竞争优胜劣汰，勇攀人生高峰

第22讲 内排序

信息学院（智能应用研究院）
欧 新 宇

建议课时：6

第21讲 内排序

考核要点

- **考核大纲**

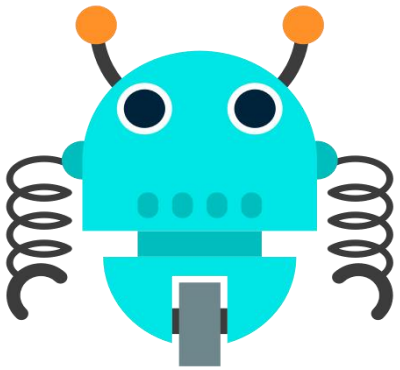
排序的基本概念、插入排序、交换排序、选择排序、二路归并排序、基数排序、外部排序及排序算法的分析和应用

- **复习要点**
  - 重点掌握各种排序算法的思想、排序过程（能动手模拟）和特征（初态影响、复杂度、稳定性和适应性等）
  - 掌握各种排序算法的关键代码，能够进行熟练编写
  - 掌握根据问题选择最优排序算法的能力

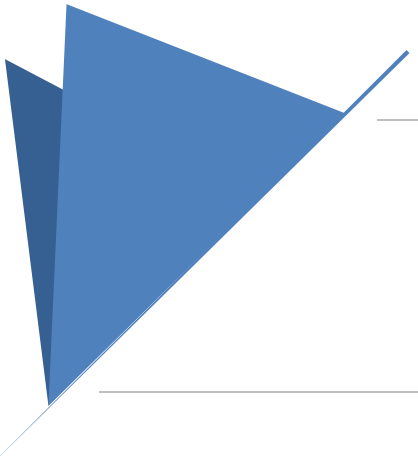
第22讲 内排序

本讲作业

- **必做题**
  - ✓ 【课堂互动22.1】排序的基本概念
  - ✓ 【课堂互动22.2】插入排序
  - ✓ 【课堂互动22.3】交换排序
  - ✓ 【课堂互动22.4】选择排序
  - ✓ 【课堂互动22.5】归并排序和基数排序
  - ✓ 【课后作业22】内排序
- **选做题**
  - ✓ 【课前自测22】内排序
  - ✓ 【扩展练习22】内排序



- 排序的基本概念
- 插入排序
- 交换排序
- 选择排序
- 归并排序和基数排序



排序的基本概念

/ 什么是排序

/ 排序算法的稳定性

/ 排序算法的衡量和比较

/ 排序算法中数据元素的存储结构

/ 排序算法的分类

排序的基本概念

什么是排序

设存在n个任意排列的数据元素序列 (D_0, D_1,..., D_{n-1})，K_i是D_i的关键字。**排序**是以提高查找速度为根本目标的一种操作，它的通过找到一组下标元素 (0, 1,..., n-1) 的一种排列 p(0), p(1),..., p(n-1)，使得n个杂乱无章的数据按照关键字的一定规律顺次序排列起来。此时，关键字是**有序的**：

- **非递减**：$K_{p(0)} \le K_{p(1)} \le \ldots \le K_{p(n-1)}$　　**递增**：$K_{p(0)} < K_{p(1)} < \ldots < K_{p(n-1)}$
- **非递增**：$K_{p(0)} \ge K_{p(1)} \ge \ldots \ge K_{p(n-1)}$　　**递减**：$K_{p(0)} > K_{p(1)} > \ldots > K_{p(n-1)}$

数据也是**有序的**：

$$D_{p(0)}, D_{p(1)}, \ldots, D_{p(n-1)}$$

排序的基本概念

排序的稳定性

若待排序序列中的两个具有相同关键字（$K_i=K_j$）的元素D_i和D_j是有序的($D_i \geq D_j$或$D_i \leq D_j$)；排序之后，顺序不变，则称所用排序算法是稳定的；反之，称该排序算法是不稳定的。

例1：存在某序列 {(张飞, 99), (夏侯惇, 95), (荀彧, 72), (甘宁, 95), (赵云, 98)}，若以各英雄武力值作为关键字进行排序，可得：

- ✓ 序列1： {(张飞, 99), (赵云, 98), (夏侯惇, 95), (甘宁, 95), (荀彧, 72)}
- ✓ 序列2： {(张飞, 99), (赵云, 98), (甘宁, 95), (夏侯惇, 95), (荀彧, 72)}

序列1的排序算法是稳定的；序列2的排序算法是不稳定的。

排序的基本概念

排序算法的衡量和比较

时空复杂度

时间复杂度(最好、最坏和平均)、比较次数和移动元素次数；空间复杂度

稳定性

具有相同关键字的元素，排序前后的位置是否保持一致

适用场合

排序算法何种初始状态和环境中使用较为合适

排序趟数

算法在手工排序过程中，每一趟的排序结果，要求趟数不能多也不能少

元素确定

每趟排序结束，都能确定部分元素的最终位置；最后一趟结束前，所有元素都无法确定最终位置

排序方法

给定排序过程/结果，判断是何种排序方法

排序的基本概念

排序算法中数据元素的存储结构

```
#define MAXSIZE 20;                 // 定义记录的总长度上限
typedef int KeyType;                // 定义关键字的数据类型为整型int
```

```
typedef struct Entry {              // 定义记录（数据元素）的结构
    KeyType key;                    // 定义关键字
    DataType data;                  // 定义数据元素的其他数据项
} Entry;
```

```
typedef struct SqList{              // 定义一个顺序表来存储排序序列
    Entry D[MaxSize+1];             // 使用静态数组来存储数据元素向量，r[0]通常为哨兵
    int length;                     // 定义顺序表的长度，即待排序数据元素的数量
} SqList;
```

排序的基本概念

内部排序和外部排序

如果待排序元素总数相对于内存而言较小，

整个排序过程可以在内存中进行，则称之为**内部排序**，简称**内排序**；

如果待排序元素总数较多，不能全部放入内存，

排序过程中需访问外存，则称之为**外部排序**，简称**外排序**。

注：外部排序时，要将数据分批调入内存来排序，中间结果还要及时放入外存，因此外部排序要复杂得多。

排序的基本概念

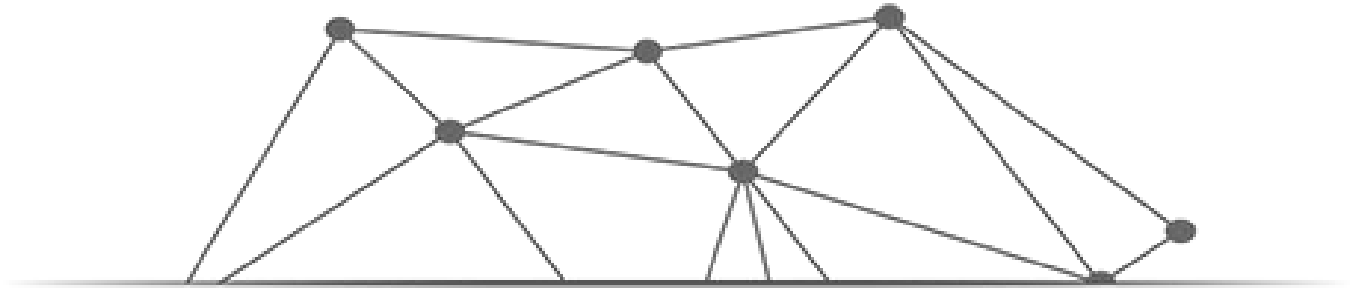
常见排序算法的分类

内排序

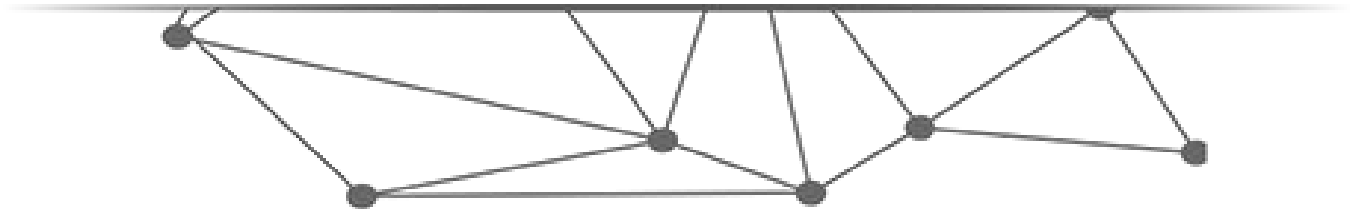
- 插入排序
  - 直接插入排序
  - 折半插入排序
  - 希尔排序
- 交换排序
  - 冒泡排序
  - 快速排序
- 选择排序
  - 简单选择排序
  - 树形选择排序
  - 堆排序
- 归并/基数排序
  - 归并排序
  - 桶排序
  - 多关键字排序
  - 链式基数排序

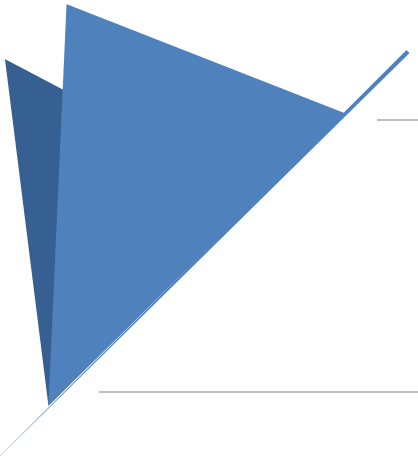
外排序

- 多路平衡归并
- 败者树
- 置换-选择排序
- 最佳归并树



课堂互动 18.1



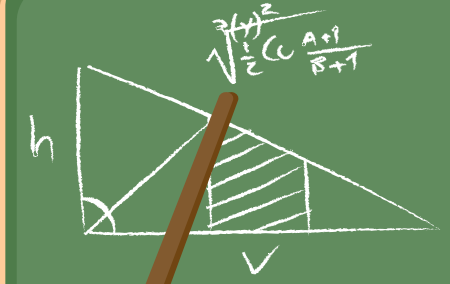


插入排序

/ 直接插入排序
/ 折半插入排序
/ 希尔排序

插入排序

插入排序算法的基本思想



1. 从只包含一个数据元素的有序序列开始；
2. 不断将待排序元素按关键字大小，**插入**到前面已经排序好的有序序列的适当位置；
3. 直到所有元素都插入到有序序列为止。

元素确定：边插入边排序，子序列始终都是有序的，但不一定是最终位置

插入排序

常见插入排序算法

- **直接插入排序** | 基于顺序查找
- **折半插入排序** | 基于折半查找
- **希尔排序** | 基于逐趟缩小增量

插入排序

直接插入排序

算法过程

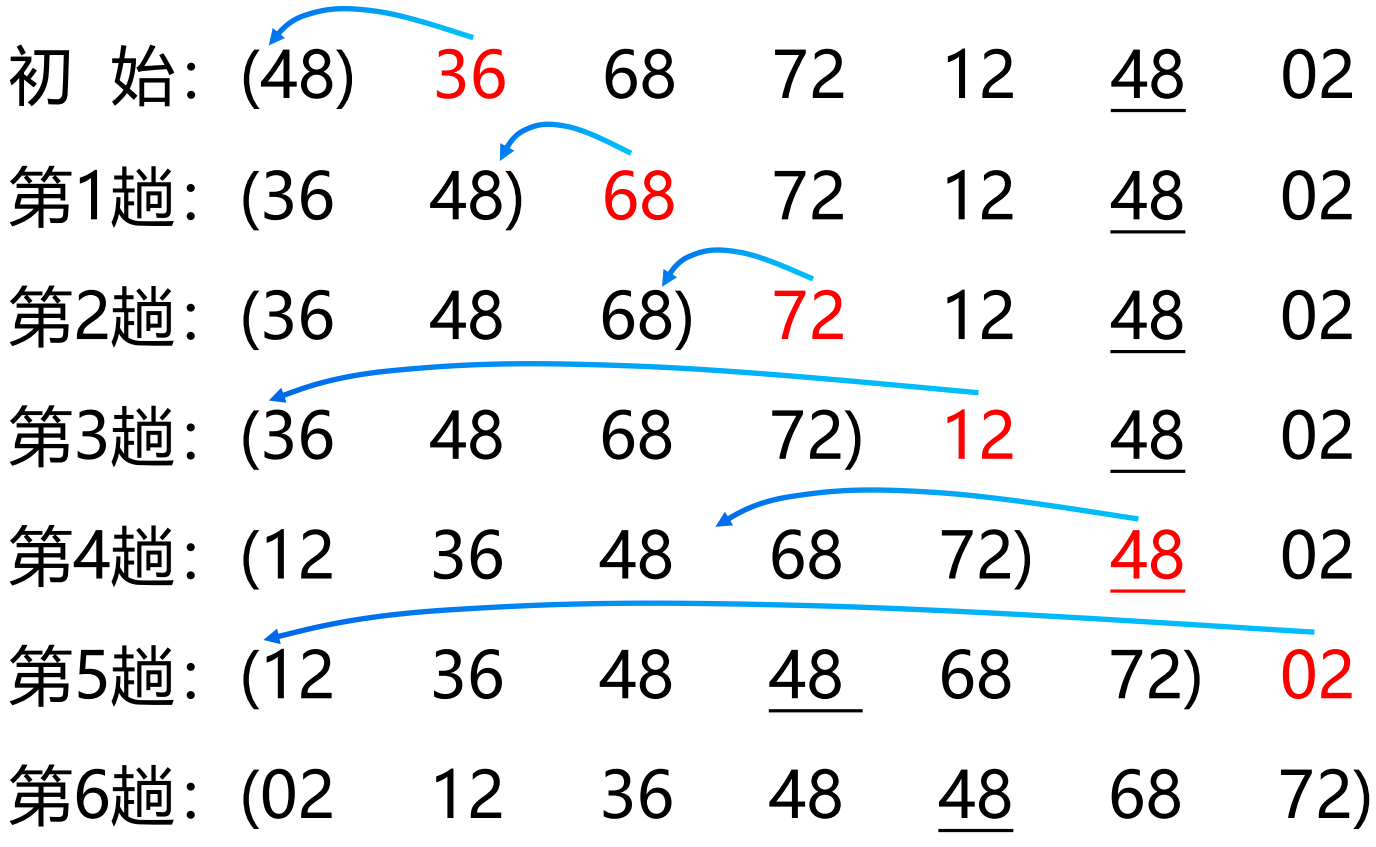
例1：给定序列 A={48, 36, 68, 72, 12, 48, 02}，试对A进行非递减排序。

- **排序过程：**

✓ **第1趟：**将序列中第1个元素作为一个有序序列，并将第2个元素按序插入到有序序列中。此刻前2个元素组成有序区，后n-2个元素组成无序区。

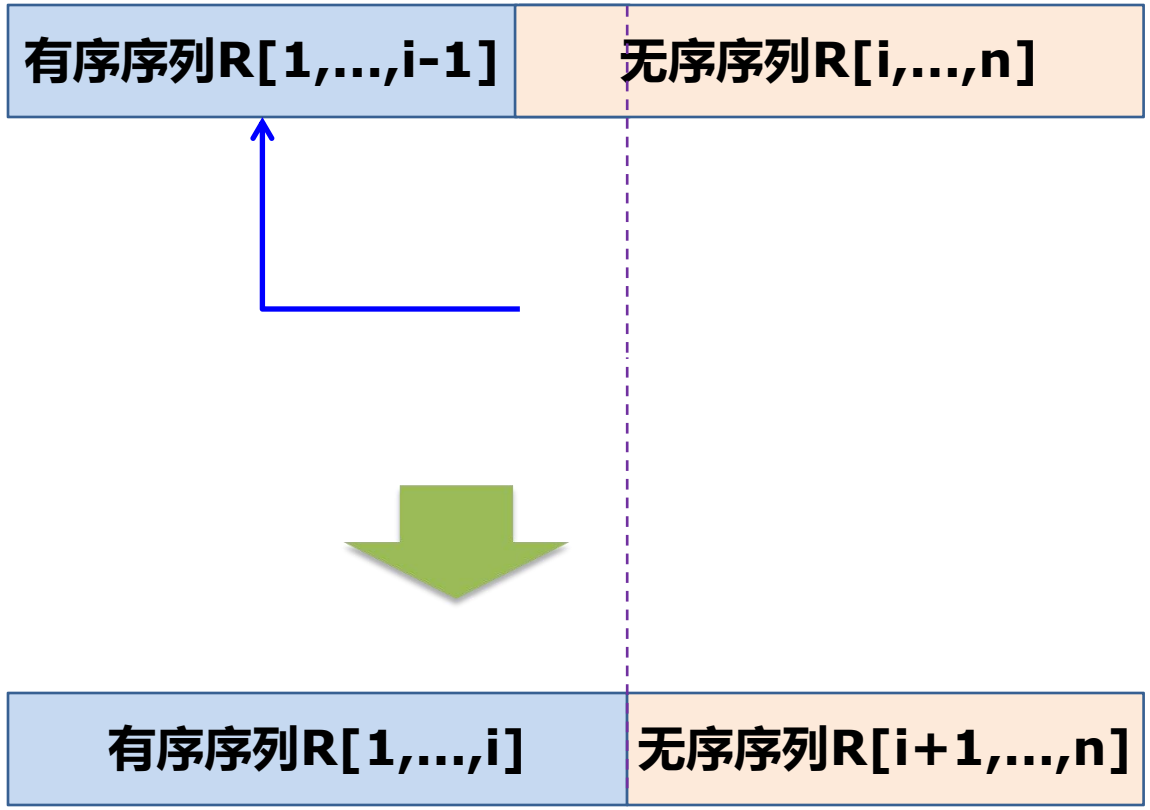
✓ **第i趟：**将序列中第1~i个元素作为有序序列，并将第i+1个元素按序插入到有序序列中。此刻前i+1个元素组成有序区，后n-i-1个元素组成无序区。

✓ **整个排序过程共执行n-1趟。**



插入排序

直接插入排序



算法的程序实现思想

01 OPTION

在R[1..i-1]中查找R[i]的插入位置，R[1..j].key≤ R[i].key<R[j+1..i-1].key；

02 OPTION

将R[j+1..i-1]中的所有记录均后移一个位置；

03 OPTION

将R[i] 插入到R[j+1]的位置上。

插入排序

直接插入排序

算法的程序实现思想（哨兵）

将R[i]作为哨兵复制到R[0]

第3趟：(36 48 68 72) 12 48 02

第1轮：12 (36 48 68 72)→72) 48 02

第2轮：12 (36 48 68→68 72) 48 02

第3轮：12 (36 48→48 68 72) 48 02

第4轮：12 (36→36 48 68 72) 48 02

第5轮：12 (12 36 48 68 72) 48 02

当**R[0]=R[j+1]**时，将R[0]插入到序列的开头

将R[i]作为哨兵复制到R[0]

第4趟：(12 36 48 68 72) 48 02

第1轮：48 (12 36 48 68 72)→72) 02

第2轮：48 (12 36 48 68→68 72) 02

第3轮：48 (12 36 48 48 68 72) 02

当**R[j]=R[j+1]**时，将R[0]插入R[j]之后

插入排序

直接插入排序

直接插入排序

```
void InsertSort (SqList &L){
  int i,j;
  for(i=2; i<=L.length; ++i)
    // 将L.r[i]插入有序子表
    if(L.r[i].key<L.r[i-1].key){
      L.r[0] = L.r[i];    // 复制为哨兵
      L.r[i] = L.r[i-1];  // r[i-1]后移1位
      for(j=i-2; L.r[0].key<L.r[j].key;--j)
          L.r[j+1] = L.r[j];   // 元素后移
      L.r[j+1] = L.r[0];       // 插入到正确位置
    }
}
```

算法的程序实现（哨兵）

01 OPTION

从R[i-1]向前进行顺序查找，哨兵设置在R[0]。

02 OPTION

当元素的关键字**大于**R[0].key时，向后移动1位；

当元素的关键字**等于**R[0].key时，跳出循环。

03 OPTION

循环结束时，R[0]将插入到位置j+1，此时：

$$R[1..j].key \le R[i].key < R[j+1..i-1].key$$

插入排序

直接插入排序

算法分析

设待排序序列元素个数为n → 直接插入排序算法总需执行n-1趟 → 比较次数和移动次数与初始排列有关

最好情况

每趟只需比较1次，不移动元素 → 总比较次数 = n-1次 ≈O(n)，不移动元素 → 顺序序列

最坏情况

第i趟比较i次，移动i+1次 → 总比较次数 = $\sum_{i=1}^{n} i = \frac{n(n-1)}{2} \approx O(n^2)$；总移动次数 = $\sum_{i=2}^{n}(i+1) = \frac{(n+4)(n-1)}{2} \approx O(n^2)$ → 逆序序列

平均情况

各种排列概率相同 → 平均比较次数和平均移动次数：$n^2/4$；时间复杂度：$O(n^2)$；空间复杂度：$O(1)$

插入排序

直接插入排序

算法分析

排序趟数	n-1趟
稳 定 性	稳定
时空复杂度	时间复杂度：最好O(n)，最坏/平均O(n²)；空间复杂度：O(1)
适 用 场 合	待排序序列基本有序递增
元 素 确 定	一趟排序后，无法确定某个元素的最终位置

插入排序

折半插入排序

为什么要使用折半查找?

当待排序元素很多时
查找插入位置将浪费
大量的时间

→ 在查找阶段使用
折半查找法替代顺序查找

→ 大幅减少关键字
的比较次数

第 i 趟的比较次数：　i → $\lfloor \log_2 i \rfloor + 1$

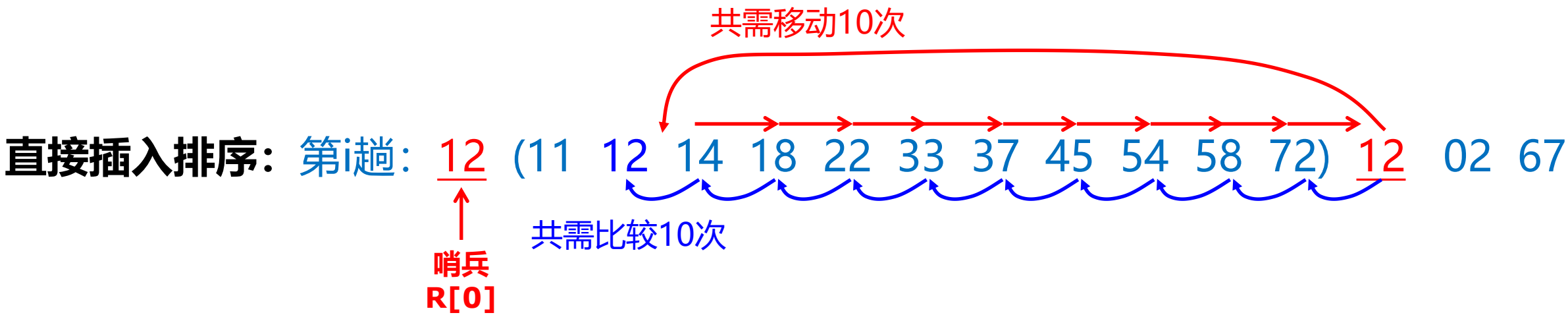
总比较次数：　n^2 → $n\lfloor \log_2 n \rfloor + 1$

插入排序

折半插入排序

算法过程

例2：假设某序列第i趟排序的初始状态如下，试给出两种插入排序的对比。

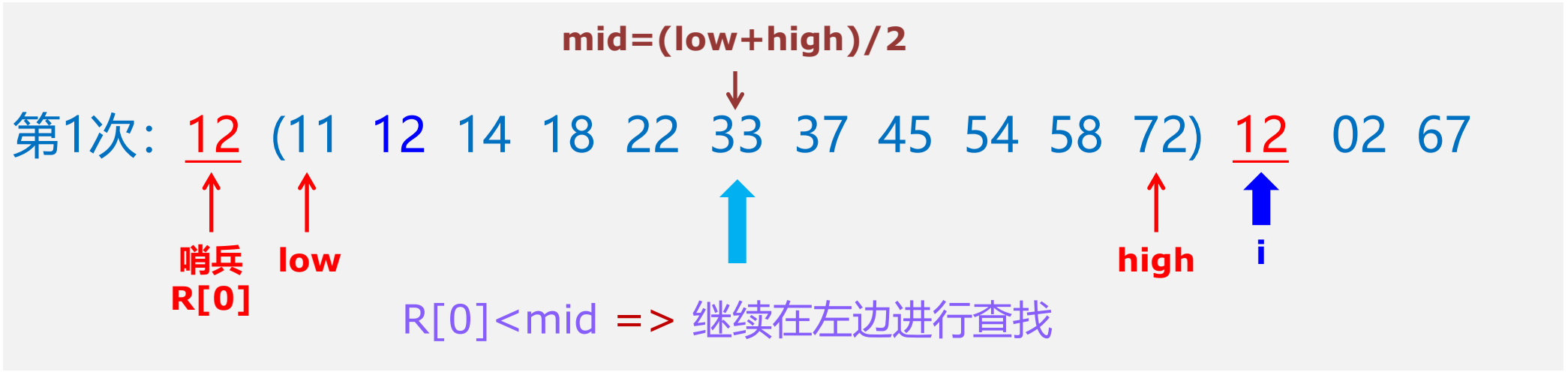


插入排序

折半插入排序

算法过程

折半插入排序：
第 i 趟

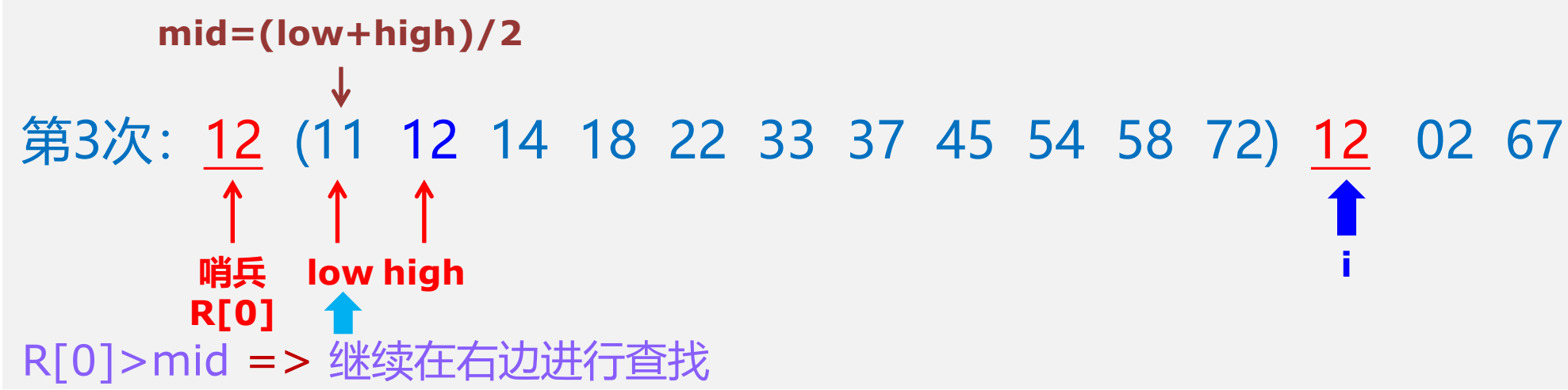


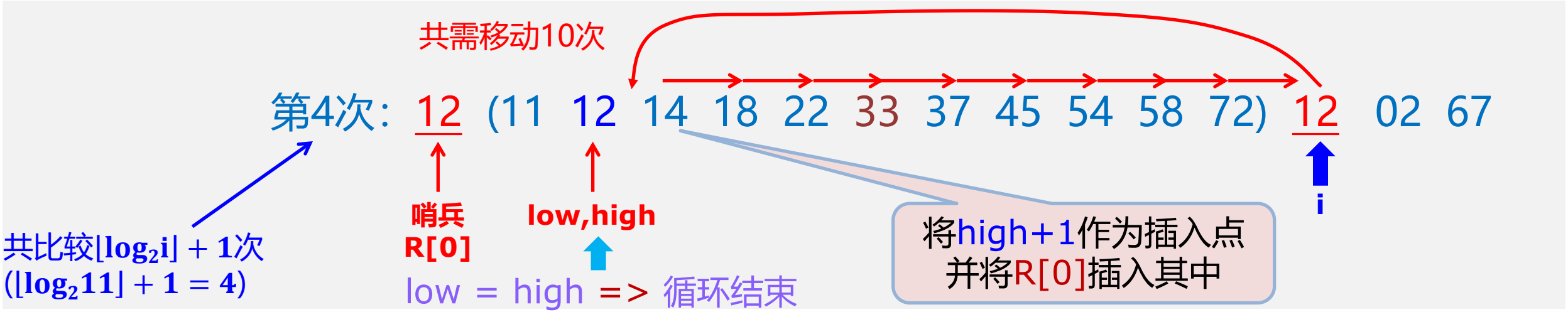
插入排序

折半插入排序

算法过程

折半插入排序：
第 i 趟





插入排序

直接插入排序

折半插入排序

```
void BinInsertSort (SqList &L){
    for(i=2; i<=L.length; ++i) {
        L.r[0] = L.r[i];
        low = 1; high = i-1;
        while(low <= high) {
            mid = (low + high) / 2 ;
            if(L.r[0].key < L.r[mid].key)
                high = mid -1;
            else
                low = mid  + 1;
        }
        for(j=i-1; j>=high+1; --j)
            L.r[j+1] = L.r[j];
        L.r[high+1] = L.r[0];
    }
}
```

算法的程序实现（哨兵）

算法步骤：

1. 依次插入第2~n个元素
2.1 将R[0]作为第i个元素的哨兵
2.2 初始化 low 和 high 指针
3. 当low小于等于high时执行循环
3.1 计算中间元素
3.2 若哨兵元素值小于中间元素值
　　下一轮查找mid的左序列
　否则，
　　下一轮查找mid的右序列

3.3 当查找过程结束后，依次将high+1之后的元素向后移动1位
3.4 在high+1的位置插入哨兵r[0]

插入排序

折半插入排序

算法分析

排序趟数	n-1趟
稳 定 性	稳定
时空复杂度	时间：最好$O(n\log_2 n)$，最坏/平均$O(n^2)$；空间：$O(1)$
适 用 场 合	折半插入排序与待排序序列初始排列无关，仅依赖于对象个数
元 素 确 定	一趟排序后，无法确定某个元素的最终位置

平均性能更好：当n比较大时，总比较次数比直接插入排序最坏情况好得多，但不如其最好情况

移动次数不变

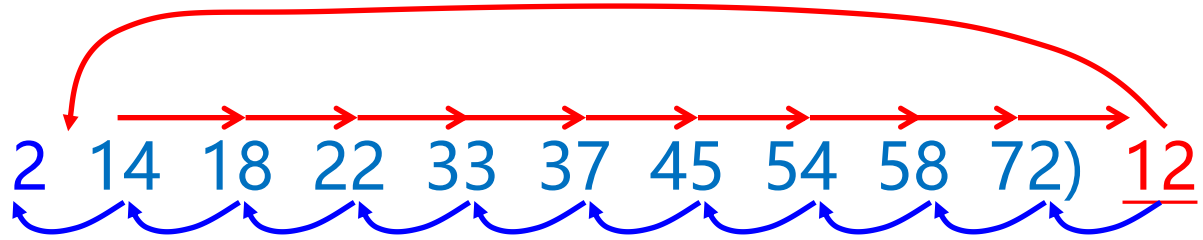
若初始队列有序或接近有序，直接插入排序比较次数更少

插入排序

希尔排序

为什么要使用希尔排序

直接插入排序：第i趟：12 (11 12 14 18 22 33 37 45 54 58 72) 12 02 67



Q1：是否可以增大移动的步数?

比较一次，移动一步 ➡ 比较一次，移动一大步

Q2：直接插入排序在什么情况下效率比较高?

待排序序列基本有序　　待排序元素个数较少

希尔排序算法的出发点

插入排序

希尔排序

算法的基本思想

- ***基本思想***

将整个待排序序列分割成若干个子序列，并分别依次进行**直接插入排序**，待整个序列中的元素“基本有序”时，再对整个序列进行一次直接插入排序。

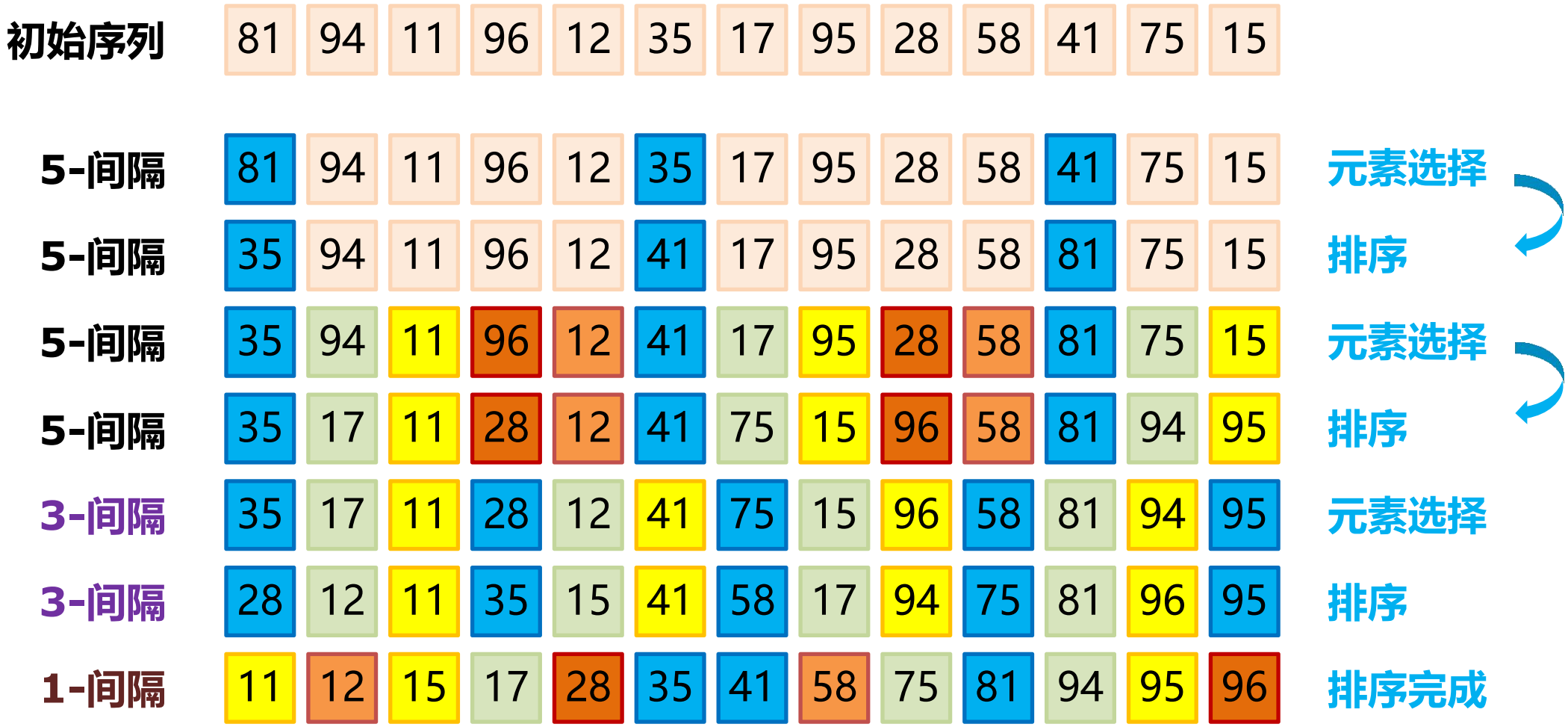
1. 定义一个缩小增量序列：$D_M > D_{M-1} > \ldots > D_1=1$，例如：5，3，1
2. 以每个D_k对待排序序列进行“D_k-间隔”排序（k=M, M-1, ..., 1）

注意：“D_k-间隔”排序相当于对待排序序列进行子序列直接插入排序；当D_k=1时，待排序序列中的元素已经“基本有序”，此时对全体元素进行一次直接插入排序。

插入排序

希尔排序

算法过程



插入排序

希尔排序

算法的程序实现

希尔排序的主程序

```
void ShellSort(SqList &L, int dlta[], int t){
    // 按增量序列dlta[0...t-1]对顺序表L作Shell排序
    for(k=0; k<t; ++k)
    // 增量为dlta[k]的一趟插入排序
        ShellInsert(L, dlta[k]);
}// ShellSort
```

希尔排序第i趟的排序方法

```
void ShellInsert(SqList &L, int dk) {
    // 对顺序表L进行一趟增量为dk的Shell排序，dk为步长因子
    for(i=dk+1; i<=L.length; ++ i)
        if(r[i].key < r[i-dk].key) { // 将r[i]插入有序增量子表
            r[0]=r[i]; // 暂存在r[0]，作为哨兵
            for(j=i-dk; j>0&&(r[0].key<r[j].key); j=j-dk)
                // 关键字较大的记录在子表中后移
                r[j+dk]=r[j];
            r[j+dk]=r[0]; // 将r[i]插入到正确位置
    }
}
```

插入排序

希尔排序

希尔排序的特点

- ✓ 使用缩小增量D_k排序，且最后一个必须是1
- ✓ 一次移动，移动位置较大，跳跃式地接近排序后的最终位置
- ✓ 执行多遍插入排序
- ✓ 增量序列应该是互质的

插入排序

希尔排序

增量不互质的Bad Case

初始序列	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
8-间隔	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
4-间隔	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
2-间隔	1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
1-间隔	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

增量元素不互质，小增量可能无法对序列起到优化作用。

插入排序

希尔排序

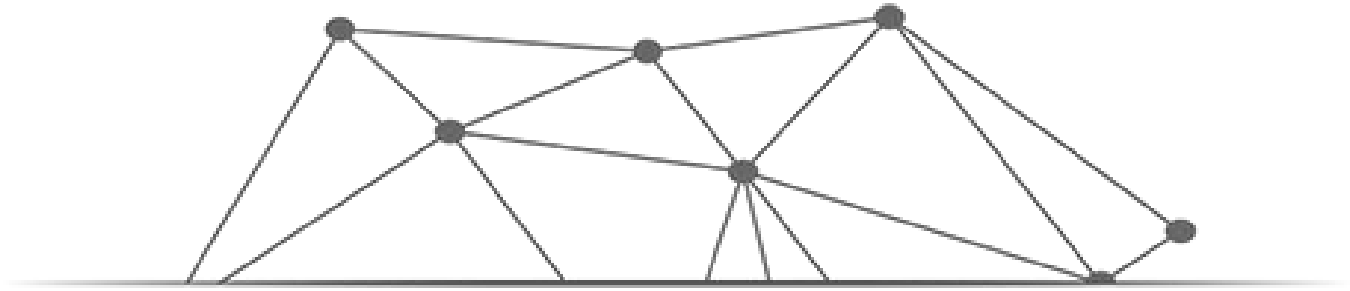
算法分析

排序趟数	M趟，M为间隔D_k的个数，其中最终一个增量之必须为1
稳 定 性	不稳定
时空复杂度	时间复杂度是n和d的函数：$O(n^{1.25})\sim O(1.6n^{1.25})$ **经验公式**； 空间复杂度：$O(1)$
适 用 场 合	与待排序序列初始排列无关，仅依赖于对象个数和间隔序列d，但目前尚无选择最佳d序列的方法
元 素 确 定	一趟排序后，无法确定某个元素的最终位置

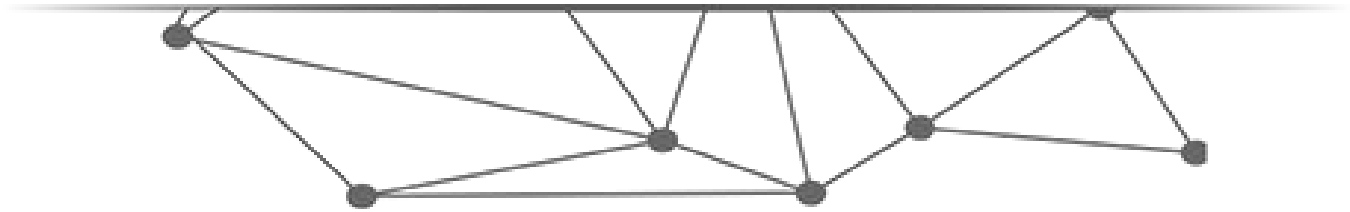
插入排序

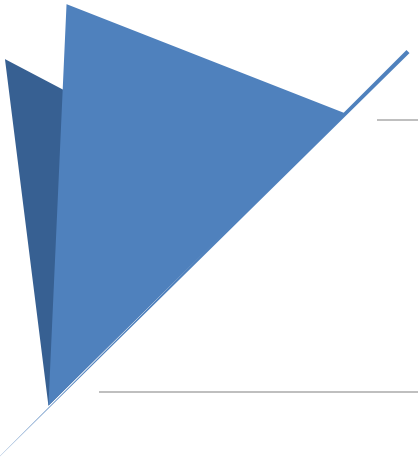
各种排序方法的比较

排序方法	时间复杂度			空间复杂度	稳定性
	最好情况	最坏情况	平均情况	辅助存储	
直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
折半插入排序	$O(n\log_2 n)$	$O(n^2)$	$O(n^2)$	O(1)	稳定
希尔排序	O(n)	$O(n^2)$	$O(n^{1.25})$	O(1)	不稳定



课堂互动 18.2



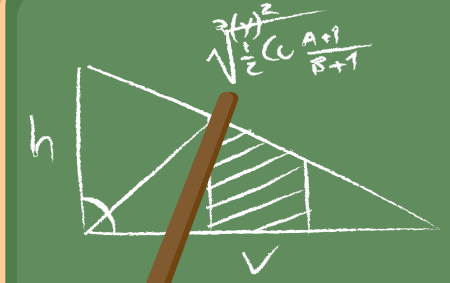


交换排序

/ 冒泡排序
/ 快速排序

交换排序

交换排序算法的基本思想



两两比较，如果发生**逆序**则**交换**，直到所有记录都排好序为止。

常见交换排序方法：冒泡排序、快速排序

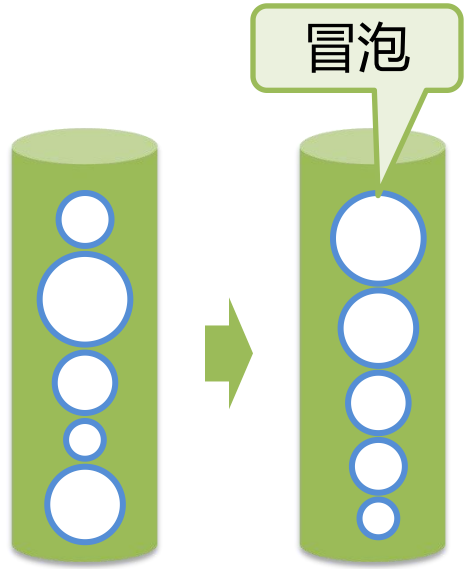
交换排序

冒泡排序

仅**逆序**才交换，**相等**不交换，因此冒泡算法是稳定的

基本思想：

1. 从前向后不断对相邻元素进行两两对比，并将**逆序**数据按“前小后大”的规则进行交换。
2. 不断执行**步骤1**，直到所有元素都不再逆序排列为止。



初 始： **48 36 68 72 12 48 02**

第1趟：

位置0,1进行比较 -> 判断 -> (逆序)交换 -> 结果：36 48 68 72 12 48 02

位置1,2进行比较 -> 判断 -> 不交换 -> 结果：36 48 68 72 12 48 02

位置2,3进行比较 -> 判断 -> 不交换 -> 结果：36 48 68 72 12 48 02

位置3,4进行比较 -> 判断 -> (逆序)交换 -> 结果：36 48 68 12 72 48 02

位置4,5进行比较 -> 判断 -> (逆序)交换 -> 结果：36 48 68 12 48 72 02

位置5,6进行比较 -> 判断 -> (逆序)交换 -> 结果：36 48 68 12 48 02 **72**

第一趟结束，最后一个元素确定

交换排序

冒泡排序

初 始: 48 36 68 72 12 48 02

第1趟结束: 36 48 68 12 48 02 72

第2趟:

位置0,1进行比较 -> 判断 -> 不交换 -> 结果: 36 48 68 12 48 02 72

位置1,2进行比较 -> 判断 -> 不交换 -> 结果: 36 48 68 12 48 02 72

位置2,3进行比较 -> 判断 -> (逆序)交换 -> 结果: 36 48 12 68 48 02 72

位置3,4进行比较 -> 判断 -> (逆序)交换 -> 结果: 36 48 12 48 68 02 72

位置4,5进行比较 -> 判断 -> (逆序)交换 -> 结果: 36 48 12 48 02 68 72

第2趟结束: 36 48 12 48 02 68 72

第3趟结束: 36 12 48 02 48 68 72

第4趟结束: 12 36 02 48 48 68 72

第5趟结束: 12 02 36 48 48 68 72

第6趟结束: 02 12 36 48 48 68 72

每结束一趟增加一个有序元素

1. n个元素总共执行 **n-1** 趟
2. 第i趟需要比较 **n-i** 次

交换排序

冒泡排序

冒泡排序

```
void BubbleSort(SqList &L){
    int i, j;
    for (i=0; i<L.length-1; j++){
        for(j=0; j<L.length-1-i; j++){
            if(L.r[j] > L.r[j+1]){
                int tmp = L.r[j];
                L.r[j] = L.r[j+1];
                L.r[j+1] = tmp;
            }
        }
    }
}
```

冒泡算法的程序实现

算法步骤：

1. 初始化指针，i为趟数，j为每趟处理的元素
2. 共循环执行 n-1 趟
3. 每一趟对比检查的样本数等于(n-1) - i
4. 不断对比相邻元素，并按照前小后大的原则进行交换
   交换过程中，使用一个中间变量tmp来完成O(1)

冒泡算法的优点：

每趟结束后，都能挤出一个最大值到最后，并且还能适当理顺前面的其他元素。

交换排序

冒泡排序

初　　始：49 38 65 97 76 13 27 30　n=8
第1趟结束：38 49 65 76 13 27 30 97
第2趟结束：38 49 65 13 27 30 76 97
第3趟结束：38 49 13 27 30 65 76 97
第4趟结束：38 13 27 30 49 65 76 97
第5趟结束：13 27 30 38 49 65 76 97
第6趟结束：13 27 30 38 49 65 76 97
第7趟结束：13 27 30 38 49 65 76 97

最后两趟为无效对比

设置一个停止标志：当不发生交换时，排序结束

冒泡算法的改进

冒泡排序

```
void BubbleSort(SqList &L){
    int i, j; bool flag; // 标识是否交换数据
    for (i=0; i<L.length-1; j++){
        flag = false;
        for(j=0; j<L.length-1-i; j++){
            if(L.r[j] > L.r[j+1]){ // 发生逆序并交换
                int tmp = L.r[j];
                L.r[j] = L.r[j+1];
                L.r[j+1] = tmp;
                flag = true;   // 发生交换，flag置为ture
            }
        }
        if(!flag) break;
    }
}
```

当某一趟不发生元素交换，则退出排序算法

交换排序

冒泡排序

算法分析

- **最好情况：**已有序，只需要一趟排序，比较n-1次，移动次数0，时间复杂度**O(n)**。
- **最坏情况：**进行n-1趟，第i趟比较 n-i 次，总比较次数N为：

$$N = (n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} n - i = \frac{n(n-1)}{2}$$

移动次数为3N次（每次交换都需借助tmp变量进行3次操作），时间复杂度为**O(n²)**。

- **平均情况：**假设算法每趟排序停止概率相同，为$\frac{1}{n-1}$，则比较次数为：

$$N = \frac{1}{n-1}\sum_{j=1}^{n-1}\sum_{i=1}^{j}(n-i) = \frac{n(n-1)}{2}$$

移动次数为3N次，时间复杂度为**O(n²)**。

交换排序

冒泡排序

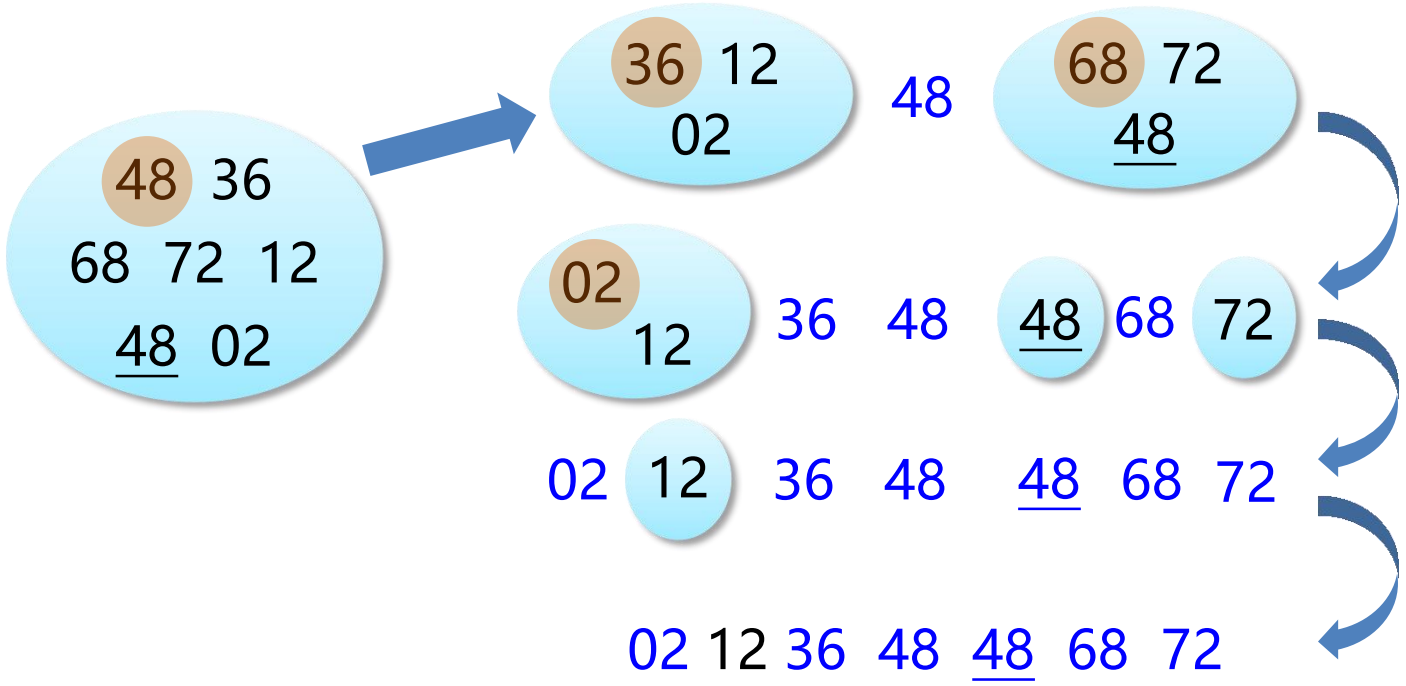
算法分析

排序趟数	1 ~ n-1趟
稳 定 性	稳定
时空复杂度	时间：最坏和平均$O(n^2)$，**最好$O(n)$**；空间复杂度：**$O(1)$**
适 用 场 合	基本有序
元 素 确 定	每趟排序后，都至少能确定1个最大/最小元素的位置

交换排序

快速排序

快速排序算法之所以命名为“快速”，是因为到目前为止，它依然是平均情况下最快的内排序算法



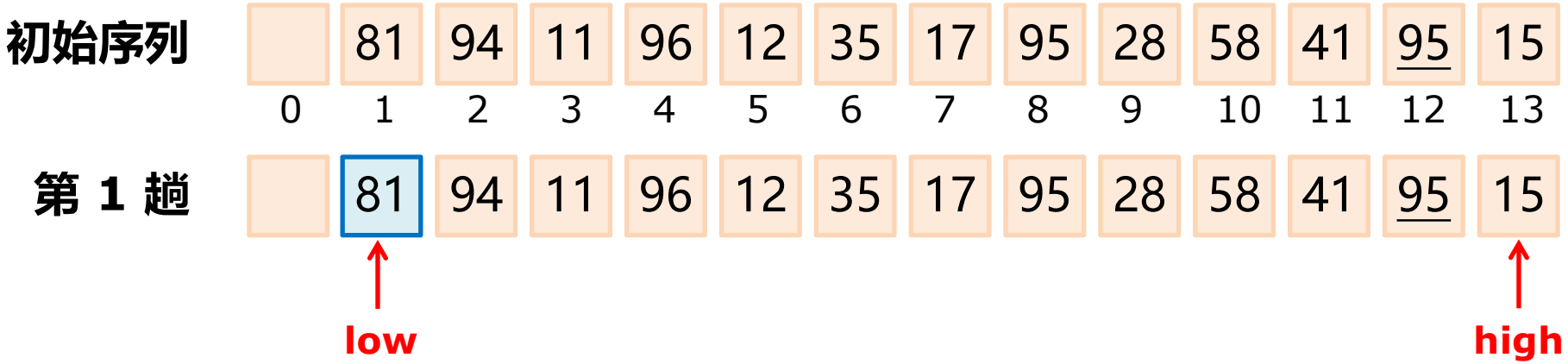
基本思想（分而治之）

1. 选择一个元素D_s作为中心分割元素（关键字K_s）；
2. 使用该中心分割元素D_s将待排序序列分成左右两个子序列，将所有关键字比K_s小的元素放到D_s的左边；所有比K_s大的元素放到D_s的右边。
3. 分别对左右子序列执行快速排序（**步骤1~2**），直至每个子序列只剩一个元素。

交换排序

快速排序

具体实现



1. 设置两个指针low和high，分别指向序列的第一个位置和最后一个位置。
2. 选择中心分割元素（通常是第一个元素、最后一个元素、中间元素或任意元素），并将其放入哨兵位置0。
3. 从最后开始选择一个比中心元素小的元素放入前面空缺的位置，high指针向前移动一位。
4. 从最前面开始选择一个比中心元素大的元素放入后面空缺的位置，low指针向后移动一位。
5. 当low和high重合，且指向空位置时，将中心分割元素（哨兵）放入当前重合的空缺位置。

交换排序

快速排序

具体实现

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
初始序列		81	94	11	96	12	35	17	95	28	58	41	95	15
第 1 趟		81	94	11	96	12	35	17	95	28	58	41	95	15
第 2 趟		15	41	11	58	12	35	17	28	81	95	96	95	94
第 3 趟		12	11	15	58	41	35	17	28	81	95	96	95	94
第 4 趟		11	12	15	58	41	35	17	28	81	95	96	95	94
第 5 趟		11	12	15	28	41	35	17	58	81	95	95	96	94

对各个子序列表操作相似，可采用递归实现

每趟子序列的形成从两头向中间交替式逼近法

交换排序

快速排序

代码实现

快速排序的主函数

```
void main(){
    QSort(L, 1, L.lenght);
}
```

快速排序的主函数

```
void QSort ( SqList &L,  int low,  int  high){
  if(low < high) {
    pivot = Partition(L, low, high ) ;  // 定义转轴
    // 将 L[Low,...,high] 一分为二
    Qsort(L, low, pivot-1);    // 快速排序左子表
    Qsort(L, pivot+1, high); // 快速排序右子表
  }
}
```

快速排序的每趟交换函数

```
int Partition(SqList &L,  int low,  int  high){
    L.r[0] = L.r[low];    // 选择中心元素作为哨兵
    pivotkey = L.r[low].key;
    while(low < high){
        while(low<high && L.r[high].key>=pivot)
            --high;
        L.r[low] = L.r[high];
        while(low<high && L.r[low].key<=pivot)
            ++low;
        L.r[high] = L.r[low];
    }
    L.r[low]=L.r[0];      // 将哨兵中心元素填入中心空缺
    return low;
}
```

交换排序

快速排序

算法分析

- **最好情况**：经过每趟排序后，若分割成的两个子序列长度相近，效率最高，时间复杂度为$O(nlog_2n)$。
- **最坏情况**：若每次分割的两个子序列有一个为空，即每趟仅生成一个子序列且长度比当前序列少1，则效率最低，时间复杂度为$O(n^2)$。
- **平均情况**：时间复杂度为$O(nlog_2n)$。
- 当序列呈现**顺序**或**逆序**时，效率最低；当序列**均匀乱序**时，效率最高

交换排序

快速排序

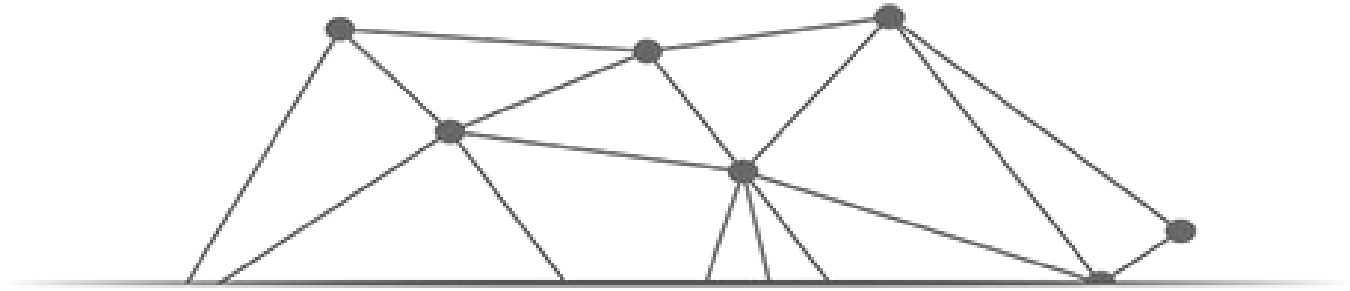
算法分析

排序趟数	最多n-1趟
稳 定 性	不稳定
时空复杂度	时间复杂度：最坏$O(n^2)$，最好和平均$O(n\log_2 n)$； 空间复杂度：非原地排序（递归-堆栈），取决于递归的深度 平均：$O(\log_2 n)$，最坏：$O(n)$
适 用 场 合	均匀乱序（非常无序）状态
元 素 确 定	能

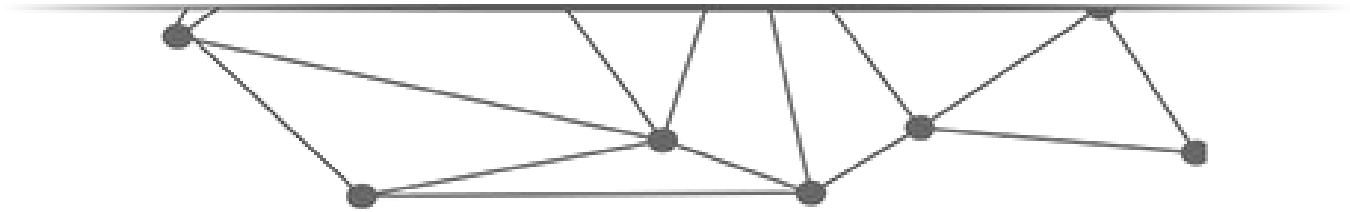
交换排序

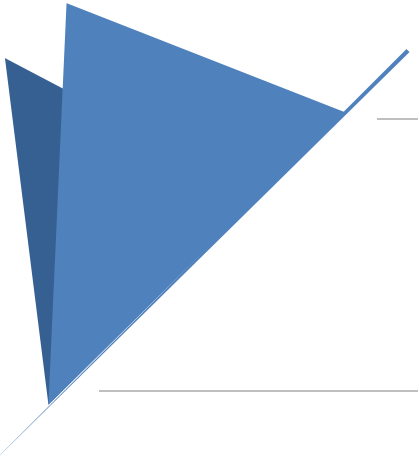
各种排序方法的比较

排序方法	时间复杂度			空间复杂度	稳定性
	最好情况	最坏情况	平均情况	辅助存储	
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
折半插入排序	$O(n\log_2 n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n)$	$O(n^2)$	$O(n^{1.3})$	$O(1)$	不稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
快速排序	$O(n^2)$	$O(n)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定



课堂互动18.3



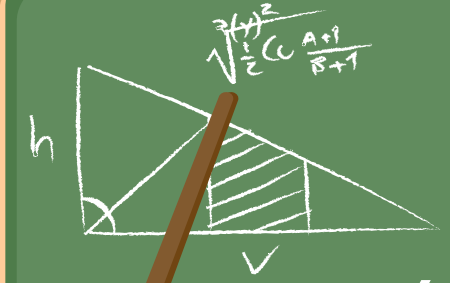


选择排序

/ 简单选择排序
/ 树形选择排序
/ 堆排序

选择排序

选择排序算法的基本思想



每趟都从待排序序列中选择最小/最大的元素放到有序序列的中，直到待排序序列为空。

常见交换排序方法：简单选择排序、树形选择排序、堆排序

选择排序

简单选择排序

基本思想：从待排序数据中选出最大/最小的元素放到其最终位置上。

基本操作：

1. 在**第1趟**排序中，在待排序序列D[0],D[1],...,D[n-1]中，通过**n-1**次关键字比较，找出关键字最小的元素，并将其与**第1个**元素D[0]进行交换；

第1趟结束后，D[0]存储的是全局关键字最小的元素，不再进入下一趟排序范围

2. 在**第2趟**排序中，在待排序序列D[1],D[2],...,D[n-1]中，通过**n-2**次关键字比较，找出关键字最小的元素，并将其与**第2个**元素D[1]进行交换；

第2趟结束后，D[1]存储的是全局关键字第二小的元素，不再进入下一趟排序范围

...

i. 在**第i趟**排序中，在待排序序列D[i-1],D[i],...,D[n-1]中，通过**n-i**次关键字比较，找出关键字最小的元素，并将其与**第i个**元素D[i-1]进行交换；

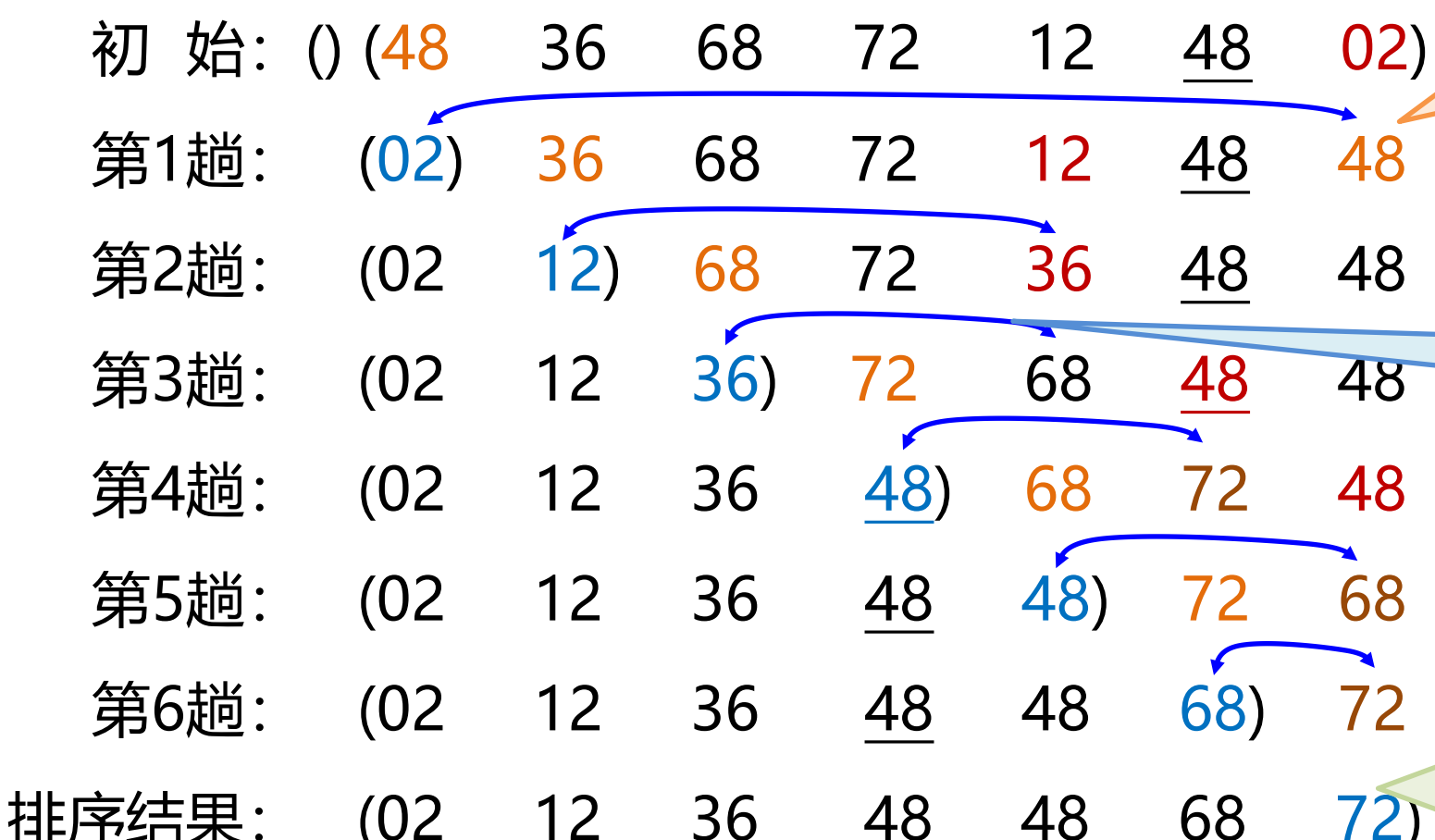
第i趟结束后，D[i-1]存储的是全局关键字第i小的元素，不再进入下一趟排序范围

n. 直到所有元素都被加入到前面的有序序列中，最后加入的是全局关键字最大的元素，排序结束。

选择排序

简单选择排序

算法过程



由于D[i]需要和当前最小元素交换位置，因此可能会产生**不稳定**。

每一趟都至少可以确定一个元素的最终位置

总共执行n-1趟排序，每趟比较n-i-1次；最好移动0次，最坏移动3(n-1)次。总比较次数：

$$\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2} \Rightarrow \boldsymbol{O(n^2)}$$

选择排序

简单选择排序

代码实现

简单选择排序

```
void SelectSort(SqList &L){
    for (i=1; i<L.length; ++i){
        k=i;
        for(j=i+1; j<=L.length; j++)
            if(L.r[j].key<L.r[k].key)
                k=j;
        if(k!=i){
            tmp = L.r[i];
            L.r[i] = L.r[k];
            L.r[k] =tmp;
    }
}
```

算法步骤：

// 从第一个元素开始，逐趟查找关键字最小的元素

1.1 定义一个指针向量k，指向当前序列中关键字最小的元素

1.2 从i开始，向后依次比较每一个元素的关键字，若其值小于指针向量k所指向的元素，则将k指向该最小元素

1.3 比较完成后，若k不等于起始元素，则将关键字最小的元素与第i个元素进行交换

选择排序

简单选择排序

排序趟数	n-1趟
稳 定 性	不稳定
时空复杂度	时间复杂度都是$O(n^2)$；空间复杂度：$O(1)$
适 用 场 合	没有特别出彩的地方
元 素 确 定	每趟排序后，都至少能确定1个最大/最小元素的位置

选择排序

树形选择排序

树形选择排序（Tree Selection Sort），又称**锦标赛排序**（Tournament Sort），是一种按照锦标赛思想进行选择排序的方法。

基本思想：

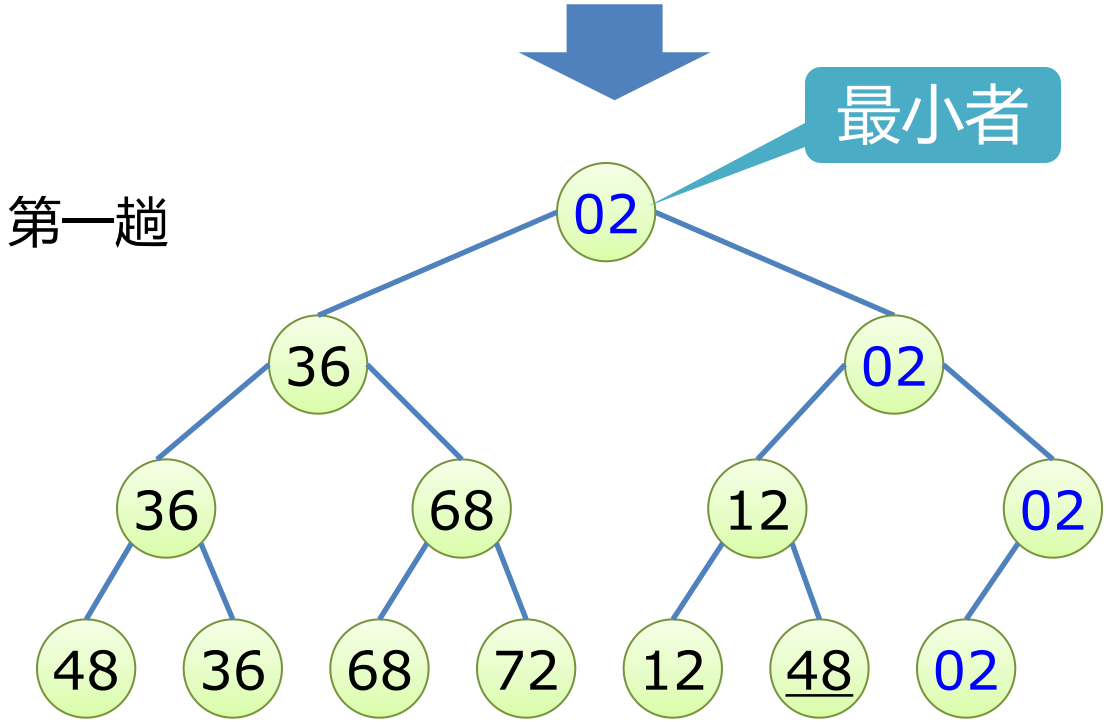
1. 将所有结点当作是叶节点，构建一棵完全二叉树，对叶节点的关键字进行两两比较，将较小者再进行两两比较，直至选出最小关键字（根结点）的元素为止；
2. 将最小关键字的结点置为“最大值”，然后从该叶子结点开始，与其左右孩子的关键字进行对比，并修改从叶子节点到根结点的路径上的各结点的关键字，直至根结点关键字，即为次小值；
3. 反复执行**步骤2**，直至所有叶节点均输出完毕。

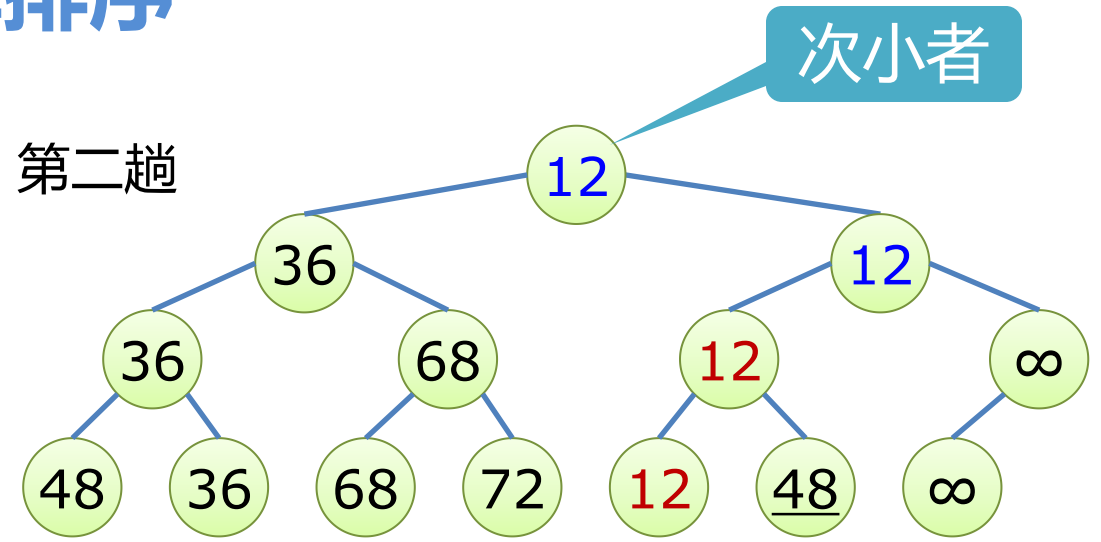
选择排序

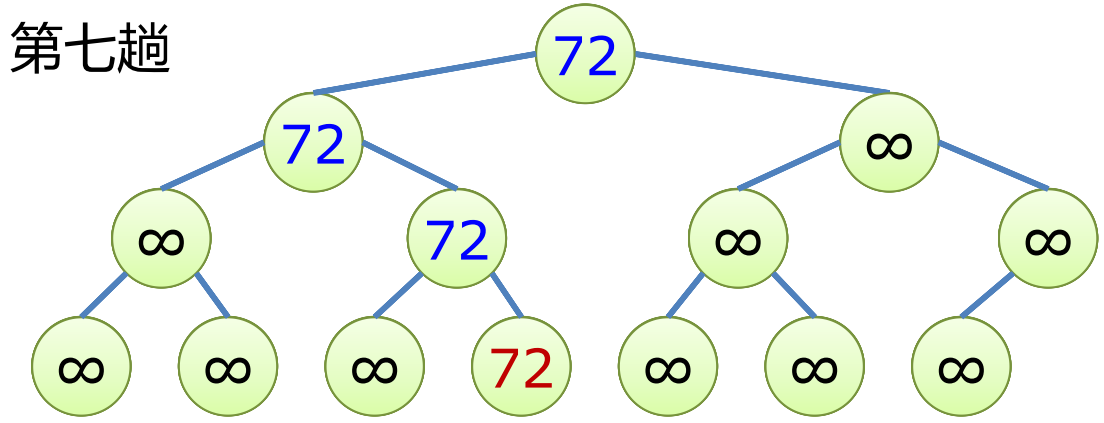
树形选择排序

树形选择排序

初 始: 48 36 68 72 12 48 02





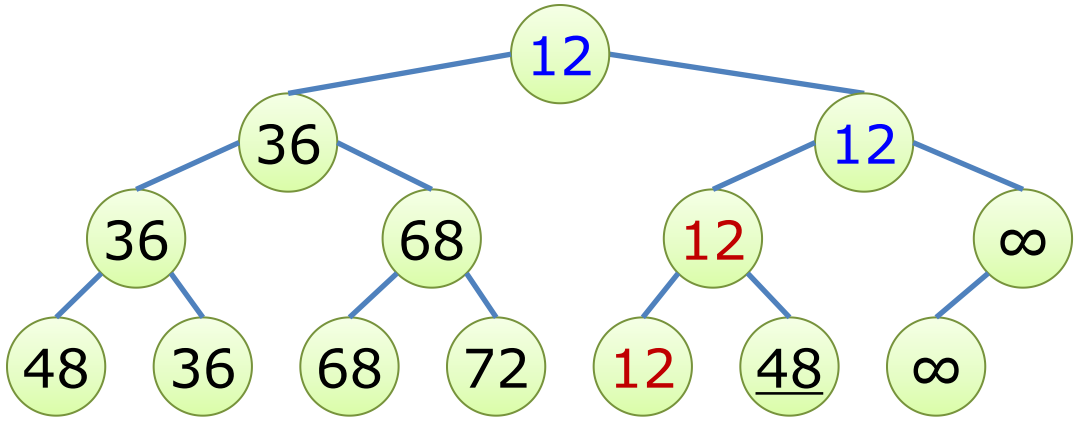


排序序列: 02 12 36 48 48 68 72

选择排序

树形选择排序

算法分析



- **时间复杂度**

1. 每次输出一个最小的关键字的元素，因此总共需要n-1趟。

2. 第一趟，每一层都需要两两对比，总共比较n-1次；后面的每一趟比较次数为$\log_2 n$；因此，总比较次数为：$(n-1)+(n-2)\log_2 n$，时间复杂度为**$O(n\log_2 n)$**。 移动次数为n。总开销的时间复杂度为**$O(n\log_2 n)$**。

- **空间复杂度**

需要额外的（n-1）个辅助空间来生成比较过程中关键字小的元素，即**$S(n)=O(n)$**

选择排序

树形选择排序

算法分析

排序趟数	n-1趟
稳 定 性	不稳定
时空复杂度	时间复杂度都是$O(nlog_2n)$；空间复杂度：$O(n)$
适 用 场 合	没有特别出彩的地方
元 素 确 定	每趟排序后，都至少能确定1个最大/最小元素的位置

选择排序

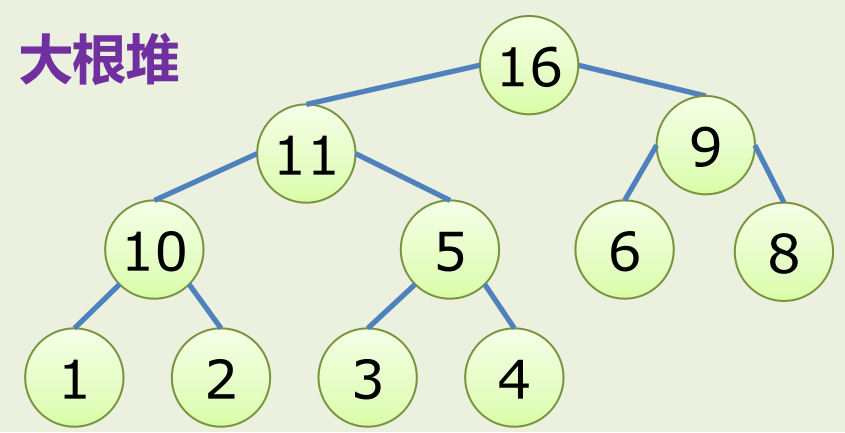
堆 排 序

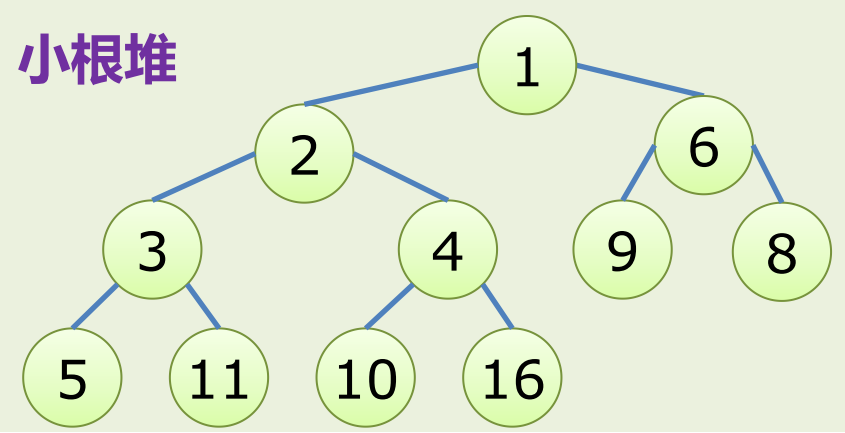
什么是堆

定义：若n个元素的序列 $\{k_1, k_2, \ldots, k_n\}$，满足如下关系，则称该序列为堆：

$$\begin{cases} k_i \le k_{2i} \\ k_i \le k_{2i+1} \end{cases} \quad \text{或} \quad \begin{cases} k_i \ge k_{2i} \\ k_i \ge k_{2i+1} \end{cases}$$

注意，此处堆只是一种利用树的结构特征来描述序列的方法，但堆的本质还是线性表，所有数据依然是存储在线性空间中。若将该序列看成是一个完全二叉树，则它任意分支结点的叶子结点均小于/大于它本身。。





选择排序

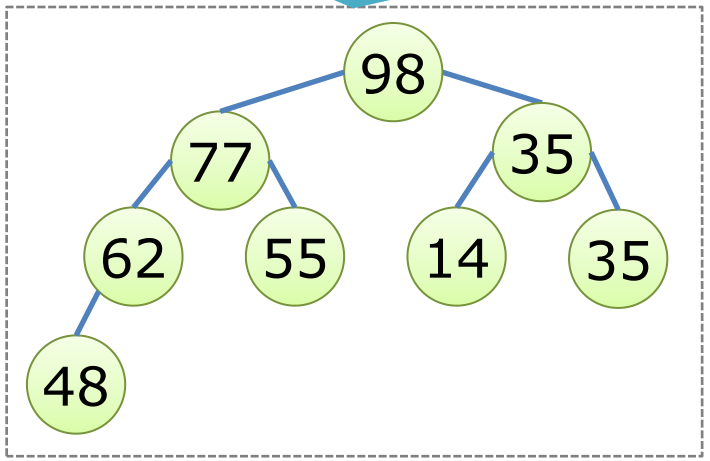
堆 排 序

什么是堆

例3：判断以下序列是否是堆?

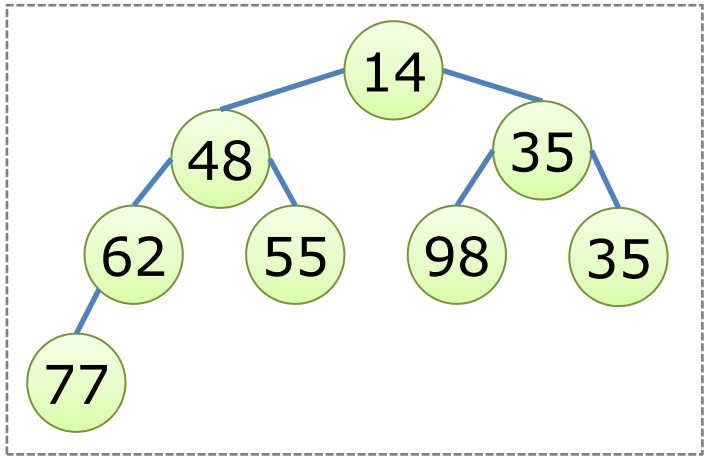
A={98,77,35,62,55,14,35,48}, B={14,48,35,62,55,98,35,77}, C={16,11,9,1,5,6,8,10,2,4}

按照层次遍历的规则构建二叉树



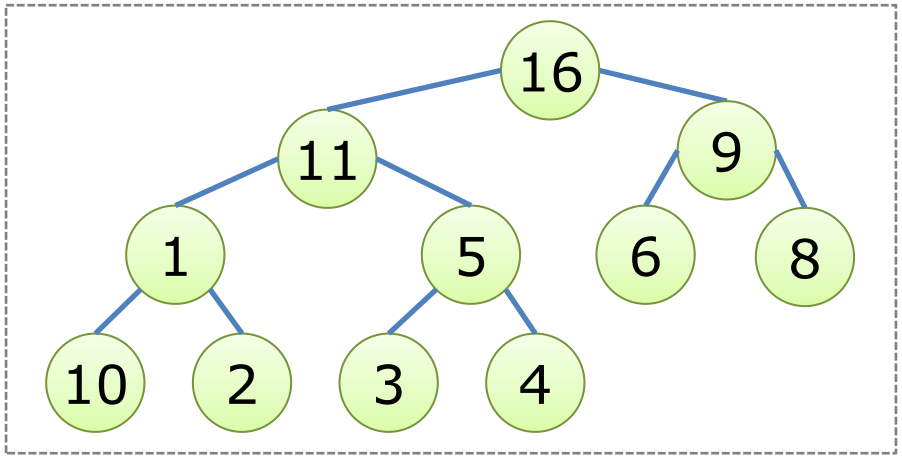
大根堆

所有分支结点都大于其两个叶结点



小根堆

所有分支结点都小于其两个叶结点



不是大根堆

结点10违反大根堆规则

选择排序

堆 排 序

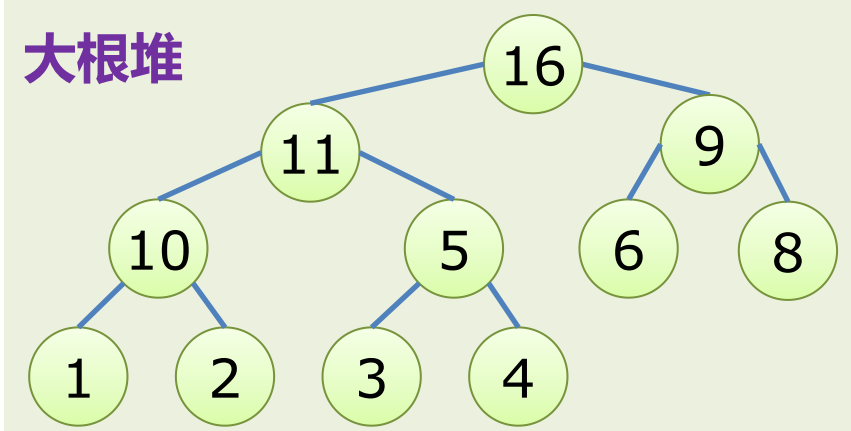
什么是堆排序

如何建堆?

如何调整?

基本思想

1. 将无序序列建成一个堆
2. 输出堆顶的最小（大）值
3. 将剩余的 n-1 个元素调整成一个堆，继续输出当前堆顶的最小（大）值
4. 重复执行**步骤2~3**，将无序序列输出成一个有序序列



选择排序

堆 排 序

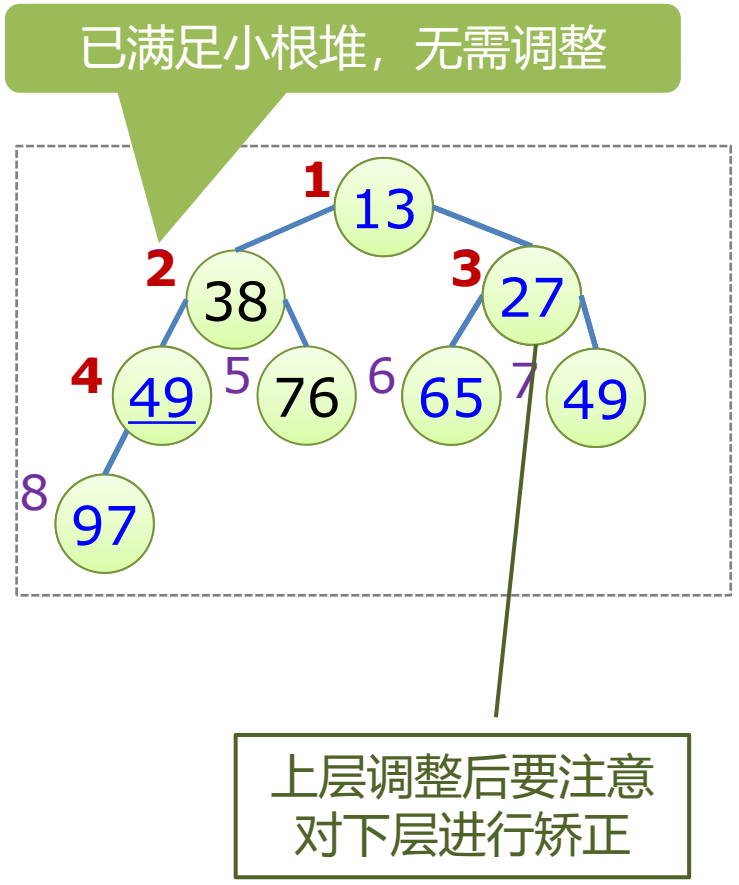
如何将无序序列建成堆

小根堆

例4：将关键字为 {49, 38, 65, 97, 76, 13, 27, 49} 的一组序列按照从小到大进行排序。

算法思想

1. **排列：**将所有元素按照序号依次排到完全二叉树中；
2. **调整：**从最后一个非叶子结点（**第n/2个**）依次向前，将以该结点为根的二叉树调整为堆。注：所有以叶子节点为根的子树都是堆，无需调整。
3. 重复执行**步骤2**，直至所有结点的子树都调整为堆



选择排序

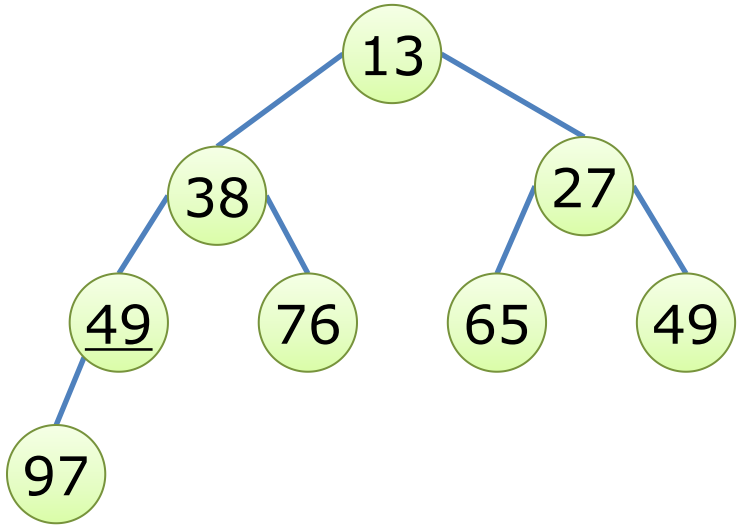
堆 排 序

如何重新调整堆并进行输出

例4：将关键字为 {49, 38, 65, 97, 76, 13, 27, 49} 的一组序列按照从小到大进行排序。

算法思想

1. **输出**：堆顶元素；
2. **替换**：用最后一个元素替代
3. **交换**：用小孩子替代根
4. **筛选**：从堆顶往叶子结点按照小根堆规则进行交换。
5. 重复执行**步骤1~4**，直至输出完毕



堆排序方法二见【项目014】习题1

排序完毕！

如何进行从大到小的排序？

大根堆

排序序列：

选择排序

堆 排 序

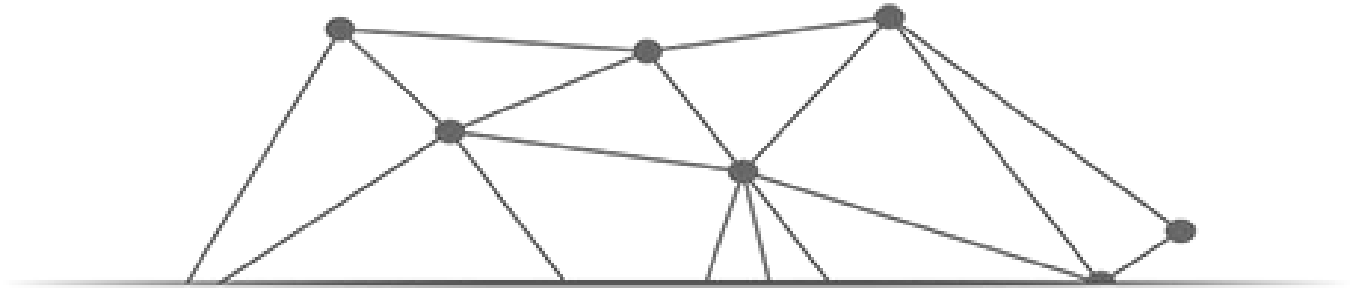
算法分析

排序趟数	n-1趟。堆排序包含两步：构造初始堆（1次）和调整排序（n-1趟）
稳 定 性	不稳定
时空复杂度	构造堆$O(n)$，调整排序堆$O(n\log_2 n)$；空间复杂度：$O(1)$
适 用 场 合	大数据量（特别是内存有限）
元 素 确 定	每趟排序后，都能确定1个最大/最小元素的位置

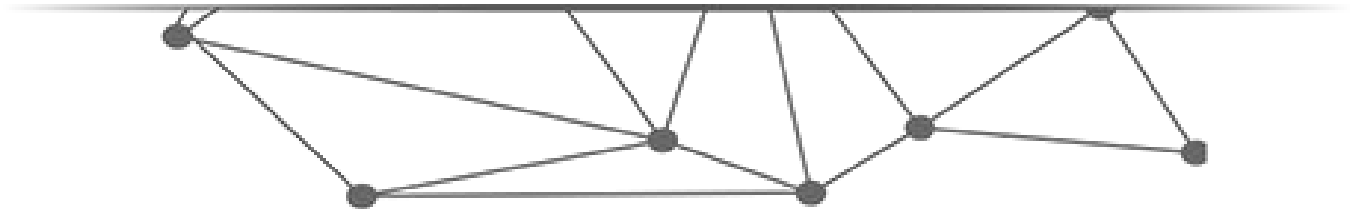
插入排序

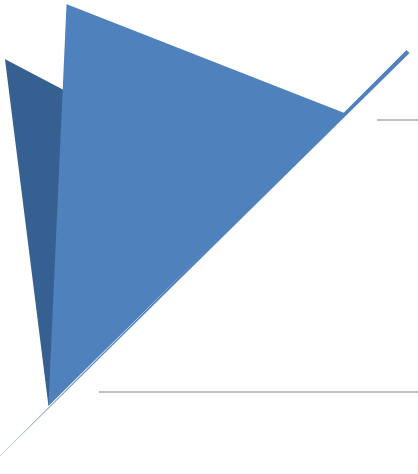
各种排序方法的比较

排序方法	时间复杂度			空间复杂度	稳定性
	最好情况	最坏情况	平均情况	辅助存储	
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
折半插入排序	$O(n\log_2 n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n)$	$O(n^2)$	$O(n^{1.3})$	$O(1)$	不稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
快速排序	$O(n^2)$	$O(n)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
树形选择排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定



课堂互动 18.4



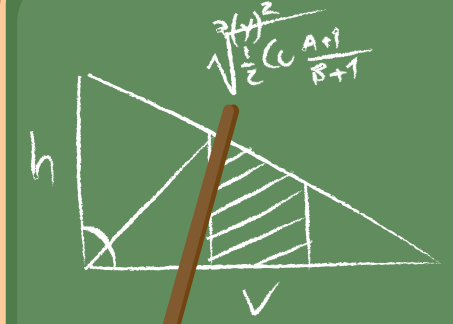


归并排序和基数排序

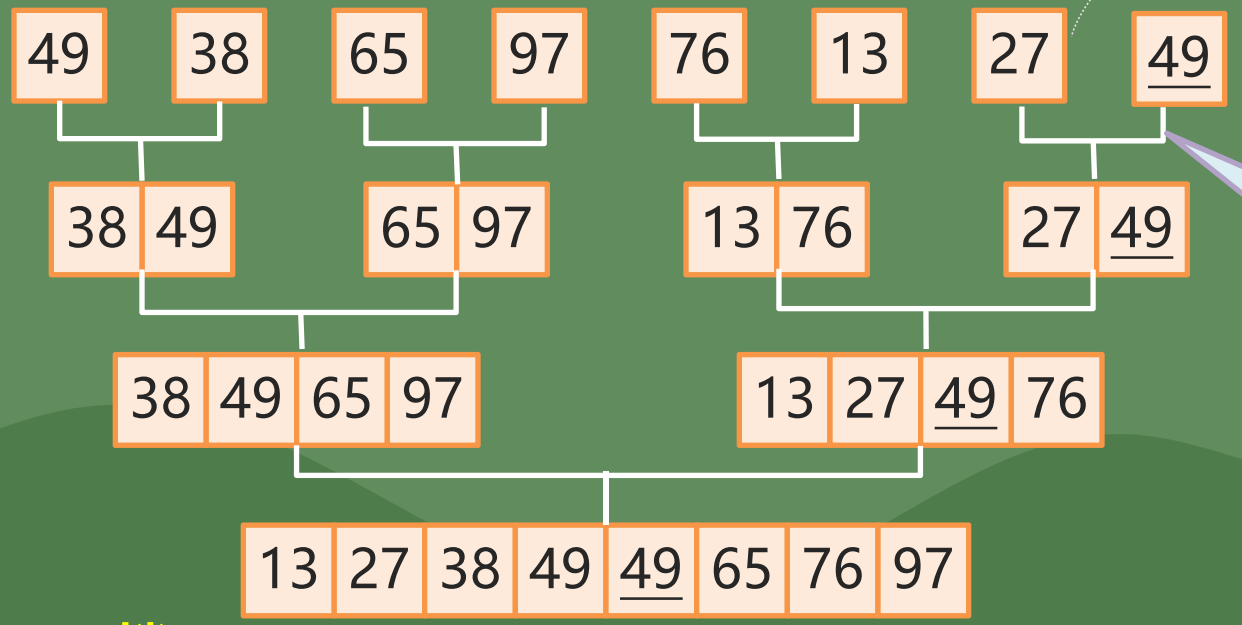
/ 归并排序
/ 桶排序
/ 基数排序
/ 多关键字排序

归并排序

基本思想



将两个或两个以上的有序表组合成一个新有序表



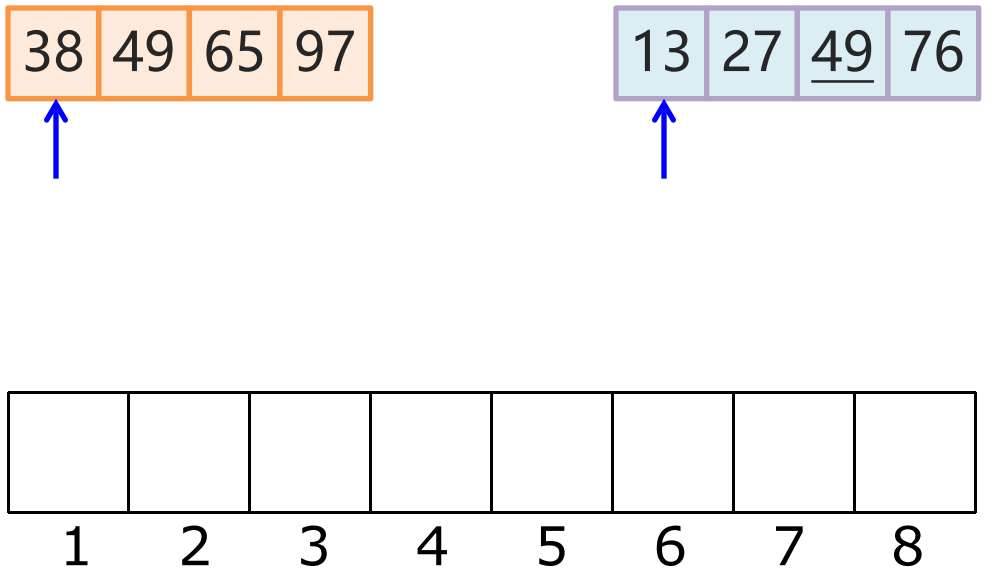
如何合并两个序列

整个归并排序仅需$\lceil log_2 n \rceil$趟

归并排序

如何将两个有序序列合并成一个有序序列?

每趟都需要对所有元素进行移动，因此每趟复杂度为O(n)



辅助空间等于源数组大小，因此复杂度为O(n)

算法步骤

1. 创建一个数组空间，其长度等于两个待合并序列的总长
2. 分别为两个序列指定一个工作指针，指向序列的第一个元素
3. 依次比较两个指针所指元素，并将较小的一个插入到新序列中，同时将指针向后移动一位
4. 当两个指针都移动到序列尾部时，合并结束

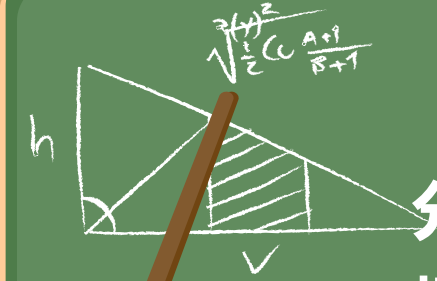
归并排序

算法分析

排序趟数	$\log_2 n$趟
稳 定 性	不稳定
时空复杂度	时间复杂度：$O(n\log_2 n)$；空间复杂度：$O(n)$
适 用 场 合	数据量大，并且对稳定性有要求的场景
元 素 确 定	排序结束后才能确定元素位置

桶排序

桶排序的基本思想



假设待排序数据是**均匀分布**的，算法将数组按**数值区间**分到**有限个**桶中，然后对每个桶**分别进行排序**，最后将每个桶中排好序的数进行**合并输出**。

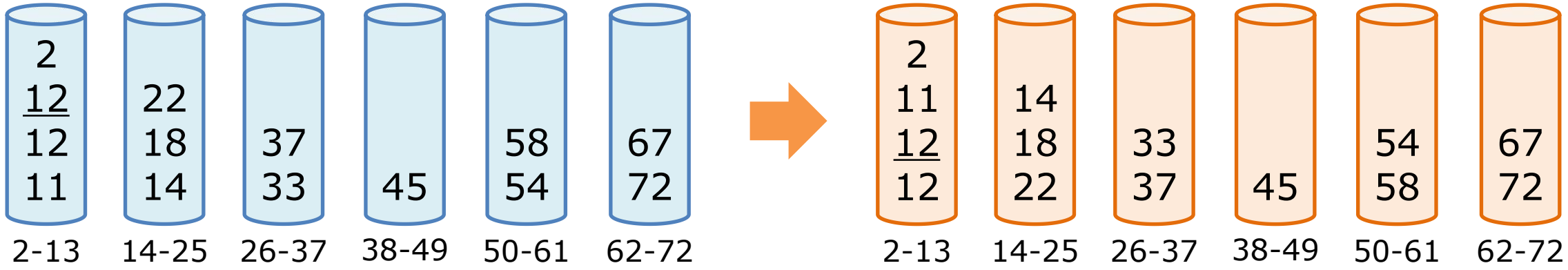
典型关键参数：

1. 确定桶的个数：m = (max - min)/n + 1
2. 确定每个桶的数值区间：k = (max - min)/m + 1

桶排序

例4：将关键字为 {11,12,14,18,22,33,37,45,54,58,72,12,02,67} 的一组序列按照从小到大进行排序。

1. 确定桶的个数：m = (max - min)/n + 1 = (72-2)/14+1 = 6
2. 确定每个桶存储的数值区间：k = (max - min)/m + 1 = (72-2)/6+1 = 12(12.6)
3. 分配元素到各个桶区间中，并分别进行排序



4. 将排序好的各个桶中的元素依次输出

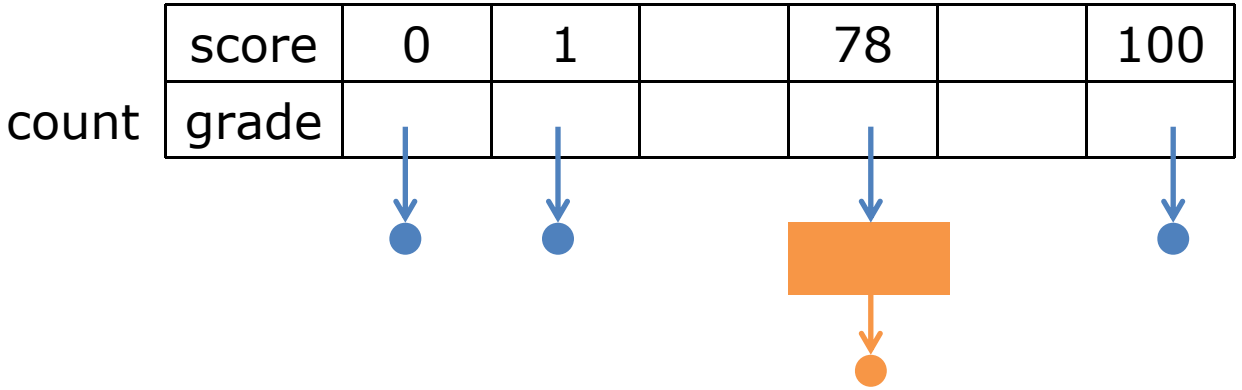
桶排序序列： 2, 11, 12, 12, 14, 18, 22, 33, 37, 45, 54, 58, 67, 72

桶的设置并不是固定的，根据具体的应用是可以进行随意调整的。

桶排序

例5：假设有N个学生，他们《数据结构》的成绩是0到100之间的整数（于是有M=101个不同的成绩）。如何在线性时间内将学生成绩按照从小到大排序呢?

1. 确定桶的个数：m = 101；每个桶存储1个分数值
2. 创建一个数组链来存储桶排序的所涉及的分数值



当M << N时，复杂度=O(N) **-> 线性**

若 M>>N怎么办?

桶排序的算法步骤

```
void BucketSort(ElementType A[] int N){
    A = count[];   // 初始化一个数组记录学生成绩
    // 1. 将数据输入到桶中
    while(读入一个学生成绩Score_i)
        将该生成绩插入 count[grade]链表;
    // 2. 输出桶中的数据（排序结果）
    for(i=0; i<M; i++){
        if(count[i])
            输出整个count[i]链表;
}
```

时间复杂度：T(M,N)=**O(M+N)**

桶数　样本数

基数排序

例6：假设有N=10个整数，每个整数的取值范围是0~999（于是有M=1000个不同的值）。此时，还能在线性时间内排序吗?

✓ 桶排序：T(M,N) = O(M+N) = O(1010) >> O(10)

输入序列：64, 8, 216, 512, 27, 729, 0, 1, 343, 125

使用“次位优先”基数排序（Least Significant Digit）[个位-十位-百位]

趟数

$$T = O(d(n+m))$$

样本数　桶数

Bucket	0	1	2	3	4	5	6	7	8	9
第1趟-个位	**0**	**1**	51**2**	34**3**	6**4**	12**5**	21**6**	2**7**	**8**	72**9**
第2趟-十位	**0** **1** **8**	5**1**2 2**1**6	1**2**5 **2**7 7**2**9		3**4**3		6**4**			
第3趟-百位	**0** **1** **8** **27** **64**	**1**25	**2**16	**3**43		**5**12		**7**29		

多关键字排序

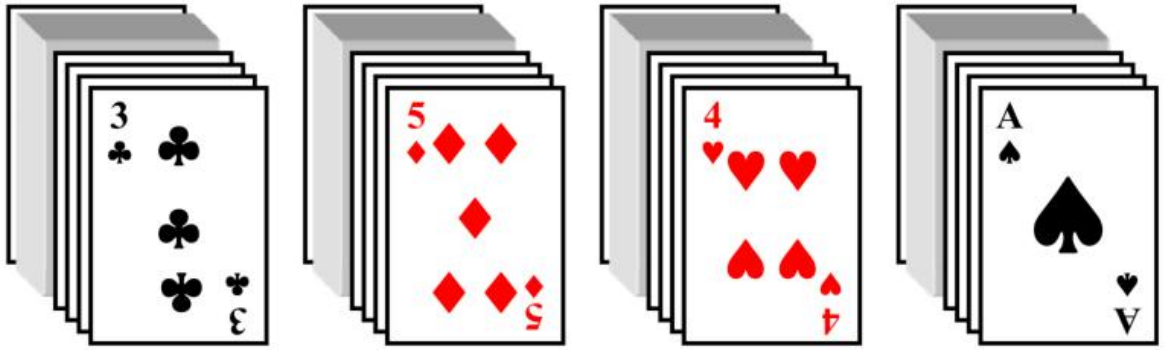
例7：将一副扑克牌按2种关键字进行排序，其中“花色”地位高于“面值”

- K^0[花色]：♥ > ♠ > ♣ > ♦
- K^1[面值]：2<3<4<5<6<7<8<9<10<J<Q<K<A

“主位优先”排序

Most Significant Digit

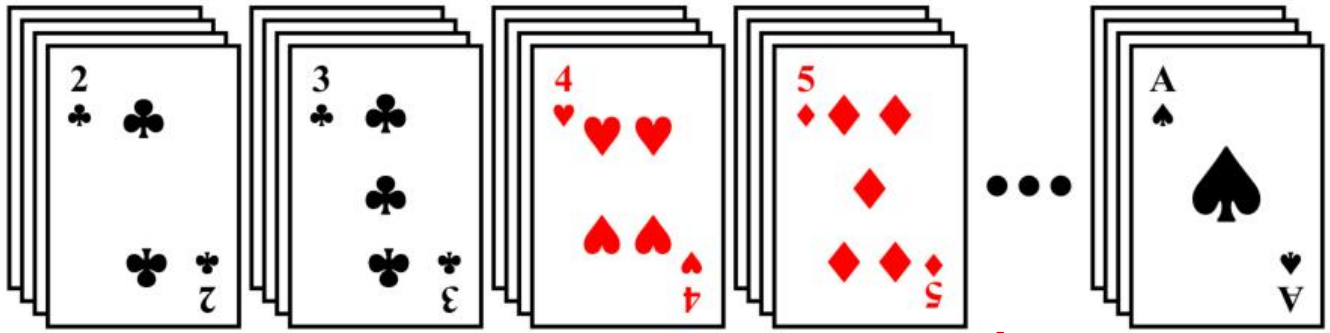
每个花色建1个桶



“次位优先”排序

Least Significant Digit

每个面值建1个桶

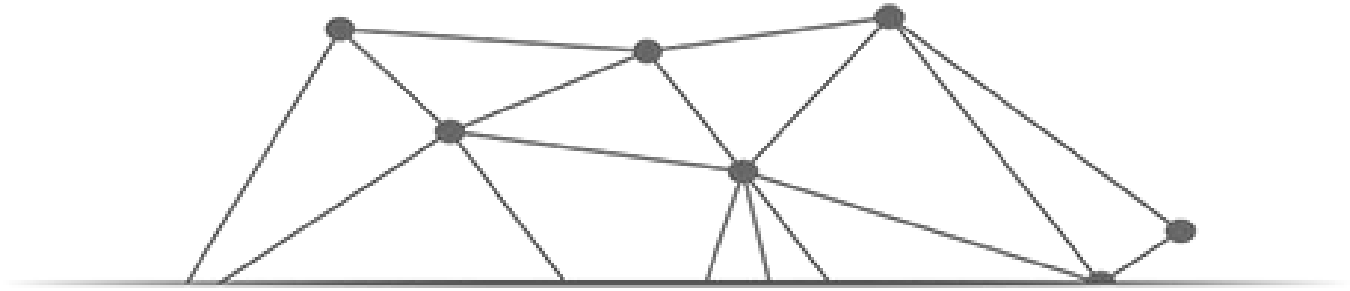


思考：MSD和LSD哪种方法更快?

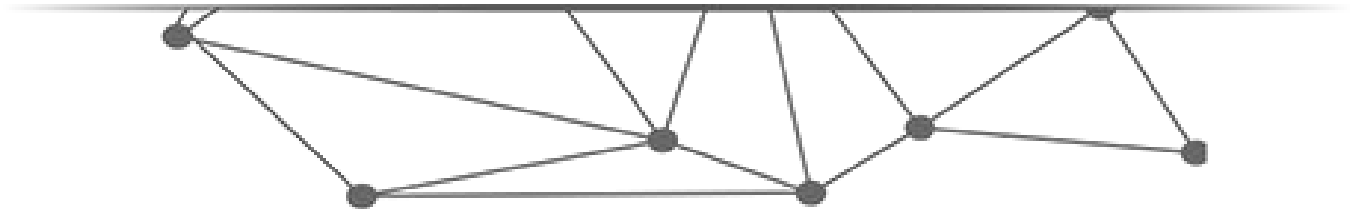
基数排序

算法分析

排序趟数	d趟
稳 定 性	稳定
时空复杂度	时间复杂度：**O(d(n+m))**；空间复杂度：**O(n+dm)**
适 用 场 合	✓ 数据元素n较大 ✓ 关键字可以拆分成较少的d组 ✓ 每组中的取值范围m较小
元 素 确 定	排序结束后才能确定元素位置



课堂互动18.5



第21讲 内排序

各种排序方法的比较

排序方法	时间复杂度			空间复杂度	稳定性
	最好情况	最坏情况	平均情况	辅助存储	
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
折半插入排序	$O(n\log_2 n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n)$	$O(n^2)$	$O(n^{1.3})$	$O(1)$	不稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
快速排序	$O(n^2)$	$O(n)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
树形选择排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序	$O(d(n+m))$	$O(d(n+m))$	$O(d(n+m))$	$O(n+dm)$	稳定

第21讲 内排序

各种排序方法的比较

- 当n较小时，n^2和$n\log_2 n$差别不大，可用简单排序；当关键字基本有序时，直接插入排序、冒泡排序速度很快。
- 当n较大时，快速排序平均性能最佳（目前比较排序中最快的），但在最坏情况时（关键字有序），时间复杂度为$O(n^2)$，空间复杂度为$O(n)$。堆排序和归并排序不存在快速排序的最坏情况，但归并排序空间复杂度较高。基数排序适用于n较大，但关键字较小的序列，但需要预先知道关键字的取值范围。因此，当n较大时：
  - ✓ 关键字随机，不要求稳定性，选择快速排序
  - ✓ 关键字基本有序，不要求稳定性，选择堆排序
  - ✓ 关键字基本有序，要求稳定性，选择归并排序
- 基数排序以及时间复杂度为$O(n^2)$的简单排序都比较稳定；性能好的快速排序、堆排序和希尔排序都不稳定

欧老师的联系方式

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Website: http://ouxinyu.cn

Tel：18687840023

地址：安宁校区 诚远楼201

南院 智能应用研究院A306-2