链滴

常见算法总结 - 排序篇

作者：valarchie

原文链接：https://ld246.com/article/1588638379663

来源网站：链滴

本文总结了常见高频的关于排序的算法考察。

1.冒泡排序

冒泡排序的思想是元素两两比较，将较大或者较小的元素往一端进行移动

```
 public static void bubble(int[] array) {

        for (int i = 0; i < array.length - 1; i++) {

            for (int j = 0; j + 1 < array.length - i; j++) {

                if (array[j] > array[j + 1]) {

                    int tmp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = tmp;

                }

            }

        }

}
```

2.快速排序

快速排序的思想是随机选取一个数字，并以这个数字为临界点，将数组分成小于临界点的的子数组和于临界点的子数组，并递归这个过程，即可实现最终的排序。

```
public static void quick(int[] array, int begin, int end) {

        if (begin >= end) {
            return;
        }

        int beginRange = begin;
        int endRange = end;

        int compareInt = array[begin];

        begin++;

        while (begin < end) {

            if (array[end] > compareInt) {
                end--;
                continue;
            }

            if (array[begin] < compareInt) {
```

```
            begin++;
            continue;
        }


        int tmp = array[begin];
        array[begin] = array[end];
        array[end] = tmp;

    }

    if (array[beginRange] > array[begin]) {

        int tmp = array[begin];
        array[begin] = array[beginRange];
        array[beginRange] = tmp;

    }

    quick(array, beginRange, begin - 1);
    quick(array, end + 1, endRange);

    return;


}
```

3.选择排序

选择排序的思想在于每一轮选择一个本轮的极大值，然后放入数组的尾部。最终可实现排序。

```
public static void selectSort(int[] array) {

    if (array.length == 0) {
        return;
    }

    for (int i = 0; i < array.length; i++) {

        int min = array[i];
        int minIndex = i;

        for (int j = i; j < array.length; j++) {
            if (array[j] < min) {
                min = array[j];
                minIndex = j;
            }
        }

        int tmp = array[i];
        array[i] = min;
        array[minIndex] = tmp;
    }
```

```
    }
```

4.归并排序

归并排序的思想在于先排序小的数组，再由小的数组合并成大数组，直到最后合并的数组大小是原数的大小时，即完成了归并排序。

```
public static void sort(int[] array, int left, int right) {

        //1.设置递归的base case
        if (left == right) {
            return;
        }
        //2.分别排两边
        int mid = left + (right - left) / 2;
        sort(array, left, mid);
        sort(array, mid + 1, right);
        //3.合并
        merge(array, left, mid + 1, right);

    }

public static void merge(int[] array, int leftPtr, int rightPtr, int rightBound) {

        int mid = rightPtr - 1;
        int[] temp = new int[rightBound - leftPtr + 1];

        int i = leftPtr, j = rightPtr, k = 0;
        while (i <= mid && j <= rightBound) {
            temp[k++] = array[i] <= array[j] ? array[i++] : array[j++];
        }

        while (i <= mid) {
            temp[k++] = array[i++];
        }
        while (j <= rightBound) {
            temp[k++] = array[j++];
        }

        //不要忘了把temp数组复制到arr中
        for (int m = 0; m < temp.length; m++) {
            array[leftPtr + m] = temp[m];
        }

    }
```

5.桶排序

桶排序的思想在于，根据数组中的最大值和最小值的差值作为区间拆分为N个桶，将元素落入这些桶中，进行排序再合并。

```
 public static void bucketSort(int[] array){
```

```
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;

        for(int i = 0; i < array.length; i++){
            max = Math.max(max, array[i]);
            min = Math.min(min, array[i]);
        }

        //桶数
        int bucketNum = (max - min) / array.length + 1;
        ArrayList<ArrayList<Integer>> bucketArr = new ArrayList<>(bucketNum);
        for(int i = 0; i < bucketNum; i++){
            bucketArr.add(new ArrayList<Integer>());
        }

        //将每个元素放入桶
        for(int i = 0; i < array.length; i++){
            int num = (array[i] - min) / (array.length);
            bucketArr.get(num).add(array[i]);
        }

        //对每个桶进行排序
        for(int i = 0; i < bucketArr.size(); i++){
            Collections.sort(bucketArr.get(i));
        }

        int k = 0;

        for (int i = 0; i < bucketArr.size(); i++) {

            for (Integer integer : bucketArr.get(i)) {
                array[k++] = integer;
            }

        }

    }
```

6.计数排序

计数排序为特殊的桶排序，一个桶的值区间为1。

```
public static int[] countSort(int[] array) {
        if (array == null || array.length == 0) {
            return null;
        }

        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;

        //找出数组中的最大最小值
        for (int i = 0; i < array.length; i++) {
            max = Math.max(max, array[i]);
            min = Math.min(min, array[i]);
```

```
        }

        int help[] = new int[max];

        //找出每个数字出现的次数
        for (int i = 0; i < array.length; i++) {
            int mapPos = array[i] - min;
            help[mapPos]++;
        }

        int index = 0;
        for (int i = 0; i < help.length; i++) {
            while (help[i]-- > 0) {
                array[index++] = i + min;
            }
        }

        return array;
    }
```

7.希尔排序

希尔排序的思想在于步长。先让元素从数组的一半作为步长，进行比较替换，再依次减小步长，当步为0时，即完成排序。它主要的优势是避免冒泡排序每次交换步长为1的低效率。

```
 public static void shellSort(int[] arr) {
        //step:步长
        for (int step = arr.length / 2; step > 0; step /= 2) {
            //对一个步长区间进行比较 [step,arr.length)
            for (int i = step; i < arr.length; i++) {
                int value = arr[i];
                int j;

                //对步长区间中具体的元素进行比较
                for (j = i - step; j >= 0 && arr[j] > value; j -= step) {
                    //j为左区间的取值，j+step为右区间与左区间的对应值。
                    arr[j + step] = arr[j];
                }
                //此时step为一个负数，[j + step]为左区间上的初始交换值
                arr[j + step] = value;
            }
        }
}
```

笔者个人总结，如有错误之处望不吝指出。