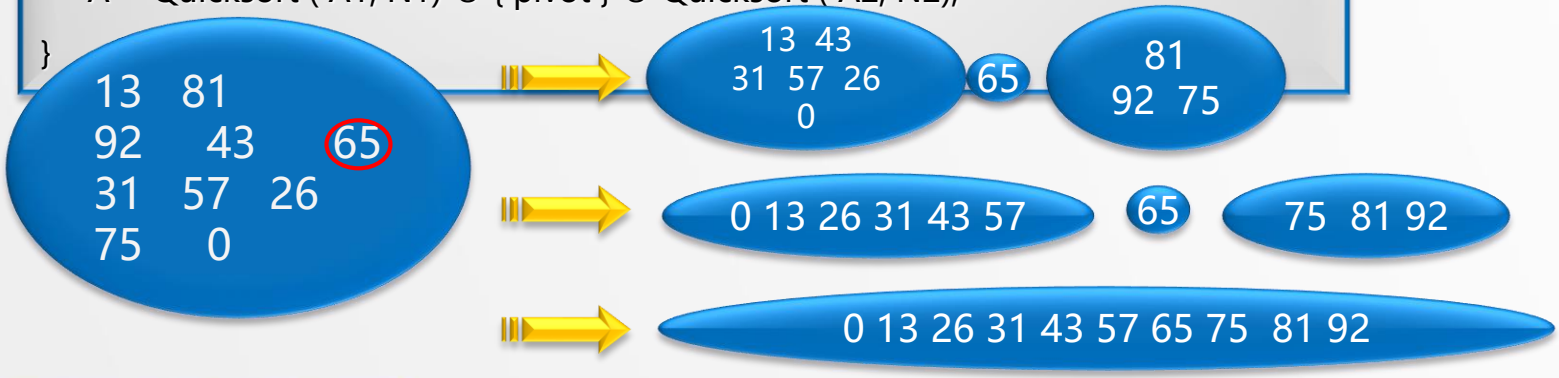
快速排序

快速排序 | 实际使用中已知的最快的算法

1. 算法思想

```
void Quicksort ( ElementType A[ ], int N )
{
    if ( N < 2 ) return;
    pivot = pick any element in A[ ];
    Partition S = { A[ ] \ pivot } into two disjoint sets:
            A1={ a∈S | a <= pivot } and A2={ a ∈ S | a >== pivot };
    A = Quicksort ( A1, N1) ∪ { pivot } ∪ Quicksort ( A2, N2);
}
```

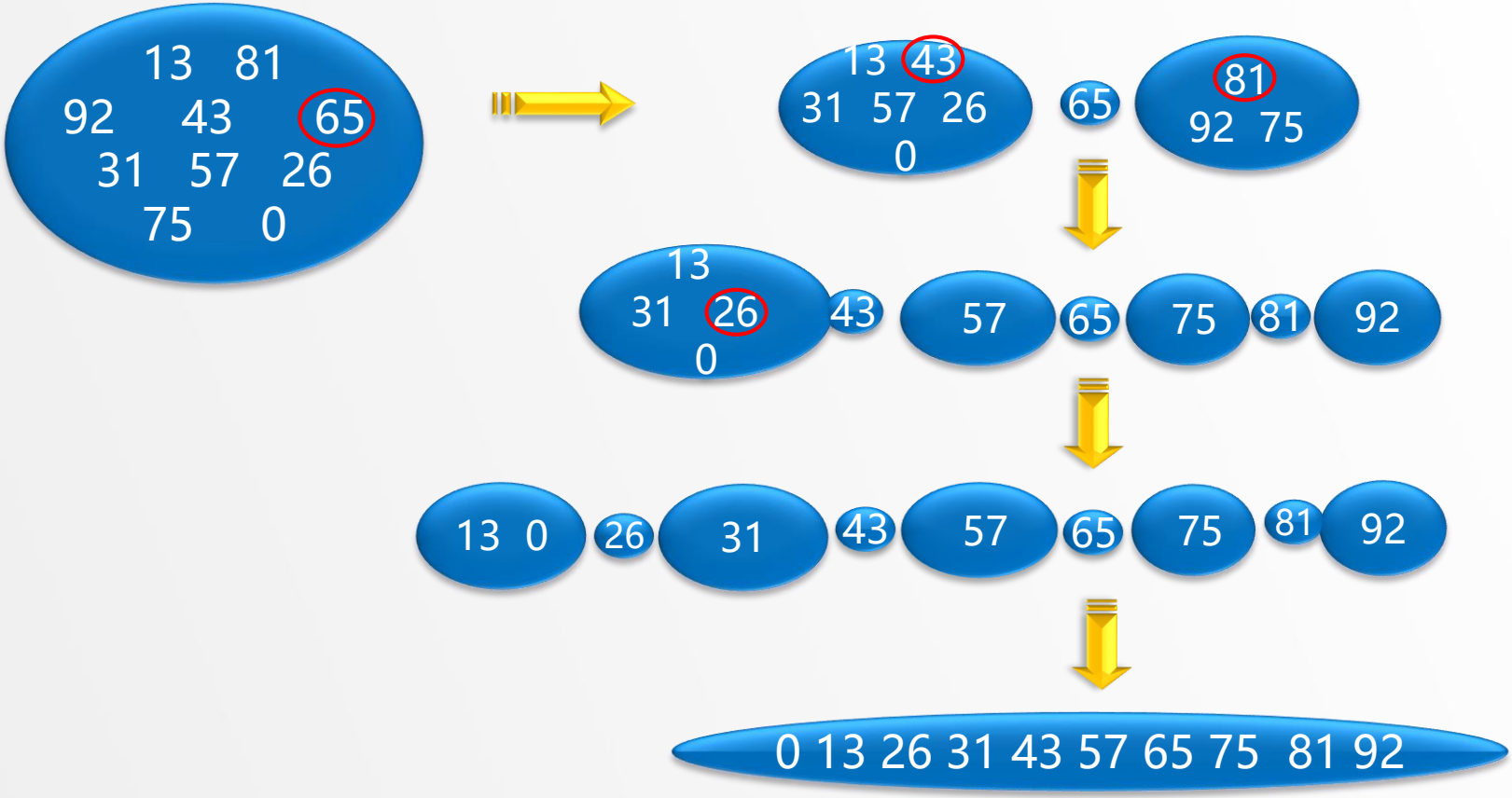


快速排序 | 实际使用中已知的最快的算法

1. 算法思想

```
void Quicksort ( ElementType A[ ], int N )
{
    if ( N < 2 )
?   pivot = pick any element in A[ ];
?   Partition S = { A[ ] \ pivot } into two disjoint sets:
            A1={ a∈S | a <= pivot } and A2={ a ∈ S | a >= pivot };
    A = Quicksort ( A1, N1) ∪ { pivot } ∪ Quicksort ( A2, N2
}
```

The best case $T(N) = O(N \log N)$


13 81
92 43 65
31 57 26
75 0
13 43
31 57 26
0
65
81
92 75
13
31 26
0
43
57
65
75
81
92
13 0
26
31
43
57
65
75
81
92
0 13 26 31 43 57 65 75 81 92

2. 枢纽

☞ 错误的方法: **Pivot = A[0]**

最糟糕的情况: A[ ] 有序或者逆序– quicksort = O(N^2) ☹

☞ 安全的方法: **Pivot = random select from A[ ]**

☹ 随机数生成很花时间(expensive)

☞ 3者取中法:

Pivot = median (left, center, right)

这种方法能够排除序列有序的枢纽是最小或者最大值情况，实际运行时间能够减少约5%.

选用第一个关键字为枢纽为例

第一个元素为枢纽

例 初始关键字：

	x							
	27	38	13	49	76	97	65	50
	i	i	i	i	j	j	j	j

完成一趟排序：(27 38 13) 49 (76 97 65 50)

分别进行快速排序：(13) 27 (38) 49 (50 65) 76 (97)

快速排序结束：13 27 38 49 50 65 76 97

排序过程：

- 对r[s……t]中记录进行一趟快速排序，附设两个指针i和j，设划分元记录rp=r[s]，x=rp.key
  - 初始时令i=s,j=t
  - 首先从j所指位置向前搜索第一个关键字小于x 的记录，并和rp交换
  - 再从i所指位置起向后搜索，找到第一个关键字大于x的记录，和rp交换
  - 重复上述两步，直至i==j为止
  - 再分别对两个子序列进行快速排序，直到每个子序列只含有一个记录为止

```
void qksort(JD r[], int t, int w){//t=low,w=high
    int i, j, k;
    JD x;
    if (t >= w)  return;
    i = t; j = w; x = r[i].key;
    while (i<j){
        while ((i<j) && (r[j].key >= x.key))  j--;//枢轴后面的值
        大于枢轴
        if (i<j) { r[i] = r[j]; i++; }//当不满足时，与枢轴交换
        while ((i<j) && (r[i].key <= x.key))  i++;//枢轴前面的值
        小于枢轴
        if (i<j) { r[j] = r[i]; j--; }//不满足，与枢轴交换
    }
    r[i].key = x;
    qksort(r, t, j - 1);
    qksort(r, j + 1, w);
}
```

算法描述

算法评价

时间复杂度

- 最好情况（每次总是选到中间值作划分元）
  $T(n)=O(n\log_2 n)$
- 最坏情况（每次总是选到最小或最大元素作划分元）
  $T(n)=O(n^2)$

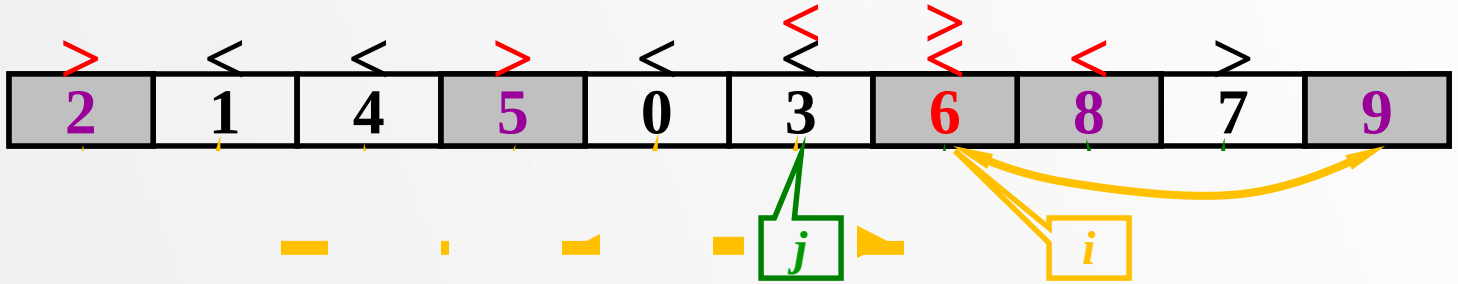
$T(n)=O(n^2)$

空间复杂度：需栈空间以实现递归

- 最坏情况：$S(n)=O(n)$
- 一般情况：$S(n)=O(\log_2 n)$

三者取中法

Input: 8 1 4 9 6 3 5 2 7 0



快速排序算法特点

- 快速排序算法是不稳定的

  对待排序序列 49 49' 38 65，

  快速排序结果为: 38 49' 49 65

- 快速排序的性能跟初始序列中**关键字的排列**和选取的**枢纽**有关
- 当初始序列按关键字有序(正序或逆序)时，性能最差，蜕化为冒泡排序，时间复杂度为$O(n^2)$
- 常用“三者取中”法来选取划分记录，即取首记录r[s].key.尾记录r[t].key和中间记录r[(s+t)/2].key三者的中间值为划分记录。
- 快速排序算法的平均时间复杂度为O(nlogn)

快速排序算法特点

- 快速排序算法是不稳定的

  对待排序序列 49 49' 38 65，

  快速排序结果为: 38 49' 49 65

请尝试用三者取中法完成快速排序，并编写程序与取第一个元素为枢纽的快速排序方法进行比较测试。然后仔细研究快排还可以做哪些改进！

- 常用“三者取中”法来选取划分记录，即取首记录r[s].key.尾记录r[t].key和中间记录r[(s+t)/2].key三者的中间值为划分记录。
- 快速排序算法的平均时间复杂度为O(nlogn)

表　　快速排序算法的性能

时间复杂度			空间复杂度	稳定性	复杂性
平均情况	最坏情况	最好情况			
O(*n*log*n*)	O(n^2)	O(*n*log*n*)	O(log*n*)	不稳定	较复杂

[例题] 对 8 个元素的线性表进行快速排序，在最好的情况下，元素间的比较次数为(D.)。

A. 7　　B. 8　　C. 12　　D. 13