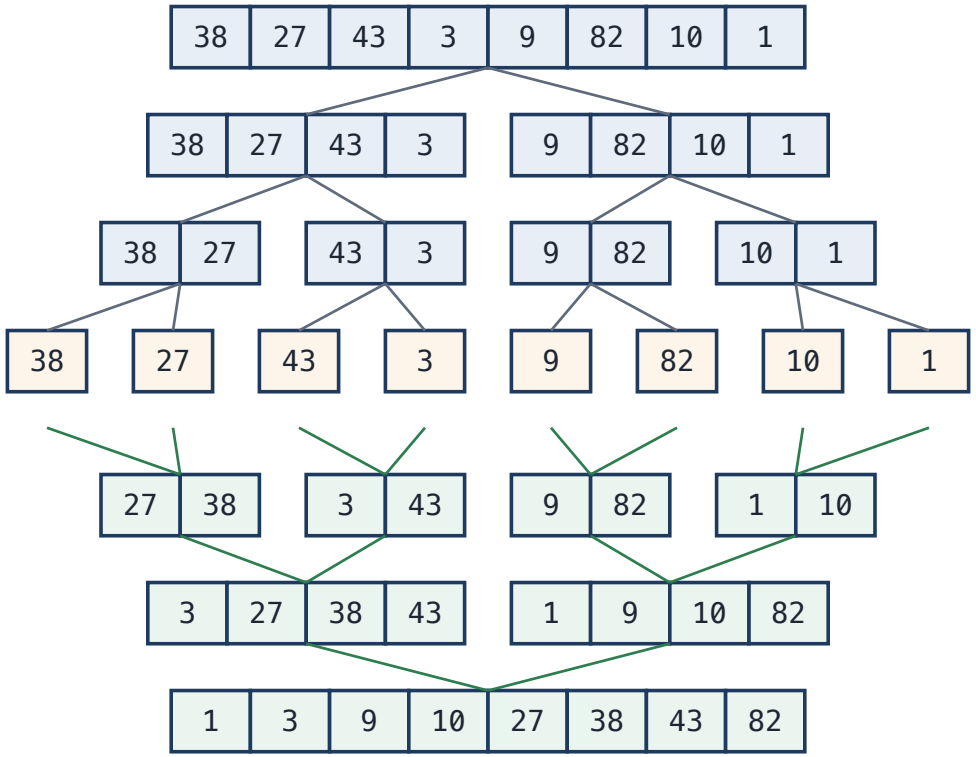
	分解	合并
归并排序	简单（从中间切开）	**工作量：**归并
快速排序	**工作量：**划分	简单（什么都不用做）
二分查找（第 2 周）	根据中间元素	–（只考察其中一半）

归并排序

归并排序 – 思路

拆分

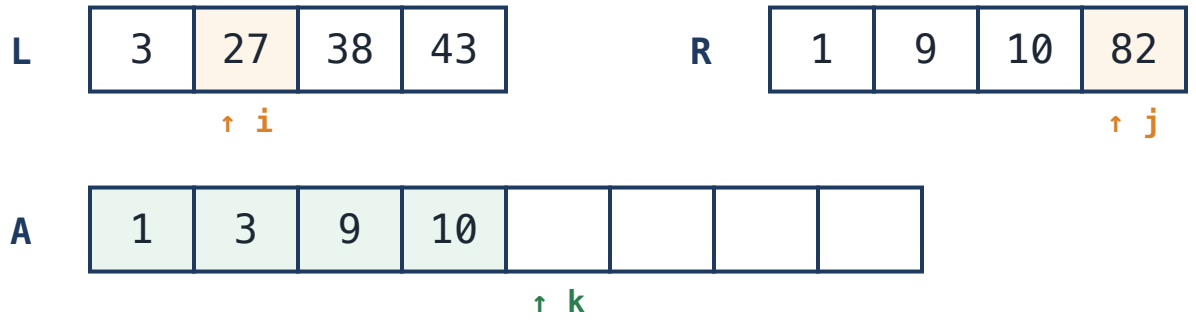
归并



$\log_2 n$ 层

每层 O(n) 的工作量

归并的步骤



`L[i] = 27 < R[j] = 82`

→ A[k] ← 27,i ← i + 1

总是取两个序列
首元素中较小的那个

- 由两个**有序**部分（L、R）得到一个有序数组（A）
- 总是把两部分**最前面**的元素中较小的那个移过去
- 如果其中一部分用完了，就把另一部分剩下的元素复制过去
- 步数：**O(n)**，额外空间：**O(n)**

归并排序 – 算法

伪代码

```
Procedure MergeSort(A, left, right)
  If left < right then
    k ← (left + right) div 2
    MergeSort(A, left, k)
    MergeSort(A, k + 1, right)
    Merge(A, left, k, right)
  End if
End procedure

Procedure Merge(A, left, k, right)
  L ← A[left..k] ; R ← A[k+1..right]   // 副本
  i ← 1 ; j ← 1 ; m ← left
  While i ≤ length(L) and j ≤ length(R)
    If L[i] ≤ R[j] then
      A[m] ← L[i] ; i ← i + 1
    else
      A[m] ← R[j] ; j ← j + 1
    End if
    m ← m + 1
  End while
  copy the rest of L and R into A      // 复制剩余部分
End procedure
```

Java

```java
static void mergeSort(int[] a, int lo, int hi) {
    if (lo >= hi) return;
    int mid = lo + (hi - lo) / 2;
    mergeSort(a, lo, mid);
    mergeSort(a, mid + 1, hi);
    merge(a, lo, mid, hi);
}

static void merge(int[] a, int lo, int mid, int hi) {
    int[] l = Arrays.copyOfRange(a, lo, mid + 1);
    int[] r = Arrays.copyOfRange(a, mid + 1, hi + 1);
    int i = 0, j = 0, k = lo;
    while (i < l.length && j < r.length)
        a[k++] = (l[i] <= r[j]) ? l[i++] : r[j++];
    while (i < l.length) a[k++] = l[i++];
    while (j < r.length) a[k++] = r[j++];
}
```

归并排序 – 分析

- 对半分的次数：**$\log_2 n$ 层**
- 每一层总共归并 **n** 个元素 → 每层 **O(n)**
- 总共：**O(n log n)** – 在**所有**情况下（最好、平均、最坏）

性质	归并排序
步数	总是 **O(n log n)**
额外空间	**O(n)** – 不是原地排序
稳定	**是**（归并中用 `≤`）
在链表上	很适合（不需要直接访问）

> $n = 10^6$ 时：$n^2 = 10^{12}$ 步，$n \log_2 n \approx 2 \cdot 10^7$ 步 – 少了 **50 000 倍**！

快速排序

快速排序 – 思路

C. A. R. Hoare(1959)

1. 选择一个元素：**枢轴**（pivot）
2. **划分**：重新排列数组，使比枢轴**小**的元素在**左边**，**大**的元素在**右边** – 这样枢轴就到达了它的**最终位置**
3. **递归地**对左右两部分排序

> 与归并排序不同，这里的工作量在**分解**上，合并是“免费”的：如果两部分都有序，整体就有序。

划分（Lomuto 方法）


起点
7 2 9 4 3 8 6
pivot = 6（最后一个元素）
j = 1
2 7 9 4 3 8 6
2 < 6 → i = 0，交换
j = 3
2 4 9 7 3 8 6
4 < 6 → i = 1，交换
j = 4
2 4 3 7 9 8 6
3 < 6 → i = 2，交换
结束
2 4 3 6 9 8 7
pivot 归位：swap(A[i+1], A[hi])
< pivot
pivot
≥ pivot

快速排序 – 算法

伪代码

```
Procedure QuickSort(A, left, right)
  If left < right then
    p ← Partition(A, left, right)
    QuickSort(A, left, p - 1)
    QuickSort(A, p + 1, right)
  End if
End procedure

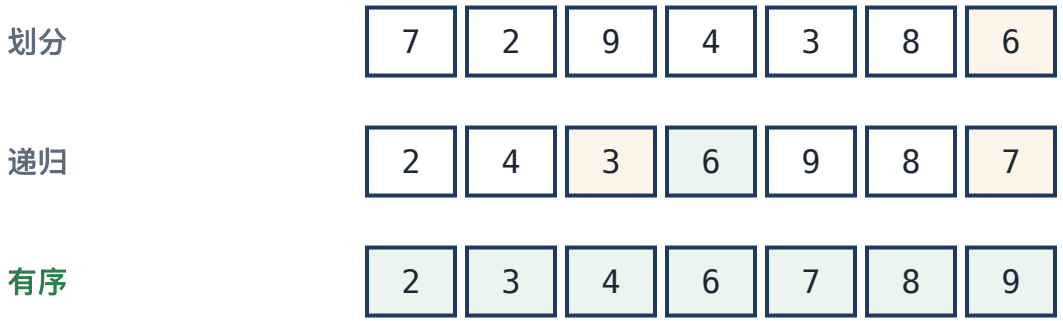
Function Partition(A, left, right)
  pivot ← A[right]
  i ← left - 1
  For j ← left to right - 1
    If A[j] < pivot then
      i ← i + 1
      swap(A[i], A[j])
    End if
  End for
  swap(A[i + 1], A[right])
  Return i + 1
End function
```

Java

```java
static void quickSort(int[] a, int lo, int hi) {
    if (lo >= hi) return;
    int p = partition(a, lo, hi);
    quickSort(a, lo, p - 1);
    quickSort(a, p + 1, hi);
}

static int partition(int[] a, int lo, int hi) {
    int pivot = a[hi];
    int i = lo - 1;
    for (int j = lo; j < hi; j++)
        if (a[j] < pivot) {
            i++;
            swap(a, i, j);
        }
    swap(a, i + 1, hi);
    return i + 1;
}
```

枢轴选择的重要性



好的 pivot（接近中位数）：对半分 → log n 层 → O(n log n)

差的 pivot（总是最小或最大值）:n 层 → $O(n^2)$

快速排序 – 分析

情况	何时?	步数
最好	枢轴总是中位数	O(n log n)
平均	随机数据	**O(n log n)**（约 $1.39 \cdot n \log_2 n$ 次比较）
最坏	枢轴总是最小值或最大值（例如**有序**数组，以最后一个元素为枢轴）	**$O(n^2)$**

实测（n = 5000，有序输入）：固定枢轴 **12 497 500** 次比较，随机枢轴 **68 648** 次

防止最坏情况：

- **随机枢轴** – 坏情况的概率可以忽略
- **三数取中**（第一个、中间和最后一个元素的中位数）
- 对较小的部分递归，对较大的部分用循环 → 栈深度 O(log n)

快速排序 – 性质

性质	快速排序
步数	平均 $O(n \log n)$，最坏 $O(n^2)$
额外空间	**$O(\log n)$**（递归的栈） – **原地**排序
稳定	**否**（由于远距离交换）
实际表现	通常**最快** – 数据移动少，对缓存友好

Java 的 `Arrays.sort` 对基本类型数组使用**双轴快速排序**，Go 的 `slices.Sort` 使用 **pdqsort**（快速排序的改进版本）。

比较

各种排序算法的比较

	最坏	平均	最好	额外空间	稳定
冒泡	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	是
选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	否
插入	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	是
归并	**$O(n \log n)$**	$O(n \log n)$	$O(n \log n)$	$O(n)$	**是**
快速	$O(n^2)$	**$O(n \log n)$**	$O(n \log n)$	$O(\log n)$	否
堆排序（第 15 周）	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	否
计数排序（第 5 周）	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	是

下界：只基于比较的排序算法在最坏情况下**不可能比 $\Omega(n \log n)$ 更快**。计数排序之所以更快，是因为它不比较而是计数（只适用于较小的取值范围）。

实测结果（随机数据）

n	插入	归并	快速	内置
1 000	0.8 ms	0.2 ms	0.2 ms	0.9 ms
10 000	13.3 ms	0.7 ms	0.6 ms	1.5 ms
100 000	**1 159 ms**	16.5 ms	4.4 ms	5.7 ms
1 000 000	（约 2 分钟）	65.6 ms	49.8 ms	43.6 ms

Java，`SortBenchmark` 程序。n 较小时内置排序看起来较慢，是因为 JVM 的预热。

外部排序

如果数据**放不进内存**怎么办（例如 100 GB 的文件，8 GB 内存）？

外部归并排序：

1. 每次读入能放下的数据量（例如 1 GB），在内存中排序，写入一个临时文件 → 得到 100 个有序的“顺串”
2. **归并**这些顺串：每个顺串只在内存中保留开头的一部分，每次输出最小的元素（**k** 路归并 – 用优先队列）

- 慢速的磁盘操作次数最少：每个数据只读写**两次**
- 数据库系统和大数据处理系统就是这样排序的

小结

- **分而治之：** 分解 → 递归求解 → 合并
- **归并排序：** 对半分 + 归并 · 总是 **O(n log n)** · 稳定 · O(n) 额外空间
- **快速排序：** 枢轴 + 划分 · 平均 **O(n log n)**，最坏 $O(n^2)$ · 原地 · 不稳定
- 枢轴的选择至关重要 – **随机枢轴**或三数取中
- 比较排序不可能比 **O(n log n)** 更快
- **外部排序：** 顺串 + **k** 路归并

下周： 树 – 基本概念、二叉树、存储表示、遍历。

复习题

1. 画出对数组 `[5, 1, 4, 2, 8, 3]` 进行归并排序的拆分树和归并树！
2. 用 Lomuto 方法对数组 `[4, 8, 1, 6, 3, 7, 5]` 进行划分（pivot = 5）！枢轴到了哪里？
3. 判断对错：快速排序即使在最坏情况下也只需 O(n log n) 步。
4. 如果稳定性很重要，并且需要保证 O(n log n)，你会选择哪种排序？
5. 如果总是以最后一个元素为枢轴，为什么快速排序在有序数组上很慢？
6. n = 1024 时，归并排序的递归树有多少层？

实践课

✎ 示例程序

程序	Java	Go	演示内容
归并排序	`MergeSort.java`	`w13/mergesort`	归并的顺序、递归深度
快速排序	`QuickSort.java`	`w13/quicksort`	划分、差的枢轴和随机枢轴
测量	`SortBenchmark.java`	`w13/benchmark`	$O(n^2)$、$O(n \log n)$ 与内置排序

运行 `SortBenchmark` 程序！**n** 增大 10 倍时，插入排序和归并排序的时间分别增大多少倍？

✎ 练习

1. **降序：**把归并排序改成降序！
2. **稳定排序：**按长度对单词排序，使长度相同的单词保持原来的顺序！
3. **O(n log n) 求逆序对：**在归并过程中统计逆序对！（与第 5 周 $O(n^2)$ 的解法比较）
4. **快速选择：**不排序，用划分找出第 **k** 小的元素 – 平均 O(n)！
5. **荷兰国旗：**只遍历一次，对只包含 0、1、2 的数组排序！
6. **中位数：**用归并求两个有序数组的共同中位数。

参考答案：`Exercises13.java`、`w13/exercises`

✎ 课后作业

1. 用**三数取中**的枢轴选择实现快速排序！在有序和逆序的输入上测量！
2. **混合排序：**如果区间少于 16 个元素，就用插入排序！根据测量，这能加速多少?
3. 为**链表**编写归并排序（不用辅助数组）！（提示：用第 9 周的龟兔赛跑方法把链表一分为二。）
4. 实现 **Hoare** 划分方法（两个下标相向移动）！与 Lomuto 方法比较交换次数！