

# 十大经典排序算法动画与解析，看我就够了！（配代码完全版）

程序员小吴 程序员乔戈里 今天



排序算法是《数据结构与算法》中最基本的算法之一。

排序算法可以分为内部排序和外部排序。

内部排序是数据记录在内存中进行排序。

而外部排序是因排序的数据很大，一次不能容纳全部的排序记录，在排序过程中需要访问外存。

常见的内部排序算法有：插入排序、希尔排序、选择排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。

用一张图概括：

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

时间复杂度与空间复杂度

关于时间复杂度：

1. 平方阶 ( $O(n^2)$ ) 排序 各类简单排序：直接插入、直接选择和冒泡排序。
2. 线性对数阶 ( $O(n\log 2n)$ ) 排序 快速排序、堆排序和归并排序；
3.  $O(n^1+\delta)$  排序， $\delta$  是介于 0 和 1 之间的常数。希尔排序
4. 线性阶 ( $O(n)$ ) 排序 基数排序，此外还有桶、箱排序。

## 关于稳定性：

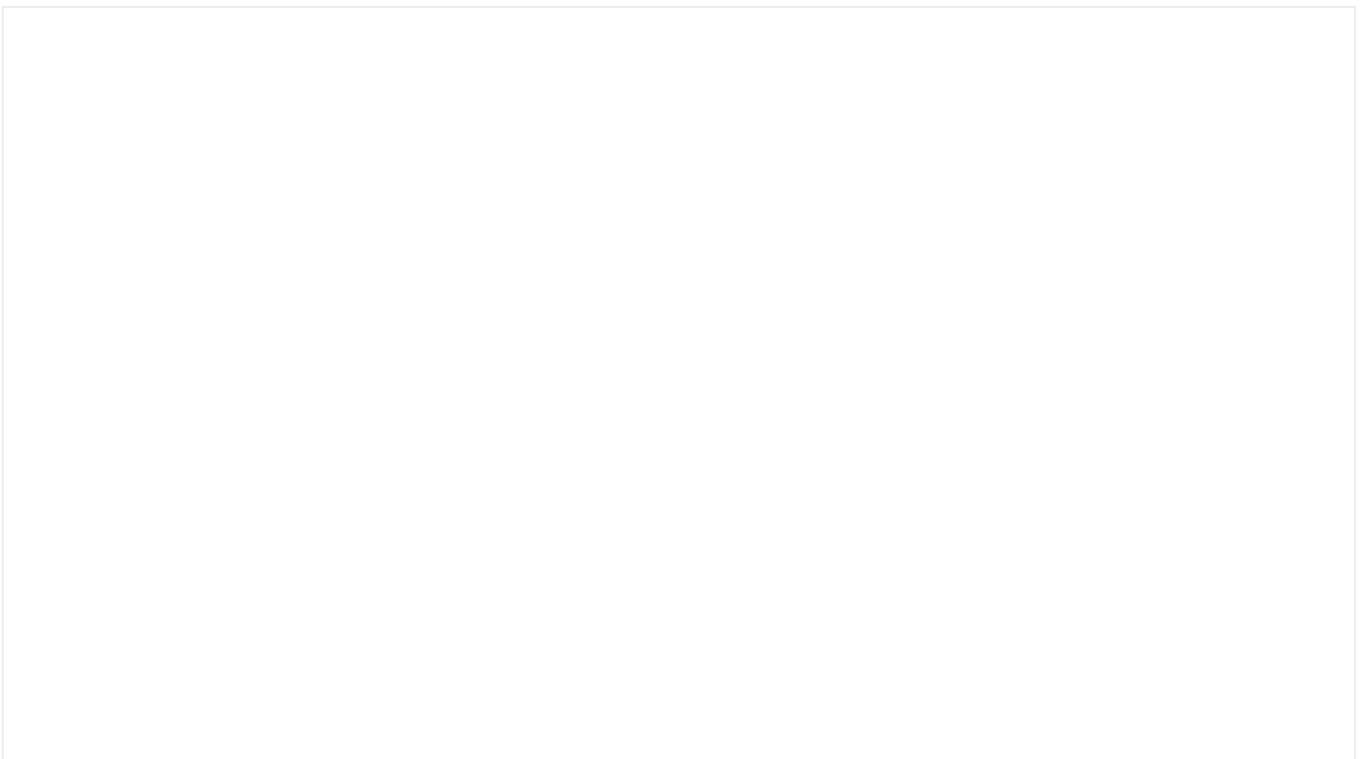
1. 稳定的排序算法：冒泡排序、插入排序、归并排序和基数排序。
2. 不是稳定的排序算法：选择排序、快速排序、希尔排序、堆排序。

## 1. 冒泡排序

### 1.1 算法步骤

- 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
- 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。
- 针对所有的元素重复以上的步骤，除了最后一个。
- 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

### 1.2 动画演示



冒泡排序动画演示

## 1.3 参考代码

```
1// Java 代码实现
2public class BubbleSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        // 对 arr 进行拷贝, 不改变参数内容
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
8
9        for (int i = 1; i < arr.length; i++) {
10            // 设定一个标记, 若为true, 则表示此次循环没有进行交换, 也就是待排序列已经有序, 排序已经完成。
11            boolean flag = true;
12
13            for (int j = 0; j < arr.length - i; j++) {
14                if (arr[j] > arr[j + 1]) {
15                    int tmp = arr[j];
16                    arr[j] = arr[j + 1];
17                    arr[j + 1] = tmp;
18
19                    flag = false;
20                }
21            }
22
23            if (flag) {
24                break;
25            }
26        }
27        return arr;
28    }
29}
```

## 2. 选择排序

### 2.1 算法步骤

- 首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置
- 再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。
- 重复第二步，直到所有元素均排序完毕。

### 2.2 动画演示

选择排序动画演示

## 2.3 参考代码

```
1//Java 代码实现
2public class SelectionSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
7
8        // 总共要经过 N-1 轮比较
9        for (int i = 0; i < arr.length - 1; i++) {
10            int min = i;
11
12            // 每轮需要比较的次数 N-i
13            for (int j = i + 1; j < arr.length; j++) {
14                if (arr[j] < arr[min]) {
15                    // 记录目前能找到的最小值元素的下标
16                    min = j;
17                }
18            }
19
20            // 将找到的最小值和i位置所在的值进行交换
21            if (i != min) {
22                int tmp = arr[i];
23                arr[i] = arr[min];
24                arr[min] = tmp;
25            }
26
27        }
28        return arr;
29    }
30}
```

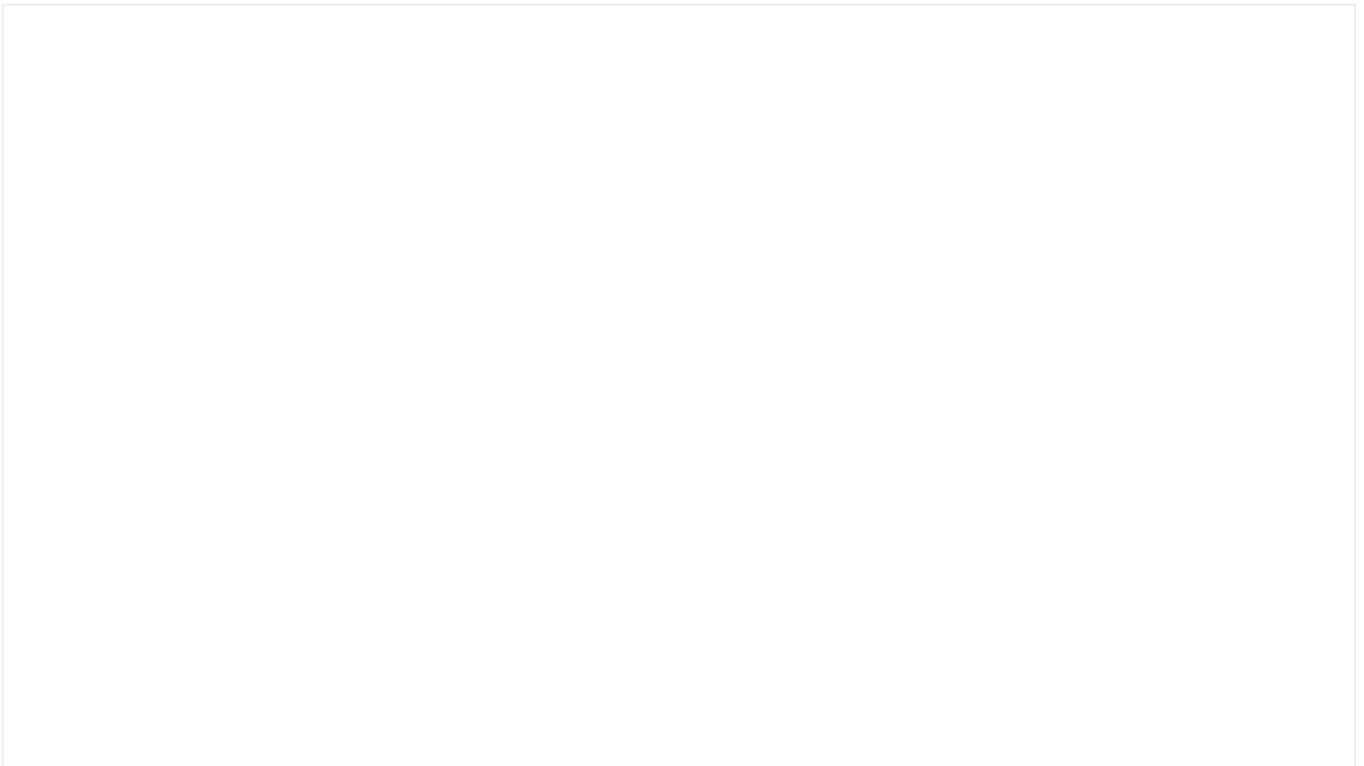
```
29    }  
30 }
```

## 3. 插入排序

### 3.1 算法步骤

- 将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。
- 从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。（如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。）

### 3.2 动画演示



插入排序动画演示

### 3.3 参考代码

```
1//Java 代码实现  
2public class InsertSort implements IArraySort {  
3  
4    @Override  
5    public int[] sort(int[] sourceArray) throws Exception {  
6        // 对 arr 进行拷贝，不改变参数内容  
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);  
8  
9        // 从下标为1的元素开始选择合适的位置插入，因为下标为0的只有一个元素，默认是有序的  
10       for (int i = 1; i < arr.length; i++) {
```

```
11
12      // 记录要插入的数据
13      int tmp = arr[i];
14
15      // 从已经排序的序列最右边的开始比较, 找到比其小的数
16      int j = i;
17      while (j > 0 && tmp < arr[j - 1]) {
18          arr[j] = arr[j - 1];
19          j--;
20      }
21
22      // 存在比其小的数, 插入
23      if (j != i) {
24          arr[j] = tmp;
25      }
26
27  }
28  return arr;
29  }
30 }
```

## 4. 希尔排序

### 4.1 算法步骤

- 选择一个增量序列  $t_1, t_2, \dots, t_k$ , 其中  $t_i > t_j, t_k = 1$ ;
- 按增量序列个数  $k$ , 对序列进行  $k$  趟排序;
- 每趟排序, 根据对应的增量  $t_i$ , 将待排序列分割成若干长度为  $m$  的子序列, 分别对各子表进行直接插入排序。仅增量因子为 1 时, 整个序列作为一个表来处理, 表长度即为整个序列的长度。

### 4.2 动画演示

希尔排序动画演示

## 4.3 参考代码

```
1//Java 代码实现
2public class ShellSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        // 对 arr 进行拷贝, 不改变参数内容
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
8
9        int gap = 1;
10       while (gap < arr.length) {
11           gap = gap * 3 + 1;
12       }
13
14       while (gap > 0) {
15           for (int i = gap; i < arr.length; i++) {
16               int tmp = arr[i];
17               int j = i - gap;
18               while (j >= 0 && arr[j] > tmp) {
19                   arr[j + gap] = arr[j];
20                   j -= gap;
21               }
22               arr[j + gap] = tmp;
23           }
24           gap = (int) Math.floor(gap / 3);
25       }
26
27       return arr;
28   }
29}
```

## 5. 归并排序

### 5.1 算法步骤

- 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列；
- 设定两个指针，最初位置分别为两个已经排序序列的起始位置；
- 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置；
- 重复步骤 3 直到某一指针达到序列尾；
- 将另一序列剩下的所有元素直接复制到合并序列尾。

### 5.2 动画演示



归并排序动画演示

### 5.3 参考代码

```
1//Java 代码实现
  public class MergeSort implements IArraySort {
2
3    @Override
4    public int[] sort(int[] sourceArray) throws Exception {
5        // 对 arr 进行拷贝，不改变参数内容
6        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
7    }
```



```
8         if (arr.length < 2) {
9             return arr;
10        }
11        int middle = (int) Math.floor(arr.length / 2);
12
13        int[] left = Arrays.copyOfRange(arr, 0, middle);
14        int[] right = Arrays.copyOfRange(arr, middle, arr.length);
15
16        return merge(sort(left), sort(right));
17    }
18
19    protected int[] merge(int[] left, int[] right) {
20        int[] result = new int[left.length + right.length];
21        int i = 0;
22        while (left.length > 0 && right.length > 0) {
23            if (left[0] <= right[0]) {
24                result[i++] = left[0];
25                left = Arrays.copyOfRange(left, 1, left.length);
26            } else {
27                result[i++] = right[0];
28                right = Arrays.copyOfRange(right, 1, right.length);
29            }
30        }
31
32        while (left.length > 0) {
33            result[i++] = left[0];
34            left = Arrays.copyOfRange(left, 1, left.length);
35        }
36
37        while (right.length > 0) {
38            result[i++] = right[0];
39            right = Arrays.copyOfRange(right, 1, right.length);
40        }
41
42        return result;
43    }
44
45}
```

## 6. 快速排序

### 6.1 算法步骤

- 从数列中挑出一个元素，称为“基准” (pivot) ；
- 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（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；
- 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；

## 6.2 动画演示



快速排序动画演示

## 6.3 参考代码

```
1//Java 代码实现
2public class QuickSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        // 对 arr 进行拷贝, 不改变参数内容
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
8
9        return quickSort(arr, 0, arr.length - 1);
10    }
11
12    private int[] quickSort(int[] arr, int left, int right) {
13        if (left < right) {
14            int partitionIndex = partition(arr, left, right);
15            quickSort(arr, left, partitionIndex - 1);
16            quickSort(arr, partitionIndex + 1, right);
17        }
18        return arr;
19    }
20
21    private int partition(int[] arr, int left, int right) {
22        // 设定基准值 (pivot)
23        int pivot = left;
24        int index = pivot + 1;
25        for (int i = index; i <= right; i++) {
26            if (arr[i] < arr[pivot]) {
```


```
27         swap(arr, i, index);
28         index++;
29     }
30 }
31 swap(arr, pivot, index - 1);
32 return index - 1;
33 }
34
35 private void swap(int[] arr, int i, int j) {
36     int temp = arr[i];
37     arr[i] = arr[j];
38     arr[j] = temp;
39 }
40
41 }
```

## 7. 堆排序

### 7.1 算法步骤

- 创建一个堆  $H[0.....n-1]$ ;
- 把堆首（最大值）和堆尾互换；
- 把堆的尺寸缩小 1，并调用 `shift_down(0)`，目的是把新的数组顶端数据调整到相应位置；
- 重复步骤 2，直到堆的尺寸为 1。

### 7.2 动画演示



堆排序动画演示

## 7.3 参考代码

```
1//Java 代码实现
2public class HeapSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        // 对 arr 进行拷贝, 不改变参数内容
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
8
9        int len = arr.length;
10
11        buildMaxHeap(arr, len);
12
13        for (int i = len - 1; i > 0; i--) {
14            swap(arr, 0, i);
15            len--;
16            heapify(arr, 0, len);
17        }
18        return arr;
19    }
20
21    private void buildMaxHeap(int[] arr, int len) {
22        for (int i = (int) Math.floor(len / 2); i >= 0; i--) {
23            heapify(arr, i, len);
24        }
25    }
26
27    private void heapify(int[] arr, int i, int len) {
28        int left = 2 * i + 1;
29        int right = 2 * i + 2;
```

```
30     int largest = i;
31
32     if (left < len && arr[left] > arr[largest]) {
33         largest = left;
34     }
35
36     if (right < len && arr[right] > arr[largest]) {
37         largest = right;
38     }
39
40     if (largest != i) {
41         swap(arr, i, largest);
42         heapify(arr, largest, len);
43     }
44 }
45
46 private void swap(int[] arr, int i, int j) {
47     int temp = arr[i];
48     arr[i] = arr[j];
49     arr[j] = temp;
50 }
51
52 }
```

## 8. 计数排序

### 8.1 算法步骤

- 花 $O(n)$ 的时间扫描一下整个序列 A，获取最小值 min 和最大值 max
- 开辟一块新的空间创建新的数组 B，长度为 ( max - min + 1)
- 数组 B 中 index 的元素记录的值为 A 中某元素出现的次数
- 最后输出目标整数序列，具体的逻辑是遍历数组 B，输出相应元素以及对应的个数

### 8.2 动画演示

计数排序动画演示

### 8.3 参考代码

```
1//Java 代码实现
2public class CountingSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        // 对 arr 进行拷贝, 不改变参数内容
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
8
9        int maxValue = getMaxValue(arr);
10
11        return countingSort(arr, maxValue);
12    }
13
14    private int[] countingSort(int[] arr, int maxValue) {
15        int bucketLen = maxValue + 1;
16        int[] bucket = new int[bucketLen];
17
18        for (int value : arr) {
19            bucket[value]++;
20        }
21
22        int sortedIndex = 0;
23        for (int j = 0; j < bucketLen; j++) {
24            while (bucket[j] > 0) {
25                arr[sortedIndex++] = j;
26                bucket[j]--;
27            }
28        }
29        return arr;
30    }
31}
```

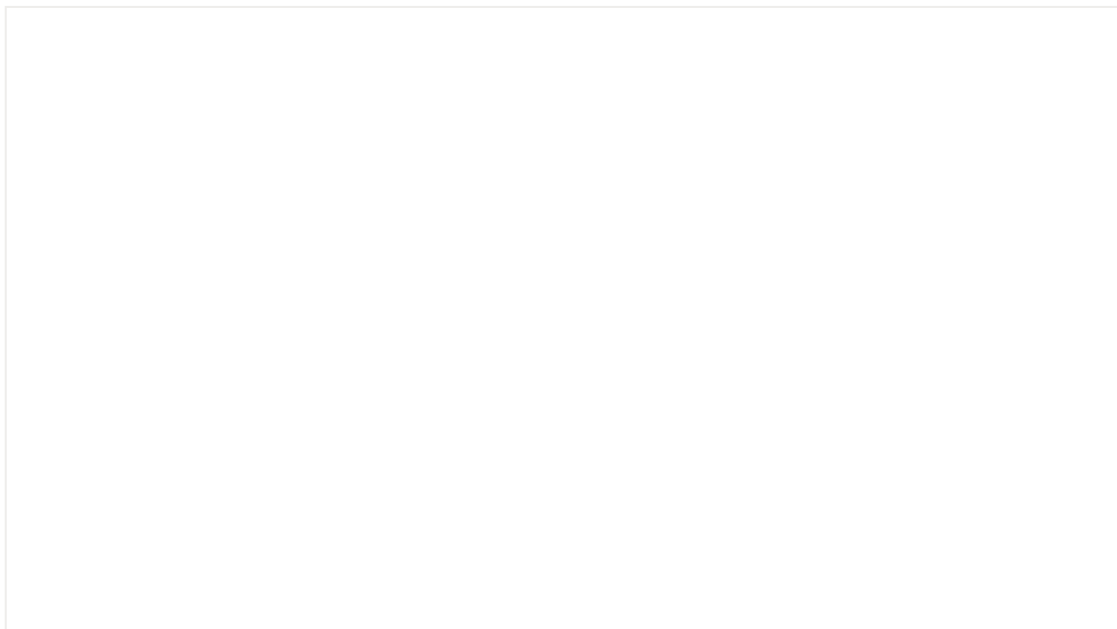
```
30     }
31
32     private int getMaxValue(int[] arr) {
33         int maxValue = arr[0];
34         for (int value : arr) {
35             if (maxValue < value) {
36                 maxValue = value;
37             }
38         }
39         return maxValue;
40     }
41
42 }
```

## 9. 桶排序

### 9.1 算法步骤

- 设置固定数量的空桶。
- 把数据放到对应的桶中。
- 对每个不为空的桶中数据进行排序。
- 拼接不为空的桶中数据，得到结果

### 9.2 动画演示



桶排序动画演示

## 9.3 参考代码

```
1//Java 代码实现
2public class BucketSort implements IArraySort {
3
4    private static final InsertSort insertSort = new InsertSort();
5
6    @Override
7    public int[] sort(int[] sourceArray) throws Exception {
8        // 对 arr 进行拷贝, 不改变参数内容
9        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
10
11        return bucketSort(arr, 5);
12    }
13
14    private int[] bucketSort(int[] arr, int bucketSize) throws Exception {
15        if (arr.length == 0) {
16            return arr;
17        }
18
19        int minValue = arr[0];
20        int maxValue = arr[0];
21        for (int value : arr) {
22            if (value < minValue) {
23                minValue = value;
24            } else if (value > maxValue) {
25                maxValue = value;
26            }
27        }
28
29        int bucketCount = (int) Math.floor((maxValue - minValue) / bucketSize) + 1;
30        int[][] buckets = new int[bucketCount][0];
31
32        // 利用映射函数将数据分配到各个桶中
33        for (int i = 0; i < arr.length; i++) {
34            int index = (int) Math.floor((arr[i] - minValue) / bucketSize);
35            buckets[index] = arrAppend(buckets[index], arr[i]);
36        }
37
38        int arrIndex = 0;
39        for (int[] bucket : buckets) {
40            if (bucket.length <= 0) {
41                continue;
42            }
43            // 对每个桶进行排序, 这里使用了插入排序
44            bucket = insertSort.sort(bucket);
45            for (int value : bucket) {
46                arr[arrIndex++] = value;
47            }
48        }
49
50        return arr;
51    }
52}
```



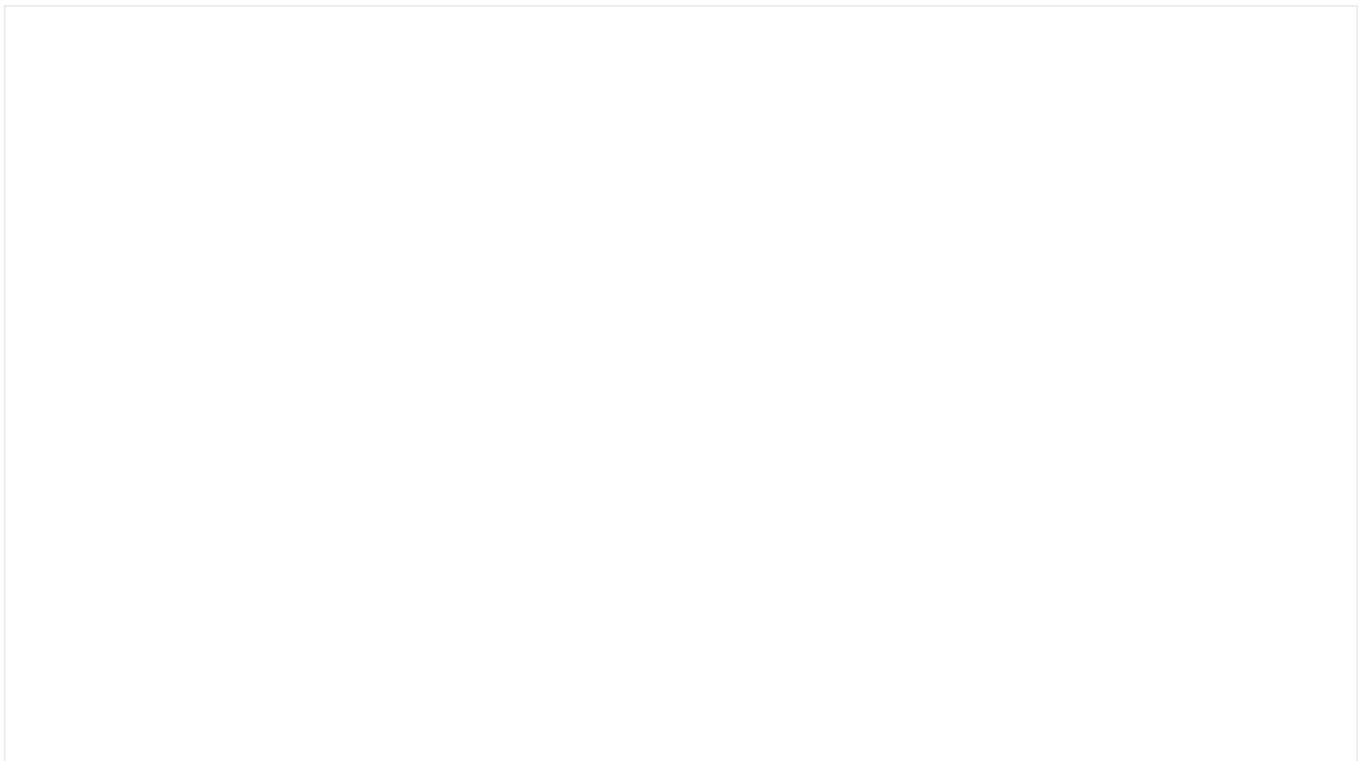
```
53  /**
54   * 自动扩容, 并保存数据
55   *
56   * @param arr
57   * @param value
58   */
59  private int[] arrAppend(int[] arr, int value) {
60      arr = Arrays.copyOf(arr, arr.length + 1);
61      arr[arr.length - 1] = value;
62      return arr;
63  }
64
65 }
```

## 10. 基数排序

### 10.1 算法步骤

- 将所有待比较数值（正整数）统一为同样的数位长度，数位较短的数前面补零
- 从最低位开始，依次进行一次排序
- 从最低位排序一直到最高位排序完成以后，数列就变成一个有序序列

### 10.2 动画演示



基数排序动画演示

## 10.3 参考代码

```
1//Java 代码实现
2public class RadixSort implements IArraySort {
3
4    @Override
5    public int[] sort(int[] sourceArray) throws Exception {
6        // 对 arr 进行拷贝, 不改变参数内容
7        int[] arr = Arrays.copyOf(sourceArray, sourceArray.length);
8
9        int maxDigit = getMaxDigit(arr);
10       return radixSort(arr, maxDigit);
11    }
12
13    /**
14     * 获取最高位数
15     */
16    private int getMaxDigit(int[] arr) {
17        int maxValue = getMaxValue(arr);
18        return getNumLenght(maxValue);
19    }
20
21    private int getMaxValue(int[] arr) {
22        int maxValue = arr[0];
23        for (int value : arr) {
24            if (maxValue < value) {
25                maxValue = value;
26            }
27        }
28        return maxValue;
29    }
30
31    protected int getNumLenght(long num) {
32        if (num == 0) {
33            return 1;
34        }
35        int lenght = 0;
36        for (long temp = num; temp != 0; temp /= 10) {
37            lenght++;
38        }
39        return lenght;
40    }
41
42    private int[] radixSort(int[] arr, int maxDigit) {
43        int mod = 10;
44        int dev = 1;
45
46        for (int i = 0; i < maxDigit; i++, dev *= 10, mod *= 10) {
47            // 考虑负数的情况, 这里扩展一倍队列数, 其中 [0-9]对应负数, [10-19]对应正数 (bucket + 10)
48            int[][] counter = new int[mod * 2][0];
49
50            for (int j = 0; j < arr.length; j++) {
51                int bucket = ((arr[j] % mod) / dev) + mod;
52                counter[bucket] = arrayAppend(counter[bucket], arr[j]);
```

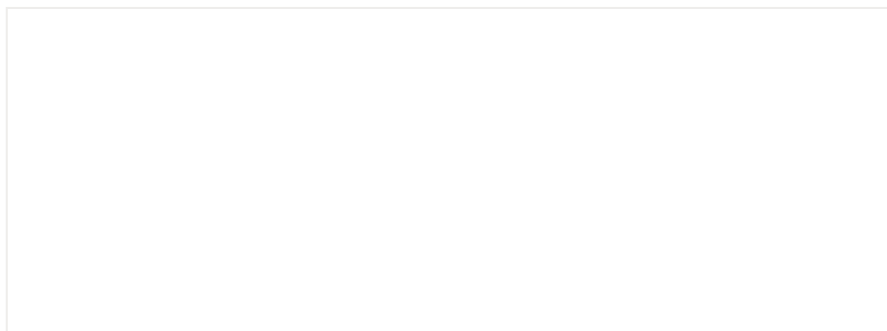
```
53         }
54
55         int pos = 0;
56         for (int[] bucket : counter) {
57             for (int value : bucket) {
58                 arr[pos++] = value;
59             }
60         }
61     }
62
63     return arr;
64 }
65 private int[] arrayAppend(int[] arr, int value) {
66     arr = Arrays.copyOf(arr, arr.length + 1);
67     arr[arr.length - 1] = value;
68     return arr;
69 }
70 }
```

本文思路来源于: <https://github.com/hustcc/JS-Sorting-Algorithm>

End

## 关于本号

作者乔戈里亲历2019秋招，哈工大计算机本硕，百度java工程师，欢迎大家关注我的**微信公众号：程序员乔戈里**，公众号有**3T编程资源**，以及我和我朋友（百度C++工程师）在秋招期间整理的近200M的面试必考的java与C++**面经**，并有**每天一道leetcode打卡群**与技术交流群，欢迎关注。



3T编程资料等你来拿

等等，先别走！[程序员乔戈里]公众号又有活动了！参与活动，不仅可以培养自己良好的习惯，还能拿到“现金红包与书籍奖励”，动作要快，姿势要帅！


[戳我看详情](#)



今日问题：  
(辣条走起) 如何看待图解算法这种方式？

留言格式：  
打卡xx天，答：xxx

文章转载自公众号

 五分钟学算法 >