南京晓庄学院
NANJING XIAOZHUANG UNIVERSITY

数据结构与算法
习题册
（课后部分参考答案）

数学与信息技术学院
School of Mathematics and Information Technology
《数据结构与算法》课程组

目录

第一章 绪论

一. 填空题

1. 从逻辑关系上讲，数据结构的类型主要分为 集合 、线性结构、树结构和 图结构。
2. 数据的存储结构主要有 顺序存储和 链式存储 两种基本方法，不论哪种存储结构，都要存储两方面的内容：数据元素 和 数据元素之间的关系 。
3. 算法具有五个特性，分别是 有穷性 、 确定性、可行性、 输入 、 输出 。
4. 算法设计要求中的健壮性指的是 算法在发生非法操作时可以作出处理的特性。

二. 选择题

1. 顺序存储结构中数据元素之间的逻辑关系是由 C 表示的，链接存储结构中的数据元素之间的逻辑关系是由 D 表示的。

A 线性结构 B 非线性结构 C 存储位置 D 指针

2. 假设有如下遗产继承规则：丈夫和妻子可以相互继承遗产；子女可以继承父亲或母亲的遗产；子女间不能相互继承。则表示该遗产继承关系的最合适的数据结构应该是 B 。

A 树 B 图 C 线性表 D 集合

3. 算法指的是 A 。

A 对特定问题求解步骤的一种描述，是指令的有限序列。

B 计算机程序 C 解决问题的计算方法 D 数据处理

三. 简答题

1. 分析以下各程序段，并用大 O 记号表示其执行时间。

(1)

```
i=1;k=0;
While(i<n-1)
{
    k=k+10*i;
    i++;
}
```

(2)

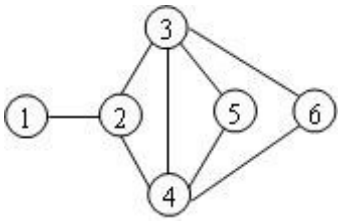
```
    i=1;k=0;
    do
    {
        k=k+10*i;
        i++;
    }while(i<=n)
```

⑴ 基本语句是 **k=k+10*i**，共执行了 **n-2** 次，所以 **T(n)=O(n)**。

⑵ 基本语句是 **k=k+10*i**，共执行了 **n** 次，所以 **T(n)=O(n)**。

2. 设有数据结构（D，R），其中 D={1, 2, 3, 4, 5, 6}，R={(1,2),(2,3),(2,4),(3,4),(3,5),(3,6),(4,5),(4,6)}。试画出其逻辑结构图并指出属于何种结构。

其逻辑结构图如下所示，它是一种图结构。



3. 求多项式 A(x)的算法可根据下列两个公式之一来设计：

⑴ $A(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$

⑵ $A(x)=(\ldots(a_nx+a_{n-1})x+\ldots+a_1)x)+a_0$

根据算法的时间复杂度分析比较这两种算法的优劣。

第二种算法的时间性能要好些。第一种算法需执行大量的乘法运算，而第二种算法进行了优化，减少了不必要的乘法运算。

第二章 线性表

一. 填空题

1. 在顺序表中，等概率情况下，插入和删除一个元素平均需移动 表长的一半 个元素，具体移动元素的个数与 表长 和 插入的位置 有关。

2. 在一个长度为 n 的顺序表的第 i（1≤i≤n+1）个元素之前插入一个元素，需向后移动 n-i+1 个元素，删除第 i（1≤i≤n）个元素时，需向前移动 n-i 个元素。

3. 在单循环链表中，由 rear 指向表尾，在表尾插入一个结点 s 的操作顺序是 s->next =rear->next; rear->next =s; rear =s;；删除开始结点的操作顺序为 q=rear->next->next; rear->next->next=q->next; delete q; 。

二. 选择题

1.数据在计算机存储器内表示时物理地址与逻辑地址相同并且是连续的，称之为： C

A 存储结构 B 逻辑结构 C 顺序存储结构 D 链式存储结构

2. 在 n 个结点的顺序表中，算法的时间复杂度是 O（1）的操作是： A

A 访问第 i 个结点（1≤i≤n）和求第 i 个结点的直接前驱（2≤i≤n）

B 在第 i 个结点后插入一个新结点（1≤i≤n）

C 删除第 i 个结点（1≤i≤n） D 将 n 个结点从小到大排序

3. 线性表 L 在 B 情况下适用于使用链式结构实现。

A 需经常修改 L 中的结点值 B 需不断对 L 进行删除插入

C L 中含有大量的结点 D L 中结点结构复杂

4. 单链表的存储密度 C

A 大于 1 B 等于 1 C 小于 1 D 不能确定

三. 判断题

1. 线性表的逻辑顺序和存储顺序总是一致的。 F

2. 线性表的顺序存储结构优于链接存储结构。 F

3. 设 p，q 是指针，若 p=q，则*p=*q。 F

4. 线性结构的基本特征是：每个元素有且仅有一个直接前驱和一个直接后继。 F

四. 简答题

1. 分析下列情况下，采用何种存储结构更好些。

(1)若线性表的总长度基本稳定，且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素。

(2)如果 n 个线性表同时并存，并且在处理过程中各表的长度会动态发生变化。

(3)描述一个城市的设计和规划。

⑴ 应选用顺序存储结构。很少进行插入和删除操作，所以空间变化不大，且需要快速存取，所以应选用顺序存储结构。

⑵ 应选用链式存储结构。链表容易实现表容量的扩充，适合表的长度动态发生变化。

⑶ 应选用链式存储结构。因为一个城市的设计和规划涉及活动很多，需要经常修改、扩充和删除各种信息，才能适应不断发展的需要。而顺序表的插入、删除的效率低，故不合适。

五. 算法设计

1. 已知数组 A[n]中的元素为整型，设计算法将其调整为左右两部分，左边所有元素为奇数，右边所有元素为偶数，并要求算法的时间复杂度为 O(n)。

数组奇偶调整算法 Adjust

```
void Adjust(int A[ ], n)
{
  i=0; j=n-1;
  while (i<j)
  {
    while (A[i] % 2!=0) i++;
    while (A[j] % 2==0) j--
    if (i<j) A[i]←→A[j];
  }
}
```

2. 线性表存放在整型数组 A[arrsize]的前 elenum 个单元中，且递增有序。编写算法，将元素 x 插入到线性表的适当位置上，以保持线性表的有序性，并且分析算法的时间复杂度。

```
int insert (datatype A[],int *elenum,datatype x)   /*设 elenum 为表的最大下标*/
{if (*elenum==arrsize-1)  return 0;    /*表已满，无法插入*/
else {i=*elenum;
      while (i>=0 && A[i]>x)     /*边找位置边移动*/
{A[i+1]=A[i];
i--;
}
      A[i+1]=x;       /*找到的位置是插入位的下一位*/
      (*elenum)++;
return 1;       /*插入成功*/
}
}
```

O（n）

第三章 栈和队列

一. 填空题

1. 设有一个空栈，栈顶指针为 1000H，现有输入序列为 1. 2. 3. 4. 5，经过 push，push，pop，push，pop，push，push 后，输出序列是 23 ，栈顶指针为 1003H 。

2. 栈通常采用的两种存储结构是 顺序栈 、 链栈 ；其判定栈空的条件分别是 TOP=-1 、 TOP=NULL ，判定栈满的条件分别是 TOP=数组长度 、存储空间满 。

3. 栈 可作为实现递归函数调用的一种数据结构。

4. 表达式 a*(b+c)-d 的后缀表达式是 abc+*d- 。

二. 选择题

1. 若一个栈的输入序列是 1，2，3，…，n，输出序列的第一个元素是 n，则第 i 个输出元素是 D 。

A 不确定 B n-i C n-i-1 D n-i+1

2. 设栈 S 和队列 Q 的初始状态为空，元素 e1. e2. e3. e4. e5. e6 依次通过栈 S，一个元素出栈后即进入队列 Q，若 6 个元素出队的顺序是 e2. e4. e3. e6. e5. e1，则栈 S 的容量至少应该是 C 。

A 6 B 4 C 3 D 2

3. 一个栈的入栈序列是 1，2，3，4，5，则栈的不可能的输出序列是 C 。

A 54321 B 45321 C 43512 D 12345

三. 判断题

1. 有 n 个元素依次进栈，则出栈序列有(n-1)/2 种。 F

2. 栈可以作为实现过程调用的一种数据结构。 T

3. 在栈满的情况下不能做进栈操作，否则将产生“上溢”。 T

4. 在循环队列中，front 指向队头元素的前一个位置，rear 指向队尾元素的位置，则队满的条件是 front=rear。 F

四. 简答题

1. 设有一个栈，元素进栈的次序为 A，B，C，D，E，能否得到如下出栈序列，若能，请写出操作序列，若不能，请说明原因。

⑴ C，E，A，B，D

⑵ C，B，A，D，E

⑴不能，因为在 C、E 出栈后，A 一定在栈中，而且在 B 的下面，不可能先于 B 出栈

⑵可以，设 I 为进栈操作，O 为入栈操作，则其操作序列为 IIIOOOIOIO。

2. 在操作序列 push(1). push(2). pop. push(5). push(7). pop. push(6)之后，栈顶元素和栈底元素分别是什么？（push(k)表示 k 入栈，pop 表示栈顶元素出栈。）

栈顶元素为 6，栈底元素为 1。

3. 在操作序列 EnQueue(1). EnQueue(3). DeQueue. EnQueue(5). EnQueue(7). DeQueue. EnQueue(9)之后，队头元素和队尾元素分别是什么？（EnQueue(k)表示整数 k 入队，DeQueue 表示队头元素出队）。

队头元素为 5，队尾元素为 9。

4. 简述以下算法的功能（栈的元素类型 SElemType 为 int）。

(1) status algo1(Stack S,int e)

```
{
    Stack T; int d; InitStack(T);
    while(!StackEmpty(S)){
        Pop(S,d);
        if(d!=e) Push(T,d);
    }
    while(!StackEmpty(T)){
        Pop(T,d);    Push(S,d);
    }
}//S 中如果存在 e，则删除它。
```

(2) void algo2(Queue &Q)

```
{
    Stack S;    int d;    InitStack(S);
    while(!QueueEmpty(Q))
    {
        DeQueue(Q, d); Push(S, d);
    }
    while(!StackEmpty(S))
    {
        Pop(S, d); EnQueue(Q, d);
    }
}//队列逆置
```

五. 算法设计

1. 试写一个判别表达式中开、闭括号是否配对出现的算法

```
BOOL BracketCorrespondency(char a[])
{
    int i=0;
    Stack s;
    InitStack(s);
    ElemType x;
    while(a[i]){
        switch(a[i]){
        case '(':
            Push(s,a[i]);
            break;
        case '[':
            Push(s,a[i]);
            break;
        case ')':
            GetTop(s,x);
            if(x=='(') Pop(s,x);
            else return FALSE;
            break;
        case ']':
            GetTop(s,x);
            if(x=='[') Pop(s,x);
            else return FALSE;
            break;
        default:
            break;
        }
        i++;
    }
    if(s.size!=0) return FALSE;
    return TRUE;
}
```

2. 假设称正读和反读都相同的字符序列为“回文”，例如，‘abba’和‘abcba’是回文，‘abcde’和‘ababab’则不是回文。试写一个算法判别读入的一个以‘@’为结束符的字符序列是否是“回文”。

```
Status SymmetryString(char* p)
{
    Queue q;
    if(!InitQueue(q)) return 0;
    Stack s;
    InitStack(s);
    ElemType e1,e2;
    while(*p){
        Push(s,*p);
        EnQueue(q,*p);
        p++;
    }
    while(!StackEmpty(s)){
        Pop(s,e1);
        DeQueue(q,e2);
        if(e1!=e2) return FALSE;
    }
    return OK;
}
```

第四章 串

一. 填空题

1. 不包含任何字符（长度为 0）的串 称为空串；由一个或多个空格（仅由空格符）组成的串 称为空白串。

2. 设 S="A;/document/Mary.doc"，则 strlen(s)= 20 , "/"的字符定位的位置为 3 。

3. 若 n 为主串长，m 为子串长，则串的经典模式匹配算法最坏的情况下需要比较字符的总次数为 (n-m+1)*m 。

二. 选择题

1. 串是一种特殊的线性表，其特殊性体现在： （ B ）

A 可以顺序存储 B 数据元素是一个字符

C 可以链式存储 D 数据元素可以是多个字符

2. 设有两个串 p 和 q，求 q 在 p 中首次出现的位置的运算称作：（ B ）

A 连接 B 模式匹配 C 求子串 D 求串长

3. 设串 s1='ABCDEFG'，s2='PQRST'，函数 con(x,y)返回 x 和 y 串的连接串，subs(s, i, j)返回串 s 的从序号 i 开始的 j 个字符组成的子串，len(s)返回串 s 的长度，则 con(subs(s1, 2, len(s2)), subs(s1, len(s2), 2))的结果串是：（ D ）

A 'BCDEF' B 'BCDEFG' C 'BCPQRST' D 'BCDEFEF'

4. 若串 S="software"，其子串的数目是（ B ）。

A 8 B 37 C 36 D 9

三. 计算题

1. 设 s='I AM A STUDENT', t='GOOD', q='WORKER', 求：

1）Replace(s,'STUDENT',q) 2）Concat(t,SubString(s,7,8)))

1、 Replace(s,'STUDENT',q)='I AM A WORKER'

2、因为 SubString(s,7,8)=' STUDENT' Concat(t,SubString(s,7,8))='GOOD STUDENT'

四. 算法设计

1. 利用 C 的库函数 strlen, strcpy（或 strncpy）写一个算法 void StrDelete(char *S,int t,int m) ，删除串 S 中从位置 i 开始的连续的 m 个字符。若 i≥strlen(S)，则没有字符被删除；若 i+m≥strlen(S)，则将 S 中从位置 i 开始直至末尾的字符均被删去。

提示：strlen 是求串长(length)函数：int strlen(char s); //求串的长度

strcpy 是串复制(copy)函数：char *strcpy(char to,char from); //该函数将串 from 复制到串 to 中，并且返回一个指向串 to 的开始处的指针。

```
void StrDelete(char *S, int i ,int m)
    { //串删除
      char Temp[Maxsize];//定义一个临时串
      if(i+m<strlen(S))
        {
          strcpy (Temp, &S[i+m]);//把删除的字符以后的字符保存到临时串中
          strcpy( &S[i],Temp);//用临时串中的字符覆盖位置 i 之后的字符
        }
      else if(i+m>=strlen(S)&& i<strlen(S))
        {
          strcpy(&S[i],"\0");//把位置 i 的元素置为'\0'，表示串结束
        }
    }
```

第五章 数组和广义表

一. 填空

1. 假设有二维数组 $A_{6\times8}$，每个元素用相邻的 6 个字节存储，存储器按字节编址。已知 A 的起始存储位置（基地址）为 1000，则数组 A 的体积（存储量）为 288 ；末尾元素 A_{57} 的第一个字节地址为 1282 ；若按行存储时，元素 A_{14} 的第一个字节地址为 1072；若按列存储时，元素 A_{47} 的第一个字节地址为 1276 。

2. 三元素组表中的每个结点对应于稀疏矩阵的一个非零元素，它包含有三个数据项，分别表示该元素的**行下标**、**列下标**和**元素值**。。

3. 广义表((a), (((b),c)),(d))的长度是 3 ，深度是 4 ，表头是 (a) ，表尾是(((b),c)),(d) 。

4. 已知广义表 LS=(a，(b，c，d)，e)，用 Head 和 Tail 函数取出 LS 中原子 b 的运算是 Head(Head(Tail(LS))) 。

二. 选择题

1. 假设有 60 行 70 列的二维数组 a[1…60, 1…70]以列序为主序顺序存储，其基地址为 10000，每个元素占 2 个存储单元，那么第 32 行第 58 列的元素 a[32,58]的存储地址为 (A)。（无第 0 行第 0 列元素）

A 16902 B 16904 C 14454 D 答案 A, B, C 均不对

2. 设矩阵 A 是一个对称矩阵，为了节省存储，将其下三角部分按行序存放在一维数组 B[1, n(n-1)/2]中，下三角部分中任一元素 $a_{i,j}(i\le j)$, 在一维数组 B 中下标 k 的值是：(B)

A i(i-1)/2+j-1 B i(i-1)/2+j C i(i+1)/2+j-1 Di(i+1)/2+j

3. 一个 n*n 的对称矩阵，用压缩存储方法处理其对称性质后存储，则其容量为：(C)。

A n*n B n*n/2 C (n+1)*n/2 D (n+1)*(n+1)/2

三. 计算题

1. 设有一个二维数组 A[m][n]，假设 A[0][0]存放位置在 644，A[2][2]存放位置在 676，每个元素占一个空间，问 A[3][3]存放在什么位置？写出计算过程。

(676-644)/1=32

A[2][2]的地址相对A[0][0]偏移量为32，则A[3][3]相较于A[0][0]的偏移量为32*3/2=48

A[3][3]地址为48+644=692

2. 设 A[9][9]是一个 10*10 对称矩阵，采用压缩存储方式存储其下三角部分，已知每个元素占用两个存储单元，其第一个元素 A[0][0]的存储位置在 1000，求下面问题的计算过程及结果：

给出 A[4][5]的存储位置。

A[4][5]是上三角部分，所以存在 A [5] [4]

若以行序为主序，则地址为 1000+2（5（1+5）/ 2+4）=1036

若以列序为主序，则地址为 1000+2（4（10+7）/ 2+5）=1062

给出存储位置为 1080 的元素下标。

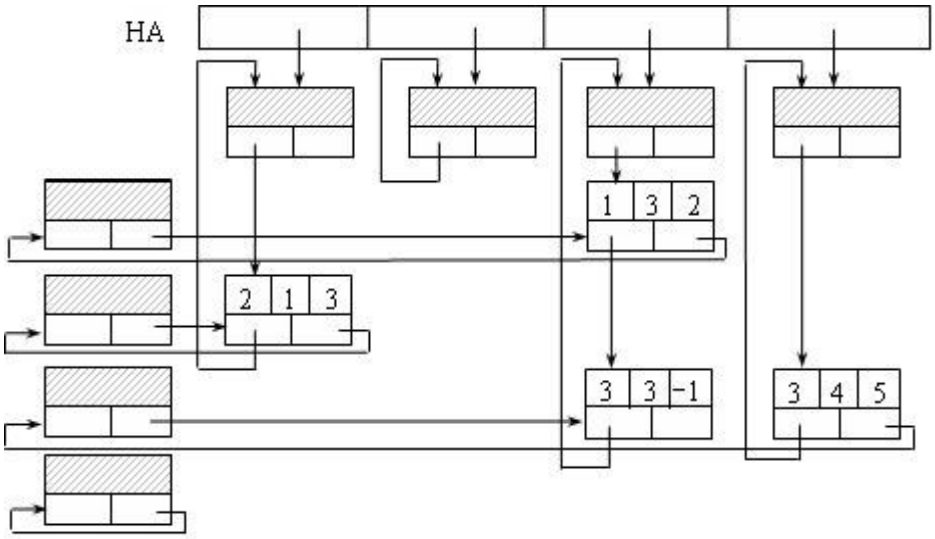
i(i+1)/2+j 或 j(10+10-j+1)/2+i=40

得 i=9，j=4 或 i=8 j=5，A [8] [5] 或 A [9] [4]

3. 一个稀疏矩阵如图所示，则对应的三元组线性表是什么？其对应的十字链表是什么？

$$\begin{pmatrix} 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

下标	行号	列号	非零元素
0	1	3	2
1	2	1	3
2	3	3	-1
3	3	4	5



4. 下列各三元组表分别表示一个稀疏矩阵，试写出它们的稀疏矩阵。

(1)
$$\begin{bmatrix} 6 & 4 & 6 \\ 1 & 2 & 2 \\ 2 & 1 & 12 \\ 3 & 1 & 3 \\ 4 & 4 & 4 \\ 5 & 3 & 6 \\ 6 & 1 & 16 \end{bmatrix}$$

$$\begin{pmatrix} 0 & 2 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 6 & 0 \\ 16 & 0 & 0 & 0 \end{pmatrix}$$

四. 算法设计

1. 设计一个算法，计算一个三元组表表示的稀疏矩阵的对角线元素之和。

```
Int diagonal (TSMatrix a,ElemType &sum)
{ int I;
Sum=0;
If(a.rows!=a.cols)
 { printf("不是对角矩阵\n");
  Return 0;
}
For(i=1;i<=a.nums;i++)
    If(a.data[i].r==a.data[i].c)
      Sum+=a.data[i].d;
Return 1;
}
```

2. 若在矩阵 A 中存在一个元素 ai,j（0≤i≤n-1，0≤j≤m-1），该元素是第 i 行元素中最小值且又是第 j 列元素中最大值，则称此元素为该矩阵的一个鞍点。假设以二维数组存储矩阵 A，试设计一个求该矩阵所有鞍点的算法，并分析最坏情况下的时间复杂度

```
void Saddle(int A[m][n])
  //A 是 m*n 的矩阵,本算法求矩阵 A 中的马鞍点.
 {int max[n]={0}, //max 数组存放各列最大值元素的行号,初始化为行号 0;
      min[m]={0}, //min 数组存放各行最小值元素的列号,初始化为列号 0;
     i, j;
  for(i=0;i<m;i++) //选各行最小值元素和各列最大值元素.
  {for(j=0;j<n;j++)
      {if(A[max[j]][j]<A[i][j]) max[j]=i;  //修改第 j 列最大元素的行号
       if(A[i][min[i]]>A[i][j]) min[i]=j;  //修改第 i 行最小元素的列号.
      }
   for (i=0;i<m;i++)
   {j=min[i];              //第 i 行最小元素的列号
    if(i==max[j])printf("A[%d][%d]是马鞍点，元素值是%d",i,j,A[i][j]);
    }
 }
}// Saddle
```

分析算法，外层 for 循环共执行 m 次，内层第一个 for 循环执行 n 次，第二个 for 循环最坏情况下执行 m 次，所以，最坏情况下的时间复杂度为 O(mn+mm)。

第六章 树和二叉树

一. 填空题

1. 树是 n（n≥0）结点的有限集合。在一棵空树中有 0 个元素；在一棵非空树中，有 且仅有一个 个根结点，其余的结点分成 m（m＞0）个 互不相交 的集合，每个集合都是根结点的子树。

2. 一棵二叉树的第 i（i≥1）层最多有 2i-1 个结点；一棵有 n（n>0）个结点的满二叉树共有 (n+1)/2 个叶子结点和 (n-1)/2 个非终端结点。

3. 设深度为 k 的二叉树上只有度为 0 和度为 2 的结点，该二叉树的结点数可能达到的最大值是 2^k-1 ，最小值是 2k-1 。

4. 深度为 k 的二叉树中，所含叶子的个数最多为 2k-1 。

5. 在顺序存储的二叉树中编号为 i 和 j 的两个结点处在同一层的条件为：$\lfloor \log_2 i \rfloor = \lfloor \log_2 j \rfloor$

二. 选择题

1. 前序遍历和中序遍历结果相同的二叉树是（ D ）。

A 根结点无左孩子的二叉树　　B 根结点无右孩子的二叉树

C 所有结点只有左子树的二叉树　D 所有结点只有右子树的二叉树

2. 序存储的方法将完全二叉树中的所有结点逐层存放在数组 A[1] ~ A[n]中，结点 A[i] 若有左子树，则左子树的根结点是（ D ）。

A　A[2i-1]　　B　A[2i+1]　　C　A[i/2]　　D　A[2i]

3. 完全二叉树中的任一结点，若其右分支下的子孙的最大层次为 h，则其左分支下的子孙的最大层次为（　C ）。

A　h　　B　h+1　　C　h 或 h+1　　D　任意

4. 在线索二叉树中，一个结点是叶子结点的充要条件为（　C ）。

A 左线索标志为 0，右线索标志为 1 B 左线索标志为 1，右线索标志为 0

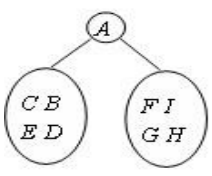
C 左. 右线索标志均为 0 D 左. 右线索标志均为 1

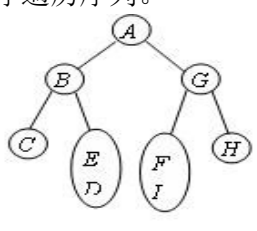
5. 由权值为{3, 8, 6, 2, 5}的叶子结点生成一棵哈夫曼树，其带权路径长度为（　C　）。

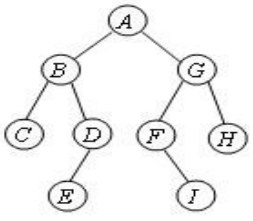
A　24　　B　48　　C　53　　D 72

三. 简答题

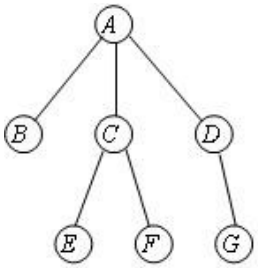
1. 已知二叉树的中序和后序序列分别为 CBEDAFIGH 和 CEDBIFHGA，请构造出此二叉树，并写出此树的后序遍历序列。

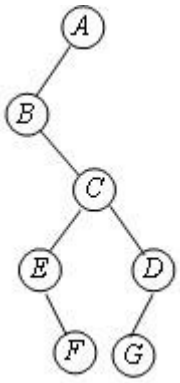




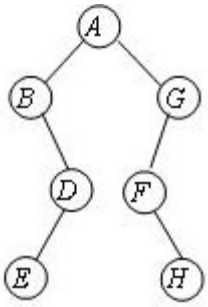


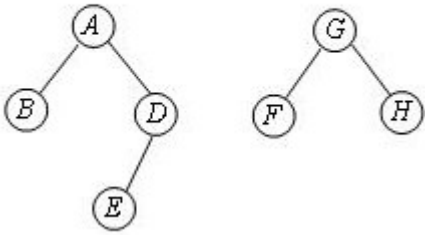
2. 将下图所示的树转换为二叉树，





3. 图 5-17 所示的二叉树转换为树或森林

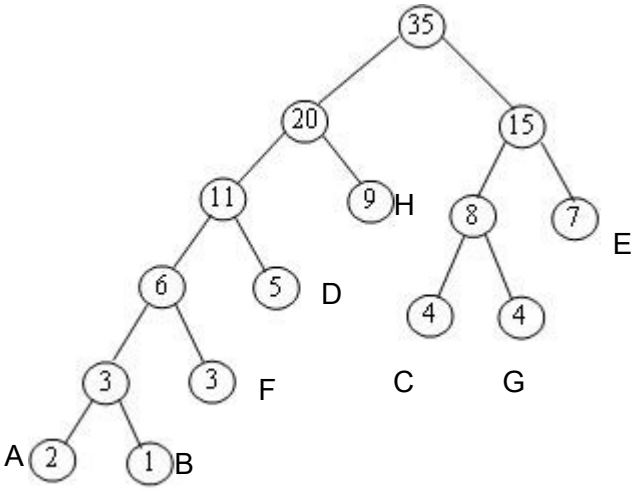




4. 已知某字符串 S 中共有 8 种字符[A,B,C,D,E,F,G,H]，分别出现 2 次. 1 次. 4 次. 5 次. 7 次. 3 次. 4 次和 9 次。

1）试构造出哈夫曼编码树，并对每个字符进行编码

2）问该字符串的编码至少有多少位。



A:00000 B:00001 C:100 D:001 E:11 F:0001 G:101 H:01

其带权路径长度=2×5+1×5+3×4+5×3+9×2+4×3+4×3+7×2=98，所以，该字符串的编码长度至少为 98 位。

四. 算法设计

1. 设计算法求二叉树的结点个数

求二叉树结点个数算法 Count

```
void Count(BiNode *root)    //n 为全局量并已初始化为 0
{
  if (root) {
          Count(root->lchild);
          n++;
          Count(root->rchild);
  }
}
```

2. 以二叉链表为存储结构，编写算法求二叉树中结点 x 的双亲

查找某结点的双亲算法 Parent

```
BiNode *Parent(BiNode *root, T x)   //p 是全局量，初值为空
{
  if (root) {
          if (root->data==x) return p;
            else {
                 p=root;
                 Parent(root->lchild, x);
                 Parent(root->rchild, x);
            }
  }
}
```

3. 编写算法交换二叉树中所有结点的左右子树。

交换左右子树算法 Exchange

```
void Exchange(BiNode *root)
{
  if (root) {
          Exchange(root->lchild);
          Exchange(root->rchild);
          root->lchild←→root->rchild;   //交换左右子树
  }
}
```

第七章 图

一. 填空题

1. 设无向图 G 中顶点数为 n，则图 G 至少有 0 条边，至多有 n(n-1)/2 条边；若 G 为有向图，则至少有 0 条边，至多有 n(n-1) 条边。

2. 任何连通图的连通分量只有一个，即是 它自身

3. 若一个有向图由邻接矩阵表示，则计算第 j 个顶点入度的方法是 求矩阵第 j 列元素之和

4. 图的深度优先遍历类似于树的 先根 遍历，广度优先遍历类似于树的 层序 遍历。

5. 对于含有 n 个顶点 e 条边的连通图，利用 Prim 算法求最小生成树的时间复杂度为 O(n^2)，利用 Kruskal 算法求最小生成树的时间复杂度为 O(elog2e) 。

二. 选择题

1. 在一个无向图中，所有顶点的度数之和等于所有边数的（ C ）倍。

A 1/2 B 1 C 2 D 4

2. n 个顶点的强连通图至少有（ A ）条边。

A n B n+1 C n-1 D n×(n-1)

3. 含 n 个顶点的连通图中的任意一条简单路径，其长度不可能超过（ C ）。

A 1 B n/2 C n-1 D n

4. 对于一个具有 n 个顶点的无向图用邻接矩阵存储，则该矩阵的大小是（ D ）。

A n B $(n-1)^2$ C n-1 D n^2

5. 设无向图 G=(V, E)和 G' =(V', E')，如果 G' 是 G 的生成树，则下面的说法中错误的是（ B ）。

A G' 为 G 的子图 B G' 为 G 的连通分量

C G' 为 G 的极小连通子图且 V= V' D G' 是 G 的一个无环子图

6. 判定一个有向图是否存在回路除了可以利用拓扑排序方法外，还可以用（ D ）。

A 求关键路径的方法 B 求最短路径的方法

C 广度优先遍历算法 D 深度优先遍历算法

7. 下面关于工程计划的 AOE 网的叙述中，不正确的是（ B ）

A 关键活动不按期完成就会影响整个工程的完成时间

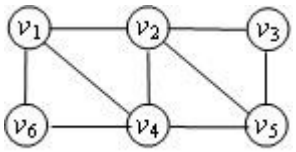
B 任何一个关键活动提前完成，那么整个工程将会提前完成

C 所有的关键活动都提前完成，那么整个工程将会提前完成

D 某些关键活动若提前完成，那么整个工程将会提前完

三. 计算题

1. 已知一个连通图如图所示，试给出图的邻接矩阵和邻接表



存储示意图，若从顶点 v1 出发对该图进行遍历，分别给出一个按深度优先遍历和广度优先遍历的顶点序列。

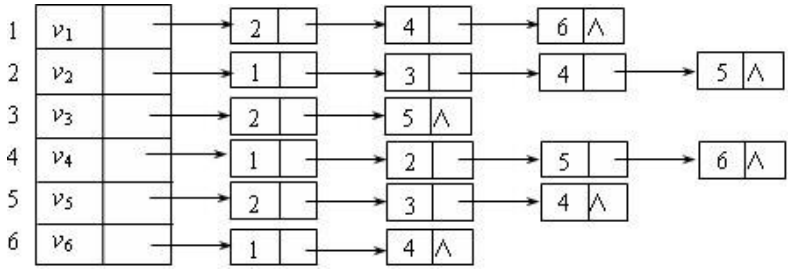
邻接矩阵表示如下：

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

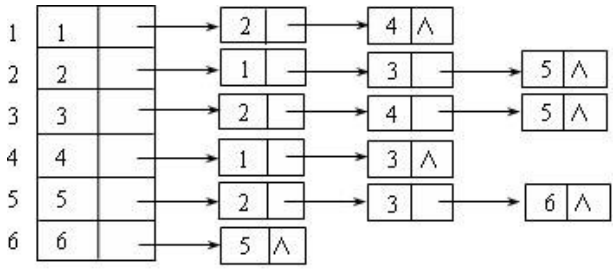
深度优先遍历序列为：v1 v2 v3 v5 v4 v6

广度优先遍历序列为：v1 v2 v4 v6 v3 v5

邻接表表示如右：

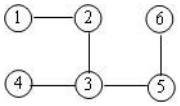


2. 已知无向图 G 的邻接表如下图所示，分别写出从顶点 1 出发的深度遍历和广度遍历序列，并画出相应的生成树。



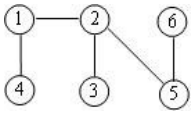
深度优先遍历序列为：1，2，3，4，5，6

对应的生成树为：

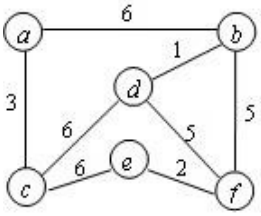


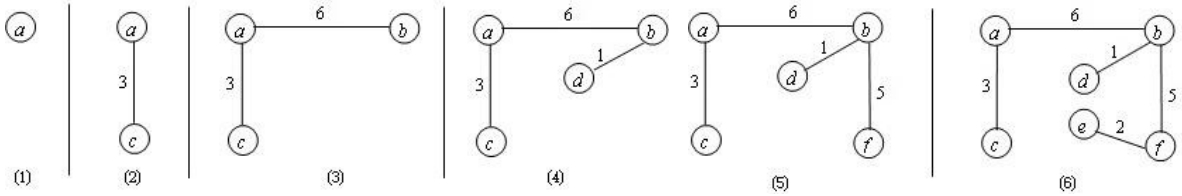
广度优先遍历序列为：1，2，4，3，5，6

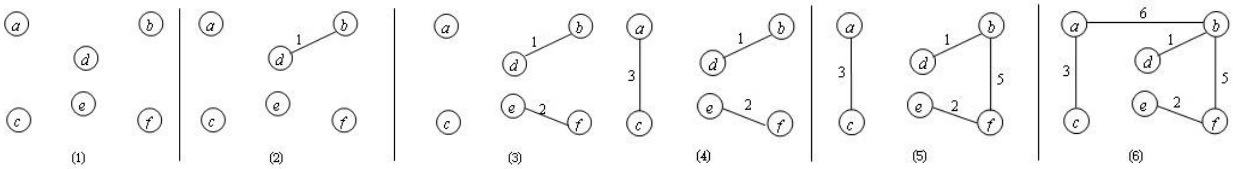
对应的生成树为：



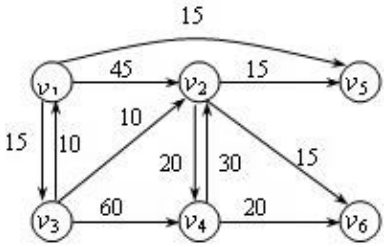
3. 下图所示是一个无向带权图，请分别按 Prim 算法和 Kruskal 算法求最小生成树。







4. 如右图所示的有向网图，利用 Dijkstra 算法求从顶点 v1 到其他各顶点的最短路径。



从源点 v1 到其他各顶点的最短路径如下表所示。

源点 终点 最短路径 最短路径长度

v1 v3 v1 v3 15

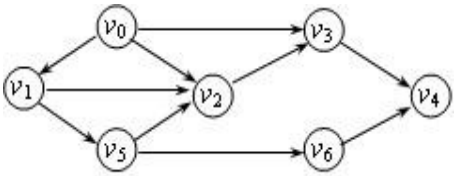
v1 v5 v1 v5 15

v1 v2 v1 v3 v2 25

v1 v6 v1 v3 v2 v6 40

v1 v4 v1 v3 v2 v4 45

5. 已知一个 AOV 网如下图所示，写出所有拓扑序列。



拓扑序列为：v0 v1 v5 v2 v3 v6 v4、 v0 v1 v5 v2 v6 v3 v4、 v0 v1 v5 v6 v2 v3 v4。

四. 算法设计

1. 设计算法，将一个无向图的邻接表转换成邻接矩阵。

邻接表转为邻接矩阵算法 ListToMat

```
void ListToMat(AdjMatrix &A, AdjList &B)
{
    A.vertexNum=B.vertexNum;
    A.arcNum=B.arcNum;
    for (i=0; i<A.vertexNum; i++)
        for (j=0; j<A.vertexNum; j++)
            A.arc[i][j]=0;
     for (i=0; i<A.vertexNum; i++)
     {
         p=B.adjlist[i].firstedge;
         while (p)
         {
             j= p->adjvex;
             a[i][j]=1;
             p=p->next;
         }
     }
}
```

2. 设计算法，计算图中出度为零的顶点个数。

统计出度为 0 的算法 SumZero

```
int SumZero(AdjMatrix A)
{
    count=0;
    for (i=0; i<A.vertexNum; i++)
    {
        tag=0;
        for (j=0; j<A.vertexNum; j++)
            if (arcs[i][j] !=0) {
                                tag=1;
                                break;
            }
        if (tag==0) count++;
    }
    return count;
}
```

3. 已知一个有向图的邻接表，编写算法建立其逆邻接表

建立逆邻接表算法 List

```
void List(AdjList A, AdjList &B)
{
    B.vertexNum= A.vertexNum;
    B.arcNum= A.arcNum;
    for (i=0; i<A.vertexNum; i++)
        B.adjlist[i].firstedge=NULL;
    for (i=0; i<A.vertexNum; i++)
    {
        p1=A.adjlist[i].firstedge;
        while (p1)
        {
            j=p1->adjvex;
            p2=new ArcNode;
            p2->adjvex=i;
            p2->next=B.adjlist[j].firstedge;
            B.adjlist[j].firstedge=p2;
            p1=p1->next;
        }
    }
}
```

第九章 查找

一. 填空题

1. 顺序查找技术适合于存储结构为 顺序和链式存储 的线性表，而折半查找技术适用于存储结构为 顺序存储 的线性表，并且表中的元素必须是 按关键码有序 。

2. 折半查找有序表（4，6，12，20，28，38，50，70，88，100），若查找表中元素20，它将依次与表中元素 28，6，12，20 比较大小。

3. 在各种查找方法中，平均查找长度与结点个数 n 无关的查找方法是 散列（哈希）查找 。

4. 为了能有效地应用哈希查找技术，必须解决的两个问题是 构造散列函数 和 解决冲突。

5. 有一个表长为 m 的散列表，初始状态为空，现将 n（n<m）个不同的关键码插入到散列表中，解决冲突的方法是用线性探测法。如果这 n 个关键码的散列地址都相同，则探测的总次数是 n(n-1)/2=（ 1＋2＋…＋n-1） 。

二. 选择题

1. 静态查找与动态查找的根本区别在于（ B ）。

A 它们的逻辑结构不一样　　B 施加在其上的操作不同

C 所包含的数据元素的类型不一样　　D 存储实现不一样

2. 用 n 个键值构造一棵二叉排序树，其最低高度为（ D ）。

A n/2　　B n　　C log2n　　D log2n+1

3. 散列技术中的冲突指的是（ D ）。

A 两个元素具有相同的序号 B 两个元素的键值不同，而其他属性相同

C 数据元素过多 D 不同键值的元素对应于相同的存储地址

4. 链表适用于 A 查找

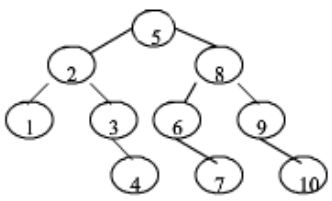
A 顺序　　B 二分法　　C 顺序，也能二分法　　D 随机

三. 简答题

1. 分别画出在线性表（a，b，c，d，e，f，g）中进行折半查找关键码 e 和 g 的过程。

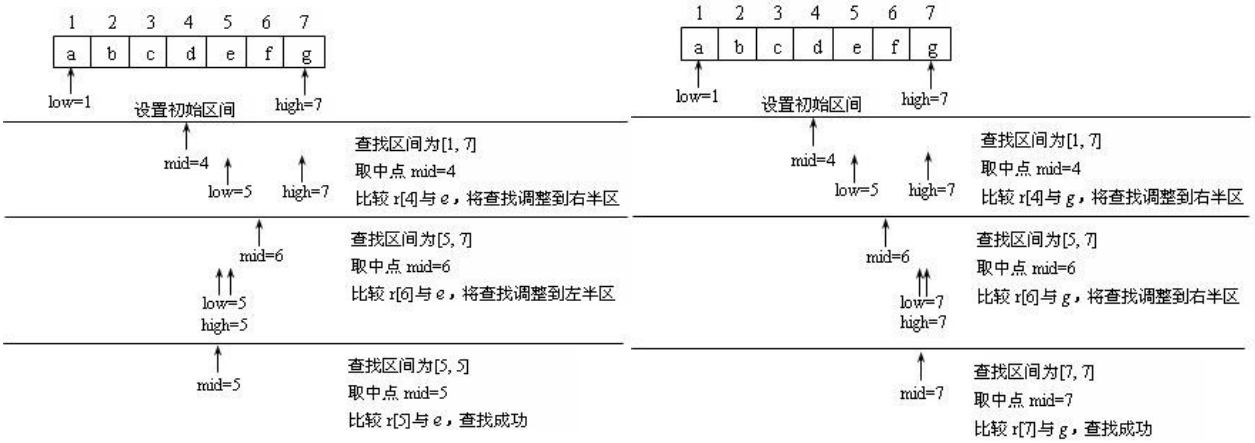
见下页

2. 画出对长度为 10 的有序表进行折半查找的判定树，并求其等概率时查找成功的平均查找长度。



ASL＝1/10（1＋2×2＋3×4＋4×3）
＝1/10（1＋4＋12＋12）
＝29/10=2.9

第 1 题答案



3. 设哈希（Hash）表的地址范围为 0～17，哈希函数为：H（K）＝K　MOD　16。K 为关键字，用线性探测法再散列法处理冲突，输入关键字序列：

（10，24，32，17，31，30，46，47，40，63，49）

构造出 Hash 表，试回答下列问题：

(1)画出哈希表的示意图；

(2)若分别查找关键字 63 和 60，分别需要依次与哪些关键字进行比较？

(3)假定每个关键字的查找概率相等，求查找成功时的平均查找长度。

解：　（1）画表如下：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
32	17	63	49					24	40	10				30	31	46	47

（2） 查找 63,首先要与 H(63)=63%16=15 号单元内容比较，即 63 vs 31 ,no;然后顺移，与 46,47,32,17,63 相比，一共比较了 6 次！

（3）查找 60,首先要与 H(60)=60%16=12 号单元内容比较，但因为 12 号单元为空（应当有空标记），所以应当只比较这一次即可。

（4） 对于黑色数据元素，各比较 1 次；共 6 次；对红色元素则各不相同，要统计移位的位数。“63”需要 6 次，“49”需要 3 次，“40”需要 2 次，“46”需要 3 次，“47”需要 3 次，所以 ASL=1/11（6＋2＋3×3）＝17/11=1.5454545454≈1.55

四. 算法设计

1. 编写算法求给定结点在二叉排序树中所在的层数

```
int Level (BiNode *root, BiNode *p)  //BiNode 请参见二查排序树的结点结构
{
   if (!p) return 0;
   if (p==root) return 1;
       else if (p->data<root->data) return Level(root->lchild, p)+1;
               else return Level (root->rchild, p)+1;
}
```

2. 一个判别给定二叉树是否为二叉排序树的算法，设此二叉树以二叉链表作存储结构。且树中结点的关键字均不同。

解：注意仔细研究二叉排序树的定义。易犯的典型错误是按下述思路进行判别：“若一棵非空的二叉树其左、右子树均为二叉排序树，且左子树的根的值小于根结点的值，又根结点的值不大于右子树的根的值，则是二叉排序树”

（刘注：即不能只判断左右孩子的情况，还要判断左右孩子与双亲甚至根结点的比值也要遵循（左小右大）原则）。

若要采用递归算法，建议您采用如下的函数首部：
bool BisortTree(BiTree T, BiTree&PRE)，其中 PRE 为指向当前访问结点的前驱的指针。
（或者直接存储前驱的数值，随时与当前根结点比较）
一个漂亮的算法设计如下：

```
int last=0, flag=1;              //  last 是全局变量，用来记录前驱结点值，只要每个结点都比前驱大就行。
int Is_BSTree(Bitree T)          //判断二叉树 T 是否二叉排序树，是则返回 1，否则返回 0
{
  if(T->lchild&&flag) Is_BSTree(T->lchild);
  if(T->data<last) flag=0;       //与其中序前驱相比较, flag=0 表示当前结点比直接前驱小，则立即返回
  last=T->data;
  if(T->rchild&&flag) Is_BSTree(T->rchild);
  return flag;
}//Is_BSTree
```

第十章 排序

一. 填空题

1. 排序的方法有很多种，插入排序法从未排序序列中依次取出元素，与已排序序列中的元素作比较，将其放入已排序序列的正确位置上。选择排序法从未排序序列中挑选元素，并将其依次放入已排序序列的一端。交换排序是对序列中元素进行一系列比较，当被比较的两元素为逆序时，进行交换；冒泡排序和快速排序是基于这类方法的两种排序方法，而快速排序是比冒泡排序效率更高的方法；堆排序法是基于选择排序的一种方法，是完全二叉树结构的一个重要应用。

2. 一组记录（54, 38, 96, 23, 15, 72, 60, 45, 83）进行直接插入排序，当把第 7 个记录 60 插入到有序表时，为寻找插入位置需比较 3 次。

3. 对 n 个待排序记录序列进行快速排序，所需要的最好时间是 $O(nlog2n)$，最坏时间是 $O(n^2)$

二. 选择题

1. 下列序列中，（A）是执行第一趟快速排序的结果。

   A [da，ax，eb，de，bb] ff [ha，gc]　　B [cd，eb，ax，da] ff [ha，gc，bb]

   C [gc，ax，eb，cd，bb] ff [da，ha]　　D [ax，bb，cd，da] ff [eb，gc，ha]

2. 堆的形状是一棵（C）。

   A 二叉排序树 B 满二叉树 C 完全二叉树 D 判定树

3. 希望用最快的速度从 5000 个元素挑选出前 10 个最大的，采用（ B ）方法最好。

   A 快速排序 B 堆排序 C 希尔排序 D 归并排序

4. 设要将序列（Q，H，C，Y，P，A，M，S，R，D，F，X）中的关键码按升序排列，则（ D ）是起泡排序一趟扫描的结果，（ B ）是增量为 4 的希尔排序一趟扫描的结果，（ E ）二路归并排序一趟扫描的结果，（ A ）是以第一个元素为轴值的快速排序一趟扫描的结果，（ C ）是堆排序初始建堆的结果。

   A（F，H，C，D，P，A，M，Q，R，S，Y，X）

   B（P，A，C，S，Q，D，F，X，R，H，M，Y）

   C（A，D，C，R，F，Q，M，S，Y，P，H，X）

   D（H，C，Q，P，A，M，S，R，D，F，X，Y）

   E（H，Q，C，Y，A，P，M，S，D，R，F，X）

5. 下述几种排序方法中，要求内存最小的是（AD ）

   A 插入排序　　B 快速排序　　C 堆排序　　D 选择排序

三. 简答题

1. 已知数据序列为(12，5，9，20，6，31，24)，对该数据序列进行排序，写出插入排

序. 起泡排序. 快速排序. 简单选择排序以及二路归并排序每趟的结果。

插入排序：

初始键值序列	[12]	5	9	20	6	31	24
第一趟结果	[5	12]	9	20	8	31	24
第二趟结果	[5	9	12]	20	6	31	24
第三趟结果	[5	9	12	20]	6	31	24
第四趟结果	[5	6	9	12	20]	31	24
第五趟结果	[5	6	9	12	20	31]	24
第六趟结果	[5	6	9	12	20	24	31]

简单选择排序：

初始键值序列	[12	5	9	20	6	31	24]
第一趟结果	5	[12	9	20	8	31	24]
第二趟结果	5	8	[9	20	12	31	24]
第三趟结果	5	8	9	[20	12	31	24]
第四趟结果	5	8	9	12	[20	31	24]
第五趟结果	5	8	9	12	20	[31	24]
第六趟结果	5	8	9	12	20	24	31

快速排序：

初始键值序列	12	5	9	20	6	31	24
第一趟结果	[6	5	9]	12	[20	31	24]
第二趟结果	[5]	6	[9]	12	20	[31	24]
第三趟结果	5	6	9	12	20	[24]	31
第四趟结果	5	6	9	12	20	24	31

起泡排序：

初始键值序列	[12	5	9	20	6	31	24]
第一趟结果	[5	9	12	6	20	24]	31
第二趟结果	[5	9	6]	12	20	24	31
第三趟结果	[5	6]	9	12	20	24	31
第四趟结果	5	6	9	12	20	24	31

堆排序：

初始键值序列	[12	5	9	20	6	31	24]
初始建堆结果	[31	20	24	5	6	9	12]
第一趟结果	[24	20	12	5	6	9]	31
第二趟结果	[20	9	12	5	6]	24	31
第三趟结果	[12	9	6	5]	20	24	31
第四趟结果	[9	5	6]	12	20	24	31
第五趟结果	[6	5]	9	12	20	24	31
第六趟结果	5	6	9	12	20	24	31

二路归并排序：

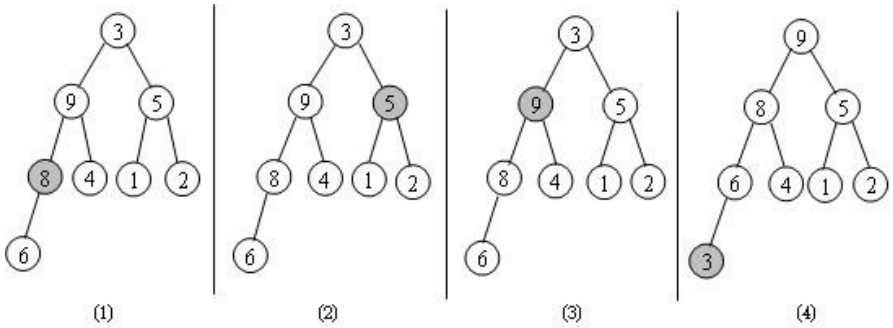
初始键值序列	12	5	9	20	6	31	24
第一趟结果	[5	12]	[9	20]	[6	31]	[24]
第二趟结果	[5	9	12	20]	[6	24	31]
第三趟结果	[5	6	9	12	20	24	31]

2. 判别下列序列是否为堆，如不是，按照堆排序思想把它调整为堆，用图表示建堆的过程。

⑴（1，5，7，25，21，8，8，42）

⑵（3，9，5，8，4，17，21，6）

【解答】序列⑴是堆，序列⑵不是堆，调整为堆（假设为大根堆）的过程如下图所示。



四. 算法设计

1. 直接插入排序中寻找插入位置的操作可以通过折半查找来实现。据此写一个改进的插入排序的算法。

```
#region 折半插入排序算法
        void HalfInsertSort(int[] array)
        {
            for(int i=2;i<array.Length;++i)
            {
                array[0] = array[i]; //或者也可以添加 if (array[i] < array[i - 1])先行判断
                int low = 1;
                int high = i - 1;
                while(low <= high)
                {
                    int mid = (low + high) / 2;
                    if (array[0] < array[mid]) high = mid - 1;
                    else low = mid + 1;
                }
                int j=0;
                for (j = i - 1; j >= high + 1; --j)
                {
                    array[j + 1] = array[j];
                }
                array[j + 1] = array[0];
            }
        }
            #endregion
```

2. 已知（$k_1, k_2, \ldots, k_n$）是堆，试写一算法将（$k_1, k_2, \ldots, k_n, k_{n+1}$）调整为堆

正负整数分开算法 Devot

```
void Devot(int r[ ], int n)
{
  i=1; j=n;
  while (i<j)
  {
     while (r[j]>0 && i<j) j--;
     while (r[i]<0 && i<j) i++;
     if (i<j) {
              r[i]←→r[j];    //交换元素
              i++;
              j--;
     }
  }
}
```