链滴

Java 排序

作者：xixiaoming

原文链接：https://ld246.com/article/1502440216894

来源网站：链滴

排序算法的分类如下：

1. 插入排序（直接插入排序、折半插入排序、希尔排序）；
2. 交换排序（冒泡排序、快速排序）；
3. 选择排序（直接选择排序、堆排序）；
4. 归并排序；
5. 基数排序；

关于排序方法的选择：

1. 若n较小(如n≤50)，可采用直接插入或直接选择排序；
2. 若文件初始状态基本有序(指正序)，则应选用直接插人、冒泡或随机的快速排序为宜；
3. 若n较大，则应采用时间复杂度为O(nlgn)的排序方法：快速排序、堆排序或归并排序；

1. 冒泡排序

- 方法：相邻两元素进行比较
- 性能：比较次数O(n^2),n^2/2；交换次数O(n^2),n^2/4

```
public static int[] bubbleSort(int[] source) {
    for (int i = 1; i < source.length; i++) {
        for (int j = 0; j < i; j++) {
            if (source[j] > source[j + 1]) {
                swap(source, j, j + 1);
            }
        }
    }
    return source;
}
```

2. 直接选择排序

- 方法：每一趟从待排序的数据元素中选出最小（或最大）的一个元素， 顺序放在已排好序的数列的后，直到全部待排序的数据元素排完。
- 性能：比较次数O(n^2),n^2/2 交换次数O(n),n
- 说明：交换次数比冒泡排序少多了，由于交换所需CPU时间比比较所需的CUP时间多，所以选择排比冒泡排序快。但是N比较大时，比较所需的CPU时间占主要地位，所以这时的性能和冒泡排序差不多，但毫无疑问肯定要快些

```
public static int[] selectSort(int[] source) {

    for (int i = 0; i < source.length; i++) {
        for (int j = i + 1; j < source.length; j++) {
            if (source[i] > source[j]) {
                swap(source, i, j);
            }
```

```
        }
    }
    return source;
}
```

3. 插入排序

● 方法：将一个记录插入到已排好序的有序表（有可能是空表）中,从而得到一个新的记录数增1的有表。

● 性能：比较次数O(n^2),n^2/4

● 说明：复制次数O(n),n^2/4 比较次数是前两者的一般，而复制所需的CPU时间较交换少，所以性上比冒泡排序提高一倍多，而比选择排序也要快

```
public static int[] insertSort(int[] source) {

    for (int i = 1; i < source.length; i++) {
        for (int j = i; (j > 0) && (source[j] < source[j - 1]); j--) {
            swap(source, j, j - 1);
        }
    }
    return source;
}
```

4. 快速排序

● 方法：快速排序使用分治法（Divide and conquer）策略来把一个序列（list）分为两个子序列（sb-lists）。 步骤为：

1. 从数列中挑出一个元素，称为 "基准"（pivot），

2. 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（同的数可以到任一边）。在这个分割之后，该基准是它的最后位置。这个称为分割（partition）操作。

3. 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。递回的最底部形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递回下去，但是这个算法总会束，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去

```
public static int[] quickSort(int[] source) {
    return qsort(source, 0, source.length - 1);
}

private static int[] qsort(int source[], int low, int high) {
    int i, j, x;
    if (low < high) {
        i = low;
        j = high;
        x = source[i];
        while (i < j) {
            while (i < j && source[j] > x) {
                j--;
            }
            if (i < j) {
                source[i] = source[j];
```

```
                i++;
            }
            while (i < j && source[i] < x) {
                i++;
            }
            if (i < j) {
                source[j] = source[i];
                j--;
            }
        }
        source[i] = x;
        qsort(source, low, i - 1);
        qsort(source, i + 1, high);
    }
    return source;
}
```

5. 基础方法

```
public static int[] swap(int[] ints, int x, int y) {
    int temp = ints[x];
    ints[x] = ints[y];
    ints[y] = temp;
    return ints;
}
```