链滴

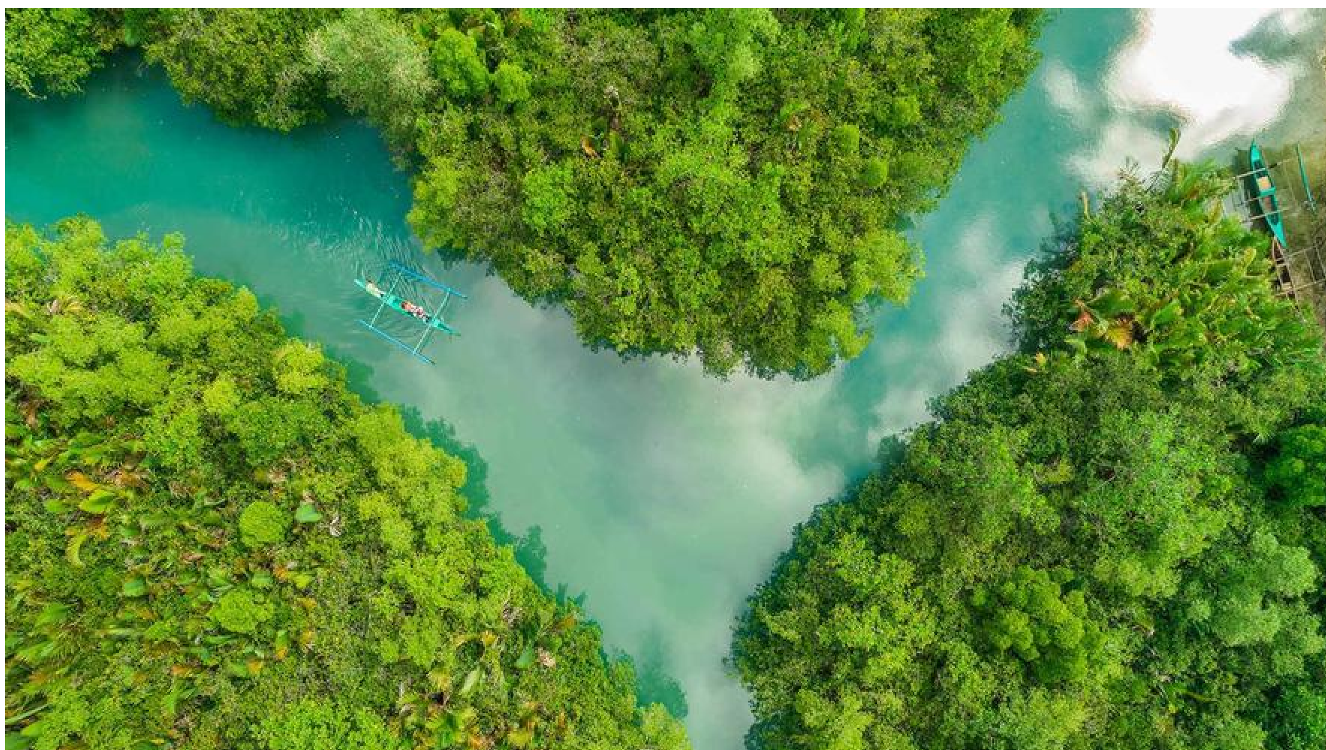
【xsong 说算法】第一期：排序算法

作者：xsong

原文链接：https://ld246.com/article/1638611183123

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



第一章：排序算法

1.1 选择排序

选择排序无论什么数据进去都是 O(N^2) 的时间复杂度，唯一好处就是不占用额外的存储空间

算法思路步骤

首先先找到最大（小） 的元素，存放起始位置，重复找最小的往后排，直到所有的元素排序完毕

- **下面是代码演示：**

```
/**
 * 使用选择排序解题
 *
 * @param nums 数组
 * @return 返回排序好的数组
 */
public int[] sortArray(int[] nums) {
    //最小值的下标
    for (int i = 0; i < nums.length ; i++) {
        int min = i;
        for (int j = i; j < nums.length; j++) {
            if (nums[j] < nums[min]) {
                min = j;
            }
        }
        if (min != i) {
            int temp = nums[i];
            nums[i] = nums[min];
```

```
                nums[min] = temp;
            }
        }
        return nums;
    }
```

选择排序的优点：交换次数最少

「选择排序」看起来好像最没有用，但是如果在交换成本较高的排序任务中，就可以使用「选择排序（《算法 4》相关章节课后练习题）。

不要对算法带有个人色彩，在面试回答问题的时候和看待一个人和事物的时候，可以参考的回答模式「具体问题具体分析，在什么什么情况下，用什么什么算法」。

1.2 冒泡排序

最快：当输入的数据已经是正序时（我还要你冒泡排序有何用）

最慢：当输入的数据是反序时（干嘛要用你冒泡排序呢，我是闲的吗）。

算法思路步骤

每次比较相邻的元素，如果第一个比第二个打，直接交换；直到没有任何一对数字需要比较

- **下面是代码演示**

```
 /**
     * 使用冒泡排序解题
     *
     * @param nums 数组
     * @return 返回排序好的数组
     */
    public int[] sortArray(int[] nums) {
        boolean mark = true;
        while (mark) {
            boolean end = false;
            for (int i = 0; i < nums.length - 1; i++) {
                if (nums[i] > nums[i + 1]) {
                    int temp = nums[i];
                    nums[i] = nums[i + 1];
                    nums[i + 1] = temp;
                    end = true;
                }
            }
            mark = end;
        }
        return nums;
    }
```

- **比较官方的解答**

```
    public int[] sortArray(int[] nums) {
        int len = nums.length;
```

```
        for (int i = len - 1; i >= 0; i--) {
            // 先默认数组是有序的，只要发生一次交换，就必须进行下一轮比较
            boolean sorted = true;
            for (int j = 0; j < i; j++) {
                if (nums[j] > nums[j + 1]) {
                    swap(nums, j, j + 1);
                    sorted = false;
                }
            }

            // 如果在内层循环中，都没有执行一次交换操作，说明此时数组已经是升序数组
            if (sorted) {
                break;
            }
        }
        return nums;
    }
    private void swap(int[] nums, int index1, int index2) {
        int temp = nums[index1];
        nums[index1] = nums[index2];
        nums[index2] = temp;
    }
```

1.3 插入排序

插入排序和冒泡排序一样，也有一种优化算法，叫做拆半插入。

算法解题步骤：

默认第一个有序，然后从后往前比较，直到找不到比自己小的，然后插进去

- **算法实现：**

```
    /**
     * 使用插入排序解题
     *
     * @param nums 数组
     * @return 返回排序好的数组
     */
    public int[] sortArray(int[] nums) {
        for (int i = 1; i < nums.length; i++) {
            int temp = nums[i];
            int j = i;
            while(j>0 && temp<nums[j-1]){
                nums[j] = nums[j-1];
                j--;
            }
            if(j != i){
                nums[j] =temp;
            }
        }
        return nums;
    }
```

1.4 希尔排序

是直接插入排序算法的一种更高效的改进版本。希尔排序是非稳定排序算法

1.5 归并排序

该算法是采用分治法(Divide and Conquer）的一个非常典型的应用。

算法解题步骤：

归并排序的核心是递归，我们需要先把数组的左右两部分分别排序，然后再把左右两部分有序的数组并到一起，形成一个有序的数组

- **代码演示：**

```
/**
 * @author song
 * 归并排序 对一个数组进行排序
 */
public class Code02_MerageSort {
    public static void main(String[] args) {
        int[] arr = new int[]{3, 5, 1, 4, 9, 7};
        process(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }

    public static void process(int[] arr, int L, int R) {
        if (L == R) {
            return;
        }
        int mid = L + ((R - L) >> 1);
        process(arr, L, mid);
        process(arr, mid + 1, R);
        merage(arr, L, mid, R);
    }

    public static void merage(int[] arr, int L, int M, int R) {
        int[] help = new int[R - L + 1];
        int i = 0;
        int p1 = L;
        int p2 = M + 1;
        //说明左右区间都有数
        while (p1 <= M && p2 <= R) {
            help[i++] = arr[p1] <= arr[p2] ? arr[p1++] : arr[p2++];
        }
        //只有左侧有数
        while (p1 <= M) {
            help[i++] = arr[p1++];
        }
        //同理只有右侧有数
        while (p2 <= R) {
            help[i++] = arr[p2++];
```

```
        }
        for (int j = 0; j < help.length; j++) {
            arr[L + j] = help[j];
        }
    }
}
```

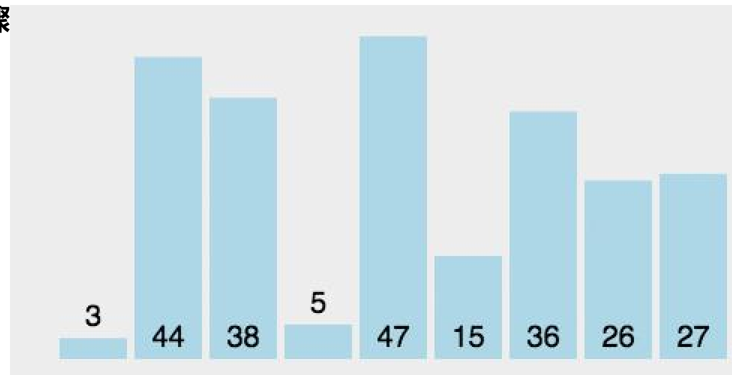
****ps：**有一个经典的例题LeetCode剑指offer51 我建议每次复习归并排序的时候，都去做一遍这题【在做这个题的时候，其核心部分在判断临界值的时候一定要小心】**

1.6 快速排序

快速排序的最坏运行情况是 $O(n^2)$，比如说顺序数列的快排。但它的平摊期望时间是 O(nlogn)，且 O nlogn) 记号中隐含的常数因子很小，比复杂度稳定等于 O(nlogn) 的归并排序要小很多。所以，对大多数顺序性较弱的随机数列而言，快速排序总是优于归并排序。

算法思想

1. **从数组中挑出来一个元素、称为基准（pivot）**
2. **重新排序，比基准小的在基准前面，泛指比基准大的在后面**
3. **递归，以基准为标准，左右两边的数分别执行一二步骤**



- **代码实现**

```
import java.util.Random;

public class Solution {

    // 快速排序 3：三指针快速排序

    /**
     * 列表大小等于或小于该大小，将优先于 quickSort 使用插入排序
     */
    private static final int INSERTION_SORT_THRESHOLD = 7;

    private static final Random RANDOM = new Random();

    public int[] sortArray(int[] nums) {
        int len = nums.length;
        quickSort(nums, 0, len - 1);
        return nums;
```

```java
}

private void quickSort(int[] nums, int left, int right) {
    // 小区间使用插入排序
    if (right - left <= INSERTION_SORT_THRESHOLD) {
        insertionSort(nums, left, right);
        return;
    }

    int randomIndex = left + RANDOM.nextInt(right - left + 1);
    swap(nums, randomIndex, left);


    int pivot = nums[left];
    int lt = left;
    int gt = right + 1;

    int i = left + 1;
    while (i < gt) {
        if (nums[i] < pivot) {
            lt++;
            swap(nums, i, lt);
            i++;
        } else if (nums[i] == pivot) {
            i++;
        } else {
            gt--;
            swap(nums, i, gt);
        }
    }
    swap(nums, left, lt);
    // 注意这里，大大减少了两侧分治的区间
    quickSort(nums, left, lt - 1);
    quickSort(nums, gt, right);
}

/**
 * 对数组 nums 的子区间 [left, right] 使用插入排序
 *
 * @param nums  给定数组
 * @param left  左边界，能取到
 * @param right 右边界，能取到
 */
private void insertionSort(int[] nums, int left, int right) {
    for (int i = left + 1; i <= right; i++) {
        int temp = nums[i];
        int j = i;
        while (j > left && nums[j - 1] > temp) {
            nums[j] = nums[j - 1];
            j--;
        }
        nums[j] = temp;
    }
}
```

```
    private void swap(int[] nums, int index1, int index2) {
        int temp = nums[index1];
        nums[index1] = nums[index2];
        nums[index2] = temp;
    }
}
```

LeetCode实战

- **LeetCode 75 颜色分类**
- **LeetCode 215 数组中第K个最大元素**