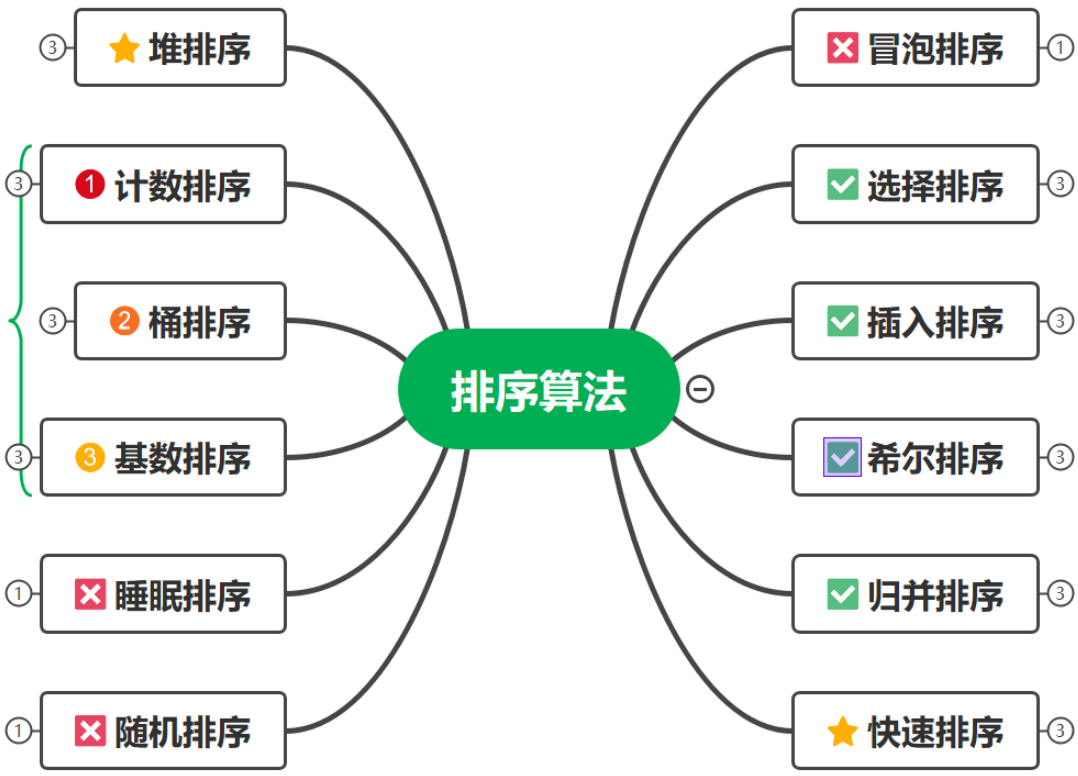
代码小抄：十大排序算法（Java实现）v1.1

整体把握



一、冒泡排序

```java
/*
 * 冒泡排序
 * 比较交换相邻元素
 * 思路简单，仅学习用
 */
public class Bubble {
    private static void sort(int[] nums) {
        int n = nums.length;
        for (int i = 0; i < n - 1; i++) {            // i表示（末尾）排序完成的个数
            for (int j = 0; j < n - i - 1; j++) {    // 注意j和j+1比，所以n-i后再-1
                if (nums[j] > nums[j + 1]) {         // 比较交换
                    int temp = nums[j];
                    nums[j] = nums[j + 1];
                    nums[j + 1] = temp;
                }
            }
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

二、选择排序

```java
/*
 * 选择排序
 * 思路：找最小元素，放到最终位置
 * 特点：时间：O(n²)、非稳定排序
 * 适用：数据量少
 */
public class Select {
    private static void sort(int[] nums) {
        int n = nums.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {   // j从i+1开始找最值
                if (nums[j] < nums[minIndex]) { // 更新最值位置
                    minIndex = j;
                }
            }
            // 交换，把最值放到正确位置
            int temp = nums[i];
            nums[i] = nums[minIndex];
            nums[minIndex] = temp;
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

三、插入排序

```java
/*
 * 插入排序
 * 思路：抓牌一样，插入当前手牌中的适当位置
 * 特点：时间：O(n²)
 * 适用：基本有序
 */
public class Insert {
    private static void sort(int[] nums) {
        int n = nums.length;
        int pos, cur;

        for (int i = 1; i < n; i++) {   // 第一张牌（i=0）在手里，从第二张牌（i=1）开始摸
            pos = i - 1;                // 手里最大（最右边）的牌
            cur = nums[i];              // 当前摸的牌。注意必须存到cur！因为pos+1可能就是i，可能被修改！
            while (pos >= 0 && cur < nums[pos]) {
                // 退出时：要么pos==-1，要么cur>=nums[pos]
                // 即：要么没找到，摸得牌放最左；要么找到小的，放在小的右边
                nums[pos + 1] = nums[pos];
                pos--;
            }
            nums[pos + 1] = cur;        // 找到位置，放牌
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

四、希尔排序

```java
/*
 * 希尔排序
 * 思路：间隔分组+自定义排序（这里给出的是冒泡）
 * 特点：时间：O(nlogn)、非稳定排序
 * 适用：数据量大
 */
public class Shell {
    private static void sort(int[] nums) {
        int n = nums.length;
        int gap = n / 2;
        while (gap > 0) {
            for (int j = gap; j < n; j++) { // j分别取第一组、第二组、第x组的最后一个值
                int i = j;                  // 对于每组，让i从后往前，比较交换
                while ((i >= gap && nums[i - gap] > nums[i])) { // 冒泡排序
                    int temp = nums[i];
                    nums[i] = nums[i - gap];
                    nums[i - gap] = temp;
                    i -= gap;
                }
            }
            gap = gap / 2;
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

五、归并排序

```java
/*
 * 归并排序
 * 思路：递归思想
 * 特点：时间：O(nlogn)、空间：O(n)--非原地
 * 适用：不受数据影响，所需空间与n成正比
 */
public class Merge {
    private static void sort(int[] nums, int start, int end) {
        // 如果start==end，数组中只有一个元素，则不用排了
        if (start < end) {
            // 分成两半
            int mid = (start + end) / 2;
            // 分别排序
            sort(nums, start, mid);
            sort(nums, mid + 1, end);
            // 进行合并
            merge(nums, start, end);
        }
    }

    // 辅助函数：合并两个有序数组
    private static void merge(int[] nums, int left, int right) {
        int[] tmp = new int[nums.length];       // 辅助数组tmp，暂存有序结果
        int mid = left + (right - left) / 2;
        int p1 = left;
        int p2 = mid + 1;
        int k = left;                           // 注意：k初始化为left（因为tmp与
nums位置要一一对应）
        while (p1 <= mid && p2 <= right) {
            if (nums[p1] <= nums[p2])
                tmp[k++] = nums[p1++];
            else
                tmp[k++] = nums[p2++];
        }
        while (p1 <= mid) {
            tmp[k++] = nums[p1++];
        }
        while (p2 <= right) {
            tmp[k++] = nums[p2++];
        }
        for (int i = left; i <= right; i++)
            nums[i] = tmp[i];
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums, 0, nums.length - 1);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

六、快速排序

```java
/*
 * 快速排序
 * 思路：选择中轴元素，比它小的放左，比它大的放右（代码过程很像 小丑在扔三个小球）
 * 特点：时间：O(nlogn)、空间：O(logn)、非稳定
 * 适用：广泛（最快）
 */
public class Quick {
    private static void sort(int[] nums, int start, int end) {
        // 终止条件
        if (start >= end)   return;

        // 核心代码
        // temp记录标记点，left和right在移动；
        // 此时left位置相当于空出，可以放置
        int left = start;
        int right = end;
        int temp = nums[left];
        // 抛三个球
        while (left < right) {
            while (left < right && nums[right] >= temp) // 位置符合
                right--;
            nums[left] = nums[right];                   // 位置不符：放到左边空位，并把自己当前位置空出
            while (left < right && nums[left] <= temp)  // 位置符合
                left++;
            nums[right] = nums[left];                   // 位置不符：放到右边空位，并把自己当前位置空出
        }
        nums[left] = temp;                              // 退出while时left==right，这是个空位，把temp标记点的数放入

        // 递归：现在以left/right为边界，左边是比它小的，右边是比它大的
        sort(nums, start, left - 1);
        sort(nums, left + 1, end);
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums, 0, nums.length - 1);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

七、堆排序

```
/*
 * 堆排序
 * 思路：升序用大顶堆，每次调整后把最大的移出，再调整...
 * 特点：时间：O(nlogn)、非稳定
 * 适用：数据结构学习
 */
public class Heap {
    public static void sort(int[] list) {
        // （1）构造初始堆
        // 从第一个非叶子节点（倒数第二行最后一个）开始调整
        // 左右孩子节点中较大的交换到父节点中
        // 注意这里i是自底往上的!
        for (int i = (list.length) / 2 - 1; i >= 0; i--) {
            headAdjust(list, list.length, i);
        }
        // （2）排序
        // 将最大的节点list[0]放在堆尾list[i]
        // 然后从根节点重新调整
        // 由于（1）的存在，这里每次最大值都会在堆顶那三个里面产生（这个想法不知道对不对，严谨性待证明）
        // 这里的i代表len，即每次把最后一个排好的忽略掉
        for (int i = list.length - 1; i >= 1; i--) {
            int temp = list[0];
            list[0] = list[i];
            list[i] = temp;
            headAdjust(list, i, 0);
        }
    }

    // 辅助函数：调整堆
    // 参数说明：list代表整个二叉树、len是list的长度、 i代表三个中的根节点
    private static void headAdjust(int[] list, int len, int i) {
        int index = 2 * i + 1; // 左孩子

        // 这步while的意义在于把较小的沉下去，把较大的提上来
        //使得最大值总在最顶上三个里面产生（这个想法不知道对不对，严谨性待证明）
        while (index < len) {
            // （1）index指向左右孩子较大的那个
            if (index + 1 < len) { // 说明还有右孩子
                if (list[index] < list[index + 1]) {
                    index = index + 1;
                }
            }
            // （2）比较交换大孩子和根节点
            if (list[index] > list[i]) {
                //交换
                int temp = list[i];     // temp暂存，现在根空了
                list[i] = list[index];  // 更新根，现在list[index]空了
                list[index] = temp;
                //更新
                i = index;
                index = 2 * i + 1;      // index指向孩子的孩子，继续
            } else {
                break;
```

```
            }
        }

    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

八、计数排序

```java
/*
 * 计数排序
 * 思路：借助足够大的辅助数组，把数字排在一个相对位置不会错的地方，最后并拢
 * 特点：时间：O(n+k)、空间：O(n+k)--非原地
 * 适用：max和min的差值不大
 */
public class Count {
    public static void sort(int[] nums, int min, int max) {
        int n = nums.length;
        int[] temp = new int[max - min + 1];
        // 哈希表
        for (int i = 0; i < n; i++) {
            temp[nums[i] - min]++;
            // 把数值信息蕴含在下标中
            // 下标 = 数值 - min
            // 即i = nums[i] - min
        }

        int index = 0; // 用于归拢
        for (int i = 0; i < temp.length; i++) {
            int cnt = temp[i];
            while (cnt != 0) {
                // 因为：下标 = 数值 - min
                // 所以：数值 = 下标 + min
                // 即 nums[i] = i + min
                nums[index] = i + min;
                index++;    // 指针++
                cnt--;      // 个数--
            }
        }
    }

    public static void main(String[] args) {
        int[] a = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 0 };
        sort(a, 0, 9);
        for (int aa : a)
            System.out.print(aa + " ");
    }
}
```

九、桶排序

```
/*
 * 桶排序
 * 思路：先粗略分类分桶，再各桶排序
 * 特点：时间：O(n+k)、空间：O(n+k)--非原地
 * 适用：均匀分布的数据
 */
public class Bucket {
    public static void sort(float[] nums) {
        // （1）初始化桶
        // 由于桶内元素会频繁的插入，所以选择 LinkedList作为桶的数据结构
        ArrayList<LinkedList<Float>> buckets = new ArrayList<LinkedList<Float>>
(); // 桶的集合
        for (int i = 0; i < 10; i++) {
            buckets.add(new LinkedList<Float>());   // 添加桶
        }
        // （2）数据放入桶中并完成排序
        for (float data : nums) {
            int index = getBucketIndex(data);       // 哪个桶
            insertSort(buckets.get(index), data);   // 桶编号，数据
        }
        // （3）从桶取出数据，放回nums
        int index = 0;
        for (LinkedList<Float> bucket : buckets) {
            for (Float data : bucket) {
                nums[index++] = data;
            }
        }
    }

    // 辅助函数一 ：计算得到输入元素应该放到哪个桶内（粗略分类）
    public static int getBucketIndex(float data) {
        // 这里例子写的比较简单，仅使用浮点数的整数部分作为其桶的索引值
        // 实际开发中需要根据场景具体设计
        return (int) data;
    }

    // 辅助函数二 ：选择插入排序作为桶内元素排序的方法（选择各桶排序方法）
    public static void insertSort(List<Float> bucket, float data) {
        ListIterator<Float> it = bucket.listIterator();
        boolean insertFlag = true;
        while (it.hasNext()) {
            if (data <= it.next()) {
                it.previous();  // 把迭代器的位置偏移回上一个位置
                it.add(data);   // 把数据插入到迭代器的当前位置
                insertFlag = false;
                break;
            }
        }
        if (insertFlag) {
            bucket.add(data);   // 否则把数据插入到链表末端
        }
    }

    //测试代码
```

```
    public static void main(String[] args) {
        // 注意输入元素均在 [0, 10) 这个区间内
        float[] nums = new float[] { 0.12f, 2.6f, 8.8f, 7.6f, 7.2f, 6.3f, 9.0f,
1.6f, 5.6f, 2.4f };
        sort(nums);
        System.out.println(Arrays.toString(nums));
    }
}
```

十、基数排序

```java
/*
 * 基数排序
 * 思路：桶排序的一种。  按数位分桶：从低位开始 -> 分桶、收集 -> 下一位...
 * 特点：时间：O(kn)、空间：O(n+k)--非原地
 * 适用：max和min的差值不大
 */
public class Radix {
    private static void sort(int[] nums, int d) {
        int n = 1;                                  // 看num从右往左第n位，相同的放在一个桶
        int length = nums.length;
        int[][] bucket = new int[10][length];    // 10是因为每位的可能数字只有0~9；
                                                    // length是因为最多length个放在同一桶
        int[] order = new int[length];           // order记录各桶放了多少个数字
        while (n < d) {                             // 当前第n位，最大第d位
            // （1）进行一轮分桶
            for (int num : nums) {
                int digit = (num / n) % 10;          // 获取num从右往左第n位，记作digit
                bucket[digit][order[digit]] = num;  // 把num放入digit桶里的order[digit]位置
                order[digit]++;                      // digit对应桶里的个数++
            }
            // （2）进行一轮排序
            int k = 0;                               // 用于遍历原数组
            for (int i = 0; i < length; i++)         // 按桶号遍历桶
            {
                if (order[i] != 0)                   // 如果该桶有数据
                {
                    for (int j = 0; j < order[i]; j++) {    // 遍历并保存（覆盖）
                        nums[k] = bucket[i][j];              // 第i桶第j个加入nums
                        k++;
                    }
                }
                order[i] = 0;                        // 重置各桶计数器
            }
            n *= 10;
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = { 5, 789, 2138, 456, 3, 1, 9, 1, 13, 4984, 3 };
        sort(nums, 10000);
        System.out.println(Arrays.toString(nums));  // 值得学习的巧妙输出！
                                                    //（因为我之前用C++刷题所以觉得这个好好用）
    }
}
```

EX1、睡眠排序

```java
/*
 * 睡眠排序
 * 思路：娱乐算法：睡醒了起来报数
 */
public class Sleep {
    public static void sleepSort(int[] array) {
        for (int i : array) {
            new Thread(() -> {
                try {
                    Thread.sleep(i);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                System.out.println(i);
            }).start();
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] array = { 10, 30, 50, 60, 100, 40, 150, 200, 70 };
        sleepSort(array);
    }
}
```

EX2、随机排序

```java
/*
 * 随机排序
 * 思路：碰运气瞎排，再看蒙对没有
 */
public class Random {
    public static void randSortX(int[] array) {
        List<Integer> list = new ArrayList<>();
        for (Integer integer : array) {
            list.add(integer);
        }
        int pre = 0;
        int index = 0;
        while (true) {
            pre = 0;
            for (index = 1; index < list.size(); index++) {
                if (list.get(index) > list.get(pre)) {
                    pre++;
                } else {
                    break;
                }
            }
            if (pre + 1 == list.size()) {
                break;
            }
            Collections.shuffle(list);
        }
        System.out.println(list.toString());
    }

    // 测试代码
    public static void main(String[] args) {
        int[] array = { 10, 30, 50, 60, 100, 40, 150, 200, 70 };
        randSortX(array);
    }
}
```