0x00 比较排序（冒泡（略）+快排）

> 快速排序

```
package me.wondertwo.quick_sort;

/**
 * O(nlogn) 不稳定  O(logn)
 *
 * Created by wondertwo on 2016/8/5.
 */
public class QuickSort {
    public static void main(String[] args) {
        int[] arr = {44, 99, 41, 17, 3, 54, 71, 10, 81, 0, 92,
                63, 59, 40, 27};
        print(arr);
        quickSort(arr, 0, arr.length-1);
        System.out.println("--------------------------------");
        print(arr);
    }

    private static void quickSort(int[] arr, int low, int high) {
        if (low >= high) return;
        int center = partition(arr, low, high);
        quickSort(arr, low, center-1);
        quickSort(arr, center+1, high);
    }

    private static int partition(int[] arr, int low, int high) {
        int centerValue = arr[getCenter(arr, low, high)]; //三数取中
        while (low < high) {
            while (low < high && arr[high]>= centerValue) {
                high--;
            }
            swap(arr, low, high);
            print(arr);
            while (low < high && arr[low] <= centerValue) {
                low++;
            }
            swap(arr, low, high);
            print(arr);
        }
        return low;
    }

    // find more suitable center value's index
    private static int getCenter(int[] arr, int low, int high) {
        int mid = (low + high) /2;
        if (arr[low] > arr[high]) {
            swap(arr, low, high);
        }
        if (arr[mid] > arr[high]) {
            swap(arr, mid, high);
```

```
        }
        if (arr[low] > arr[mid]) {
            swap(arr, low, mid);
        }
        return low; //???????????为什么返回low?
    }

    private static void swap(int[] data, int i, int j) {
        if (i == j) return;
        data[i] = data[i] + data[j];
        data[j] = data[i] - data[j];
        data[i] = data[i] - data[j];
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
        System.out.println();
    }
}
```

0x01 插入排序（直插 + 希尔）

> 直接插入排序

```
package me.wondertwo.insert_sort;

import java.util.Arrays;

/**
 * 直接插入排序    稳定  O(n^2)
 *
 * Created by wondertwo on 2016/8/2.
 */
public class InsertSort {
    public static void main(String[] args) {
        //length=8,其中arr[0]作为辅助空间，可传入原数组后做处理，
        //在数组最前面添加一个元素0，即arr[0]=0
        int[] arr = {0, 99, 41, 17, 3, 54, 71, 100};
        insertSort(arr);
        System.out.println(Arrays.toString(arr));
    }

    private static void insertSort(int[] arr) {
        int i, j;
        //以arr[1]作为基准，从arr[2]开始判断
        for (i = 2; i < arr.length; i++) {
            //将arr[i]插入有序子表
            if (arr[i] < arr[i-1]) {
                //作为辅助空间，暂存需要插入的值
                arr[0] = arr[i];
                for (j = i-1; arr[j] > arr[0]; j--) {
                    arr[j+1] = arr[j]; //记录后移
                }
                arr[j+1] = arr[0]; //插入到正确的位置
```

```
            }
        }
        //循环结束，排序结束。arr[0]为辅助空间，将其值重新恢复为0
        arr[0] = 0;
    }
}
```

希尔排序

```
package me.wondertwo.shell_sort;

/**
 * 希尔排序    不稳定    O(nlogn)
 *
 * Created by wondertwo on 2016/8/2.
 */
public class ShellSort {
    public static void main(String[] args) {
        //length=8,其中arr[0]作为辅助空间，可传入原数组后做处理，
        //在数组最前面添加一个元素0，即arr[0]=0
        int[] arr = {0, 99, 41, 17, 3, 54, 71, 100, 101, 29};
        print(arr);
        shellSort(arr);
        print(arr);
    }

    private static void shellSort(int[] arr) {
        int i, j;
        int increment = arr.length; //定义分割间隔
        do {
            //通过increment把数组分为前后两部分
            increment = increment/3 + 1;
            //此时increment标记后部分元素的首位元素下标，记为后1,从后1开始循环，
            //依次成对和前部分数组元素进行大小比较,即后1和前1进行比较，后2和前2比较，
            //前1是指arr[1]，因为arr[0]是辅助空间
            for (i = increment+1; i < arr.length; i++) {
                //比较后1和1前1的大小关系，若后1值较小则进行位置交换
                if (arr[i] < arr[i-increment]) {
                    //arr[0]作为辅助空间，暂存后1值
                    arr[0] = arr[i];
                    //此处循环的作用：将前1值赋值给后1
                    for (j=i-increment; j>0&&arr[0]<arr[j]; j-=increment) {
                        arr[j+increment] = arr[j];
                    }
                    //将arr[0]赋值给前1
                    arr[j+increment] = arr[0];
                }
            }
        } while (increment > 1);
        //循环结束，排序结束。arr[0]为辅助空间，将其值重新恢复为0
        arr[0] = 0;
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
```

```
            System.out.println();
        }
    }
```

0x02 选择排序（简单选择（略） + 堆排）

> 堆式排序一

```
package me.wondertwo.heap_sort;

/**
 * 堆排序  不稳定   O(nlogn)
 *
 * Created by wondertwo on 2016/8/3.
 */
public class HeapSort {
    public static void main(String[] args) {
        int[] arr = {0, 99, 41, 17, 71, 44, 92, 63, 59, 40, 27};
        print(arr);
        heapSort(arr);
        print(arr);
    }

    private static void heapSort(int[] arr) { //递归实现
        int n = arr.length; //数组元素个数
        //对所有数组元素构建大顶堆
        for(int i = n/2-1; i >= 0; i--) {toMaxHeap(arr, i, n);}
        for(int j = n-1; j >= 1; j--) {
            //此时堆顶元素是最大元素，交换堆顶元素和数组最后一位元素
            swap(arr, 0, j);
            //继续把前面n-1个元素构建大顶堆
            toMaxHeap(arr, 0, j);
        }
    }

    private static void toMaxHeap(int[] arr, int idx, int max) {
        int left = 2*idx + 1; //左孩子下标
        int right = 2*idx + 2; //左孩子下标
        int largest; //寻找3个节点中最大值节点下标
        if(left < max && arr[left] > arr[idx]) {
            largest = left;
        } else {
            largest = idx;
        }
        if(right < max && arr[right] > arr[largest]) {
            largest = right;
        }
        if(largest != idx) {
            swap(arr, largest, idx);
            toMaxHeap(arr, largest, max); //递归
        }
    }

    private static void swap(int[] data, int i, int j) {
```

```
        if (i == j) {return;}
        data[i] = data[i] + data[j];
        data[j] = data[i] - data[j];
        data[i] = data[i] - data[j];
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
        System.out.println();
    }
}
```

堆式排序二

```
package me.wondertwo.heap_sort;

/**
 * Created by wondertwo on 2016/8/4.
 */
public class HeapSort2 {

    public static void main(String[] args) {
        int[] arr = new int[] { 5, 3, 6, 2, 1, 9, 4, 8, 7 };
        print(arr);
        heapSort(arr);
        print(arr);
    }

    private static void heapSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
            createMaxHeap(data, data.length - 1 - i);
            swap(data, 0, data.length - 1 - i);
            print(data);
        }
    }

    private static void createMaxHeap(int[] data, int lastIndex) {
        for (int i = (lastIndex - 1) / 2; i >= 0; i--) {
            // 保存当前正在判断的节点
            int k = i; // 若当前节点的子节点存在
            while (2 * k + 1 <= lastIndex) {
                // biggerIndex总是记录较大节点的值,先赋值为当前判断节点的左子节点
                int biggerIndex = 2 * k + 1;
                if (biggerIndex < lastIndex) {
                    // 若右子节点存在，否则此时biggerIndex应该等于 lastIndex
                    if (data[biggerIndex] < data[biggerIndex + 1]) {
                        // 若右子节点值比左子节点值大，则biggerIndex++
                        biggerIndex++;
                    }
                }
                if (data[k] < data[biggerIndex]) {
                    // 若当前节点值小于子节点最大值，则交换两者的值
                    swap(data, k, biggerIndex);
                    k = biggerIndex; // 交换后将biggerIndex值赋值给k
                } else {
```

```
                    break;
                }
            }
        }
    }

    private static void swap(int[] data, int i, int j) {
        if (i == j) {return;}
        data[i] = data[i] + data[j];
        data[j] = data[i] - data[j];
        data[i] = data[i] - data[j];
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
        System.out.println();
    }
}
```

0x03 归并排序

```
package me.wondertwo.merge_sort;

/**
 * 归并排序     稳定     O(n+logn)
 *
 * Created by wondertwo on 2016/8/4.
 */
public class MergeSort {
    public static void main(String[] args) {
        int[] arr = {99, 41, 100, 71, 54, 101, 44, 92, 63, 59, 40, 27};
        print(arr);
        mergeSort(arr);
        print(arr);
    }

    // 归并排序
    private static void mergeSort(int[] data) {
        sort(data, 0, data.length - 1);
    }

    private static void sort(int[] data, int left, int right) {
        if (left >= right) return;
        int center = (left + right) / 2;  // 找出中间索引
        sort(data, left, center);         // 对左边数组进行递归
        sort(data, center + 1, right);    // 对右边数组进行递归
        merge(data, left, center, right); // 合并
        // print(data);// 打印排序过程
    }

    /**
     * 将两个数组进行归并，归并前面2个数组已有序，归并后依然有序
     * @param data    数组对象
```

```
     * @param left   左数组的第一个元素的索引
     * @param center 左数组的最后一个元素的索引，center+1是右数组第一个元素的索引
     * @param right  右数组最后一个元素的索引
     */
    private static void merge(int[] data, int left, int center, int right) {
        // 临时数组
        int[] tmpArr = new int[data.length];
        // 右数组第一个元素索引
        int mid = center + 1;
        // third 记录临时数组的索引
        int third = left;
        // 缓存左数组第一个元素的索引
        int tmp = left;
        while (left <= center && mid <= right) {
            // 从两个数组中取出最小的放入临时数组
            if (data[left] <= data[mid]) {
                tmpArr[third++] = data[left++];
            } else {
                tmpArr[third++] = data[mid++];
            }
        }
        // 剩余部分依次放入临时数组（实际上两个while只会执行其中一个）
        while (mid <= right) {
            tmpArr[third++] = data[mid++];
        }
        while (left <= center) {
            tmpArr[third++] = data[left++];
        }
        // 将临时数组中的内容拷贝回原数组中
        // 原left-right范围的内容被复制回原数组
        while (tmp <= right) {
            data[tmp] = tmpArr[tmp++];
        }
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
        System.out.println();
    }
}
```

0x04 基数排序（计数 + 基数 + 桶排）

> 计数排序

```
package me.wondertwo.radix_sort.counting_sort;

/**
 * 计数排序      稳定       线性时间
 *
 * Created by wondertwo on 2016/8/6.
 */
public class CountingSort {
    public static void main(String[] args) {
```

```
        //待排序元素所在区间[0, max]， 所以待排元素最大值为max=100
        int[] arr = {64, 44, 99, 41, 17, 44, 54, 71, 10, 81, 100, 0,
                92, 63, 71, 40, 27};
        print(arr);
        System.out.println("--------------------------------");
        print(countingSort(arr, 100));
    }

    private static int[] countingSort(int[] arr, int max) {
        // 辅助数组，存储最终的输出序列
        int[] OutputTemp = new int[arr.length];
        // 记录数组，记录每个输入元素在输出序列中的位置
        int[] CacheCount = new int[max+1];

        // 初始化记录数组，初始值0
        for (int b = 0; b <= max; b++) {CacheCount[b] = 0;}

        // 循环遍历，统计每个数据出现的次数，保存在CacheCount相应位置
        for (int anArr : arr) {
            CacheCount[anArr] = CacheCount[anArr] + 1;
        }
        // 循环结束，记录数组中已经记录着每个输入元素的个数

        // 从第一个位置开始遍历记录数组，满足条件：
        // CacheCount[j] = CacheCount[j] + CacheCount[j-1]
        // 此公式可计算出每个输入元素，对应在输出序列中位置
        for (int j = 1; j <= max; j++) {
            CacheCount[j] += CacheCount[j-1];
        }

        // 倒叙遍历待排序数组，将每个元素放到辅助数组中其对应的输出位置即可
        for (int t = arr.length-1; t >= 0; t--) {
            // 为了与待排数组下标对应，输出数组OutputTemp[]下标减1
            OutputTemp[CacheCount[arr[t]]-1] = arr[t];
            // 可能待排数组中有重复元素，CacheCount 减1后，
            // 方便第二个出现的重复元素放在倒数第二个重复的位置上
            CacheCount[arr[t]]--;
        }
        return OutputTemp;
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
        System.out.println();
    }
}
```

基数排序

```
package me.wondertwo.radix_sort.radix_sort;

import java.util.Arrays;

/**
 * 基数排序    稳定    线性时间
```

```
 * Created by wondertwo on 2016/8/6.
 */
public class RadixSort {
    public static void main(String[] args) {
        int[] data = {1100, 192, 101, 998, 34, 901, 2344, 7689,
                41, 221, 12, 23};
        print(data);
        radixSort(data, 10, 4);
        System.out.println("--------------------");
        print(data);
    }

    /**
     * @param radix 待排数据为十进制，为计算每个数据的子关键字，radix=10
     * @param d     待排数据最高位是 d 位数
     */
    private static void radixSort(int[] data, int radix, int d) {
        // 缓存数组
        int[] tmpArr = new int[data.length];
        // buckets用于记录待排序元素的信息
        // buckets数组定义了max-min个桶
        int[] buckets = new int[radix];

        // 低位优先，循环变量 i 由0到(d-1)
        for (int i = 0, rate = 1; i < d; i++) {

            // 重置 buckets 数组，开始统计下一个关键字
            Arrays.fill(buckets, 0);
            // 将data中的元素完全复制到tmp数组中
            System.arraycopy(data, 0, tmpArr, 0, data.length);

            // 计算每个待排序数据的子关键字
            for (int j = 0; j < data.length; j++) {
                int subKey = (tmpArr[j] / rate) % radix;
                buckets[subKey]++;
            }

            for (int j = 1; j < radix; j++) {
                buckets[j] = buckets[j] + buckets[j - 1];
            }

            // 按子关键字对指定的数据进行排序
            for (int m = data.length - 1; m >= 0; m--) {
                int subKey = (tmpArr[m] / rate) % radix;
                data[--buckets[subKey]] = tmpArr[m];
            }
            rate *= radix;
        }
    }

    private static void print(int[] data) {
        for (int aData : data) {System.out.print(aData + "\t");}
        System.out.println();
    }
}
```

桶式排序

```
package me.wondertwo.radix_sort.bucket_sort;

import java.util.ArrayList;
import java.util.Collections;

/**
 * 桶排序    稳定    线性时间
 *
 * Created by wondertwo on 2016/8/5.
 */
public class BucketSort {

    public static void main(String[] args) {
        // 所有待排元素满足：均匀分布在区间[0, 1)
        double[] array = { 0.78, 0.17, 0.39, 0.26, 0.72, 0.94,
                0.21, 0.12, 0.23, 0.68 };
        print(array);
        bucketSort(array);
        System.out.println("----------------------");
        print(array);
    }

    private static void bucketSort(double[] array) {
        /**
         *  把待排元素区间均分为 bucketNum 个桶
         *  每个桶是一个list，存放落在此桶上的元素
         */
        int bucketNum = array.length; // 桶的个数
        ArrayList[] arrList = new ArrayList[bucketNum];

        // 存放落在每个桶中的元素
        for (double anArray : array) {
            //0.7到0.79放在第8个桶里，编号7；第一个桶放0到0.09
            int temp = (int) Math.floor(10 * anArray);
            if (null == arrList[temp]) {
                arrList[temp] = new ArrayList();
            }
            arrList[temp].add(anArray);
        }

        // 对每个桶中的数进行插入排序
        for (int i = 0; i < bucketNum; i++) {
            if (null != arrList[i]) {
                // 此处排序方法不定，越快越好，除了三大线性排序外，都没有
                // Collections和Arrays里的sort好，因为这是调优后的快排，
                // Arrays里也有，在基数排序里用过copyOf和fill方法
                Collections.sort(arrList[i]);
            }
        }

        // 输出类似鸽巢排序
        int index = 0;
```

```
        for (int i = 0; i < bucketNum; i++) {
            if (null != arrList[i]) {
                for (Object o : arrList[i]) {
                    Double d = (Double) o;
                    array[index] = d;
                    index++;
                }
            }
        }
    }

    private static void print(double[] data) {
        for (double aData : data) {
            System.out.print(aData + "\t");
        }
        System.out.println();
    }
}
```