十大排序算法

1、冒泡排序（Bubble Sort）

```
/**
     * 冒泡排序
     *
     * @param array
     * @return
     */
    public static int[] bubbleSort(int[] array) {
        if (array.length == 0)
            return array;
        for (int i = 0; i < array.length; i++)
            for (int j = 0; j < array.length - 1 - i; j++)
                if (array[j + 1] < array[j]) {
                    int temp = array[j + 1];
                    array[j + 1] = array[j];
                    array[j] = temp;
                }
        return array;
    }
```

2、选择排序（Selection Sort）

```
/**
     * 选择排序
     * @param array
     * @return
     */
    public static int[] selectionSort(int[] array) {
        if (array.length == 0)
            return array;
        for (int i = 0; i < array.length; i++) {
            int minIndex = i;
            for (int j = i; j < array.length; j++) {
                if (array[j] < array[minIndex]) //找到最小的数
                    minIndex = j; //将最小数的索引保存
            }
            int temp = array[minIndex];
            array[minIndex] = array[i];
            array[i] = temp;
        }
        return array;
    }
```

3、插入排序（Insertion Sort）

```
/**
     * 插入排序
     * @param array
     * @return
     */
    public static int[] insertionSort(int[] array) {
        if (array.length == 0)
            return array;
        int current;
        for (int i = 0; i < array.length - 1; i++) {
            current = array[i + 1];
            int preIndex = i;
            while (preIndex >= 0 && current < array[preIndex]) {
                array[preIndex + 1] = array[preIndex];
                preIndex--;
            }
            array[preIndex + 1] = current;
        }
        return array;
    }
```

4、希尔排序（Shell Sort）

```
/**
     * 希尔排序
     *
     * @param array
     * @return
     */
    public static int[] ShellSort(int[] array) {
        int len = array.length;
        int temp, gap = len / 2;
        while (gap > 0) {
            for (int i = gap; i < len; i++) {
                temp = array[i];
                int preIndex = i - gap;
                while (preIndex >= 0 && array[preIndex] > temp) {
                    array[preIndex + gap] = array[preIndex];
                    preIndex -= gap;
                }
                array[preIndex + gap] = temp;
            }
            gap /= 2;
        }
        return array;
    }
```

5、归并排序（Merge Sort）

```
/**
     * 归并排序
     *
```

```
     * @param array
     * @return
     */
    public static int[] MergeSort(int[] array) {
        if (array.length < 2) return array;
        int mid = array.length / 2;
        int[] left = Arrays.copyOfRange(array, 0, mid);
        int[] right = Arrays.copyOfRange(array, mid, array.length);
        return merge(MergeSort(left), MergeSort(right));
    }
    /**
     * 归并排序——将两段排序好的数组结合成一个排序数组
     *
     * @param left
     * @param right
     * @return
     */
    public static int[] merge(int[] left, int[] right) {
        int[] result = new int[left.length + right.length];
        for (int index = 0, i = 0, j = 0; index < result.length; index++) {
            if (i >= left.length)
                result[index] = right[j++];
            else if (j >= right.length)
                result[index] = left[i++];
            else if (left[i] > right[j])
                result[index] = right[j++];
            else
                result[index] = left[i++];
        }
        return result;
    }
```

6、快速排序（Quick Sort）

```
/**
     * 快速排序方法
     * @param array
     * @param start
     * @param end
     * @return
     */
    public static int[] QuickSort(int[] array, int start, int end) {
        if (array.length < 1 || start < 0 || end >= array.length || start > end) return null;
        int smallIndex = partition(array, start, end);
        if (smallIndex > start)
            QuickSort(array, start, smallIndex - 1);
        if (smallIndex < end)
            QuickSort(array, smallIndex + 1, end);
        return array;
    }
    /**
     * 快速排序算法——partition
     * @param array
     * @param start
     * @param end
     * @return
```

```
     */
    public static int partition(int[] array, int start, int end) {
        int pivot = (int) (start + Math.random() * (end - start + 1));
        int smallIndex = start - 1;
        swap(array, pivot, end);
        for (int i = start; i <= end; i++)
            if (array[i] <= array[end]) {
                smallIndex++;
                if (i > smallIndex)
                    swap(array, i, smallIndex);
            }
        return smallIndex;
    }

    /**
     * 交换数组内两个元素
     * @param array
     * @param i
     * @param j
     */
    public static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
```

7、堆排序（Heap Sort）

```
//声明全局变量，用于记录数组array的长度；
static int len;
    /**
     * 堆排序算法
     *
     * @param array
     * @return
     */
    public static int[] HeapSort(int[] array) {
        len = array.length;
        if (len < 1) return array;
        //1.构建一个最大堆
        buildMaxHeap(array);
        //2.循环将堆首位（最大值）与末位交换，然后在重新调整最大堆
        while (len > 0) {
            swap(array, 0, len - 1);
            len--;
            adjustHeap(array, 0);
        }
        return array;
    }
    /**
     * 建立最大堆
     *
     * @param array
     */
    public static void buildMaxHeap(int[] array) {
        //从最后一个非叶子节点开始向上构造最大堆
```

```
        for (int i = (len/2 - 1); i >= 0; i--) { //感谢 @让我发会呆 网友的提醒，此处应该为 i = (len/2 - 1)
            adjustHeap(array, i);
        }
    }
    /**
     * 调整使之成为最大堆
     *
     * @param array
     * @param i
     */
    public static void adjustHeap(int[] array, int i) {
        int maxIndex = i;
        //如果有左子树，且左子树大于父节点，则将最大指针指向左子树
        if (i * 2 < len && array[i * 2] > array[maxIndex])
            maxIndex = i * 2;
        //如果有右子树，且右子树大于父节点，则将最大指针指向右子树
        if (i * 2 + 1 < len && array[i * 2 + 1] > array[maxIndex])
            maxIndex = i * 2 + 1;
        //如果父节点不是最大值，则将父节点与最大值交换，并且递归调整与父节点交换的位置。
        if (maxIndex != i) {
            swap(array, maxIndex, i);
            adjustHeap(array, maxIndex);
        }
    }
```

8、计数排序（Counting Sort）

```
/**
     * 计数排序
     *
     * @param array
     * @return
     */
    public static int[] CountingSort(int[] array) {
        if (array.length == 0) return array;
        int bias, min = array[0], max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max)
                max = array[i];
            if (array[i] < min)
                min = array[i];
        }
        bias = 0 - min;
        int[] bucket = new int[max - min + 1];
        Arrays.fill(bucket, 0);
        for (int i = 0; i < array.length; i++) {
            bucket[array[i] + bias]++;
        }
        int index = 0, i = 0;
        while (index < array.length) {
            if (bucket[i] != 0) {
                array[index] = i - bias;
                bucket[i]--;
                index++;
            } else
                i++;
```

```
        }
        return array;
    }
```

9、桶排序（Bucket Sort）

```
/**
     * 桶排序
     *
     * @param array
     * @param bucketSize
     * @return
     */
    public static ArrayList<Integer> BucketSort(ArrayList<Integer> array, int bucketSize) {
        if (array == null || array.size() < 2)
            return array;
        int max = array.get(0), min = array.get(0);
        // 找到最大值最小值
        for (int i = 0; i < array.size(); i++) {
            if (array.get(i) > max)
                max = array.get(i);
            if (array.get(i) < min)
                min = array.get(i);
        }
        int bucketCount = (max - min) / bucketSize + 1;
        ArrayList<ArrayList<Integer>> bucketArr = new ArrayList<>(bucketCount);
        ArrayList<Integer> resultArr = new ArrayList<>();
        for (int i = 0; i < bucketCount; i++) {
            bucketArr.add(new ArrayList<Integer>());
        }
        for (int i = 0; i < array.size(); i++) {
            bucketArr.get((array.get(i) - min) / bucketSize).add(array.get(i));
        }
        for (int i = 0; i < bucketCount; i++) {
            if (bucketSize == 1) { // 如果带排序数组中有重复数字时  感谢 @见风任然是风 朋友指出错误
                for (int j = 0; j < bucketArr.get(i).size(); j++)
                    resultArr.add(bucketArr.get(i).get(j));
            } else {
                if (bucketCount == 1)
                    bucketSize--;
                ArrayList<Integer> temp = BucketSort(bucketArr.get(i), bucketSize);
                for (int j = 0; j < temp.size(); j++)
                    resultArr.add(temp.get(j));
            }
        }
        return resultArr;
    }
```

10、基数排序（Radix Sort）

```
/**
     * 基数排序
     * @param array
```

```
     * @return
     */
    public static int[] RadixSort(int[] array) {
        if (array == null || array.length < 2)
            return array;
        // 1.先算出最大数的位数；
        int max = array[0];
        for (int i = 1; i < array.length; i++) {
            max = Math.max(max, array[i]);
        }
        int maxDigit = 0;
        while (max != 0) {
            max /= 10;
            maxDigit++;
        }
        int mod = 10, div = 1;
        ArrayList<ArrayList<Integer>> bucketList = new ArrayList<ArrayList<Integer>>();
        for (int i = 0; i < 10; i++)
            bucketList.add(new ArrayList<Integer>());
        for (int i = 0; i < maxDigit; i++, mod *= 10, div *= 10) {
            for (int j = 0; j < array.length; j++) {
                int num = (array[j] % mod) / div;
                bucketList.get(num).add(array[j]);
            }
            int index = 0;
            for (int j = 0; j < bucketList.size(); j++) {
                for (int k = 0; k < bucketList.get(j).size(); k++)
                    array[index++] = bucketList.get(j).get(k);
                bucketList.get(j).clear();
            }
        }
        return array;
    }
```