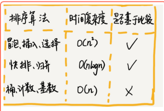
7.O(n^2)排序

1.常见排序算法分类



排序算法	时间复杂度	是否基于比较
冒泡、插入、选择	O(n²)	✓
快排、归并	O(nlogn)	✓
桶、计数、基数	O(n)	✗

2.排序算法分析

效率评价

1. 最好情况、最坏情况、平均情况时间复杂度
为什么要区分这三种时间复杂度呢?
第一，有些排序算法会区分，为了好对比，所以我们最好都做一下区分。
第二，对于要排序的数据，有的接近有序，有的完全无序。有序度不同的数据，对于排序的执行时间肯定是有影响的，我们要知道排序算法在不同数据下的性能表现。

2. 时间复杂度的系数、常数 、低阶
时间复杂度反应的是数据规模 n 很大的时候的一个增长趋势，所以它表示的时候会忽略系数、常数、低阶。但是实际的软件开发中，我们排序的可能是 10 个、100 个、1000 个这样规模很小的数据，所以，在对同一阶时间复杂度的排序算法性能对比的时候，我们就要把系数、常数、低阶也考虑进来

3. 比较次数和交换（或移动）次数
基于比较的排序算法的执行过程，会涉及两种操作，一种是元素比较大小，另一种是元素交换或移动。所以，如果我们在分析排序算法的执行效率的时候，应该把比较次数和交换（或移动）次数也考虑进去

内存消耗

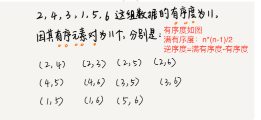
主要是用来分析空间复杂度，原地排序特指空间复杂度为O(1)的

排序稳定性

排序稳定性的含义是：两个相同的数据，例如排序前有两个3，排序后两个3的位置灭没有变化就是稳定排序，反之是不稳定排序

使用场景：例如对某一个对象的两个属性排序，先按照订单金额，金额相同则按照下单时间，这种情况下使用稳定排序先按照下单时间排序，再使用金额排序，就会更有效率，即便遇到金额相同的也会保持之前的顺序

逆序度



2，4，3，1，5，6 这组数据的有序度为11，
因其有序元素对为11个，分别是：

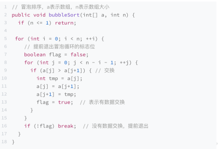
(2，4)　(2，3)　(2，5)　(2，6)
(4，5)　(4，6)　(3，5)　(3，6)
(1，5)　(1，6)　(5，6)

有序度如图
满有序度：n*(n-1)/2
逆序度=满有序度-有序度

3.冒泡排序

特性
1.排序过程中只占用临时空间，属于原地排序算法
2.两个相同数的不会发生交换，所以属于稳定排序
3.最好、最坏、平均复杂度
最好是本身数据有序，只做一次冒泡时间复杂度O(n)
最坏是全部逆序，复杂度是O(n^2)
平均也是O(n^2)

冒泡排序和插入排序对比
我们把执行一个赋值语句的时间粗略地计为单位时间（unit_time），然后分别用冒泡排序和插入排序对同一个逆序度是 K 的数组进行排序。用冒泡排序，需要 K 次交换操作，每次需要 3 个赋值语句，所以交换操作总耗时就是 3*K 单位时间。而插入排序中数据移动操作只需要 K 个单位时间



```java
// 冒泡排序，a表示数组，n表示数组大小
public void bubbleSort(int[] a, int n) {
  if (n <= 1) return;

 for (int i = 0; i < n; ++i) {
    // 提前退出冒泡循环的标志位
    boolean flag = false;
    for (int j = 0; j < n - i - 1; ++j) {
      if (a[j] > a[j+1]) { // 交换
        int tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
        flag = true;  // 表示有数据交换
      }
    }
    if (!flag) break;  // 没有数据交换，提前退出
  }
}
```

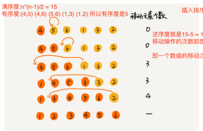
代码

4.插入排序

特性
1.排序过程中只占用临时空间，属于原地排序算法
2.两个相同数的不会发生交换，所以属于稳定排序
3.最好、最坏、平均复杂度
最好是本身数据有序，只做一次冒泡时间复杂度O(n)
最坏是全部逆序，复杂度是O(n^2)
平均也是O(n^2)

插入排序说明

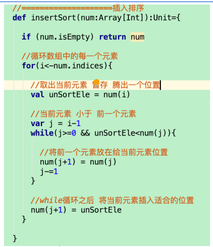
满序度:n*(n-1)/2 = 15
有序度:(4,5) (4,6) (5,6) (1,3) (1,2) 所以有序度是5



逆序度就是15-5 = 10
移动操作的次数如图3+3+4=10

即一个数组的移动次数是固定的

插入排序图解

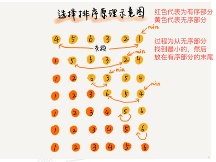


```scala
//=====================插入排序
def insertSort(num:Array[Int]):Unit={

  if (num.isEmpty) return num

  //循环数组中的每一个元素
  for(i<-num.indices){

    //取出当前元素 暂存 腾出一个位置
    val unSortEle = num(i)

    //当前元素 小于 前一个元素
    var j = i-1
    while(j>=0 && unSortEle<num(j)){

      //将前一个元素放在给当前元素位置
      num(j+1) = num(j)
      j-=1
    }

    //while循环之后 将当前元素插入适合的位置
    num(j+1) = unSortEle
  }

}
```

理解：左边开始作为有序部分，逐个+1，纳入外循环，内循环再逐个比较

5.选择排序

特性
空间复杂度为 O(1)，是一种原地排序算法。
最好情况时间复杂度、最坏情况和平均情况时间复杂度都为 O(n2)



红色代表为有序部分
黄色代表无序部分

过程为从无序部分找到最小的，然后放在有序部分的末尾

排序过程

XMind:ZEN | 试用模式