

Lesson6--排序

【本节目标】

- 1.排序的概念及其运用
- 2.常见排序算法的实现
- 3.排序算法复杂度及稳定性分析

1.排序的概念及其运用

1.1排序的概念

排序：所谓排序，就是使一串记录，按照其中的某个或某些关键字的大小，递增或递减的排列起来的操作。

稳定性：假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中， $r[i]=r[j]$ ，且 $r[i]$ 在 $r[j]$ 之前，而在排序后的序列中， $r[i]$ 仍在 $r[j]$ 之前，则称这种排序算法是稳定的；否则称为不稳定的。

内部排序：数据元素全部放在内存中的排序。

外部排序：数据元素太多不能同时放在内存中，根据排序过程的要求不能在内外存之间移动数据的排序。

1.2排序运用

综合销量评论数新品价格

配送至陕西省西安市灞桥区京东物流货到付款仅显示有货京东国际可配送全球新品PLUS专享拍拍二手焕新季

为了方便您找到满意的商品，我们已为您屏蔽部分不相关的商品，[查看更多商品](#) >

按某个商品维度排序

陕西预定



焕新季

¥4499.00

商品手机 Apple iPhone 11 (A2223) 64GB 紫色 移动联通电信4G手机 双卡双

500万+条评价 去看二手

Apple产品京东自营...

自营 (赠)

陕西无货



焕新季

¥3999.00

Apple iPhone XR (A2108) 64GB 黑色 移动联通电信4G手机 双卡双待

200万+条评价 去看二手

Apple产品京东自营...

自营

陕西无货



焕新季

¥1048.00

商品手机 Redmi Note8 4800万全场景四摄 4000mAh长续航 高通骁龙665 18W快

100万+条评价

小米京东自营旗舰店

自营 (秒杀) (赠)

陕西无货



焕新季

¥899.00

荣耀9i 4GB+64GB 幻夜黑 移动联通电信4G全面屏手机 双卡双待

100万+条评价

荣耀京东自营旗舰店

自营 (券980-00)

陕西无货



焕新季

¥699.00

Redmi 8A 5000mAh大电量 大字体大音量 大内存 骁龙八核处理器 AI人脸解锁 莱茵

100万+条评价

小米京东自营旗舰店

自营 (赠)

排名	院校名称	总分	类型	所在地	批次	历年排名
1	北京大学	100	综合类	北京	本科一批	↗
2	清华大学	99.58	理工类	北京	本科一批	↗
3	浙江大学	82.56	综合类	浙江	本科一批	↗
4	复旦大学	82.17	综合类	上海	本科一批	↗
5	中国人民大学	81.51	综合类	北京	本科一批	↗
6	上海交通大学	81.5	综合类	上海	本科一批	↗
7	武汉大学	81.49	综合类	湖北	本科一批	↗
8	南京大学	80.7	综合类	江苏	本科一批	↗
9	解放军国防科学技术大学	80.31	军事类	湖南	本科一批	↗
10	中山大学	76.16	综合类	广东	本科一批	↗
11	吉林大学	75.99	综合类	吉林	本科一批	↗
12	华中科技大学	75.04	综合类	湖北	本科一批	↗
13	四川大学	74.5	综合类	四川	本科一批	↗
14	天津大学	73.54	理工类	天津	本科一批	↗
15	南开大学	73.5	综合类	天津	本科一批	↗
16	西安交通大学	73.5	综合类	陕西	本科一批	↗

1.3 常见的排序算法



```
1 // 排序实现的接口
2
3 // 插入排序
4 void InsertSort(int* a, int n);
5
6 // 希尔排序
```

```

7 void ShellSort(int* a, int n);
8
9 // 选择排序
10 void SelectSort(int* a, int n);
11
12 // 堆排序
13 void AdjustDwon(int* a, int n, int root);
14 void HeapSort(int* a, int n);
15
16 // 冒泡排序
17 void BubbleSort(int* a, int n)
18
19 // 快速排序递归实现
20 // 快速排序hoare版本
21 int PartSort1(int* a, int left, int right);
22 // 快速排序挖坑法
23 int PartSort2(int* a, int left, int right);
24 // 快速排序前后指针法
25 int PartSort3(int* a, int left, int right);
26 void QuickSort(int* a, int left, int right);
27
28 // 快速排序 非递归实现
29 void QuickSortNonR(int* a, int left, int right)
30
31 // 归并排序递归实现
32 void MergeSort(int* a, int n)
33 // 归并排序非递归实现
34 void MergeSortNonR(int* a, int n)
35
36 // 计数排序
37 void CountSort(int* a, int n)
38
39 // 测试排序的性能对比
40 void TestOP()
41 {
42     srand(time(0));
43     const int N = 100000;
44     int* a1 = (int*)malloc(sizeof(int)*N);
45     int* a2 = (int*)malloc(sizeof(int)*N);
46     int* a3 = (int*)malloc(sizeof(int)*N);
47     int* a4 = (int*)malloc(sizeof(int)*N);
48     int* a5 = (int*)malloc(sizeof(int)*N);
49     int* a6 = (int*)malloc(sizeof(int)*N);
50
51     for (int i = 0; i < N; ++i)
52     {
53         a1[i] = rand();
54         a2[i] = a1[i];
55         a3[i] = a1[i];
56         a4[i] = a1[i];
57         a5[i] = a1[i];
58         a6[i] = a1[i];
59

```

```
60     }
61
62     int begin1 = clock();
63     InsertSort(a1, N);
64     int end1 = clock();
65
66     int begin2 = clock();
67     ShellSort(a2, N);
68     int end2 = clock();
69
70     int begin3 = clock();
71     SelectSort(a3, N);
72     int end3 = clock();
73
74     int begin4 = clock();
75     HeapSort(a4, N);
76     int end4 = clock();
77
78     int begin5 = clock();
79     QuickSort(a5, 0, N-1);
80     int end5 = clock();
81
82     int begin6 = clock();
83     MergeSort(a6, N);
84     int end6 = clock();
85
86     printf("InsertSort:%d\n", end1 - begin1);
87     printf("ShellSort:%d\n", end2 - begin2);
88     printf("SelectSort:%d\n", end3 - begin3);
89     printf("HeapSort:%d\n", end4 - begin4);
90     printf("QuickSort:%d\n", end5 - begin5);
91     printf("MergeSort:%d\n", end6 - begin6);
92
93     free(a1);
94     free(a2);
95     free(a3);
96     free(a4);
97     free(a5);
98     free(a6);
99 }
```

排序OJ(可使用各种排序跑这个OJ)[OJ链接](#)

2. 常见排序算法的实现

2.1 插入排序

2.1.1 基本思想:

直接插入排序是一种简单的插入排序法，其基本思想是：

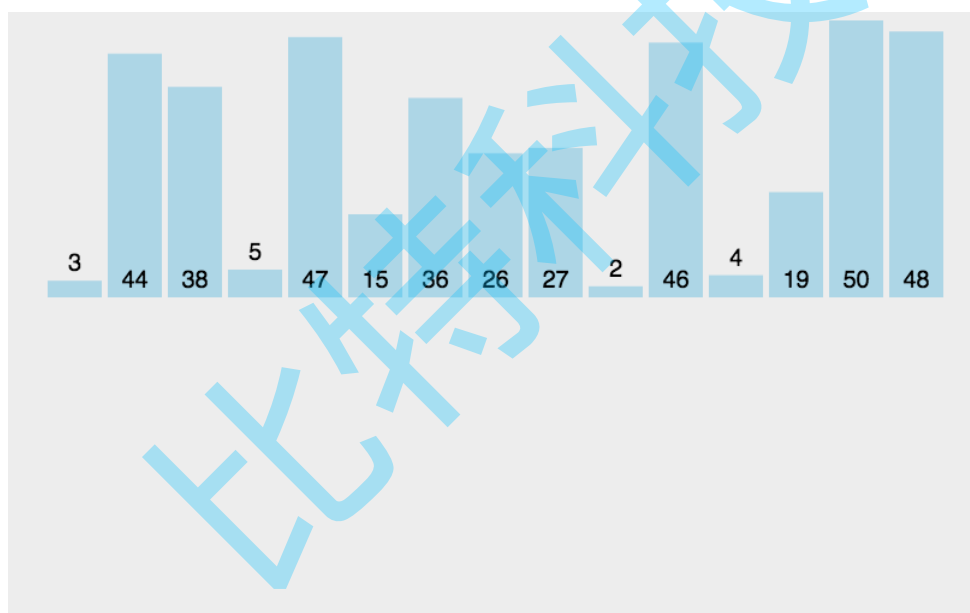
把待排序的记录按其关键码值的大小逐个插入到一个已经排好序的有序序列中，直到所有的记录插入完为止，得到一个新的有序序列。

实际中我们玩扑克牌时，就用了插入排序的思想



2.1.2直接插入排序:

当插入第 $i(i \geq 1)$ 个元素时，前面的 $array[0], array[1], \dots, array[i-1]$ 已经排好序，此时用 $array[i]$ 的排序码与 $array[i-1], array[i-2], \dots$ 的排序码顺序进行比较，找到插入位置即将 $array[i]$ 插入，原来位置上的元素顺序后移

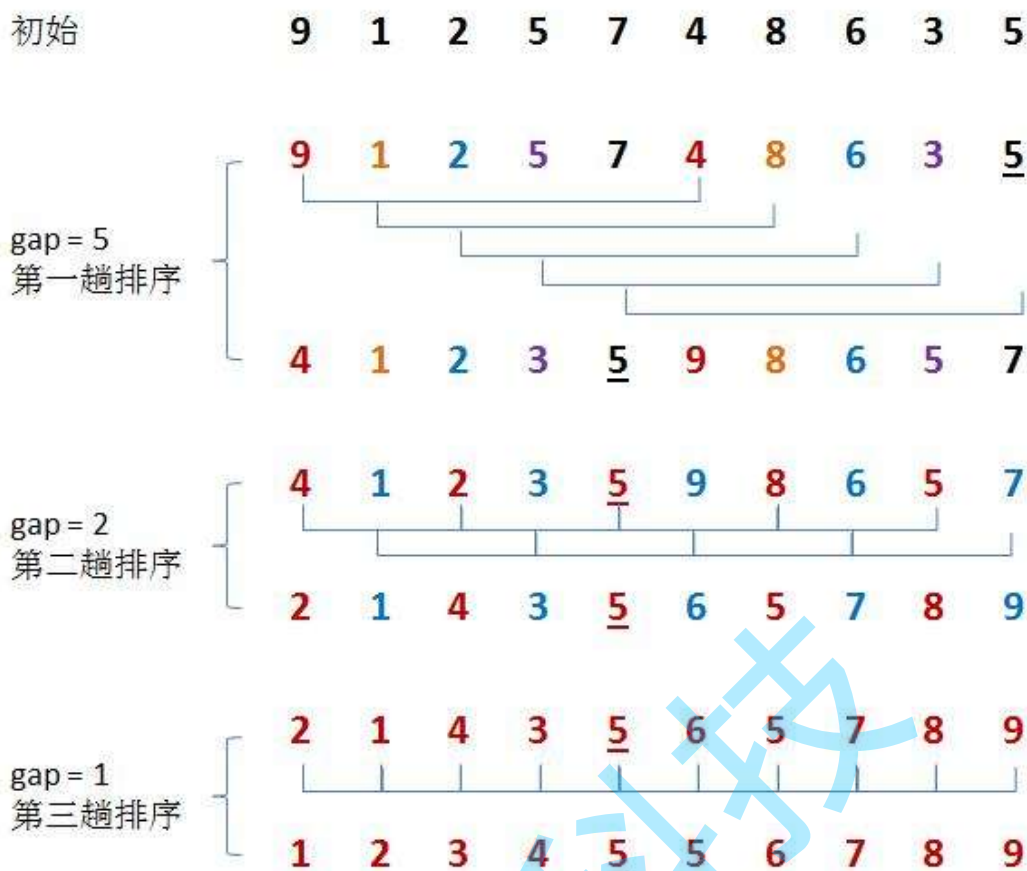


直接插入排序的特性总结:

1. 元素集合越接近有序，直接插入排序算法的时间效率越高
2. 时间复杂度: $O(N^2)$
3. 空间复杂度: $O(1)$ ，它是一种稳定的排序算法
4. 稳定性: 稳定

2.1.3 希尔排序(缩小增量排序)

希尔排序法又称缩小增量法。希尔排序法的基本思想是：先选定一个整数，把待排序文件中所有记录分成个组，所有距离为的记录分在同一组内，并对每一组内的记录进行排序。然后，取，重复上述分组和排序的工作。当到达=1时，所有记录在统一组内排好序。



希尔排序的特性总结：

1. 希尔排序是对直接插入排序的优化。
2. 当gap > 1时都是预排序，目的是让数组更接近于有序。当gap == 1时，数组已经接近有序的了，这样就会很快。这样整体而言，可以达到优化的效果。我们实现后可以进行性能测试的对比。
3. 希尔排序的时间复杂度不好计算，因为gap的取值方法很多，导致很难去计算，因此在好些书中给出的希尔排序的时间复杂度都不固定：

《数据结构(C语言版)》--- 严蔚敏

希尔排序的分析是一个复杂的问题，因为它的时间是所取“增量”序列的函数，这涉及一些数学上尚未解决的难题。因此，到目前为止尚未有人求得一种最好的增量序列，但大量的研究已得出一些局部的结论。如有人指出，当增量序列为 $\text{delta}[k] = 2^{t-k+1} - 1$ 时，希尔排序的时间复杂度为 $O(n^{3/2})$ ，其中 t 为排序趟数， $1 \leq k \leq t \leq \lfloor \log_2(n+1) \rfloor$ 。还有人在大量的实验基础上推出：当 n 在某个特定范围内，希尔排序所需的比较和移动次数约为 $n^{1.3}$ ，当 $n \rightarrow \infty$ 时，可减少到 $n(\log_2 n)^{2^{[2]}}$ 。增量序列可以有各种取法^①，但需注意：应使增量序列中的值没有除 1 之外的公因子，并且最后一个增量值必须等于 1。

《数据结构-用面向对象方法与C++描述》--- 殷人昆

gap 的取法有多种。最初 Shell 提出取 $gap = \lfloor n/2 \rfloor$, $gap = \lfloor gap/2 \rfloor$, 直到 $gap = 1$, 后来 Knuth 提出取 $gap = \lfloor gap/3 \rfloor + 1$ 。还有人提出都取奇数为好, 也有人提出各 gap 互质为好。无论哪一种主张都没有得到证明。

对希尔排序的时间复杂度的分析很困难, 在特定情况下可以准确地估算关键码的比较次数和对象移动次数, 但想要弄清关键码比较次数和对象移动次数与增量选择之间的依赖关系, 并给出完整的数学分析, 还没有人能够做到。在 Knuth 所著的《计算机程序设计技巧》第 3 卷中, 利用大量的实验统计资料得出, 当 n 很大时, 关键码平均比较次数和对象平均移动次数大约在 $n^{1.25}$ 到 $1.6n^{1.25}$ 范围内, 这是在利用直接插入排序作为子序列排序方法的情况下得到的。

因为咱们的 gap 是按照 Knuth 提出的方式取值的, 而且 Knuth 进行了大量的试验统计, 我们暂时就按照: $O(n^{1.25})$ 到 $O(1.6 * n^{1.25})$ 来算。

4. 稳定性: 不稳定

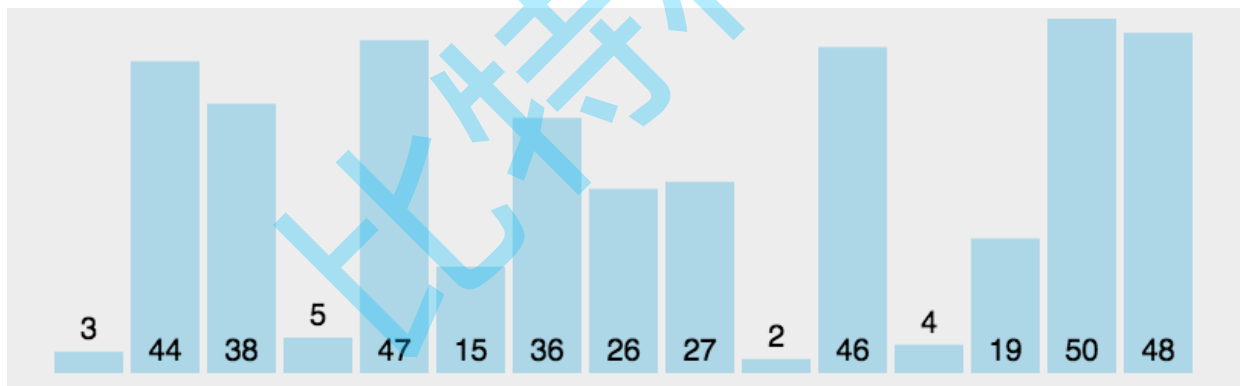
2.2 选择排序

2.2.1 基本思想:

每一次从待排序的数据元素中选出最小 (或最大) 的一个元素, 存放在序列的起始位置, 直到全部待排序的数据元素排完。

2.2.2 直接选择排序:

- 在元素集合 $array[i] \sim array[n-1]$ 中选择关键码最大 (小) 的数据元素
- 若它不是这组元素中的最后一个 (第一个) 元素, 则将它与这组元素中的最后一个 (第一个) 元素交换
- 在剩余的 $array[i] \sim array[n-2]$ ($array[i+1] \sim array[n-1]$) 集合中, 重复上述步骤, 直到集合剩余 1 个元素

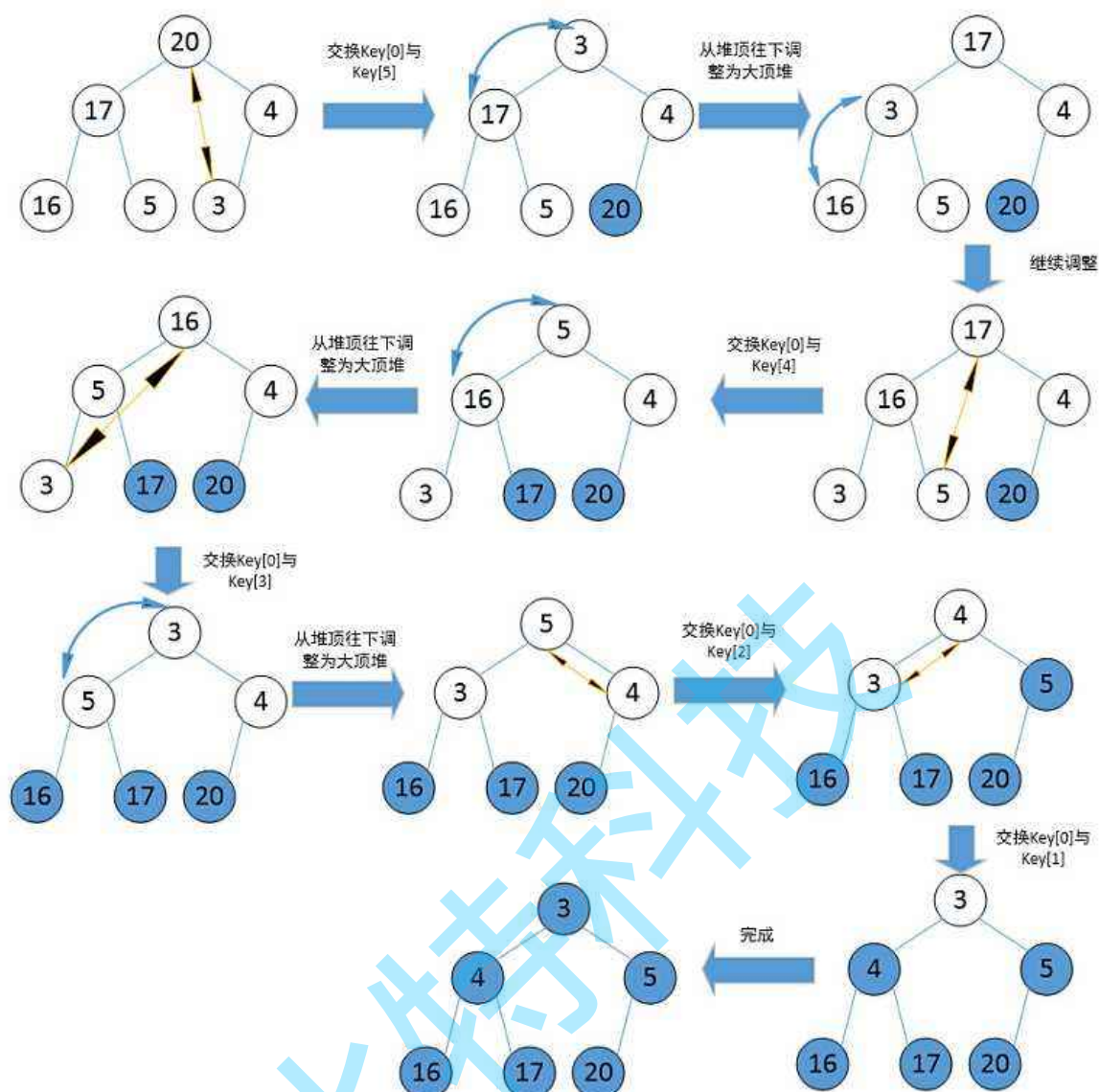


直接选择排序的特性总结:

- 直接选择排序思考非常好理解, 但是效率不是很好。实际中很少使用
- 时间复杂度: $O(N^2)$
- 空间复杂度: $O(1)$
- 稳定性: 不稳定

2.2.3 堆排序

堆排序 (Heapsort) 是指利用堆积树 (堆) 这种数据结构所设计的一种排序算法, 它是选择排序的一种。它是通过堆来进行选择数据。需要注意的是排升序要建大堆, 排降序建小堆。



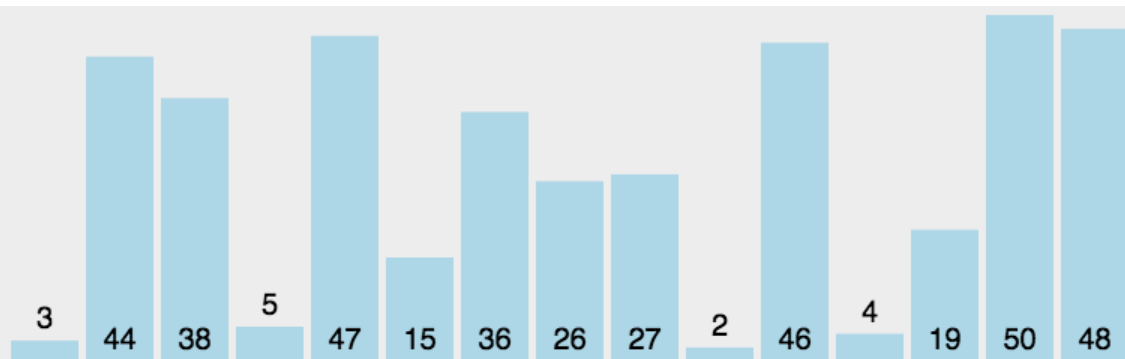
直接选择排序的特性总结:

1. 堆排序使用堆来选数，效率就高了很多。
2. 时间复杂度: $O(N \cdot \log N)$
3. 空间复杂度: $O(1)$
4. 稳定性: 不稳定

2.3 交换排序

基本思想: 所谓交换, 就是根据序列中两个记录键值的比较结果来对换这两个记录在序列中的位置, 交换排序的特点是: 将键值较大的记录向序列的尾部移动, 键值较小的记录向序列的前部移动。

2.3.1 冒泡排序



冒泡排序的特性总结：

1. 冒泡排序是一种非常容易理解的排序
2. 时间复杂度： $O(N^2)$
3. 空间复杂度： $O(1)$
4. 稳定性：稳定

2.3.2 快速排序

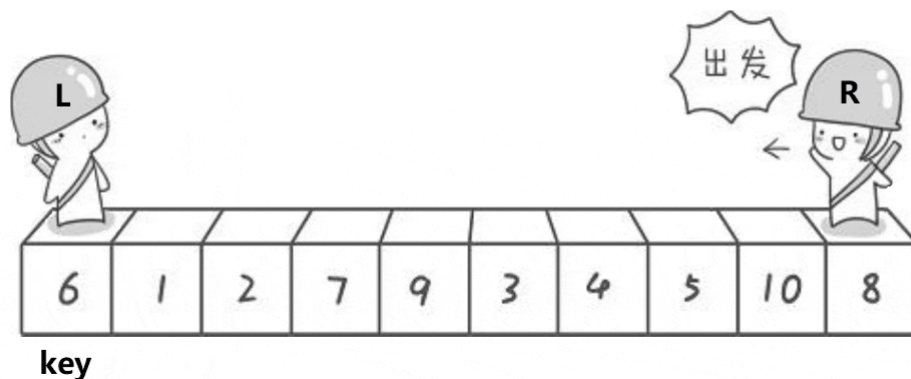
快速排序是Hoare于1962年提出的一种二叉树结构的交换排序方法，其基本思想为：**任取待排序元素序列中的某元素作为基准值，按照该排序码将待排序集合分割成两子序列，左子序列中所有元素均小于基准值，右子序列中所有元素均大于基准值，然后最左右子序列重复该过程，直到所有元素都排列在相应位置上为止。**

```
1 // 假设按照升序对array数组中[left, right)区间中的元素进行排序
2 void QuickSort(int array[], int left, int right)
3 {
4     if(right - left <= 1)
5         return;
6
7     // 按照基准值对array数组的 [left, right)区间中的元素进行划分
8     int div = partion(array, left, right);
9
10    // 划分成功后以div为边界形成了左右两部分 [left, div) 和 [div+1, right)
11    // 递归排[left, div)
12    QuickSort(array, left, div);
13
14    // 递归排[div+1, right)
15    QuickSort(array, div+1, right);
16 }
```

上述为快速排序递归实现的主框架，发现与二叉树前序遍历规则非常像，同学们在写递归框架时可想想二叉树前序遍历规则即可快速写出来，后序只需分析如何按照基准值来对区间中数据进行划分的方式即可。

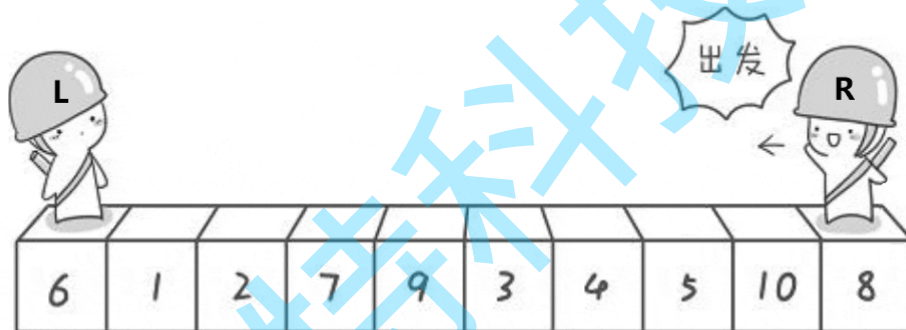
将区间按照基准值划分为左右两半部分的常见方式有：

1. hoare版本



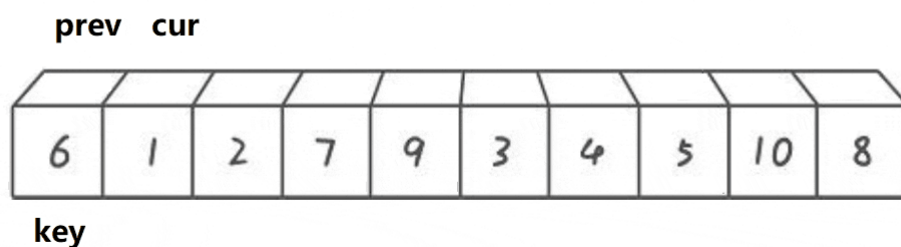
2. 挖坑法

先将第一个数据存放在临时变量 `key` 中，形成一个坑位
`key =`



3. 前后指针版本

初始时，`prev` 指针指向序列开头，
`cur` 指针指向 `prev` 指针的后一个位置




2.3.2 快速排序优化

1. 三数取中法选 `key`
2. 递归到小的子区间时，可以考虑使用插入排序

```

#define MAX_LENGTH_INSERT_SORT 7  /* 数组长度阈值 */
/* 对顺序表 L 中的子序列 L.r[low..high] 作快速排序 */
void QSort ( SqList &L, int low, int high )
{
    int pivot;
    if ( ( high-low ) > MAX_LENGTH_INSERT_SORT )
    { /* 当 high-low 大于常数时用快速排序 */
        pivot = Partition ( L, low, high ); /* 将 L.r[low..high] 一分为二, */
                                              /* 并算出枢轴值 pivot */
        QSort ( L, low, pivot-1 );          /* 对低子表递归排序 */
        QSort ( L, pivot+1, high );         /* 对高子表递归排序 */
    }
    else /* 当 high-low 小于等于常数时用直接插入排序 */
        InsertSort ( L );
}

```



2.3.2 快速排序非递归

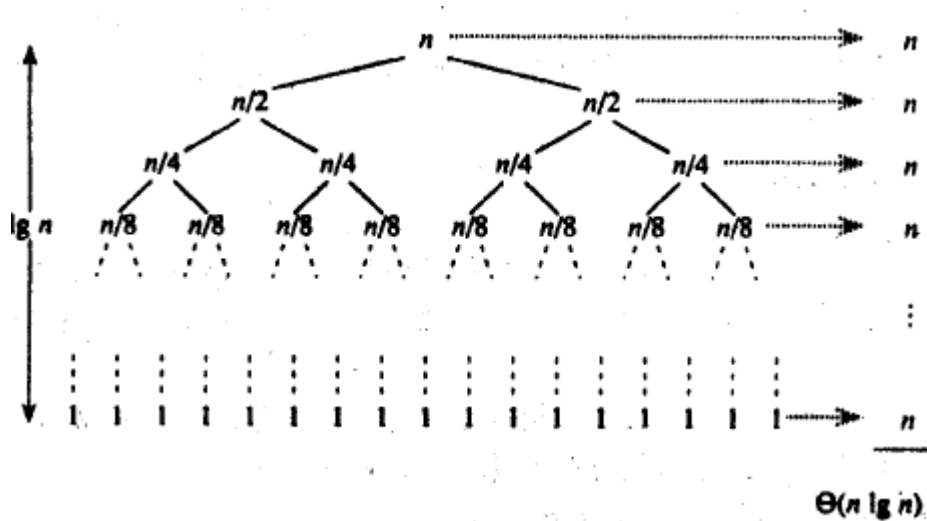
```

1 void QuickSortNonR(int* a, int left, int right)
2 {
3     Stack st;
4     StackInit(&st);
5     StackPush(&st, left);
6     StackPush(&st, right);
7
8     while (StackEmpty(&st) != 0)
9     {
10         right = StackTop(&st);
11         StackPop(&st);
12         left = StackTop(&st);
13         StackPop(&st);
14
15         if(right - left <= 1)
16             continue;
17
18         int div = PartSort1(a, left, right);
19         // 以基准值为分割点, 形成左右两部分: [left, div) 和 [div+1, right)
20         StackPush(&st, div+1);
21         StackPush(&st, right);
22
23         StackPush(&st, left);
24         StackPush(&st, div);
25     }
26
27     StackDestroy(&st);
28 }

```

快速排序的特性总结:

1. 快速排序整体的综合性能和使用场景都是比较好的, 所以才敢叫**快速**排序
2. 时间复杂度: $O(N \cdot \log N)$



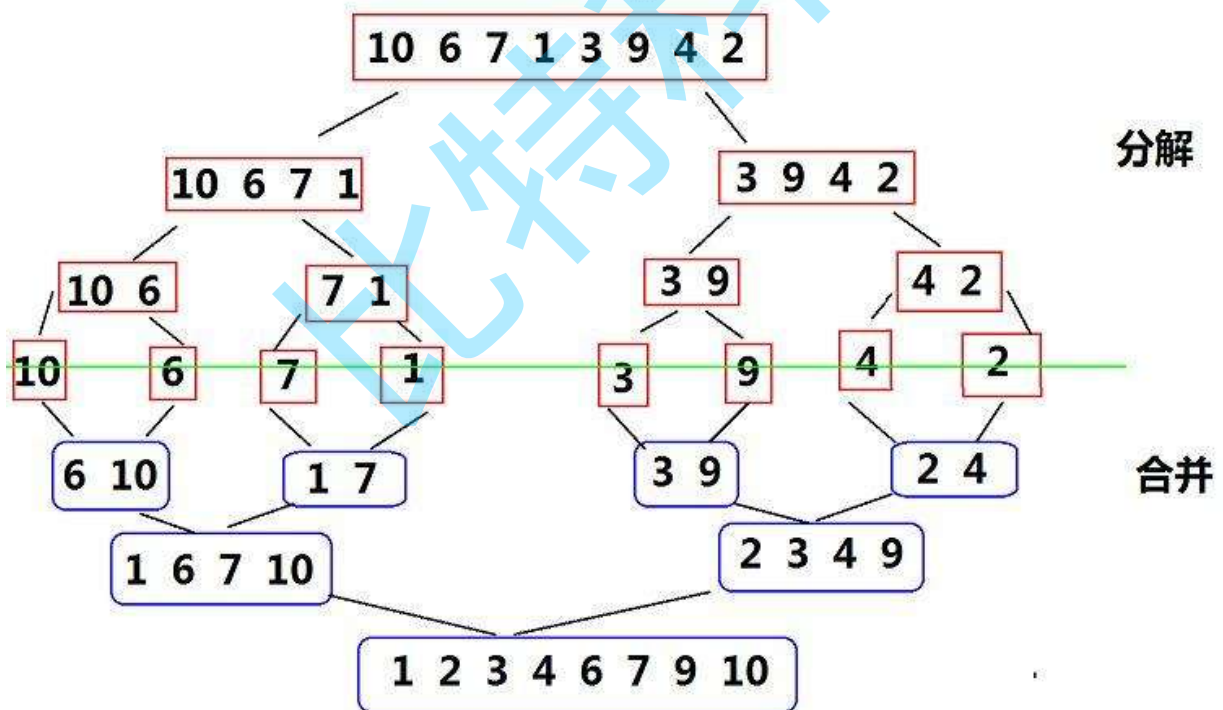
3. 空间复杂度: $O(\log N)$

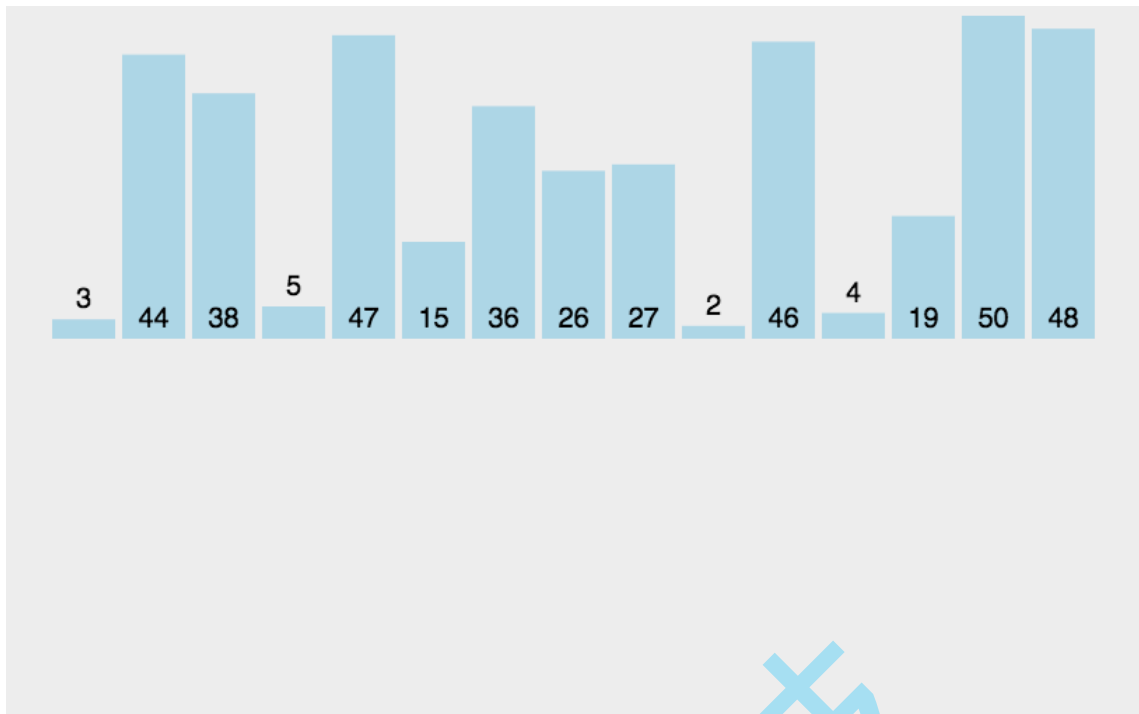
4. 稳定性: 不稳定

2.4 归并排序

基本思想:

归并排序 (MERGE-SORT) 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 (Divide and Conquer) 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。若将两个有序表合并成一个有序表,称为二路归并。归并排序核心步骤:





归并排序的特性总结:

1. 归并的缺点在于需要 $O(N)$ 的空间复杂度, 归并排序的思考更多的是解决在磁盘中的外排序问题。
2. 时间复杂度: $O(N \cdot \log N)$
3. 空间复杂度: $O(N)$
4. 稳定性: 稳定

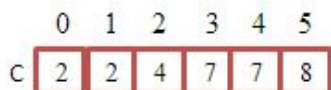
2.5 非比较排序

思想: 计数排序又称为鸽巢原理, 是对哈希直接定址法的变形应用。操作步骤:

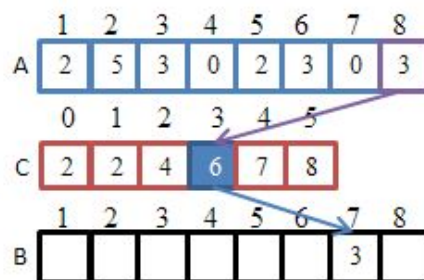
1. 统计相同元素出现次数
2. 根据统计的结果将序列回收到原来的序列中



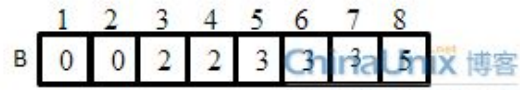
a) A 为待排序的数组。C 记录 A 中各值的元素数目。值为 0 的有 2 个元素, 值为 1 的有 0 个元素, ..., 值为 5 的有一个元素



b) 将 $C[i]$ 转换为值小于等于 i 的元素个数。



c) 为 A 数组从后向前的每个元素找到对应的 B 中的位置。每次从 A 中复制一个元素到 B 中, C 中相应的计数减一。



d) 当 A 中的元素都复制到 B 中后, B 就是排好序的结果。

计数排序的特性总结:

1. 计数排序在数据范围集中时, 效率很高, 但是适用范围及场景有限。
2. 时间复杂度: $O(\text{MAX}(N, \text{范围}))$
3. 空间复杂度: $O(\text{范围})$

4. 稳定性：稳定

3.排序算法复杂度及稳定性分析



排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n \log n) \sim O(n^2)$	$O(n^{1.3})$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n) \sim O(n)$	不稳定

4.选择题练习

1. 快速排序算法是基于 () 的一个排序算法。
A 分治法
B 贪心法
C 递归法
D 动态规划法
2. 对记录 (54, 38, 96, 23, 15, 72, 60, 45, 83) 进行从小到大的直接插入排序时, 当把第8个记录45插入到有序表时, 为找到插入位置需比较 () 次? (采用从后往前比较)
A 3
B 4
C 5
D 6
3. 以下排序方式中占用 $O(n)$ 辅助存储空间的是
A 简单排序
B 快速排序

- 16 C 堆排序
17 D 归并排序
18
19 4. 下列排序算法中稳定且时间复杂度为 $O(n^2)$ 的是 ()
20 A 快速排序
21 B 冒泡排序
22 C 直接选择排序
23 D 归并排序
24
25 5. 关于排序, 下面说法不正确的是
26 A 快排时间复杂度为 $O(N \cdot \log N)$, 空间复杂度为 $O(\log N)$
27 B 归并排序是一种稳定的排序, 堆排序和快排均不稳定
28 C 序列基本有序时, 快排退化成冒泡排序, 直接插入排序最快
29 D 归并排序空间复杂度为 $O(N)$, 堆排序空间复杂度的为 $O(\log N)$
30
31 6. 下列排序法中, 最坏情况下时间复杂度最小的是 ()
32 A 堆排序
33 B 快速排序
34 C 希尔排序
35 D 冒泡排序
36
37 7. 设一组初始记录关键字序列为(65, 56, 72, 99, 86, 25, 34, 66), 则以第一个关键字65为基准而得到的一趟快速排序结果是 ()
38 A 34, 56, 25, 65, 86, 99, 72, 66
39 B 25, 34, 56, 65, 99, 86, 72, 66
40 C 34, 56, 25, 65, 66, 99, 86, 72
41 D 34, 56, 25, 65, 99, 86, 72, 66

- 1 答案:
2 1.A
3 2.C
4 3.D
5 4.B
6 5.D
7 6.A
8 7.A