稳定与不稳定

排序前序列Ri领先于Rj（即i < j），若在排序后的序列中Ri依然领先于Rj，称其是稳定的

稳定的

- 堆排序、快速排序、希尔排序、直接选择排序是不稳定的排序算法
- 冒泡排序、直接插入排序、折半插入排序、归并排序是稳定的排序算法。

直接插入排序

名称	作用	备注
时间复杂度	O(n^2)	-
-	-	-
是否稳定	稳定	-

```
/**
 * 直接插入排序
 * 升序
*/
void StraightInsertionSort(ElementType *array, int length){
    //i：用来区分已排序和未排序部分
    //已排序[0,i-1] 未排序[i,end]
    for(int i = 1;i < length;i++){
        //用未排序部分的第一个元素和已排序的最后一个元素比较，判断是否需要后移以让出合适
位置
        if(array[i] < array[i - 1]){
            //需要后移让出合适位置；因为可能会被覆盖，所以将未排序部分的第一个元素保存
            int j;
            int temp = array[i];

            //边判断边后移；这个代码没有哨兵，所以要记得做边界检查
            for(j = i - 1;(j >= 0) && (temp < array[j]);j--){
                array[j + 1] = array[j];
            }
            //到循环结束时，j+1便是可以插入未排序部分第一个元素的合适位置
            array[j + 1] = temp;
        }
    }
}
```

折半插入排序

名称	作用	备注
时间复杂度	O(n^2)	-
-	-	-
是否稳定	稳定	-

```
/**
 * 折半插入排序
 *
 * 基于直接插入排序，只是减少了比较次数，实际上时间复杂度还是O(n^2)
*/
void BinaryInsertionSort(ElementType *array, int length){
    //i：用来区分已排序和未排序部分
    //已排序[0,i-1] 未排序[i,end]
    for(int i = 1;i < length;i++){
        //折半查找找出合适位置
        int k, temp = array[i];
        int index = BinarySearch(array, i + 1, temp);

        //在已排序部分加未排序部分的第一个元素，合适位置之后的元素后移一位
        for(k = i - 1;k >= index;k--){
            array[k + 1] = array[k];
        }
        //合适位置放置未排序部分的第一个元素
        array[k + 1] = temp;
    }
}
```

希尔排序

名称	作用	备注
时间复杂度	O(n^1.25)	-
-	-	-
是否稳定	不稳定	-

```
/**
 * 希尔排序
 * Shell's Sort
*/
void SheelSort(ElementType *array, int length){
    for(int i = length / 2;i > 0;i /= 2){
        //printf("i %d\n", i);
        StraightInsertionSortWithStepSize(array, length, i);
```

```
    }
}
```

起泡排序

名称	作用	备注
时间复杂度	O(n^2)	-
-	-	-
是否稳定	稳定	-

```
/**
 * 起泡（冒泡）排序
 * bubble sort
*/
void BubbleSort(ElementType *array, int length){
    for(int i = 0;i < length - 1;i++){
        int isSwap = FALSE;

        //[length-i,length-1]   已排序区间
        //[0,length-i-1]        未排序区间
        for(int j = 0;j < length - i - 1;j++){
            if(array[j + 1] < array[j]){
                int temp = array[j + 1];
                array[j + 1] = array[j];
                array[j] = temp;

                isSwap = TRUE;
            }
        }

        //如果前面都没有交换，那么说明前面是有序的，那么剩下的不需要再比较了
        if(!isSwap){
            break;
        }
    }
}
```

快速排序

名称	作用	备注
时间复杂度	O(nlogn)	-
-	-	-

名称	作用	备注
是否稳定	不稳定	-

```
/**
 * 交换顺序表中子表left至right的记录，使得枢轴记录到位
*/
int partition(ElementType *array, int left, int right);

void QuickSort(ElementType *array, int left, int right){
    if(left < right){
        //交换顺序表中子表left至right的记录，使得枢轴记录到位
        //枢轴左边的数均小于枢轴
        //枢轴右边的数均大于枢轴
        int pivotLocation = partition(array, left, right);

        //对低子表进行递归排序
        QuickSort(array, left, pivotLocation - 1);
        //对高子表进行递归排序
        QuickSort(array, pivotLocation + 1, right);
    }
}
```