插入排序

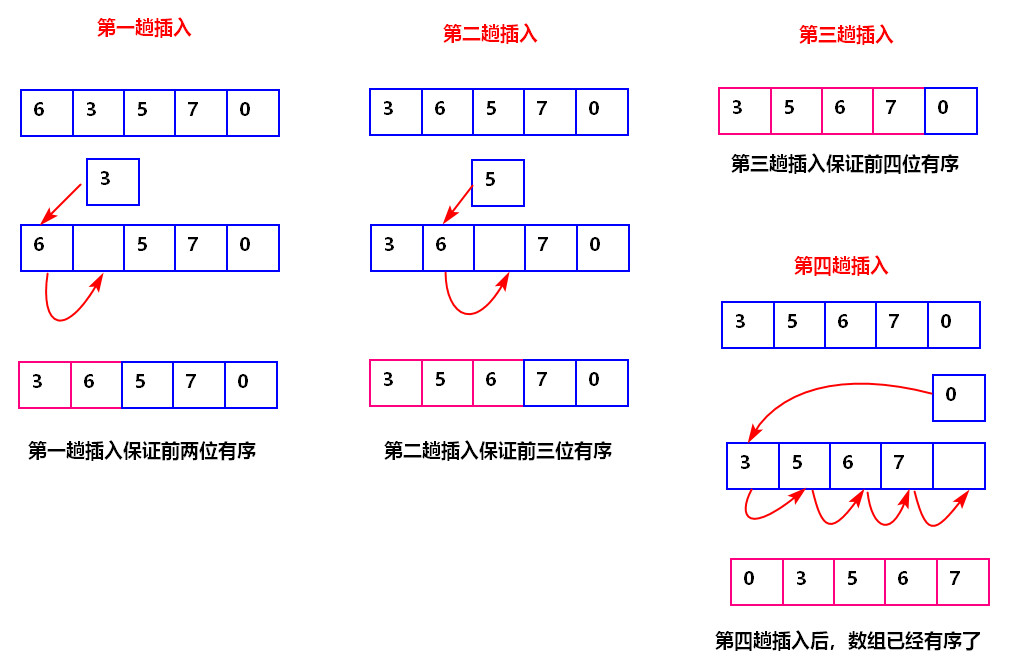
在计算机的实现中，为了给要插入的元素腾出空间，我们需要将其余所有元素在插入之前都向右移动一位。这种算法叫做插入排序

与选择排序一样，当前索引左边的所有元素都是有序的，但它们的最终位置还不确定，为了给更小的元素腾出空间，它们可能就会被移动。但是当索引到达数组的右边时，数组排序就完成了。

和选择排序不同的是，插入排序所需的时间取决于输入中元素的初始顺序。

例如：对一个很大且其中国的元素已经有序（或接近有序）的数组进行排序将会比对随机顺序的数组或是逆序数组进行排序要快得多。

对于随机排列的长度为N且主键不重复的数组，平均情况下插入排序需要～N2/4次比较以及～N2/4次交换。最坏情况下需要～N2/2次比较和～N2/2次比较，最好情况下需要N-1次比较和0次交换。

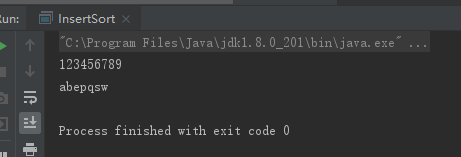


图摘自<https://cloud.tencent.com/developer/article/1350569>

插入排序代码如下：

1. **package** com.lagoon.sort;
3. /\*\*
4. \* @Author WinkiLee
5. \* @Date 2019/4/11 9:44
6. \* @Description
7. \*/
8. **public** **class** InsertSort {
10. **public** **static** **void** sort(Comparable[] a){
11. /\*\*
12. \* 算法编写区域
13. \*/
14. //将a[]按升序排列
15. **int** N=a.length;
16. **for** (**int** i=1;i<N;i++){
17. //将a[i]插入到a[i-1],a[i-2],a[i-3]....中
18. //如果当前a[i]能在左侧插入，也就是较小的话
19. **for** (**int** j=i;j>0 && less(a[j],a[j-1]);j--){
20. exch(a,j,j-1);
21. }
22. }
23. }
25. /\*\*
26. \* 比较大小
27. \* @param v
28. \* @param w
29. \* @return false or true
30. \*/
31. **private** **static** **boolean** less(Comparable v,Comparable w){
32. **return** v.compareTo(w)<0;
33. }
35. /\*\*
36. \* 交换数
37. \* @param a
38. \* @param i
39. \* @param j
40. \*/
41. **private** **static**  **void** exch(Comparable[] a,**int** i,**int** j){
42. Comparable t=a[i];
43. a[i]=a[j];
44. a[j]=t;
45. }
47. /\*\*
48. \* 打印结果
49. \* @param a
50. \*/
51. **private**  **static** **void** show(Comparable[] a){
52. //在单行中打印数组
53. **for** (**int** i=0;i<a.length;i++){
54. System.out.print(a[i]+"");
55. }
56. System.out.println("");
57. }
59. /\*\*
60. \* 测试元素数组是否有序
61. \* @param a
62. \* @return true or false
63. \*/
64. **public** **static** **boolean** isSorted(Comparable[] a){
65. //测试元素数组是否有序
66. **for** (**int** i=1;i<a.length;i++){
67. **if** (less(a[i],a[i-1])){
68. **return** **false**;
69. }
70. }
71. **return** **true**;
72. }
74. /\*\*
75. \* main方法及测试
76. \* @param args
77. \*/
78. **public** **static** **void** main(String[] args) {
80. Comparable[] a ={6,4,1,3,5,9,7,8,2};
81. Comparable[] b ={'a','e','w','p','s','b','q'};
82. sort(a);
83. sort(b);
84. show(a);
85. show(b);
86. }
88. }

测试结果如下：



有以下几种典型的部分有序数组：

1. 数组中的每个元素距离它的最终位置都不远
2. 一个有序的大数组接一个小数组
3. 数组中只有几个元素的位置不正确

插入排序对这样的数组很有效，而选择排序则不然。事实上，当倒置的数量很少时，插入排序很可能比其他任何算法都要快

插入排序需要的交换操作和数组中倒置的数量相同，需要的比较次数大于等于倒置的数量，小于等于倒置的数量加上数组的大小再减一

倒置指的是数组中两个顺序颠倒的元素。例如EXAMPLE中有11对倒置，E-A, X-A, X-M, X-P, X-L, M-E, P-L, P-E, L-E.

如果数组中倒置的数量小于数组大小的某个倍数，可以说这个数组就是部分有序的

要大幅提高插入排序的速度并不难，只需要在内循环中将较大的元素都向右移动而不总是交换两个元素（这样访问的数组就能减半）。

总的来说插入排序对于部分有序的数组十分高效，也很适合小规模数组，这很重要，因为这些类型的数组在实际应用中经常出现。