快速排序算法

题目难度：★★★

知识点标签：快速排序算法

学习时长：20分钟

题目描述

描述一下快速排序算法。

解题思路

> 面试官问题可以从几个方面来回答：快速排序算法原理、伪代码实现方式，时间复杂度，如何避免最坏情况

快速排序算法原理

快速排序的核心思想是：分治思想。

简单而言，比较排序算法是对两个元素进行比较来决定两个元素的在输出序列中相对位置（谁在前面谁在后面）的排序算法。

快速排序的主旨很简单：找一个**标杆数**，称为**X**，然后根据X把数组的数分堆，**小于X**的全放左边，**大于X**的全放右边，就可以啦。对于实际情况呢，我们还需要考虑**等于X**的情况，我将其与小于归为一起，即数组排列后，形成“**小于等于X**” + “**大于X**”两部分。

就是说，快速排序的主要步骤就是：找X + 跟X比大小排列?

你可能会疑惑，只是按“比X大或比X小”排列数组，怎么能得到完整的排序呢。一次排列几乎不可能排好，但我们可以将排了一次的数组上，切分为“小于等于X”和“大于X”两块，再对这两块分别再找标杆数X'和X''，接着再分别排序。最后组合再一起，就得到了排列了两次的数组，其顺序肯定更接近完美序列。那么我们继续这么“切分→找标杆数→排列”操作下去呢？在此例中，由于每一分堆总小于右侧的分堆，而大于左侧的分堆，同时每个分堆内部已排好序，因此整个序列排序完成。

以上这种操作叫做“递归”，可以对数组不断地切分并采用同样的排列模式进行排列，直到递归条件不再满足，则停止递归。在这里，我们选择切分后数组的长度大小，作为递归的条件。

伪代码实现方式

```
    public <T extends Comparable<T>> T[] sort(T[] array) {
        doSort(array, 0, array.length - 1);
        return array;
    }
```

```
    private static <T extends Comparable<T>> void doSort(T[] array, int left,
int right) {
        if (left < right) {
            int pivot = randomPartition(array, left, right);
            doSort(array, left, pivot - 1);
            doSort(array, pivot, right);
        }

    }

    private static <T extends Comparable<T>> int randomPartition(T[] array,
int left, int right) {
        int randomIndex = left + (int)(Math.random() * (double)(right - left +
1));
        SortUtils.swap(array, randomIndex, right);
        return partition(array, left, right);
    }

    private static <T extends Comparable<T>> int partition(T[] array, int
left, int right) {
        int mid = (left + right) / 2;
        Comparable pivot = array[mid];

        while(left <= right) {
            while(SortUtils.less(array[left], pivot)) {
                ++left;
            }

            while(SortUtils.less(pivot, array[right])) {
                --right;
            }

            if (left <= right) {
                SortUtils.swap(array, left, right);
                ++left;
                --right;
            }
        }

        return left;
    }
```

常见的比较排序算法

1. 冒泡排序（Bubble Sort）时间复杂度和空间复杂度为$\Theta(n^2), \Theta(n)$。
2. 插入排序（Insertion Sort） 时间复杂度下界，上界以及空间复杂度为$\Omega(n), O(n^2)\ and\ \Theta(n)$。

3. 归并排序（Merge Sort）时间复杂度和空间复杂度为 $\Theta(n \cdot log_2(n)) \; and \; \Theta(n \cdot log_2(n))$ 。可见虽然归并排序的时间复杂度符合理论最小值，但它的空间复杂度也很高。更蛋疼的是归并排序的时间复杂度分析是建立在RAM模型（Random Access Memory）上，而实际的计算机内存模型并不是RAM而是有cache的。因为归并排序算法在每次递归迭代过程中都会申请新的内存然后在新申请内存上进行操作。这不匹配于cache内存机制，所以归并排序算法在真实硬件上的表现不很理想。不过随着并行，分布式系统的兴起，这个算法又有了新的生机。它非常适合并行优化，并且已经有了相应分析和研究（如果这篇反响好我第二篇就写这个内容）。

时间复杂度，空间复杂度

时间复杂度为O(nlogn)，空间复杂度为O(n)

如何避免最坏情况

1. 采用三位中值法选择标杆数
2. 采用随机数的方式选择标杆数

总结

充分理解快速排序算法的运行原理，可以形象的描述清楚。

快速排序可实现理论最优效率，这可能是快速排序比较重要的原因吧。