排序(10大排序算法)

本笔记来自于李刚《Java程序员的基本修养》

- 选择排序
    - 直接选择排序
    - 堆排序
- 归并排序
- 交换排序
    - 快速排序
    - 冒泡排序
- 基数排序
- 桶排序
- 插入排序
    - 直接插入
    - 折半插入
    - 希尔排序

1: 选择排序

选择排序有两种：直接选择排序和堆排序。直接选择直观但低效，堆排序复杂但高效。

1.1: 直接选择排序

选择排序思维简单，容易实现。就是简单的做比较。

1. 首先为第一个位置的元素做比较，如果存在比其元素更小的，交换两者，然后继续；
2. 第一个元素结束后，为第二个，第三个，第四个元素直至最后一个元素做比较；
   看代码：

```java
//直接选择排序,这里是对整形数组排序。事实上可以对任何一种数组包括自定义类型组进行排序。
private void selectSort(int[] nums) {
    for (int i = 0; i < nums.length - 1; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[i] > nums[j]) {
                int temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
            }
        }
    }
}
```

时间复杂度$O(n^2)$.空间复杂度为$O(1)$.

1.2: 堆排序

堆排序的想法利用了二叉树的特性。对于一棵完全二叉树而言，在其顺序存储结构中，在数组位置为k的节点的左右子节点是$2k+1$和$2k+2$.在这个基础上，定义了小顶堆和大顶堆。
小顶堆：
如果这个数组满足

$$nums[k] <= nums[2k+1]\text{且}$$

$$nums[k] <= nums[2k+2]$$

的话，那么就把这个数组称为小顶堆数组

大顶堆：
如果这个数组满足

$$nums[k] >= nums[2k+1]\text{且}$$

$$nums[k] >= nums[2k+2]$$

的话，那么就把这个数组称为大顶堆数组

算法步骤：

1. 给整个数组进行大顶堆或者小顶堆构造，这样其实就找出了整个数组的最大值或者最小值；
2. 交换数组第一个元素和数组倒数第$n-1-i$个元素；

看代码：

```java
private void heapSort(int[] nums) {
    for (int i = 0; i < nums.length - 1; i++) {
        buildHeap(nums, nums.length - 1 - i);
        swap(nums, 0, nums.length - 1 - i);
    }
}

private void buildHeap(int[] nums, int lastIndex) {
    //排序的最终结果是赠序，因此构造大顶堆数组
    for (int i = (lastIndex - 1)/2; i >=0; i--) {
        int k = i;
        while (2*k+1 <= lastIndex) {
            int biggerIndex = 2 * k + 1;
            if (biggerIndex < lastIndex) {
                if (nums[biggerIndex] < nums[biggerIndex + 1]) {
                    biggerIndex++;
                }
            }
            if (nums[k] < nums[biggerIndex]) {
                swap(nums, k, biggerIndex);
                k = biggerIndex; //这步非常重要，以确保一次交换之后，交换之后还能保证大顶堆性质.
            }else {
                break;
            }
        }
    }
}
private void swap(int[] nums, int i, int j) {
    int temp = nums[i];
    nums[i] = nums[j];
    nums[j] = nums[i];
}
```

堆排序的复杂度：1）构造大小顶堆是$log(n)$；2）总共构造$n - 1$次。所以复杂度是$nlog(n)$。空间复杂度是$O(1)$。

2: 交换排序

2.1: 冒泡排序

冒泡排序就是模拟冒泡过程，先将大的数据沉入数组底部，然后一步步将数组排序。这种算法广为人知，直接上代码：

```java
private void bubbleSort(int nums) {
    for (int i = 0; i < nums.length - 1; i++) {
        for (int j = 0; j < nums - 1 - i; j++) {
            if (nums[j] > nums[j + 1]) {
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }
        }
    }
}
```

这种算法的时间复杂度是$O(n^2)$，空间复杂度是$O(1)$。

2.2: 快速排序

快速排序的关键在于选择一个pivot，比pivot小的放在数组左边，比pivot大的放在数组右边。不断重复这个过程，最终数组会排序。
看代码：

```java
private void quickSort(int[] nums) {
    quickSort(nums, 0, nums.length - 1);
}
private void quickSort(int[] nums, int first, int end) {
    if (first >= end) return;
    int pivot = nums[first];
    int i = first + 1;
    int j = end
    while (true) {
        while (i < end && nums[i] < piovt) i++;
        while (j > first && nums[j] > pivot) j--;
        if (i < j) {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }else {
            break;
        }
    }
    swap(nums, first, j); //交换最后一个坑
    quickSort(nums, first, j-1);
    quickSort(nums, j+1, end);
}
```

快速排序的平均时间复杂度是$nlog(n)$,空间复杂度是$log(n)$。

3: 插入排序

插入排序包含直接插入排序，折半插入排序和希尔排序

3.1: 直接插入排序

插入排序的主要思想就是假定之前的数组已经有序，然后将当前元素插入到有序数组的合适位置。
看代码：

```java
private void insertSort(int[] nums) {
    for (int i = 1; i < nums.length; i++) {
        int temp = nums[i];
        if (nums[i] < nums[i - 1]) {
            for (int j = i - 1; j >=0 && nums[j]>nums[i]; j--){
                nums[j + 1] = nums[j];
            }
            nums[j + 1] = temp;
        }
    }
}
```

时间复杂度$O(n^2)$，空间复杂度$O(1)$。

3.2: 折半插入排序

折半插入排序和直接插入排序的效果差不多，只是在直接插入排序中，我们是通过一个个比较去确定插入排序的合适位置，而折半排序是通过二分查找来确定即将插入的位置。对直接排序算法稍作修改即可。
看代码：

```java
private void insertBinarySort(int[] nums) {
    for (int i = 1; i < nums.length; i++) {
        int temp = nums[i];
        if (nums[i] < nums[i - 1]) {
            int first = 0;
            int end = i - 1;
            while (first <= end) {
                int mid = first + (end - first) / 2;
                if (nums[mid] > temp) end = mid - 1;
                if (nums[mid] < temp) first = mid + 1;
            }
            for (int j = i; j > first; j--) nums[j] = nums[j-1];
            nums[first] = temp;
        }
    }
}
```

3.3: 希尔排序

希尔排序的主要思想就是利用一个增量h来进行插入排序，然后对每h的数组进行插入排序。通常来说增量h的算法是$h = 3*h+1$。
看代码：

```java
private void shellSort(int[] nums) {
    int length = nums.length;
    int h = 1;
    while (h < length / 3) h = h * 3 + 1;
    while (h >= 1) {
        for (int i = h; i < length; i++) {
            int temp = nums[i];
            if (nums[i] < nums[i - h]) {
                for (int j = i - h; j >=0&&nums[j]>temp; j-=h) {
                    nums[j + h] = nums[j];
                }
                nums[j + h] = temp;
            }
        }
        h = h / 3;
    }
}
```

4: 归并排序

归并排序的核心在于原地归并。看代码：

```java
private void mergeSort(int[] nums) {
    sort(nums, 0, nums.length - 1);
}
private void sort(int[] nums, int first, int end) {
    if (first >= end) return;
    int mid = first + (end - first) / 2;
    sort(nums, first, mid);
    sort(nums, mid + 1, end);
    merge(nums, first, mid, end);
}
private void merge(int[] nums, int first, int mid, int end) {
    int[] arr = new int[nums.length];
    int i = first;
    int j = mid + 1;
    for (int k = first; k <= end; k++) arr[k] = nums[k];
    for (int k = first; k <= end; k++) {
        if (i > mid) nums[k] = arr[j++];
        else if (j > end) nums[k] = arr[i++];
        else if (arr[i] < arr[j]) {
            nums[k] = arr[i];
            i++;
        }else {
            nums[k] = arr[j];
            j++;
        }
    }
}
```

归并排序时间复杂度$nlog(n)$。

5: 桶排序

桶排序的主要思想在于统计每个元素出现的频率，然后计算每个元素在数组中的位置。桶排序的缺点在于空间开销较大，因此如果想利用桶排序，最好满足：

- 数据元素的范围是可以枚举的
- 数据量最好集中，不要过于稀疏

看代码，在代码中解释：

```java
private void bucketSort(int[]nums, int min, int max) {
    int length = nums.length;
    int[] temp = new int[length];
    for (int i = 0; i < length; i++) temp[i] = nums[i];
    int[] buckets = new int[max - min];
    for (int i = 0; i < length; i++) {
        buckets[nums[i] - min]++;
    }
    for (int i = 1; i < buckets.length; i++) {
        buckets[i] = buckets[i] + buckets[i-1];
    }
    //重点在这里.
    for (int k = length - 1; k >= 0; k--) {
        nums[--buckets[temp[k] - min]] = temp[k];
    }
}
```

6: 基数排序

基数排序法又可称为多关键字排序法，有两种具体实现方式：

- MSD(Most significant digit)
- LSD(Least significant digit)

通常实现LSD。值得注意的是，基数排序对于每一位的排序必须是稳定的，也就是说即使某两个数字在某一位相同，他们的相对位置在这位排序上不会变化。否则会出现错误。在上面的介绍中，桶排序是稳定的，因此对于每一位上的排序可以使用桶排序。
看代码：

```java
private void radixSort(int[] nums, int radix, int d) {
    //radix 代表基数，这里是10. d代表最高位数。
    int length = nums.length;
    int[] temp = new int[length];
    int[] buckets = new int[radix];
    int rate = 1;
    for (int i = 0; i < d; i++) {
        for (int j = 0; j < radix; j++) buckets[j] = 0;
        for (int j = 0; j < length; j++) temp[j] = nums[j];
        for (int j = 0; j < length; j++) {
            int key = ( temp[j] / rate ) % radix;
            buckets[key]++;
        }
        for (int j = 1; j < buckets.length; j++) {
            buckets[j] = buckets[j] + buckets[j-1];
        }
        for (int j = length - 1; j >=0; j--) {
            int key = ( temp[j] / rate ) % radix;
            nums[--buckets[key]] = temp[j];
        }
        rate *= radix;
    }
}
```