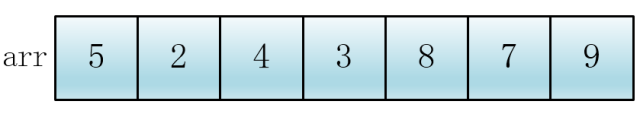
快速排序

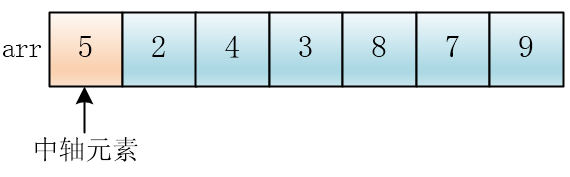
快速排序，顾名思义，是一种排序速度非常快的排序方法，该算法之所以很快是因为高度优化的内部循环。

快速排序是一种采用分治思想，在实践中通常运行较快的一种排序算法，思路如下：

对于一个无序数组

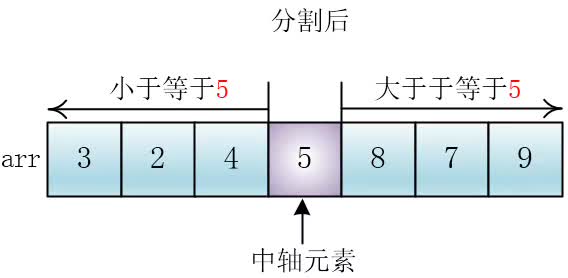


首先，任意选取一个元素，通常选取数组的第一个元素，该元素成为中轴元素



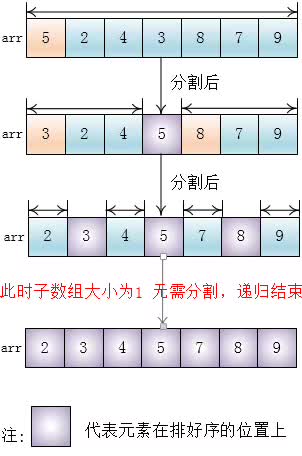
然后将大于或者等于中轴元素的元素放在右边，小于或者等于中轴元素的元素放在左边

上面两个过程，选元素和调整位置成为分割（partition）



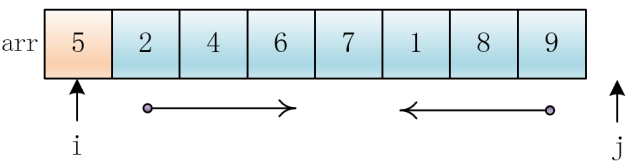
然后对左右两个子数组分别按照同样的方法进行分割操作（递归进行）

一直递归分割到子数组只有一个或者零个元素为止，此时整个数组有序



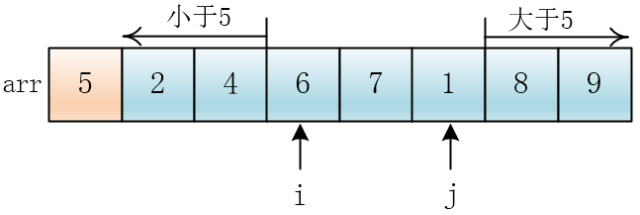
分割操作的进行过程：

首先，用两个变量i和j从数组两边开始向中间扫描，i从右走，j从左走

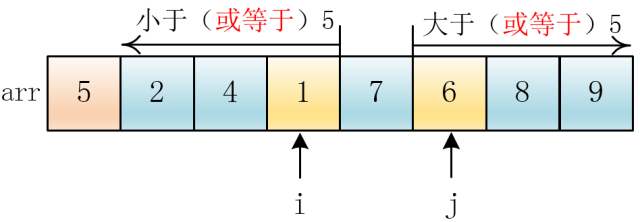


i初始化为第一个元素的下标，j初始化为最后一个元素的下标加1

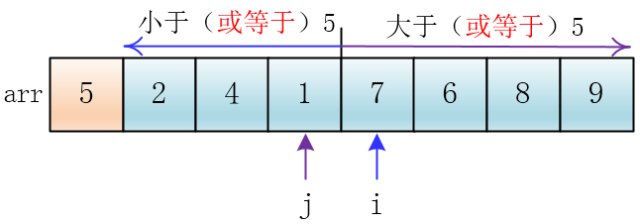
i往右走，直到遇见比中轴元素大的（或等于）元素停止移动，j向左走，直到遇到比中轴元素小的（或等于）的元素停止移动。



此时，如果i<j，则交换i，j所指向的元素

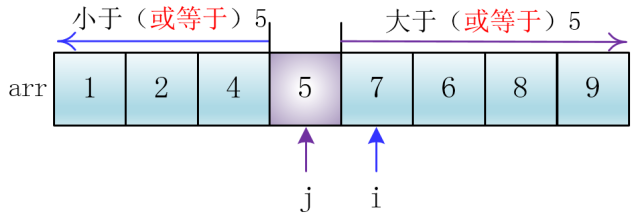


然后继续向右走向左走，直到i>=j，整个扫描停止



此时i对应元素的左边（不包含arr[i]）必定小于或等于5，j对应元素的右边（不包含arr[j]）必定大于或等于5

交换中轴元素5与arr[j]



分割完成

对数组arr[low…high]进行快速排序

首先进行分割操作，返回中轴元素下标j，然后对左数组arr[low…j-1]和右数组arr[j+1…high]分别递归进行排序

什么时候递归终止？当然是数组大小为小于等于1（0或1）时

代码如下

1. **package** com.lagoon.sort;
3. **import** java.util.Arrays;
5. /\*\*
6. \* @Author WinkiLee
7. \* @Date 2019/4/12 15:35
8. \* @Description
9. \*/
10. **public** **class** QuickSort {
12. /\*\*
13. \* 对数组arr[low...high]进行快速排序
14. \*/
15. **private** **static** **void** quickSort(**int**[] arr, **int** low, **int** high) {
17. //数组小于等于1时递归终止，否则对数组进行排序
18. **if** (low < high) {
19. //对数组arr[low...high]进行分割，返回中轴元素下标
20. **int** j = partition(arr, low, high);
22. //对左数组arr[low...j-1]进行快速排序
23. quickSort(arr, low, j - 1);
25. //对右数组arr[j+1...high]进行快速排序
26. quickSort(arr, j + 1, high);
27. }
28. }

31. /\*\*
32. \* 选出中轴元素，将数组arr[low..high]分割成三部分
33. \* arr[low...j-1]<=arr[j]<=arr[j+...high]
34. \*
35. \* @param arr
36. \* @param low
37. \* @param high
38. \* @return
39. \*/
40. **private** **static** **int** partition(**int**[] arr, **int** low, **int** high) {
41. /\*\*
42. \* 左右扫描指针
43. \*/
44. **int** i = low, j = high + 1;
46. /\*\*
47. \* 中轴元素v
48. \*/
49. **int** v = arr[low];
51. //i,j扫描数组
52. **while** (**true**) {
53. //遇到比中轴元素v大的（或等于）元素时（break）
54. **while** (arr[++i] < v) {
55. **if** (i == high) {
56. **break**;
57. }
58. }
59. **while** (arr[--j] > v) {
60. **if** (j == low) {
61. **break**;
62. }
63. }
64. //如果i>=j，扫描结束
65. **if** (i >= j) {
66. **break**;
67. } **else** {
68. exch(arr, i, j);
69. }
70. }
71. exch(arr, low, j);
72. **return** j;
73. }
75. /\*\*
76. \* 交换数
77. \*
78. \* @param a
79. \* @param i
80. \* @param j
81. \*/
82. **private** **static** **void** exch(**int**[] a, **int** i, **int** j) {
83. **int** t = a[i];
84. a[i] = a[j];
85. a[j] = t;
86. }
88. /\*\*
89. \* main方法
90. \*/
91. **public** **static** **void** main(String[] args) {
92. **int**[] arr=**new** **int**[]{5,1,2,8,4,9,7,10,0,14,22};
93. quickSort(arr,0,arr.length-1);
94. System.out.println(Arrays.toString(arr));
95. }
96. }

