排序算法

简单排序

冒泡排序

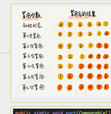
前置条件

- 实现Comparable接口
- 重写compareTo方法

排序原理

1. 比较相邻的元素。如果第一个元素比后一个元素大，就交换这两个元素的位置。
2. 对每一对相邻元素做同样的工作，从开始第一对元素到结尾的最后一对元素。最终最后位置的元素是最大值。

原理图解



代码实例

```
public static void sort(Comparable[] a){
    for(int i=a.length-1;i>0;i--){
        for(int j=0;j<i;j++){
            //几几几几几
            //交换数组中相邻的两个元素的值
            if (greater(a[j],a[j+1])){
                exch(a,j,j+1);
            }
        }
    }
}
```

时间复杂度

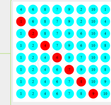
O(N-2)

选择排序

排序原理

1. 第一次遍历的过程中，都假定第一个索引处的元素是最小值，和其他索引处的值或次次进行比较，如果找到索引处的值大于索引为第零个索引处的值，则假定其索引处的值，到假定其他某个索引处的值小于假定的值大小值，最后可以找到最小值的索引。
2. 交换第一个索引处和最小值所在的索引处的值。

原理图解



代码示例

```
public static void sort(Comparable[] a){
    for(int i=0;i<a.length-1;i++){
        //定义一个变量，记录最小值的索引，默认为第一个元素要参与的最大，假定该最小值的索引是当前的
        int minIndex = i;
        for(int j=i+1;j<a.length;j++){
            //获取当前索引处的值和minIndex索引处的值进行比较
            if (a[j].compareTo(a[minIndex]).<0){
                //记录最小值
                minIndex=j;
            }
        }
        //交换最小值和当前索引处的值
        exch(a,i,minIndex);
    }
}
```

时间复杂度

O(N-2)

插入排序

排序原理

1. 把所有的元素分为两组，已经排序的和未排序的；
2. 找到未排序的组中的第一个元素，向已经排序的组中进行插入；
3. 倒叙遍历已经排序的元素，依次和待插入的元素进行比较，直到找到一个元素小于等于待插入元素，那么就把待插入元素放到这个位置，其他的元素向后移动一位；

原理图解

代码示例

```
public static void sort(Comparable[] a){
    for(int i=1;i<a.length;i++){
        //定义一个变量，记录当前要参与排序的元素
        int j=i;
        //拿此时j索引处的值和j-1索引处的值进行比较，如果j-1索引处的值大，则交换数据
        //直到遇到一个j，使其前面的元素都比j小，后面的都比j大
        if (greater(a[j-1],a[j])){
            exch(a,j,j-1);
            j--;
        }else{
            break;
        }
    }
}
```

时间复杂度

O(N-2)

高级排序

希尔排序

排序原理

1. 选定一个增长量h，按照增长量h作为数据分组的依据，对数据进行分组；
2. 对分好组的每一组数据均成插入排序；
3. 减小增长量，最小减为1，重复第二步操作。

原理图解

代码示例

```
public static void sort(Comparable[] a){
    //先定义希尔算法的增长量h，默认情况下该值为数组长度
    int h = a.length;
    while (h >= 1){
        h = h/2;
    }
    //从h开始进行排序
    while(h>=1){
        //定义一个变量保存插入的元素
        for (int i=h;i<a.length;i++){
            //定义一个变量，记录当前要参与排序的元素
            for (int j=i;j>=h;j-=h){
                //获取j-h索引处的元素，与j的元素进行比较
                if (greater(a[j-h],a[j])){
                    //交换元素
                    exch(a, j-h,j);
                }else{
                    //如果插入元素已经找到了合适的位置，就重新开始
                    break;
                }
            }
        }
        //分组h的组
        h= h/2;
    }
}
```

常见增量

- Shell增量序列
  - ht = N / 2,N=N/2 = h[k] / 2，即N/2, (N / 2)/2, ... 1)
  - 使用希尔增量时希尔排序的最坏情形运行时间为O(N-2)
- Hibbert增量序列
  - h[k] = 2^k - 1,即0, 3, ... 2^k-1)
  - 使用Hibbert增量时的最坏情形运行时间为O(N-3/2)
  - 使用Hibbert增量时的平均情形运行时间为O(N-5/4) (尚未证明)
- Sedgewick增量序列
  - (1, 5, 19, 41, 109, ...)该序列中的项或者是9*4i - 9*2i + 1的形式，或者是4*i - 3*2i + 1的形式
  - 使用Sedgewick增量时的最坏情形运行时间为O(N-4/3)
  - 使用Sedgewick增量时的平均情形运行时间猜测为O(N-7/6)

归并排序

递归

- 定义
  - 定义方法时，在方法内部调用方法本身，称之为递归。
- 作用
  - 它通常把一个大型复杂的问题，层层转换为一个与原问题相似的、规模较小的问题来求解，递归策略只需要少量的程序就可以描述出解题过程所需要的大量数目，大大地减少了程序的代码量。
- 注意事项
  - 在递归中，不要无限制的调用自己，必须要有边界条件，能够让递归结束，因为每一次递归调用都会在栈内存开辟新的空间，重新开辟方法，如果递归的层级太深，很容易造成栈内存溢出。

排序原理

0. 归并排序是建立在归并操作上的一种有效的排序算法，该算法是采用分治法的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若两个有序表合并成一个有序表，称为二路归并。
1. 尽可能的一组数据拆分成两个元素相等的子项，并对每一个子项继续拆分，直到拆分出现每个子项的元素个数是1为止。
2. 将相邻的两个子项进行合并成一个有序的大项。
3. 不断的重复步骤2，直到最终只有一个项为止。

原理图解

代码示例

```
/**
* 对指定数组进行归并
* 参数：a：要进行归并的数组
* 参数：aux：辅助数组，用于在进行归并操作的过程中使用
* 参数：lo：要进行归并的数组的开始索引
* 参数：mid：要进行归并的数组的中间索引
* 参数：hi：要进行归并的数组的结束索引
* 作用：将a数组从lo到mid和mid+1到hi的两个有序子数组进行归并，使a数组从lo到hi成为有序的数组
* 注意：aux数组的长度一定要和a数组的长度相等，且aux数组的索引应该从0开始
```