

# 复习

## 一、要求

1. 熟练掌握各种数据结构的基本概念、存储表示方法及基本操作的实现算法。
  - (1) 记忆（记住一些概念的关键点）
  - (2) 理解、分析
  - (3) 灵活运用
2. 对所学知识有一定的综合应用能力。
3. 掌握算法的时间和空间复杂度的分析方法。

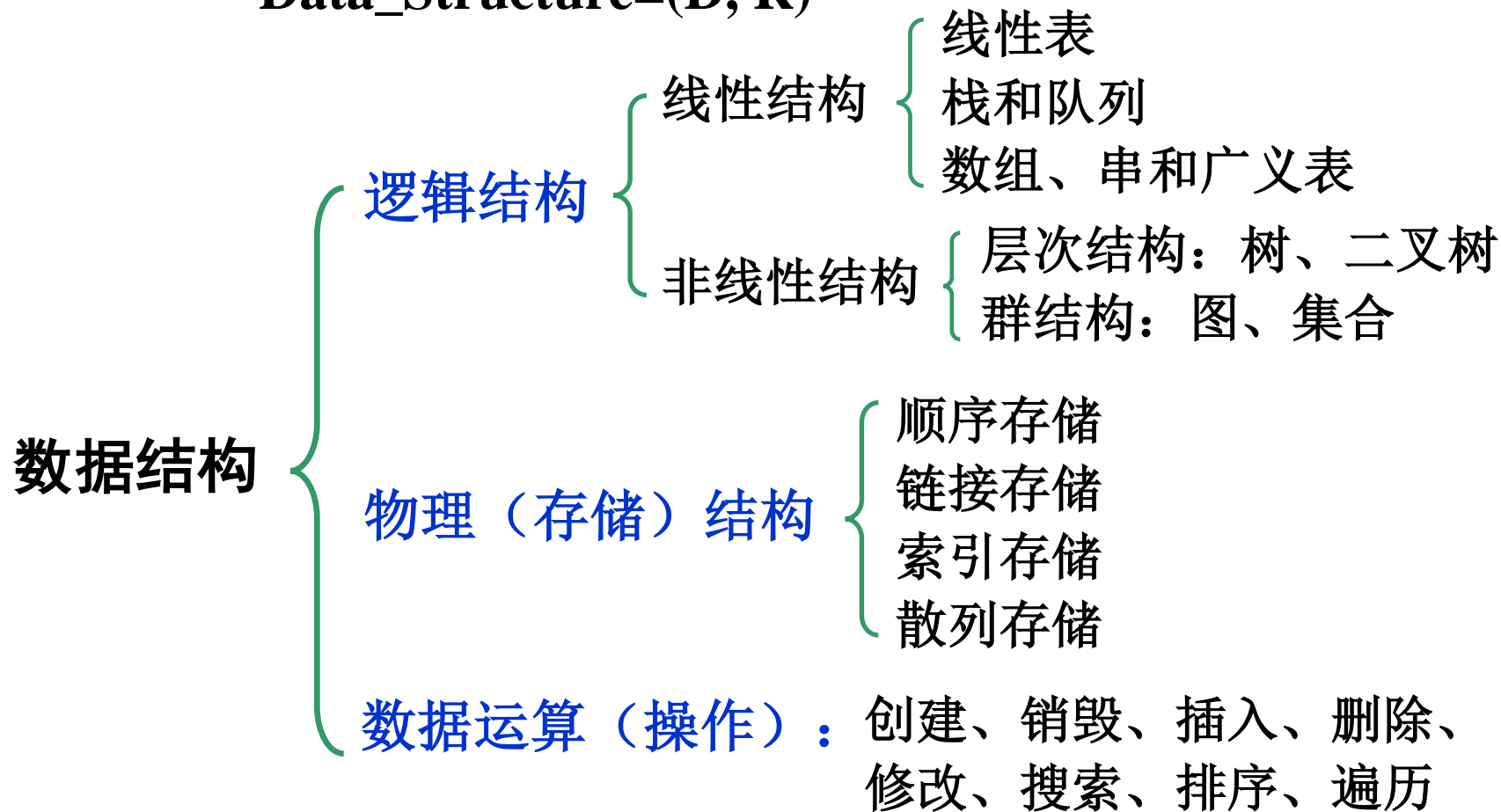
## 二、题型

选择、解答、算法设计

# 第一章 绪论

## 1. 数据结构的概念

$\text{Data\_Structure} = (D, R)$



## 2. 抽象数据类型（ADT）的概念、特点

## 3. 算法和算法分析

- ◆ 算法的定义，算法与程序的区别；
- ◆ 算法的特性：有穷性、确定性、能行性、输入、输出；
- ◆ 算法设计的目标：正确性、可读性、健壮性、效率（算法执行时间和空间利用率）；
- ◆ 算法分析的含义  
算法的事前分析： $\begin{cases} \text{时间复杂度: } T(n)=O(f(n)) \\ \text{空间复杂度: } S(n)=O(g(n)) \end{cases}$   
掌握时间和空间复杂度的估算方法。

## 第二章 线性表

1. 线性表的基本概念。
2. 线性表的两种存储结构的不同特点及其适用场合。
  - ◆ **顺序存储**：借助元素在存储器中的相对位置来表示元素之间的逻辑关系。
  - ◆ **链式存储**：借助指示元素存储地址的指针来表示元素之间的逻辑关系。
  - ◆ **顺序表和链表的优缺点比较**（正好是相对的）。
3. 在线性表的两种存储结构上实现基本操作的算法。
  - ◆ 创建、搜索、插入、删除、归并、分解、就地逆置等

**注意：**单链表与双向链表、普通链表与循环链表的区别。

**双向链表中结点的特征：**

$$p == p \rightarrow \text{lLink} \rightarrow \text{rLink} == p \rightarrow \text{rLink} \rightarrow \text{lLink}$$

因此，在删除时，对单链表，必须要知道第*i*-1个结点的地址，而对双向链表则不必如此。

例1. 按递增有序输出单链表中的元素值。

例2. 实现集合的交、并、差等运算。

## 4. 线性表的应用

{ 约瑟夫环  
一元多项式的表示及运算

## 第三章 栈和队列

### 1. 栈的定义、特征以及抽象数据类型定义。

会灵活运用栈的特征（后进先出）。

例：设输入元素为1, 2, 3, P, A，输入次序为123PA，元素经过栈后到达输出序列，当所有元素均到达输出序列后，有哪些序列可以作为高级语言的变量名？

### 2. 栈的存储结构 { 顺序栈：栈满和栈空的条件 链式栈

入栈、出栈、取栈顶元素等基本操作的实现算法。

### 3. 栈的应用 { 表达式计算：了解基本思想、栈在算法中的作用 实现递归算法

4. 队列的定义、特征以及抽象数据类型定义。

5. 队列的存储结构 { (顺序) 循环队列  
链队列

入队、出队、取队头元素等基本操作的实现算法。

循环队列: { 入队:  $\text{elements}[\text{rear}] = x;$   
 $\text{rear} = (\text{rear} + 1) \% \text{maxSize};$   
出队:  $x = \text{elements}[\text{front}];$   
 $\text{front} = (\text{front} + 1) \% \text{maxSize};$

判断队满和队空: (1) 少用一个存储单元;  
(2) 设置一个标志位(tag);  
(3) 设置一个计数器(length)。

6. 队列的应用: 结合后面二叉树和图的遍历算法。

## 第四章 数组、串和广义表

1. 数组采用行优先或列优先顺序存储时，数组元素的存储地址的计算方法（重点掌握二维、三维数组）。
2. 特殊矩阵进行压缩存储时下标变换公式的推导方法。  
要求掌握上三角矩阵、下三角矩阵、对称矩阵、三对角矩阵分别按行优先或列优先压缩存储时的下标变换公式。
3. 了解稀疏矩阵的压缩存储中，三元组顺序表的描述方法及实现矩阵运算的基本思想。

**注：**串和广义表（略）



## 第五章 树和二叉树

1. 树、二叉树的结构特性；二叉树的五种基本形态。
2. 满二叉树和完全二叉树的定义、特点；二叉树的性质（五条，其中有条只对完全二叉树满足）；二叉树的存储结构。
3. 二叉树的遍历策略及算法（递归和非递归）。由前序和中序遍历序列、中序和后序遍历序列能唯一构造出一棵二叉树。

会灵活运用遍历算法求解其他问题。

例1. 输出某类结点的值或求某类结点的个数（如叶子结点）。

例2. 判定一棵二叉树是否为二叉搜索树（采用中序遍历策略，判断正在访问的结点与它的前驱结点之间是否递增有序）。

4. 二叉树的线索化的目的以及方法，会画线索二叉树；在线索二叉树中查找结点的前驱、后继的方法。

**注意：**在线索链表中判断结点是否有左、右孩子的方法与二叉链表有何不同；查找时要深刻理解遍历策略。

5. 树的存储结构及其特点（**广义表表示法**、**双亲表示法**、**子女链表表示法**和**子女-兄弟链表表示法**（也称二叉树表示法），重点掌握**子女-兄弟链表表示法**）；树、森林与二叉树的转换方法。

6. 树和森林的遍历方法

**注意：** { 先根（对应二叉树的前序遍历）  
后根（对应二叉树的中序遍历）

7. 堆的定义；最小堆的建立（向下调整）方法；最小堆的插入（向上调整）和删除元素的方法。

8. Huffman树的概念及构造方法，WPL值的计算；哈夫曼编码的设计。

**注意：**Huffman树是扩充二叉树（即树中只含度为0和度为2的结点）。

## 第六章 集合与字典

### 1. 并查集的概念以及实现方法。

- ◆ 用一棵树表示一个集合，采用树的双亲表示法
- ◆ 加权的Union算法
- ◆ 压缩路径的Find算法
- ◆ 了解并查集的应用：等价类划分、Kruskal算法的实现等

### 2. 了解字典的概念以及组织方式。

### 3. 散列

- ◆ 基本概念：散列表、冲突、装载因子、堆积（或聚集）等；
- ◆ 散列表的构造：散列函数、处理冲突的方法（重点掌握闭散列法和开散列法（也称链地址法））；
- ◆ 散列表的搜索、插入和删除方法；
- ◆ 根据公式计算等概率情况下的搜索成功和不成功的ASL。

## 第七章 搜索结构

1. 静态搜索表的顺序搜索、折半搜索的实现算法；描述搜索过程的判定树的构造方法；分块搜索的基本思想。
2. 二叉搜索树（也称**二叉排序树**）的概念和性质；二叉搜索树的构造以及搜索、插入和删除操作的实现方法。  
**注意：**二叉搜索树按**中序遍历**是**递增有序**序列。
3. AVL树（平衡的二叉搜索树）的概念；插入或删除后调整平衡的方法（即重点掌握平衡化旋转的4种方法）。
4. 了解红黑树的概念和性质。
5. B树和B+树的概念；B树的搜索、插入和删除方法。
6. 根据公式计算各种搜索方法在等概率情况下的ASL。

## 第八章 图

### 1. 图的定义、基本术语。

（如：无向完全图、有向完全图、路径、回路、连通图和连通分量、强连通图和强连通分量、连通图的生成树）

**注意：**边数与顶点数、度之间的关系：
$$e = \frac{1}{2} \sum_{i=1}^n TD(V_i)$$

完全图、连通图以及生成树的特点。

判断图中是否存在回路的方法。

### 2. 图的各种存储结构（重点掌握邻接矩阵和邻接表）的特点及构造方法。

**注意：**图和网的区别。

3. 图的两种遍历策略及遍历算法（**深度优先**和**广度优先**）。  
会**灵活运用**遍历算法求解其他问题。（例：求路径）

#### 4. 图的应用

（1）生成树的定义、类型（**深度优先**和**广度优先生成树**）；  
最小生成树的定义；**Kruskal**算法和**Prim**算法求网的最小生成树的方法（**注意**：记住算法特点，不要混淆）。

（2）**Dijkstra**算法求单源最短路径的方法（有向网和无向网都适用）。

（3）拓扑排序的方法及算法实现。

**注意**：可以用拓扑排序的方法来判断有向图中是否存在回路；拓扑排序的算法实现**要设一个栈（或队列）**保存入度为0的顶点序号。

（4）关键路径问题的求解方法。

## 第九章 内部排序

1. 排序的定义，排序方法“稳定”或“不稳定”的含义。

**注意：**希尔排序、快速排序、直接选择排序、堆排序是不稳定的排序算法。

2. 各种排序方法的基本思想、算法特点、排序过程以及它们的时间和空间复杂度分析。

（包括基本算法及它们的改进算法：直接插入排序、折半插入排序、希尔排序、起泡排序、快速排序、直接选择排序、堆排序、归并排序及基数排序）

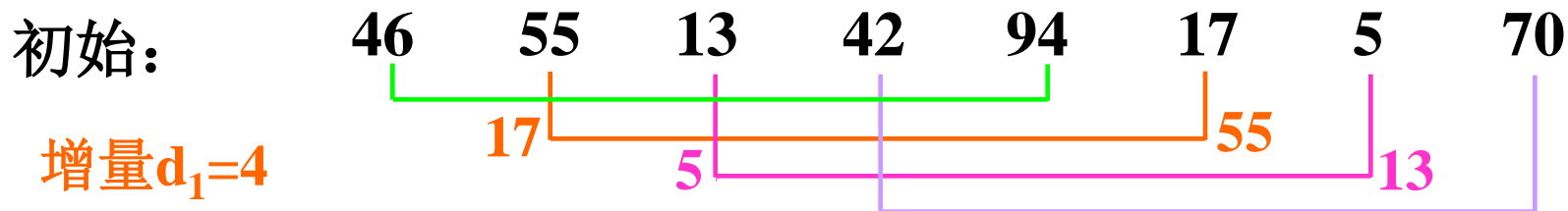
会根据实际应用情况**选择合适的排序算法**。



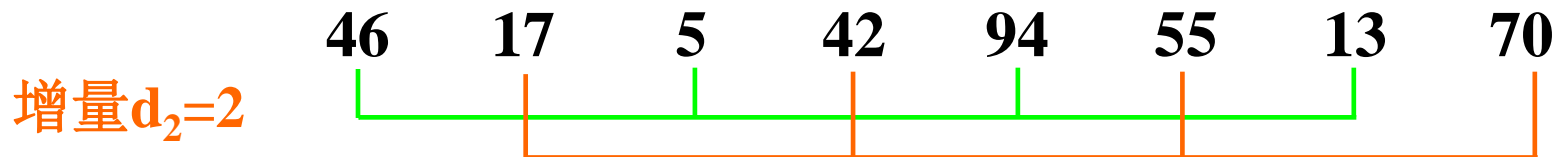
**后面补充的是内部排序的例子以及其他  
教材实现快速排序和归并排序的方法**

**(算法部分选读)**

■ 例：记录数 $n=8$ ，进行**希尔排序**  
(按Shell的方法选择增量(间隔))



第一趟排序结果:



第二趟排序结果:



第三趟排序结果: 5 13 17 42 46 55 70 94

■ 例： **起泡排序**（从前往后两两比较， **最大值放到最后**）：

初始： [44 55 22 33 99 11 66 77]

第一趟排序之后： [44 22 33 55 11 66 77] **99**

第二趟排序之后： [22 33 44 11 55 66] **77** 99

第三趟排序之后： [22 33 11 44 55] **66** 77 99

第四趟排序之后： [22 11 33 44] **55** 66 77 99

第五趟排序之后： [11 22 33] **44** 55 66 77 99

第六趟排序之后： [11 22] **33** 44 55 66 77 99

此趟没有记录进行  
交换，所以不必执  
行第七趟


## ■ 例：快速排序：

(从两端向中间检测)


初始：[46 55 13 42 94 5 17 70]




进行一次交换后：[17] 55 13 42 94 5 □ [70]




进行二次交换后：[17] □ 13 42 94 5 [55 70]



进行三次交换后：[17 5] 13 42 94 □ [55 70]



进行四次交换后：[17 5 13 42] □ [94 55 70]



完成一趟排序：[17 5 13 42] 46 [94 55 70]

二趟排序之后：[13 5] 17 [42] 46 [70 55] 94

三趟排序之后：[5] 13 17 42 46 [55] 70 94

## ■ 快速排序算法：

```
void QSort (DataType a[], int low, int high)
```

```
    //对记录a[low]~a[high]作快速排序
```

```
    { int i,j; DataType temp;
```

```
      i=low; j=high;
```

```
      temp=a[low];      //序列中的第一个记录作支点（枢轴记录）
```

```
      while (i<j)      //序列长度大于1
```

```
        { while (i<j&& a[j].key>=temp.key) j--;
```

```
          if(i<j) { a[i]=a[j]; i++; }
```

```
          while (i<j&& a[i].key<=temp.key) i++;
```

```
          if(i<j) { a[j]=a[i]; j--; }
```

```
        }
```

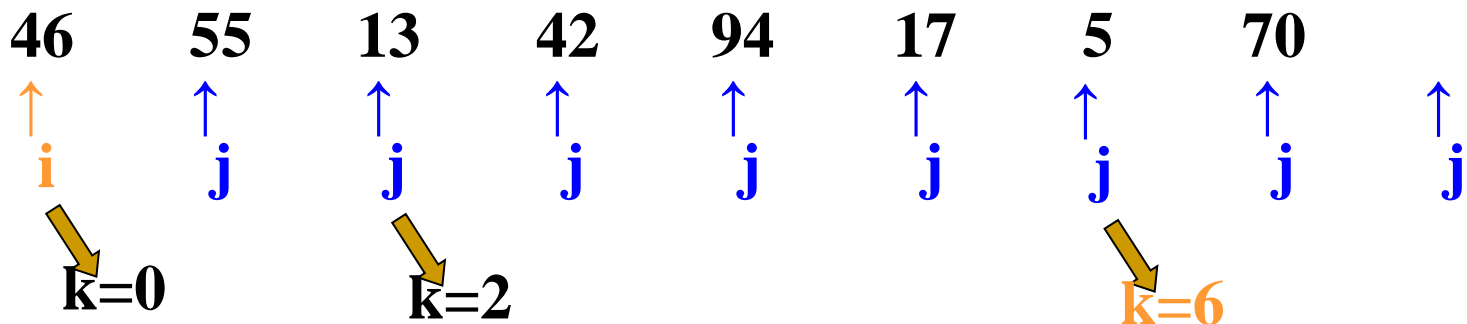
```
      a[i]=temp;      //枢轴记录到位
```

```
      if (low<i-1) QSort(a,low,i-1);      //对左边的子序列递归排序
```

```
      if (i+1<high) QSort(a,i+1,high);    //对右边的子序列递归排序
```

```
    }
```

## ■ 例：直接选择排序：



∵  $k \neq i$ , 交换

第一趟排序结果: [5] 55 13 42 94 17 46 70

第二趟排序结果: [5 13] 55 42 94 17 46 70

第三趟排序结果: [5 13 17] 42 94 55 46 70

第四趟排序结果: [5 13 17 42] 94 55 46 70

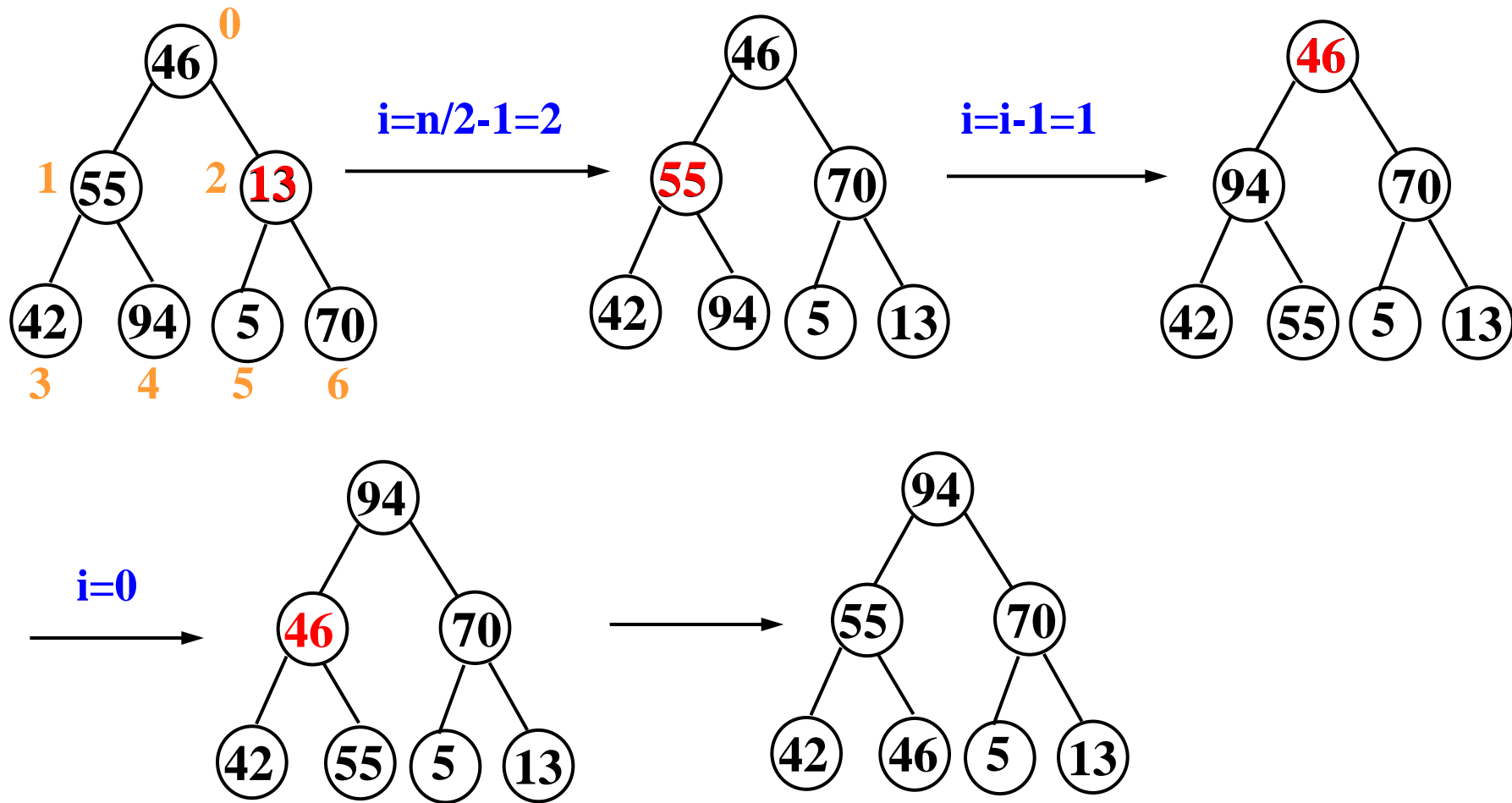
第五趟排序结果: [5 13 17 42 46] 55 94 70

第六趟排序结果: [5 13 17 42 46 55] 94 70

第七趟排序结果: [5 13 17 42 46 55 70] 94

■ 例：对关键字序列 { 46, 55, 13, 42, 94, 5, 70 } 进行堆排序

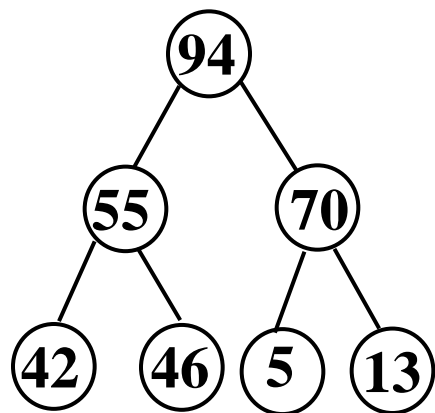
(1) 建立初始最大堆：



初始堆（最大堆）

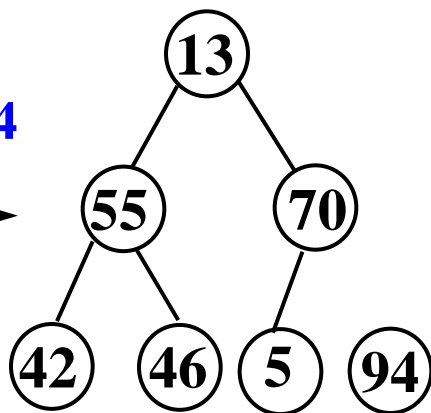
序列： 94, 55, 70, 42, 46, 5, 13

## (2) 交换元素并重新调整为堆:

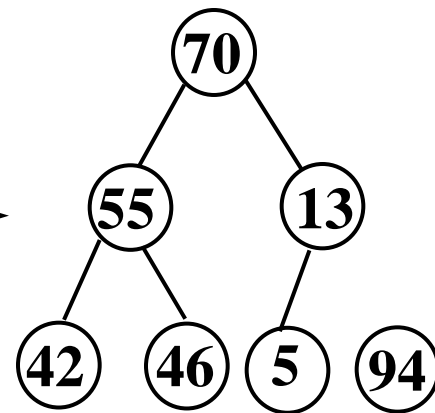


初始堆

交换94  
与13

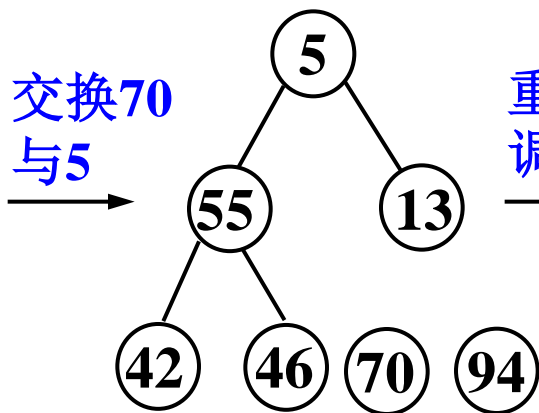


重新  
调整

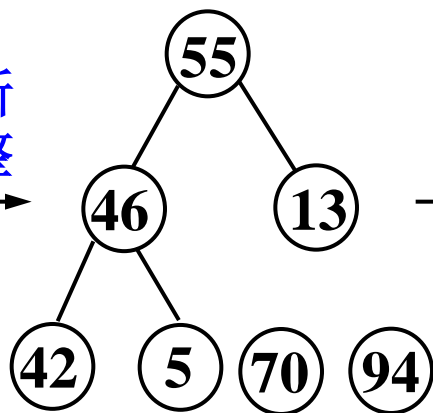


第一趟排序结果

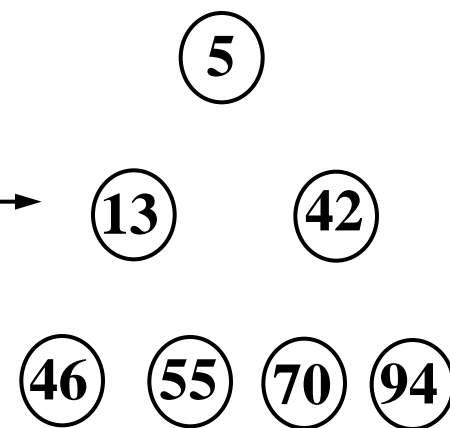
序列: 70, 55, 13, 42, 46, 5, 94



重新  
调整



...



第二趟排序结果

序列: 55, 46, 13, 42, 5, 70, 94

结果序列: 5, 13, 42, 46, 55, 70, 94



## ■ 例：2-路归并排序：

关键字序列：46, 55, 13, 42, 94, 5, 17, 70

初始： [46] [55] [13] [42] [94] [5] [17] [70]

第一趟归并后： [46, 55] [13, 42] [5, 94] [17, 70]

```
graph TD; A1[46] --- B1[46, 55]; A2[55] --- B1; A3[13] --- B2[13, 42]; A4[42] --- B2; A5[5] --- B3[5, 94]; A6[94] --- B3; A7[17] --- B4[17, 70]; A8[70] --- B4;
```

第二趟归并后： [13, 42, 46, 55] [5, 17, 70, 94]

```
graph TD; B1[46, 55] --- C1[13, 42, 46, 55]; B2[13, 42] --- C1; B3[5, 94] --- C2[5, 17, 70, 94]; B4[17, 70] --- C2;
```

第三趟归并后： [5, 13, 17, 42, 46, 55, 70, 94]

```
graph TD; C1[13, 42, 46, 55] --- D[5, 13, 17, 42, 46, 55, 70, 94]; C2[5, 17, 70, 94] --- D;
```

## ■ 归并排序算法（非递归）

### 1. 一趟2-路归并排序算法

```
void Merge(DataType a[],int n,DataType swap[],int k)
```

```
//对序列a[0]~a[n-1]进行一趟二路归并排序，每个有序  
//子序列的长度为k
```

```
{ int i,j,l1,l2,u1,u2,m;
```

```
  m=0; l1=0;
```

```
  while (l1+k<=n-1)
```

```
    { l2=l1+k; u1=l2-1;
```

```
      u2=(l2+k-1<=n-1)?l2+k-1:n-1;
```

```
      //将两个相邻的有序子序列进行归并
```

```
      for (i=l1,j=l2; i<=u1&& j<=u2; m++)
```

```
        if (a[i].key<=a[j].key) { swap[m]=a[i]; i++; }
```

```
        else { swap[m]=a[j]; j++; }
```

**while (i<=u1)**                      **//子序列2已归并完**

**{ swap[m]=a[i];**

**m++;**

**i++;**

**}**

**while (j<=u2)**                      **//子序列1已归并完**

**{ swap[m]=a[j];**

**m++;**

**j++;**

**}**

**l1=u2+1;**

**}**

**//若待归并的子序列数为奇数，则将最后一个子序列直接**

**//复制到swap中**

**for (i=l1;i<n;i++,m++) swap[m]=a[i];**

**}**

## 2. 将一个无序表按2-路归并为一个有序表

```
void MergeSort(DataType a[],int n)
```

```
    //用二路归并排序法对记录a[0]~a[n-1]排序
```

```
    { int i, k;    DataType swap[n];
```

```
        k=1;                //归并长度由1开始
```

```
        while (k<n)
```

```
            { Merge(a,n,swap,k);
```

```
                for (i=0;i<n;i++) a[i]=swap[i]; //将记录从数组swap放回a中
```

```
                k=2*k;                //归并长度加倍
```

```
            }
```

```
    }
```

## ■ LSD链式基数排序：

每个链队列设立两个指针：

**front[i]**：队列的**头指针**，指向第一个进入队列的关键字

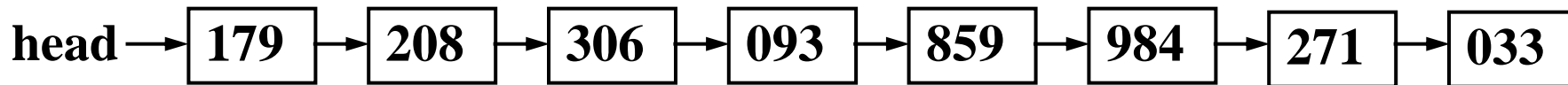
**rear[i]**：队列的**尾指针**，指向当前队列中刚进入的关键字

例：待排序列的初始状态是一个单链表，含有8个记录，其关键字分别为 179, 208, 306, 093, 859, 984, 271, 033，它们是十进制数，所以基数  $r=10$ ，关键字的位数  $d=3$ 。

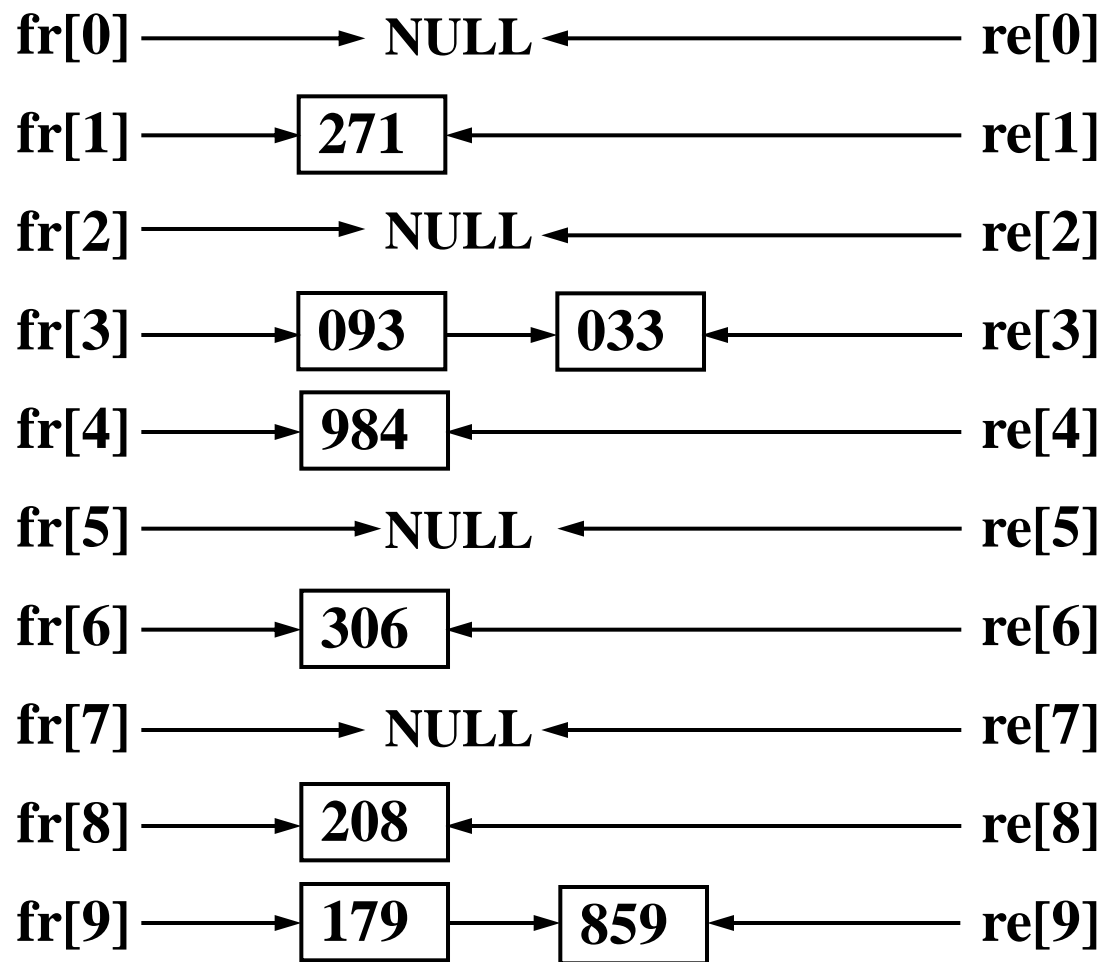


(a)初态

链式基数排序的排序过程如下：



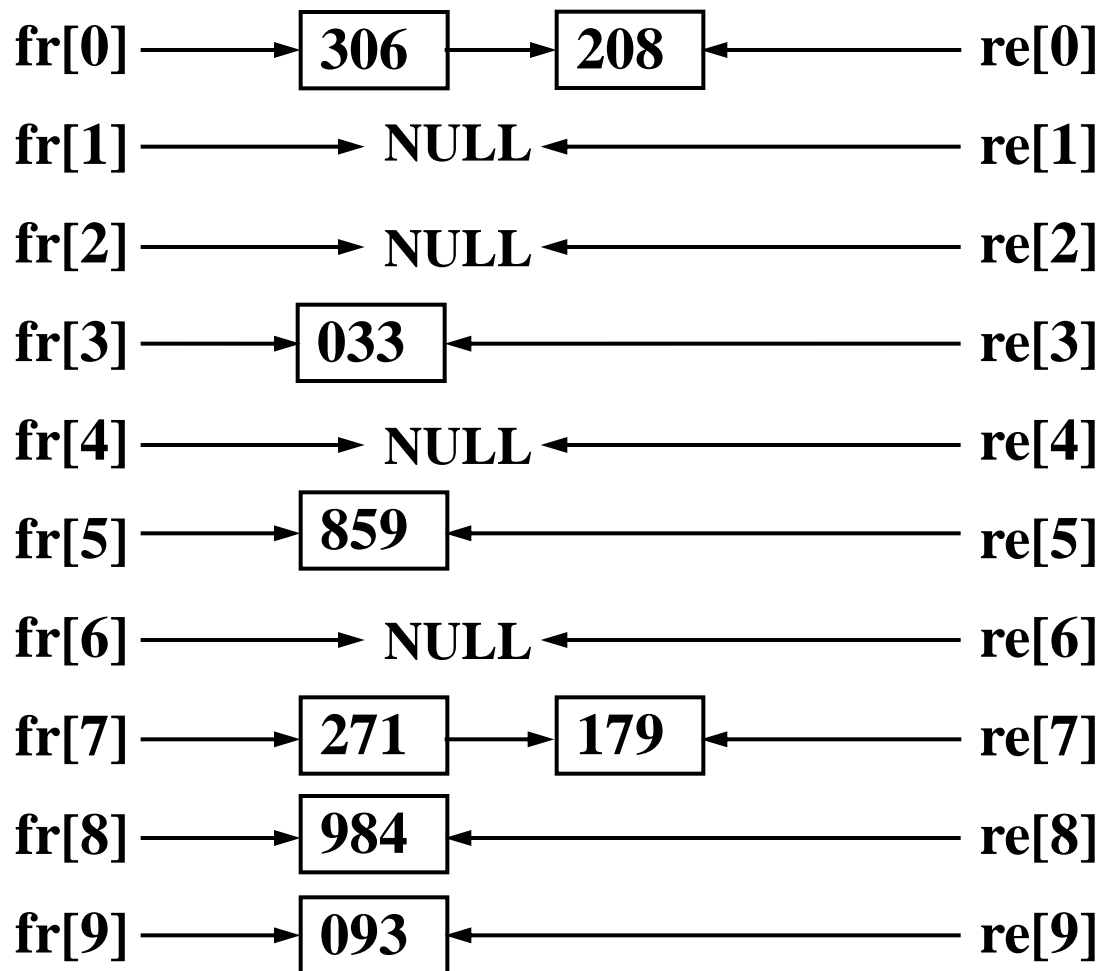
(a)初态



(b)第一趟分配之后



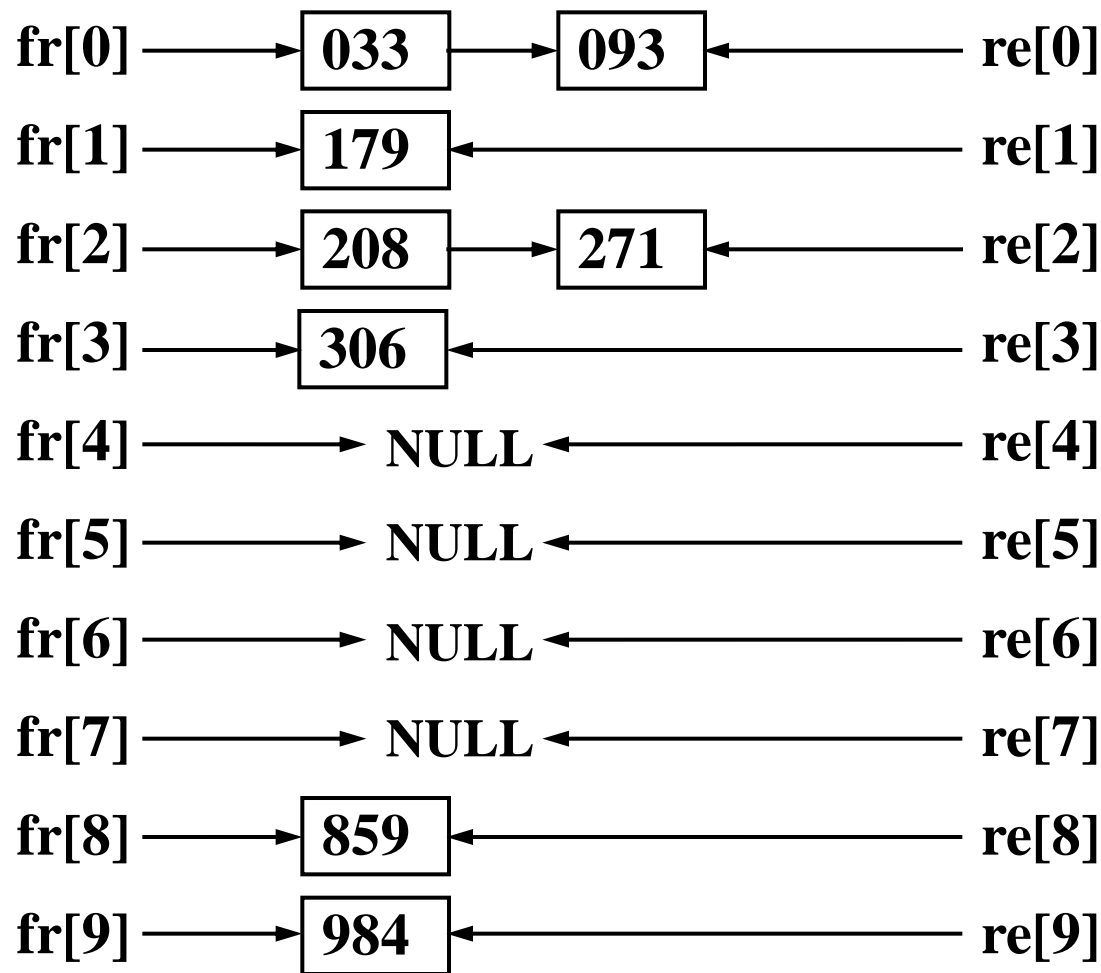
(c)第一趟收集之后



(d)第二趟分配之后



(e)第二趟收集之后



(f)第三趟分配之后





(g)第三趟收集之后，得有序序列