**目录**

[要点](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E8%A6%81%E7%82%B9)  
[算法分析](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E7%AE%97%E6%B3%95%E5%88%86%E6%9E%90)  
[快速排序算法的性能](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95%E7%9A%84%E6%80%A7%E8%83%BD)  
[时间复杂度](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)  
[空间复杂度](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)  
[算法稳定性](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E7%AE%97%E6%B3%95%E7%A8%B3%E5%AE%9A%E6%80%A7)  
[完整参考代码](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E5%AE%8C%E6%95%B4%E5%8F%82%E8%80%83%E4%BB%A3%E7%A0%81)  
[JAVA版本](https://www.cnblogs.com/jingmoxukong/p/4302891.html#java%E7%89%88%E6%9C%AC)  
[参考资料](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E5%8F%82%E8%80%83%E8%B5%84%E6%96%99)  
[相关阅读](https://www.cnblogs.com/jingmoxukong/p/4302891.html#%E7%9B%B8%E5%85%B3%E9%98%85%E8%AF%BB)

**要点**

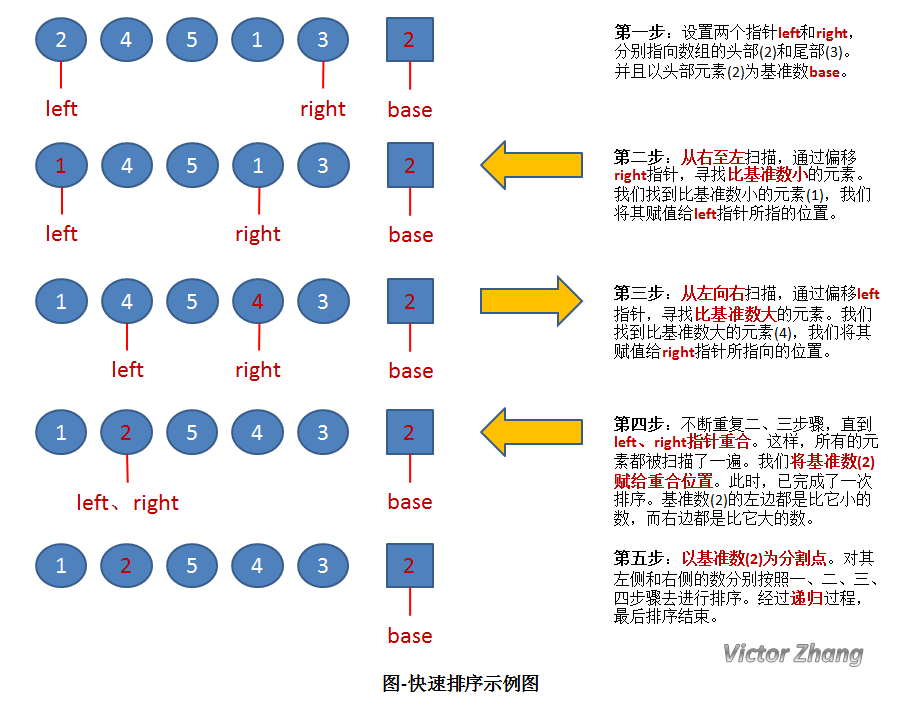
快速排序是一种**交换排序**。

快速排序由C. A. R. Hoare在1962年提出。

它的基本思想是：通过一趟排序将要排序的数据**分割**成独立的两部分：**分割点左边都是比它小的数，右边都是比它大的数**。

然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

详细的图解往往比大堆的文字更有说明力，所以直接上图：

[](http://images2015.cnblogs.com/blog/318837/201604/318837-20160421155746320-2138416137.png)

上图中，演示了快速排序的处理过程：

**初始状态**为一组无序的数组：2、4、5、1、3。

经过以上操作步骤后，完成了**第一次的排序**，得到新的数组：1、2、5、4、3。

新的数组中，以2为分割点，左边都是比2小的数，右边都是比2大的数。

因为2已经在数组中找到了合适的位置，所以不用再动。

2左边的数组只有一个元素1，所以显然不用再排序，位置也被确定。（注：这种情况时，left指针和right指针显然是重合的。因此在代码中，我们可以通过设置判定条件**left必须小于right，如果不满足，则不用排序了**）。

而对于2右边的数组5、4、3，设置left指向5，right指向3，开始继续**重复图中的一、二、三、四步骤**，对新的数组进行排序。

**核心代码**

[复制代码](javascript:void(0);)

public int division(int[] list, int left, int right) {  
    // 以最左边的数(left)为基准  
    int base = list[left];  
    while (left < right) {  
        // 从序列右端开始，向左遍历，直到找到小于base的数  
        while (left < right && list[right] >= base)  
            right--;  
        // 找到了比base小的元素，将这个元素放到最左边的位置  
        list[left] = list[right];  
   
        // 从序列左端开始，向右遍历，直到找到大于base的数  
        while (left < right && list[left] <= base)  
            left++;  
        // 找到了比base大的元素，将这个元素放到最右边的位置  
        list[right] = list[left];  
    }  
   
    // 最后将base放到left位置。此时，left位置的左侧数值应该都比left小；  
    // 而left位置的右侧数值应该都比left大。  
    list[left] = base;  
    return left;  
}  
   
private void quickSort(int[] list, int left, int right) {  
   
    // 左下标一定小于右下标，否则就越界了  
    if (left < right) {  
        // 对数组进行分割，取出下次分割的基准标号  
        int base = division(list, left, right);  
   
        System.out.format("base = %d:\t", list[base]);  
        printPart(list, left, right);  
   
        // 对“基准标号“左侧的一组数值进行递归的切割，以至于将这些数值完整的排序  
        quickSort(list, left, base - 1);  
   
        // 对“基准标号“右侧的一组数值进行递归的切割，以至于将这些数值完整的排序  
        quickSort(list, base + 1, right);  
    }  
}

[复制代码](javascript:void(0);)

**算法分析**

**快速排序算法的性能**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **排序类别** | **排序方法** | **时间复杂度** | | | **空间复杂度** | **稳定性** | **复杂性** |
| **平均情况** | **最坏情况** | **最好情况** |
| 交换排序 | 快速排序 | O(Nlog2N) | O(N2) | O(Nlog2N) | O(Nlog2N) | 不稳定 | 较复杂 |

**时间复杂度**

当数据有序时，以第一个关键字为基准分为两个子序列，前一个子序列为空，此时执行效率最差。

而当数据随机分布时，以第一个关键字为基准分为两个子序列，两个子序列的元素个数接近相等，此时执行效率最好。

所以，数据越随机分布时，快速排序性能越好；数据越接近有序，快速排序性能越差。

**空间复杂度**

快速排序在每次分割的过程中，需要 1 个空间存储基准值。而快速排序的大概需要 Nlog2N次的分割处理，所以占用空间也是 Nlog2N 个。

**算法稳定性**

在快速排序中，相等元素可能会因为分区而交换顺序，所以它是不稳定的算法。

**完整参考代码**

**JAVA版本**

**代码实现**



[复制代码](javascript:void(0);)

 1 public class QuickSort {  
 2    
 3     public int division(int[] list, int left, int right) {  
 4         // 以最左边的数(left)为基准  
 5         int base = list[left];  
 6         while (left < right) {  
 7             // 从序列右端开始，向左遍历，直到找到小于base的数  
 8             while (left < right && list[right] >= base)  
 9                 right--;  
10             // 找到了比base小的元素，将这个元素放到最左边的位置  
11             list[left] = list[right];  
12    
13             // 从序列左端开始，向右遍历，直到找到大于base的数  
14             while (left < right && list[left] <= base)  
15                 left++;  
16             // 找到了比base大的元素，将这个元素放到最右边的位置  
17             list[right] = list[left];  
18         }  
19    
20         // 最后将base放到left位置。此时，left位置的左侧数值应该都比left小；  
21         // 而left位置的右侧数值应该都比left大。  
22         list[left] = base;  
23         return left;  
24     }  
25    
26     private void quickSort(int[] list, int left, int right) {  
27    
28         // 左下标一定小于右下标，否则就越界了  
29         if (left < right) {  
30             // 对数组进行分割，取出下次分割的基准标号  
31             int base = division(list, left, right);  
32    
33             System.out.format("base = %d:\t", list[base]);  
34             printPart(list, left, right);  
35    
36             // 对“基准标号“左侧的一组数值进行递归的切割，以至于将这些数值完整的排序  
37             quickSort(list, left, base - 1);  
38    
39             // 对“基准标号“右侧的一组数值进行递归的切割，以至于将这些数值完整的排序  
40             quickSort(list, base + 1, right);  
41         }  
42     }  
43    
44     // 打印序列  
45     public void printPart(int[] list, int begin, int end) {  
46         for (int i = 0; i < begin; i++) {  
47             System.out.print("\t");  
48         }  
49         for (int i = begin; i <= end; i++) {  
50             System.out.print(list[i] + "\t");  
51         }  
52         System.out.println();  
53     }  
54    
55     public static void main(String[] args) {  
56         // 初始化一个序列  
57         int[] array = {  
58                 1, 3, 4, 5, 2, 6, 9, 7, 8, 0  
59         };  
60    
61         // 调用快速排序方法  
62         QuickSort quick = new QuickSort();  
63         System.out.print("排序前:\t\t");  
64         quick.printPart(array, 0, array.length - 1);  
65         quick.quickSort(array, 0, array.length - 1);  
66         System.out.print("排序后:\t\t");  
67         quick.printPart(array, 0, array.length - 1);  
68     }  
69 }

[复制代码](javascript:void(0);)

**运行结果**

[复制代码](javascript:void(0);)

排序前:    1  3  4  5  2  6  9  7  8  0   
base = 1: 0  1  4  5  2  6  9  7  8  3   
base = 4:       3  2  4  6  9  7  8  5   
base = 3:       2  3   
base = 6:                5  6  7  8  9   
base = 7:                      7  8  9   
base = 8:                         8  9   
排序后:    0  1  2  3  4  5  6  7  8  9

[复制代码](javascript:void(0);)

**参考资料**

《数据结构习题与解析》（B级第3版）