本文内容是对快速排序的梳理和总结，本文内容包括：

快速排序(Quick Sort)
- 算法步骤
- 图解算法
- 代码实现
- 算法分析

1 快速排序(Quick Sort)

快速排序用到了分治思想，同样的还有归并排序。乍看起来快速排序和归并排序非常相似，都是将问题变小，先排序子串，最后合并。不同的是快速排序在划分子问题的时候经过多一步处理，将划分的两组数据划分为一大一小，这样在最后合并的时候就不必像归并排序那样再进行比较。但也正因为如此，划分的不定性使得快速排序的时间复杂度并不稳定。

快速排序的基本思想：通过一趟排序将待排序列分隔成独立的两部分，其中一部分记录的元素均比另一部分的元素小，则可分别对这两部分子序列继续进行排序，以达到整个序列有序。

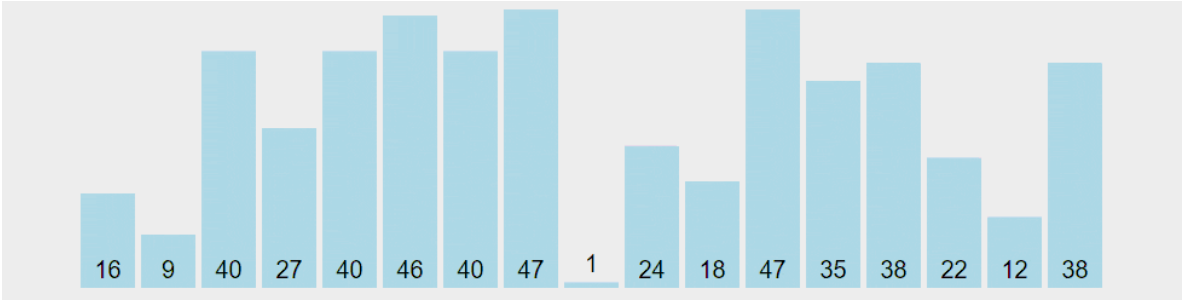
1.1 算法步骤

快速排序使用分治法策略来把一个序列分为较小和较大的 2 个子序列，然后递回地排序两个子序列。具体算法描述如下：

1. 从序列中**随机**挑出一个元素，做为 “基准”(`pivot`);
2. 重新排列序列，将所有比基准值小的元素摆放在基准前面，所有比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个操作结束之后，该基准就处于数列的中间位置。这个称为分区（partition）操作;
3. 递归地把小于基准值元素的子序列和大于基准值元素的子序列进行快速排序。

1.2 图解算法

如果下图不动，点击这里查看在线的图解



1.3 代码实现

```
public static void main(String[] args) {
    int[ ] data = { 19 , 22 , 98 , 100 , 56 , 77 , 45 , 72 , 0 , -1 };
    quickSort(data,0,data.length-1);
}
private static int partition(int[] arr, int startIndex, int endIndex) {
    int pivot = arr[startIndex];
    int leftPoint = startIndex;
    int rightPoint = endIndex;
    while (leftPoint < rightPoint) {
        // 从右向左找出比pivot小的数据
        while (leftPoint < rightPoint && arr[rightPoint] > pivot) {
            rightPoint--;
        }
        // 从左向右找出比pivot大的数据
        while (leftPoint < rightPoint  && arr[leftPoint] <= pivot) {
            leftPoint++;
        }
        // 没有过界则交换
        if (leftPoint < rightPoint) {
            int temp = arr[leftPoint];
            arr[leftPoint] = arr[rightPoint];
            arr[rightPoint] = temp;
        }
    }
    // 最终将分界值与当前指针数据交换
    arr[startIndex] = arr[rightPoint];
    arr[rightPoint] = pivot;
    // 返回分界值所在下标
    return rightPoint;
}
```

```java
public static void quickSort(int[] array, int low, int high)
{
    if (low < high) {
        int position = partition(array, low, high);
        quickSort(array, low, position - 1);
        quickSort(array, position + 1, high);
    }
}
```

1.4 算法分析

- **稳定性**： 不稳定
- **时间复杂度**： 最佳：O(nlogn)， 最差：O(n^2)，平均：O(nlogn)
- **空间复杂度**： O(nlogn)