排序

8.1 排序的基本概念

8.1.1 稳定性

稳定性指排序前后关键字的相对位置，未变化就是稳定的，否则就是不稳定的。
如果关键字不能重复，则排序结果是唯一的，那么排序算法稳定性就无关紧要；如果可以重复，就要考虑稳定性。

8.1.2 分类

1.插入类的排序
在一个已经有序的序列中，插入一个新的关键字。如：**直接插入排序、折半插入排序、希尔排序。**
2.交换类的排序
冒泡排序、快速排序
3.选择类的排序
简单选择排序、堆排序
4.归并类排序
二路归并排序
5.基数类排序
多关键字排序

8.2 插入类排序

8.2.1 直接插入排序

1.算法

```
void InsertSort(int R[], int n) {
    int i, j;
    int temp;
    for(i = 1; i < n; ++i) {
        temp = R[i];
        j = j-1;
        while(j >= 0 && temp < R[j]) {
            R[j+1] = R[j];
            --j;
        }
        R[j+1] = temp;
```

```
    }
}
```

2.算法性能分析

(1) 时间复杂度分析

- 考虑**最坏**的情况，整个序列都是逆序的，则 `temp < R[j]` 始终成立，最内层循环此时达到i次，i取值为1到i−1，总执行次数为n(n−1)/2，时间复杂度为O(n^2)。
- 考虑**最好**的情况，整个序列已经有序排列，则 `temp < R[j]` 始终不成立，只执行外层循环时间复杂度为O(n)。

综上，本算法平均时间复杂度为O(n^2)。

(2)空间复杂度分析

算法所需辅助存储空间不会随待排序规模变化而变化，空间复杂度为O(1)。

8.2.2 折半插入排序

1.执行流程

举一趟排序为例：

13 18 49 65 76 97 27 <u>49</u>

此时数组的情况是：

已排序 未排序

关键字	13	38	49	65	76	97	27	49
数组下标	0	1	2	3	4	5	6	7

1）low=0，high=5，m=⌊(0+5)/2⌋=2，下标为 2 的关键字是 49，27<49，所以 27 应该插入到 49 的低半区，改变 high=m−1=1，low 仍然是 0。

2）low=0，high=1，m=⌊(0+1)/2⌋=0，下标为 0 的关键字是 13，27>13，所以 27 应该插入到 13 的高半区，改变 low=m+1=1，high 仍然是 1。

3）low=1，high=1，m=⌊(1+1)/2⌋=1，下标为 1 的关键字是 38，27<38，所以 27 应该插入到 38 的低半区，改变 high=m−1=0，low 仍然是 1，此时 low>high，折半查找结束，27 的插入位置在下标为 high 的关键字之后，即 13 之后。

4）依次向后移动关键字 97，76，65，49，38，然后将 27 插入，这一趟折半插入排序结束。

执行完这一趟排序的结果为：

13 27 38 49 65 76 97 <u>49</u>

2.算法性能分析

(1) 时间复杂度分析

最好情况O($n\log_2 n$),最差O(n^2),平均O(n^2)

（2）空间复杂度

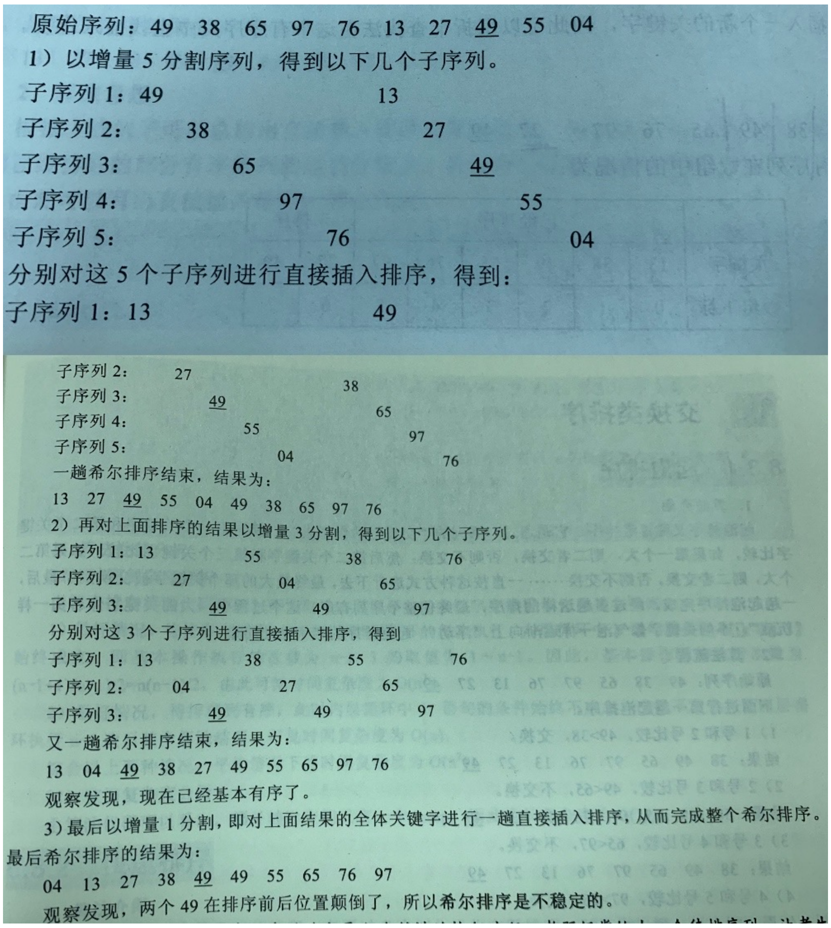
O(1)

8.2.3 希尔排序

1.算法介绍

希尔排序又称作**缩小增量排序**，将待排序列按某种规则分成几个序列，分别对这几个子序列进行直接插入排序。这个规则的体现就是增量的选取，若增量为1，就是直接插入序列。

2.执行流程



原始序列：49 38 65 97 76 13 27 49 55 04

1）以增量 5 分割序列，得到以下几个子序列。

子序列 1：49 13

子序列 2：38 27

子序列 3：65 49

子序列 4：97 55

子序列 5：76 04

分别对这 5 个子序列进行直接插入排序，得到：

子序列 1：13 49

子序列 2：27 38

子序列 3：49 65

子序列 4：55 97

子序列 5：04 76

一趟希尔排序结束，结果为：

13 27 49 55 04 49 38 65 97 76

2）再对上面排序的结果以增量 3 分割，得到以下几个子序列。

子序列 1：13 55 38 76

子序列 2：27 04 65

子序列 3：49 49 97

分别对这 3 个子序列进行直接插入排序，得到

子序列 1：13 38 55 76

子序列 2：04 27 65

子序列 3：49 49 97

又一趟希尔排序结束，结果为：

13 04 49 38 27 49 55 65 97 76

观察发现，现在已经基本有序了。

3）最后以增量 1 分割，即对上面结果的全体关键字进行一趟直接插入排序，从而完成整个希尔排序。

最后希尔排序的结果为：

04 13 27 38 49 49 55 65 76 97

观察发现，两个 49 在排序前后位置颠倒了，所以希尔排序是不稳定的。

3.算法性能分析

（1）时间复杂度

时间复杂度与增量的选择有关，希尔排序的增量选组规则有很多，常见的有以下两个：

- 希尔自己提出的：

  $n/2, n/4, ...n/2^k, ...2, 1$

  每次将增量除以2并向下取整，其中n为序列长度，此时时间复杂度为$O(n^2)$。
- 怕佩尔诺夫和斯塔舍维奇提出的:

  $2^k+1, ..., 65, 33, 17, 9, 5, 3, 1$

  此时时间复杂度为$O(n^{1.5})$

（2）空间复杂度分析

O(1)

8.3 交换类排序

8.3.1 冒泡排序

1.算法

```
void BubbleSort(int R[], int n) {
    int i, j, flag;
    int temp;
    for (i = n-1; i >= 1; --i) {
        flag = 0;
        for(j = 1; j <= i; ++j) {
            if(R[j-1] < R[j] {
            temp = R[j];
            R[j] = R[j-1];
            R[j-1] = temp;
            flag = 1;
            }
        }
        if(flag == 0)
            return;
    }
}
```

2.算法性能分析

（1）时间复杂度

- 最坏情况，执行(n−1+1)(n−1)/2 = n(n−1)/2，时间复杂度为O(n^2)。
- 最好情况，不交换，内层循环执行n−1次，时间复杂度为O(n)。

（2）空间复杂度

O(1)

7.3.2 快速排序

1.算法：

```
void QuickSort(int R[], int low, int high) {
    int temp;
    int i = low, j = high;
    if(low < high) {
```

```
        temp = R[low];
        while(i < j) {
            while(j > i && R[j] >= temp)
                --j;
            if(i < j) {
                R[i] = R[j];
                ++i;
            }
            while(i < j && R[i] < temp)
                ++i;
            if(i < j) {
                R[j] = R[i];
                --j;
            }
        }
        R[i] = temp;
        QuickSort(R, low, i-1);
        QuickSort(R, i + 1, high);
    }
}
```

3.算法性能分析

（1）时间复杂度

最好情况是$O(nlog_2n)$，待排序列越接近无序，本算法效率越高，最坏情况是$O(n^2)$，平均情况下时间复杂度为$O(nlog_2n)$。

（2）空间复杂度

$O(log_2n)$

8.4选择类排序

8.4.1 简单选择排序

1.执行流程

把整个序列分成有序序列和无序序列，开始时整个序列为无序序列。

进行第一趟排序，从无序序列中选取一个最小的关键字，使其与无序序列的第一个关键字交换，此时产生了仅含一个关键字的有序序列。无序序列中关键字减少一个。重复此步骤，知道无序序列中空。

代码：

```
void SelectSort(int R[], int n) {
    int i, j, k;
    int temp;
    for(i = 0; i < n; ++i) {
```

```
            k = i;
            for(j = i+i; j < n; ++j) {
                if(R[k] > R[j]) {
                    k = j;
                }
            }
            temp = R[i];
            R[i] = R[k];
            R[k] = temp;
        }
    }
```

3.算法复杂度分析

（1）时间复杂度

外层循环n次，内层n+1次，执行(n−1+1)(n−1)/2 = n(n−1)/2，时间复杂度为$O(n^2)$。

（2）空间复杂度

O(1)

8.4.2 堆排序

1.算法介绍

堆可以看作一个完全二叉树，任何一个非叶结点的值都不大于（或不小于）其左右孩子结点的值，若**父亲大孩子小**，则称为**大顶堆**；若**父亲小孩子大**，则称为**小顶堆**。

2.具体过程

见书P245

算法：

```
//完成在数组R[low]和R[high]范围内对在为止low上的结点调整
void Sift(int R[], int low, int high) {
    int i = low, j = 2*i;
    int temp = R[i];
    while(j <= high) {
        if(j < high && R[j] < R[j+1]) {
            ++j;
        }
        if(temp < R[j]) {
            R[i] = R[j];
            i = j;
            j = 2*i;
        }
        else
            break;
    }
```

```
    R[i] = temp;
}

//堆排序
void heapSort(int R[], int n) {
    int i;
    int temp;
    for(i = n/2; i >= 1; --i) {
        Sift(R, i, n);                  //建立初始堆
    }
    for(i = n; i >= 2; --i) {
        temp = R[i];
        R[1] = R[i];
        R[i] = temp;
        Sift(R, 1, i-1);                //减少了一个关键字的无序序列中进行调整
    }
}
```

3.性能分析

（1）时间复杂度分析

Sift函数中，完全二叉树高度为$\log_2 n$，时间复杂度为$O(\log_2 n)$；

heapSort中，第一个循环为$O(\log_2 n)*n/2$，第二个循环是$O(\log_2 n)*(n-1)$，整个算法复杂度为$O(\log_2 n)*n/2+O(\log_2 n)*(n-1)$，化简后得$O(n\log_2 n)$。

（2）空间复杂度

O(1)

8.5 二路归并排序

见书P247

算法：

```
int *B = (int *)malloc((n+1)*sizeof(int) //辅助数组B

void Merge(int A[], int low, int mid, int high) {
    for(int k = low; k <= high; ++k) {
        B[k] = A[k];                    //将A中元素复制到B中
    }
    for( i = low, j = mid + 1, k = i; i <= mid && j <= high; k++) {
        if(B[i] <= B[j])                //比较左右两段中的元素
            A[k] = B[i++];              //较小值复制到A中
        else
            A[k] = B[j++];
    }
```

```
    while(i <= mid)  A[k++] = B[i++];    //若第一个表检测完复制
    while(j <= high) A[k++] = B[j++];    //若第一个表检测完复制
}

void mergeSort(int A[], int low, int high) {
    if(low < high) {
        int mid = (low + high)/2;
        mergeSort(A, low, mid);
        mergeSort(A, mid+1, high);
        merge(A, low, mid, high);
    }
}
```

2.算法性能

（1）时间复杂度

$O(n\log_2 n)$

（2）空间复杂度

$O(n)$

8.6 基数排序

1.算法介绍

基数排序的思想是“多关键字排序”，有两种实现方式：第一种**最高为优先**，即先按最高位拍成若干字序列，再对每个字序列按次高位排序。第二种是**最低位优先**，这种方式不必分成字序列，妹子排序全体关键字参与。最低位可以优先这样进行，不通过比较，而是通过“分配”和“收集”。

2.执行流程

P249

3.算法性能分析

时间复杂度：$O(d(n+r_d)$

空间复杂度：$O(r_d)$

其中，n为序列中关键字的个数，d为关键字的关键字位数，

如何判断序列是否是堆：

大顶堆：若满足a[i] > a[2*i*]且*a*[*i*] > *a*[2i+1]

小顶堆：若满足a[i] < a[2*i*]且*a*[*i*] < *a*[2i+1]