

极客时间算法训练营

第九课

排序

李煜东

《算法竞赛进阶指南》作者



目录

1. 基于比较的各类排序算法
2. 其他排序算法，不同排序算法的适用场景
3. 第K大数、中位数、逆序对等应用

排序算法分类

基于比较的排序

通过比较大小来决定元素间的相对次序

可以证明时间复杂度下界为 $O(N\log N)$ —— 不可能突破这个复杂度达到更快

非比较类排序

不通过比较大小来决定元素间的相对次序

时间复杂度受元素的范围以及分布等多种因素影响，不单纯取决于元素数量 N

排序算法

```
graph LR; A[排序算法] --- B[比较类排序]; A --- C[非比较排序]; B --- D[交换排序]; B --- E[插入排序]; B --- F[选择排序]; B --- G[归并排序]; D --- D1[冒泡排序]; D --- D2[快速排序]; E --- E1[简单插入排序]; E --- E2[希尔排序]; F --- F1[简单选择排序]; F --- F2[堆排序]; G --- G1[二路归并排序]; G --- G2[多路归并排序]; C --- H[计数排序]; C --- I[桶排序]; C --- J[基数排序]
```

比较类排序

交换排序

冒泡排序

快速排序

插入排序

简单插入排序

希尔排序

选择排序

简单选择排序

堆排序

归并排序

二路归并排序

多路归并排序

非比较排序

计数排序

桶排序

基数排序

基于比较的各类排序算法

初级排序算法

选择排序 (Selection Sort) —— “该放哪个数了？”

每次从未排序数据中找最小值，放到已排序序列的末尾

插入排序 (Insertion Sort) —— “这个数该放哪儿？”

从前到后依次考虑每个未排序数据，在已排序序列中找到合适位置插入

冒泡排序 (Bubble Sort)

不断循环扫描，每次查看相邻的元素，如果逆序，则交换

平均时间复杂度均为 $O(N^2)$

堆排序

堆排序（Heap Sort）是对选择排序的优化——利用二叉堆高效地选出最小值

建立一个包含所有 N 个元素的二叉堆

重复 N 次从堆中取出最小值，即可得到有序序列

时间复杂度 $O(N\log N)$

```
void heap_sort(int a[], int n) {  
    priority_queue<int> q;  
    for(int i = 0; i < n; i++) {  
        q.push(-a[i]);  
    }  
    for(int i = 0; i < n; i++) {  
        a[i] = -q.top();  
        q.pop();  
    }  
}
```

希尔排序（选做）

希尔排序（Shell Sort）是对插入排序的优化——增量分组插入排序

图示

希尔排序的时间复杂度取决于增量序列（步长序列）的选取

目前已知的最好序列可以做到 $O(N^{4/3})$ 或 $O(N\log^2 N)$

增量序列列表

归并排序

归并排序（Merge Sort）是一个基于分治的算法

时间复杂度 $O(N\log N)$

原问题：把数组排序

子问题：把数组前半、后半分别排序

然后再合并左右两半（两个有序数组）就可以了

```
public static void mergeSort(int[] arr, int l, int r) { // sort arr[l..r]
    if (l >= r) return;
    int mid = (l + r) >> 1; // (l + r) / 2
    mergeSort(arr, l, mid);
    mergeSort(arr, mid + 1, r);
    merge(arr, l, mid, r);
}
```

归并排序

```
static void merge(int[] arr, int left, int mid, int right) {  
  
    int[] temp = new int[right - left + 1]; // 临时数组  
    int i = left, j = mid + 1;  
  
    for (int k = 0; k < temp.length; k++) { // 合并两个有序数组  
        if (j > right || (i <= mid && arr[i] <= arr[j]))  
            temp[k] = arr[i++];  
        else  
            temp[k] = arr[j++];  
    }  
  
    for (int k = 0; k < temp.length; k++) { // 拷回原数组  
        arr[left + k] = temp[k];  
    }  
}
```

快速排序

快速排序（Quick Sort）也是一个基于分治的算法

- 从数组中选取中轴元素 pivot
- 将小元素放在 pivot 左边，大元素放在右边
- 然后分别对左边和右边的子数组进行快排

快速排序和归并排序具有相似性，但步骤顺序相反

- 归并排序：先排序左右子数组，然后合并两个有序数组
- 快速排序：先调配出左右子数组，然后对左右子数组分别进行排序

随机选取pivot，期望时间复杂度 $O(N\log N)$

快速排序

快速排序可以通过适当的交换来原地实现数组调配，避免占用额外空间

最经典和高效的调配方式叫作 Hoare Partition

Hoare Partition 动画：<https://www.bilibili.com/video/BV1q64y1S7Ax>

这个应该是大多数人在课本上学到的，双指针一边向中间扫描一边交换

```
public static void quickSort(int[] arr, int l, int r) {  
    if (l >= r) return;  
    int pivot = partition(arr, l, r);  
    quickSort(arr, l, pivot);  
    quickSort(arr, pivot + 1, r);  
}
```


快速排序

```
static int partition(int[] a, int l, int r) {  
    int pivot = l + (int)(Math.random() * (r - l + 1));  
    int pivotVal = a[pivot];  
  
    while (l <= r) {  
        while (a[l] < pivotVal) l++;  
        while (a[r] > pivotVal) r--;  
        if (l == r) break;  
        if (l < r) {  
            int temp = a[l]; a[l] = a[r]; a[r] = temp;  
            l++; r--;  
        }  
    }  
    return r;  
}
```

非比较类排序算法

非比较类排序

计数排序 (Counting Sort)

计数排序要求输入的数据必须是有确定范围的整数。将输入的数据作为 key 存储在额外的数组中，然后依次把计数大于 1 的填充回原数组

时间复杂度 $O(N+M)$ ，N为元素个数，M为数值范围

桶排序 (Bucket Sort)

桶排序假设输入数据服从均匀分布，将数据分到有限数量的桶里，每个桶再分别排序（有可能使用别的排序算法，或是以递归方式继续使用桶排序）

时间复杂度 $O(N) \sim O(N^2)$

基数排序 (Radix Sort)

基数排序把数据切割成一位位数字（0-9），从低位到高位对每一位分别进行计数排序

时间复杂度 $O(NK)$ ，K 为数字位数

排序的稳定性

对于序列中存在的若干个关键字相等的元素

如果排序前后它们的相对次序一定保持不变，就称排序算法是稳定的
否则就称排序算法是不稳定的

插入、冒泡、归并、计数、基数和桶排序是稳定的
选择、希尔、快速、堆排序是不稳定的

排序阵营九宫格

	稳定	不稳定	
$O(n^2)$	<div>守序划水</div> <div><u>插入排序</u></div> <div>都有地儿，一个个来嘛...</div>	<div>中立划水</div> <div><u>冒泡排序</u></div> <div>有逆序？容我交换一下...</div>	<div>混乱划水</div> <div><u>选择排序</u></div> <div>随便找个最小的？好的马上...</div>
???	<div>守序中立</div> <div><u>计数排序 / 基数排序</u></div> <div>不比较？那从数值范围入手吧</div>	<div>绝对中立</div> <div><u>桶排序</u></div> <div>我就分个组，具体咋排你们定</div>	<div>混乱中立</div> <div><u>希尔排序</u></div> <div>插排慢？这年头不会增量分组吗</div>
$O(n\log n)$	<div>守序内卷</div> <div><u>归并排序</u></div> <div>$n\log n$稳定可靠，你值得拥有 合并有序数组大家都会吧...</div>	<div>中立内卷</div> <div><u>堆排序</u></div> <div>优化也是要讲基本法的... 选最小值不考虑一下堆吗？</div>	<div>混乱内卷</div> <div><u>快速排序</u></div> <div>$n\log n$里常数最小了解一下？ swap是一门艺术，不服不要玩</div>

排序算法实战应用

实战

排序数组

<https://leetcode-cn.com/problems/sort-an-array/>

模板练习题

实战

数组的相对排序

<https://leetcode-cn.com/problems/relative-sort-array/>

比较类排序

- 利用哈希表对 arr2 建立数值到索引的映射
- 自定义比较函数

非比较类排序

- 计数排序
- 第一遍把计数数组中出现在 arr2 的数值填充回 arr1
- 第二遍把剩下的填充回 arr1

实战

合并区间

<https://leetcode-cn.com/problems/merge-intervals/>

方法一：对区间进行双关键字排序（左右端点）
然后扫描合并（排序后能合并的区间一定是连续的）

方法二：差分思想、关键事件思想

把每个区间 $[l, r]$ 看作一次+1的覆盖，进一步转化为 “ l 处 +1”、“ $r+1$ 处 -1” 两个事件

把 $2n$ 个事件排序、扫描，用一个计数变量记录覆盖次数，0变1、1变0时就找到了合并后的区间端点

实战

数组中的第K个最大元素

<https://leetcode-cn.com/problems/kth-largest-element-in-an-array/>

实战

货仓选址

<https://www.acwing.com/problem/content/description/106/>

在一条数轴上有 N 家商店，它们的坐标分别为 $A_1 \sim A_n$ 。

现在需要在数轴上建立一家货仓，每天清晨，从货仓到每家商店都要运送一车商品。

为了提高效率，求把货仓建在何处，可以使得货仓到每家商店的距离之和最小。

$1 \leq N \leq 100000$

实战

翻转对

<https://leetcode-cn.com/problems/reverse-pairs/>

区间和的个数 (Homework, 困难, 选做)

<https://leetcode-cn.com/problems/count-of-range-sum/>

总结:

在一个数组中统计满足特定大小关系的pair数量, 可以考虑基于归并排序求解

THANKS

 极客时间 | 训练营