第 2 部分 习题解析

第 1 章　绪论

1.1　选择题

1. 算法的时间复杂度取决于（　C　）

A）问题的规模　　B） 待处理数据的初态　　C） A 和 B

【答案】C

2.计算机算法指的是解决问题的步骤序列，它必须具备（B　　） 这三个特性。

A）可执行性、可移植性、可扩充性　　B） 可执行性、确定性、有穷性

C） 确定性、有穷性、稳定性　　D） 易读性、稳定性、安全性

【答案】B

5．从逻辑上可以把数据结构分为（　C　）两大类。

A）动态结构、静态结构　　B）顺序结构、链式结构

C）线性结构、非线性结构　　D）初等结构、构造型结构

【答案】C

6．在下面的程序段中，对 x 的赋值的语句频度为（　C　）

```
for(i=0;i<n;i++)
    for(j=0;j<n;j++)  x=x+1;
```

A） O(2n)　　B）O(n)　C．$O(n^2)$　D．$O(\log_2 n)$

【答案】C

7．下面的程序段中， n 为正整数，则最后一行的语句频度在最坏情况下是（D　　）

```
for(i=n-1;i>=1;i--)
    for(j=1;j<=i;j++)
      if (A[j]>A[j+1])
          A[j]与 A[j+1]对换;
```

A. O（n）　　B） $O(n\log_2 n)$　　C） $O(n^3)$　　D） $O(n^2)$

【答案】D

1.2　填空题

2. 对于给定的 n 个元素,可以构造出的逻辑结构有_____，_____，_____，_____四种。

【答案】（1）集合　（2）线性结构　（3）树形结构（4）图状结构或网状结构

4．数据结构中评价算法的两个重要指标是_____。

【答案】算法的时间复杂度和空间复杂度。

5. 数据结构是研讨数据的_____和_____，以及它们之间的相互关系，并对与这种结构定义相应的_____，设计出相应的_____。

【答案】（1）逻辑结构（2）物理结构（3）操作（运算）（4）算法。

6．一个算法具有 5 个特性：_____、_____、_____，有零个或多个输入、有一个或多个输出。

【答案】（1）有穷性（2）确定性　　（3）可行性。

9．已知如下程序段

```
for(i=n;i>0;i--)                    {语句 1}
{  x=x+1;                           {语句 2}
   for(j=n;j>=i;j--)                {语句 3}
     y=y+1;                    {语句 4}
}
```

语句 1 执行的频度为_____；语句 2 执行的频度为_____；语句 3 执行的频度为_____；语句 4 执行的频度为_____。

【答案】（1）n+1　（2）n　（3）n(n+3)/2　（4）n(n+1)/2

10．在下面的程序段中，对 x 的赋值语句的频度为_____（表示为 n 的函数）

```
for(i=0;i>n;i++)
   for(j=0;j>i;j++)
       for(k=0;k>j;k++)
           x = x +delta;
```

【答案】1+（1+2）+（1+2+3）+…+（1+2+…+n）=n(n+1)(n+2)/6 ， O(n^3)

11.下面程序段中带下划线的语句的执行次数的数量级是_____。

```
i=1; while(i<n) i=i*2;
```

【答案】$\log_2 n$

12. 计算机执行下面的语句时，语句 s 的执行次数为_____。

```
for(i=l;i<n-l;i++)
  for(j=n;j>=i;j--)  s;
```

【答案】(n+3)(n-2)/2

13. 下面程序段的时间复杂度为_____。(n>1)

```
sum=1;
for (i=0;sum<n;i++) sum+=1;
```

【答案】O(n)

第 2 章 线性表

2.1 选择题

1．对于线性表最常用的操作是查找指定序号的元素和在末尾插入元素，则选择（ ）最节省时间

A）顺序表　　B）带头结点的双循环链表

C）单链表　　D）带尾结点的单循环链表

【答案】A

2．若长度为 n 的线性表采用顺序存储结构，在其第 i 个位置插入一个新元素的算法时间复杂度为（ ）(1≤i≤n+1)。

A） O(0)　　B） O(1)　　C） O(n)　　D） O(n^2)

【答案】C

3．双向链表中有两个指针域，prior 和 next，分别指向前驱及后继，设 p 指向链表中的一个结点，q 指向一待插入结点，现要求在 p 前插入 q，则正确的插入为（ ）

A） p->prior=q; q->next=p; p->prior->next=q; q->prior=p->prior;

B） q->prior=p->prior; p->prior->next=q; q->next=p; p->prior=q->next;

C） q->next=p; p->next=q; p->prior->next=q; q->next=p;

D） p->prior->next=q; q->next=p; q->prior=p->prior; p->prior=q;

【答案】D

4．在一个具有 n 个结点的有序单链表中插入一个新结点并仍然保持有序的时间复杂度是（ ）

A）O($n\log_2 n$）　　B） O(1）　　C） O(n）　　D） O(n^2）

【答案】C

5． 在一个以 h 为头指针的单循环链中，p 指针指向链尾结点的条件是（ ）

A）p->next==NULL　　B） p->next==h

C）p->next->next==h　　D） p->data==-1

【答案】B

6．对于一个具有 n 个结点的线性表，建立其单链表的时间复杂度是（ ）

A）O(n)　　B） O(1)　　C）O($n\log_2 n$)　　D） O(n^2)

【答案】A

8．在双向链表存储结构中，删除 p 所指的结点时须修改指针（ ）

A）p->prior->next=p->next　　p->next->prior=p->prior;

B）p->prior=p->prior->prior　　p->prior->next=p;

C）p->next->prior=p　　p->next=p->next->next

D）p->next=p->prior->prior　　p->prior=p->next->next;

【答案】A

9．线性表采用链式存储时，其元素地址（ ）

A）必须是连续的　　B）一定是不连续的

C）部分地址是连续的　　D）连续与否均可

【答案】D

2.2 填空题

1．线性表 L=（a_1，a_2，…，a_n）用数组表示，假定删除表中任一元素的概率相同，则删除一个元素

平均需要移动元素的个数是_____。

【答案】（n-1）/2

2．在单链表中设置头结点的作用是_____。

【答案】主要是使插入和删除等操作统一，在第一个元素之前插入元素和删除第一个结点不必另作判断。另外，不论链表是否为空，链表头指针不变。

3．线性表的顺序存储是通过 _____来反应元素之间的逻辑关系，而链式存储结构是通过_____来反应元素之间的逻辑关系。

【答案】（1）数据元素的前后顺序　（2）元素中的指针

4．当对一个线性表经常进行的是存取操作，而很少进行插入和删除操作时，则采用_____存储结构最节省时间，相反当经常进行插入和删除操作时，则采用_____存储结构最节省时间。

【答案】（1）顺序　（2）链式

5．对于 一个 具有 n 个结点 的单链 表， 在已 知的 结点 *p 后插入一个 新结 点的 时间 复杂度为_____，在给定值为 x 的结点后插入一个新结点的时间复杂度为_____。

【答案】（1）O(1)　（2）O(n)

7. 对于双向链表，在两个结点之间插入一个新结点需修改的指针共 _____个，单链表为_____个。

【答案】（1）4　（2）2

8. 循环单链表的最大优点是_____。

【答案】从任一结点出发都可访问到链表中每一个元素。

9．若要在一个不带头结点的单链表的首结点*p 结点之前插入一个*s 结点时，可执行下列操作：

```
s->next=_____;
p->next=s;
t=p->data;
p->data=_____;
s->data=_____;
```

【答案】（1）p->next　（2）s->data　（3） t

10．某线性表采用顺序存储结构，每个元素占据 4 个存储单元，首地址为 100，则下标为 11 的（第 12 个）元素的存储地址为_____。

【答案】144

11．带头结点的双循环链表 L 中只有一个元素结点的条件是_____。

【答案】L->next->next==L

2.3　判断题

1．取线性表的第 i 个元素的时间同 i 的大小有关（　　）

【答案】×

2．线性表的特点是每个元素都有一个前驱和一个后继（　　）

【答案】×

3．顺序存储方式的优点是存储密度大，且插入、删除运算效率高（　　）

【答案】×

4．线性表采用链表存储时，结点的存储空间可以是不连续的（　　）

【答案】√

5．链表是采用链式存储结构的线性表，进行插入、删除操作时，在链表中比在顺序存储结构中效率高（　　）

【答案】√

6．顺序存储方式只能用于存储线性结构（　　）

【答案】×

【解析】线性结构、树型结构和图状结构均可用顺序存储表示。

9．顺序存储结构的主要缺点是不利于插入或删除操作（　　）

【答案】√

10．顺序存储方式插入和删除时效率太低，因此它不如链式存储方式好（　　）

【答案】×

2.4　程序设计题

1．设顺序表 va 中的数据元素递增有序。试设计一个算法，将 x 插入到顺序表的适当位置上，以保持

该表的有序性。

【算法源代码】

```
void  Insert_SqList(SqList va,int x)/*把 x 插入递增有序表 va 中*/
{ int i;
  if(va.length> MAXSIZE) return;
  for(i=va.length-1;va.elem[i]>x&&i>=0;i--)
        va.elem[i+1]=va.elem[i];
  va.elem[i+1]=x;
  va.length++;
}/*Insert_SqList*/
```

2．设 A=（a_1，a_2，…，a_m） 和 B=（b_1，b_2，…，b_n）均为顺序表，试设计一个比较 A，B 大小的算法（请注意：在算法中，不要破坏原表 A 和 B）。

【算法分析】比较顺序表 A 和 B，并用返回值表示结果，值为 1，表示 A>B；值为-1，表示 A<B；值为 0，表示 A=B。

1）当两个顺序表可以互相比较时，若对应元素不等，则返回值为 1 或-1；

2）当两个顺序表可以互相比较的部分完全相同时，若表长也相同，则返回值为 0；否则，哪个较长，哪个就较大

【算法源代码】

```
int ListComp(SqList A,SqList B)
{
  for(i=1;i<=A.length&&i<=B.length;i++)
    if(A.elem[i]!=B.elem[i])
      return A.elem[i]>B.elem[i]?1:-1;
  if(A.length==B.length) return 0;
  return A.length>B.length?1:-1;
/*当两个顺序表可以互相比较的部分完全相同时，哪个较长，哪个就较大*/
}/*ListComp */
```

3．已知指针 ha 和 hb 分别指向两个单链表的头结点，并且已知两个链表的长度分别为 m 和 n。试设计一个算法将这两个链表连接在一起（即令其中一个表的首元结点连在另一个表的最后一个结点之后），假设指针 hc 指向连接后的链表的头结点，并要求算法以尽可能短的时间完成连接运算。

【算法分析】

1）单链表 ha 的头结点作为连接后的链表的头结点，即 hc=ha；

2）查找单链表 ha 的最后一个结点，由指针 p 指向，即 p->next==NULL；

3）将单链表 hb 的首元结点（非头结点）连接在 p 之后，即 p->next=hb->next；

4）回收单链表 hb 的头结点空间

【算法源代码】

```
void ListConcat(LinkList ha,LinkList hb,LinkList *hc)
/*把链表 hb 接在 ha 后面形成链表 hc*/
{
  *hc=ha;
  p=ha;/*由指针 p 指向 ha 的尾元结点*/
       p=p->next;
  p->next=hb->next;
  free(hb);
}/*ListConcat */
```

4．试设计一个算法，在无头结点的动态单链表上实现线性表操作 INSERT（L，i，b），并和在带头结点的动态单链表上实现相同操作的算法进行比较。

【算法分析】

1）生成新结点存放元素 b，由指针 new 指向；

2）将 new 插入在单链表的第 i 个元素的位置上：若 i==1，new 插在链表首部；否则查找第 i-1 个结点，由指针 p 指向，然后将 new 插在 p 之后。

【算法源代码】

```
void  Insert(LinkList *L,int i,int b)
{ LinkList  new;
```

```
 new=(LinkList*)malloc(sizeof(LNode));
 new->data=b;
 if(i==1)
 {/*插入在链表头部*/
  New->next=*L;
  *L=new;
 }
 else
 { /*插入在第 i 个元素的位置*/
  p=*L;
  while(--i>1) p=p->next;
  new->next=p->next;p->next=new;
 }
}/*Insert */
```

5．已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试设计一个高效的算法，删除表中所有值大于 mink 且小于 maxk 的元素（若表中存在这样的元素），同时释放被删结点空间（注意：mink 和 maxk 是给定的两个参变量。它们的值可以和表中的元素相同，也可以不同）。

【算法分析】

1）查找最后一个不大于 mink 的元素结点，由指针 p 指向；

2）如果还有比 mink 更大的元素，查找第一个不小于 maxk 的元素，由指针 q 指向；

3）p->next=q，即删除表中所有值大于 mink 且小于 maxk 的元素。

【算法源代码】

```
void  Delete_Between(LinkList *L,int mink,int maxk)
{
 p=*L;
 while(p->next->data<=mink) p=p->next;  /*p 是最后一个不大于 mink 的元素*/
 if(p->next)   /*如果还有比 mink 更大的元素*/
 {
  q=p->next;
  while(q->data<maxk) q=q->next;  /*q 是第一个不小于 maxk 的元素*/
  p->next=q;
 }
}/*Delete_Between */
```

6．已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试设计一个高效的算法，删除表中所有值相同的多余元素（使得操作后的线性表中所有元素的值均不相同），同时释放被删结点空间。

【算法分析】

1）初始化指针 p 和 q，分别指向链表中相邻的两个元素；

2）当 p->next 不为空时，做如下处理：

①若相邻两元素不相等时，p 和 q 都向后推一步；

②否则，当相邻元素相等时，删除多余元素。

【算法源代码】

```
void  Delete_Equal(LinkList *L)
{
 p=(*L)->next;q=p->next; /*p 和 q 指向相邻的两个元素*/
 while(p->next)
 {
  if(p->data!=q->data) /*若相邻两元素不相等时，p 和 q 都向后推一步*/
     {
        p=p->next;
        q=p->next;
      }
  else
  {
   while(q->data==p->data) /*当相邻元素相等时删除多余元素*/
     {
```

```
        r=q;
        q=q->next;
        free(r);
       }
     p->next=q;p=q;q=p->next;
    }/*else*/
   }/*while*/
}/*Delete_Equal */
```

7．试设计一个算法，对带头结点的单链表实现就地逆置。

【算法分析】

1）空表或长度为 1 的表，不做任何处理；

2）表长大于 2 时，做如下处理：

①首先将整个链表一分为二，即从链表的第一元素结点处断开；

②逐个地把剩余链表的当前元素 q 插入到链表的头部。

【算法源代码】

```
void LinkList_reverse(LinkList  L)
{ if(!L->next||!L->next->next)  return;
  p=L->next;  q=p->next;  s=q->next;
  p->next=NULL;   /*从链表的第一元素结点处断开*/
  while(s->next)
  {q->next=p;p=q;
    q=s;s=s->next;   /*把 L 的元素逐个插入新表表头*/
  }
  q->next=p;s->next=q;L->next=s;
}/*LinkList_reverse*/
```

8．设线性表 A=（a_1，a_2，…，a_m） 和 B=（b_1，b_2，…，b_n），试设计一个按下列规则合并 A，B 为线性表 C 的算法，即使得

C=（a_1，b_1，…，a_m，b_m，b_{m+1}，…，b_n）当 m≤n 时；

或者

C=（a_1，b_1，…，a_n，b_n，a_{n+1}，…，a_m）当 m＞n 时。

线性表 A，B 和 C 均以单链表作存储结构，且 C 表利用 A 表和 B 表中的结点空间构成。注意：单链表的长度值 m 和 n 均未显式存储。

【算法分析】

1）初始化指针 p 指向链表 A 的当前元素，指针 q 指向链表 B 的当前元素；

2）当链表 A 和 B 均为结束时，做如下处理：

①将 B 的元素插入

②若 A 非空，将 A 的元素插入

③指针 p 和 q 同时后移

【算法源代码】

```
void merge1(LinkList A,LinkList B,LinkList *C)
 {p=A->next;q=B->next;*C=A;
  while(p&&q)
  { s=p->next;p->next=q;  /*将 B 的元素插入*/
    if(s)
    { t=q->next;
      q->next=s; /*若 A 非空，将 A 的元素插入*/
    }
    p=s;q=t;                    /*指针 p 和 q 同时后移*/
  }/*while*/
}/*merge1 */
```

9．假设有两个按元素值递增有序排列的线性表 A 和 B，均以单链表作存储结构，请设计一个算法将 A 表和 B 表归并成一个按元素值递减有序（即非递增有序，允许表中含有值相同的元素）排列的线性表 C，并要求利用原表（即 A 表和 B 表）的结点空间构造 C 表。

【算法分析】按从小到大的顺序依次把 A 和 B 的元素插入新表的头部 pc 处，最后处理 A 或 B 的剩余

元素。

【算法源代码】

```
void reverse_merge(LinkList A,LinkList B,LinkList *C)
 { LinkList pa,pb,pre;
  pa=A->next;pb=B->next; /*pa 和 pb 分别指向 A 和 B 的当前元素*/
  pre=NULL;
  while(pa||pb)
  { if(pa->data<pb->data||!pb) /*将 A 的元素插入新表*/
     { pc=pa;q=pa->next;pa->next=pre;pa=q;     }
   else    /*将 B 的元素插入新表*/
    { pc=pb;q=pb->next;pb->next=pre;pb=q;     }
   pre=pc;
  }
 *C=A;
A->next=pc; /*构造新表头*/
}/*reverse_merge*/
```

10．已知 A，B 和 C 为三个递增有序的线性表，现要求对 A 表作如下操作：删去那些既在 B 表中出现又在 C 表中出现的元素。试对顺序表编写实现上述操作的算法，并分析你的算法的时间复杂度（注意：题中没有特别指明同一表中的元素值各不相同）。

【算法分析】先从 B 和 C 中找出共有元素，记为 same，再在 A 中从当前位置开始， 凡小于 same 的元素均保留(存到新的位置)，等于 same 的就跳过，到大于 same 时就再找下一个 same。

【算法源代码】

```
void SqList_Intersect_Delete(SqList *A,SqList B,SqList C)
{i=0;j=0;k=0;m=0;     /*i 指示 A 中元素原来的位置，m 为移动后的位置*/
 while(i<(*A).length&&j<B.length&& k<C.length)
 { if(B.elem[j]<C.elem[k]) j++;
    else if(B.elem[j]>C.elem[k]) k++;
      else{
    same=B.elem[j];                    /*找到了相同元素 same*/
    while(B.elem[j]==same) j++;
    while(C.elem[k]==same) k++;        /*j 和 k 后移到新的元素*/
    while(i<(*A).length&&(*A).elem[i]<same)
      (*A).elem[m++]=(*A).elem[i++];     /*需保留的元素移动到新位置*/
    while(i<(*A).length&&(*A).elem[i]==same) i++; /*跳过相同的元素*/
   }
 }/*while*/
 while(i<(*A).length)
   (*A).elem[m++]=(*A).elem[i++];       /*A 的剩余元素重新存储*/
 (*A).length=m;
}/* SqList_Intersect_Delete*/
```

11．设 L 为单链表的头结点地址，其数据结点的数据都是正整数且无相同的，试设计利用直接插入的原则把该链表整理成数据递增的有序单链表的算法。

【算法分析】本题明确指出单链表带头结点，其结点数据是正整数且不相同，要求利用直接插入原则把链表整理成递增有序链表。这就要求从第二结点开始，将各结点依次插入到有序链表中。

【算法源代码】

```
void  InsertSort (LinkList la)
{if(la->next!=NULL) /*链表不为空表*/
 {p=la->next->next;       /*p 指向第一结点的后继*/
  la->next->next=NULL;
 /*直接插入原则认为第一元素有序，然后从第二元素起依次插入*/
  while(p!=NULL)
 {r=p->next;/*暂存 p 的后继*/
  q=la;
  while(q->next!=NULL&&q->next->data<p->data)q=q->next;/*查找插入位置*/
```

```
  p->next=q->next;/*将 p 结点链入链表*/
  q->next=p;
  p=r;
 }
```

12．设有一个双向循环链表，每个结点中除有 prior，data 和 next 三个域外，还增设了一个访问频度域 freq。在链表被起用之前，频度域 freq 的值均初始化为零，而每当对链表进行一次 LOCATE（L，X）的操作后，被访问的结点（元素值等于 X 的结点）中的频度域 freq 的值便增 1，同时调整链表中结点之间的次序，使其按访问频度非递增的次序顺序排列，以便始终保持被频繁访问的结点总是靠近表头结点。试编写符合上述要求的 LOCATE 操作的算法。

【算法分析】

1）在双向链表中查找数据值为 x 的结点，由指针 p 指向，若找不到，直接返回，否则执行第 2 步；

2）修改 x 结点的访问频度 freq，并将结点从链表上摘下；

3）顺结点的前驱链查找该结点的位置，即找到一个结点的访问频度大于 x 结点的访问频度，由指针 q 指向；若 q 和 p 不是相邻结点，调整位置，把 p 插在 q 之后。

【算法源代码】

```
DuLNode * Locate_DuList(DuLinkList *L,int x)
{ p=(*L)->next;
  while(p.data!=x&&p!= (*L)) p=p->next;
  if(p==(*L)) return NULL;                          /*没找到 x 结点*/
  p->freq++;
  p->pre->next=p->next;p->next->pre=p->pre;  /*将 x 结点从链表上摘下*/
  q=p->pre;
  while(q->freq<=p->freq&&p!= (*L)) q=q->pre;    /*查找插入位置*/
  if(q!=p->pre)                                     /*将 x 结点插入*/
  {q->next->pre=p;p->next=q->next;
   q->next=p;p->pre=q;                              /*调整位置*/
  }
  return p;
}/*Locate_DuList */
```

13．已知三个带头结点的线性链表 A、B 和 C 中的结点均依元素值自小至大非递减排列（可能存在两个以上值相同的结点），编写算法对 A 表进行如下操作：使操作后的链表 A 中仅留下三个表中均包含的数据元素的结点，且没有值相同的结点，并释放所有无用结点。限定算法的时间复杂度为 O（m+n+p），其中 m、n 和 p 分别为三个表的长度。

【算法分析】留下三个链表中公共数据，首先查找两表 A 和 B 中公共数据，再去 C 中找有无该数据。要消除重复元素，应记住前驱，要求时间复杂度 O（m+n+p），在查找每个链表时，指针不能回溯。

【算法源代码】

```
LinkList Common(LinkList A, LinkList B, LinkList C)
{pa=A->next;pb=B->next; pc=C->next;  /*pa，pb 和 pc 是工作指针*/
pre=A;
while(pa && pb && pc)        /*当三表均不空时，查找共同元素*/
{  while(pa && pb)
  if(pa->data<pb->data)             /*处理 pa 结点，后移指针*/
    {u=pa;pa=pa->next;free(u);}
     else if(pa->data> pb->data)pb=pb->next;
       else if (pa && pb)           /*处理 A 和 B 表元素值相等的结点*/
        {while(pc && pc->data<pa->data)pc=pc->next;
          if(pc)
            {if(pc->data>pa->data) /*处理 pa 结点，后移指针*/
              {u=pa;pa=pa->next;free(u);}
          else
           {if(pre==A)         /*结果表中第一个结点*/
                  { pre->next=pa;pre=pa;pa=pa->next}
            else if(pre->data==pa->data)     /*重复结点不链入 A 表*/
                  {u=pa;pa=pa->next;free(u);}
```

```
            else
                  {pre->next=pa;pre=pa;pa=pa->next;}/*将新结点链入 A 表 */
               pb=pb->next;pc=pc->next; /* 链表的工作指针后移*/
               }
            }
         else
            if(pa==NULL)pre->next=NULL; /*若 A 表已结束，置 A 表表尾*/
         else     /*处理原 A 表未到尾而 B 或 C 到尾的情况*/
            {pre->next=NULL;   /*置 A 表表尾标记*/
             while(pa!=NULL)  /*删除原 A 表剩余元素。*/
                  {u=pa;pa=pa->next;free(u);}
             }
      }
```

14．设 head 为一单链表的头指针，单链表的每个结点由一个整数域 data 和指针域 next 组成，整数在单链表中是无序的。编一函数，将 head 链中结点分成一个奇数链和一个偶数链，分别由 p，q 指向，每个链中的数据按由小到大排列。程序中不得使用 malloc 申请空间。

【算法分析】本题要求将一个链表分解成两个链表，两个链表都要有序，两链表建立过程中不得使用 malloc 申请空间，这就是要利用原链表空间，随着原链表的分解，新建链表随之排序。

【算法源代码】

```
discreat(LinkList p, LinkList q, LinkList head)
{ p=NULL;  q=NULL;/*p 和 q 链表初始化为空表*/
   s=head;
   while(s!=NULL)
     {r=s->next;            /*暂存 s 的后继*/
      if(s->data%2==0)       /*处理偶数*/
        if (p==NULL) {p=s;p->next=NULL;}  /*第一个偶数结点*/
        else { pre=p;
             if(pre->data>s->data)
                {s->next=pre;p=s;}/*插入当前最小值结点*/
           else
                {while (pre->next!=NULL)
                   if (pre->next->data<s->data) pre=pre->next;/*查找插入位置*/
                  s->next=pre->next;     /*链入结点*/
                  pre->next=s;}
          }
     else/*处理奇数链
       if (q==NULL) {q=s;q->next=NULL;}          /*第一奇数结点*/
       else
          {pre=q;
           if (pre->data>s->data) {s->next=pre; q=s;}            /*修改头指针*/
           else
                {while (pre->next!=NULL)          /*查找插入位置*/
                 if (pre->next->data<s->data)  pre=pre->next;
                 s->next=pre->next;      /*链入结点*/
                 pre->next=s; }
          }/*结束奇数链结点*/
      s=r;   /*s 指向新的待排序结点*/
     }
}
```

第 3 章　栈和队列

3.1　选择题

1．一个栈的输入序列为　123…n，若输出序列的第一个元素是　n，输出第　i（1≤i≤n）个元素是（　　）

A）不确定　　B）n-i+1　　C）i　　D）n-i

【答案】B

【解析】根据栈的性质（LIFO），若输出的第一个元素是 n，则表明所有的元素已经入栈，则出栈顺序为 n，n-1，…，3，2，1。

2．设栈 S 和队列 Q 的初始状态为空，元素 e1，e2，e3，e4，e5 和 e6 依次通过栈 S，一个元素出栈后即进队列 Q，若 6 个元素出队的序列是 e2，e4，e3，e6，e5，e1 则栈 S 的容量至少应该是（　　）

A）6　　B）4　　C）3　　D）2

【答案】C

【解析】根据栈的性质（LIFO）得，e2 出栈前，栈中存有 e1 和 e2 两个元素，e4 出栈前，栈中存有 e1、e3 和 e4 三个元素，e4 和 e3 出栈以后，e5 和 e6 入栈，栈中同样存在 e1、e5 和 e6 三个元素，然后三个元素依次出栈，所以栈的容量至少应该为 3。

3．若一个栈以向量 V[1..n]存储，初始栈顶指针 top 为 n+1，则下面 x 进栈的正确操作是（　　）

A）top=top+1; V[top]=x　　B）V[top]=x; top=top+1

C）top=top-1; V[top]=x　　D）V[top]=x; top=top-1

【答案】C

【解析】栈式运算受限的线性表，只允许在栈顶进行插入和删除操作。本题中栈顶指针为 n+1，该数组将栈顶放在了下标大的一端，所以在进行入栈操作时 top 指针应该进行减一操作。通常元素进栈的操作为：先移动栈顶指针后存入元素。

4．如果我们用数组 A[1..100]来实现一个大小为 100 的栈，并且用变量 top 来指示栈顶，top 的初值为 0，表示栈空。请问在 top 为 100 时，再进行入栈操作，会产生（ ）

A）正常动作　　B）溢出　　C）下溢　　D）同步

【答案】B

【解析】当 top 为 100 时，表示栈已经满了，此时再进行入栈操作，则会造成溢出。

5．栈在（　　）中应用。

A）递归调用　　B）子程序调用　　C）表达式求值　D）A，B，C

【答案】D

6．表达式 3* 2^(4+2*2-6*3)-5 求值过程中当扫描到 6 时，对象栈和算符栈为（　　），其中^为乘幂 。

A）3,2,4,1,1；(*^(+*-　　B)3,2,8；(*^-

C）3,2,4,2,2；(*^(-　　D)3,2,8；(*^(-

【答案】D

【解析】根据表达式求值的基本思想，在扫描表达式时，依次读入表达式的每个字符，若是操作数则进对象栈，若为运算符则和运算符栈的栈顶运算符比较优先级后做相应的操作。

7．用链接方式存储的队列，在进行删除运算时（　　）

A）仅修改头指针　　B）仅修改尾指针

C）头、尾指针都要修改　　D）头、尾指针可能都要修改

【答案】D

【解析】若队列中的元素多于一个，删除队列中的队尾元素，只需修改队尾指针；若队列中只有一个元素，删除该元素后，队头队尾指针都需要修改。

8．循环队列 A[0..m-1]存放其元素值，用 front 和 rear 分别表示队头和队尾，则当前队列中的元素数是(　　)

A）(rear-front+m)%m　　B）rear-front+1

C）rear-front-1　　D）rear-front

【答案】A

【解析】循环队列是解决假溢出的问题，通常把一维数组看成首尾相接。在循环意义下的求元素个数的运算可以利用求模运算。

9．若用一个大小为 6 的数组来实现循环队列，且当前 rear 和 front 的值分别为 0 和 3，当从队列中删除一个元素，再加入两个元素后，rear 和 front 的值分别为多少？( )

A）1 和 5　　B）2 和 4　　C）4 和 2　　D）5 和 1

【答案】B

【解析】循环队列是解决假溢出的问题，通常把一维数组看成首尾相接。在循环意义下的加 1 运算通常用求模运算来实现。所以入队和出队时的操作分别为：rear=(rear+1)%m，front=(front+1)%m。

10．栈和队列的共同点是（　　）

A）都是先进先出　　B）都是先进后出

C）只允许在端点处插入和删除元素　　D）没有共同点

【答案】C

【解析】栈和队列都是运算受限的线性表，只允许在表端点处进行操作。

11．在一个链队列中，假定 front 和 rear 分别为队头和队尾指针，则插入*s 结点的操作为（　　）

A）front->next=s;front=s;　　B）s->next=rear;rear=s;

C）rear->next=s;rear=s;　　D）s->next=front;front=s;

【答案】C

【解析】队列是运算受限的线性表（FIFO），插入元素只能插在队尾，所以需修改队尾指针。

12．判定一个栈 S（元素个数最多为 MAXSIZE）为空和满的条件分别为（　　）

A）S->top!=-1　S->top!=MAXSIZE-1

B）S->top=-1　S->top=MAXSIZE-1

C）S->top=-1　S->top!=MAXSIZE-1

D）S->top!=-1　S->top=MAXSIZE-1

【答案】B

13．循环顺序队列中是否可以插入下一个元素（　　）

A）与队头指针和队尾指针的值有关

B）只与队尾指针的值有关，与队头指针的值无关

C）只与数组大小有关，而与队头指针和队尾指针的值无关

D）与曾经进行过多少次插入操作有关

【答案】A

【解析】在循环队列中判断队满的条件为：(q.rear+1)%m==q.front 是否为真，从中可以看出，与队头指针和队尾指针的值有关。

14．最不适合用作链队的链表是（　　）

A）只带队头指针的非循环双链表　　B）只带队头指针的循环双链表

C）只带队尾指针的非循环双链表　　D）只带队尾指针的循环单链表

【答案】A

【解析】链队是在链表的两端进行操作，而在 A 中查找链表最后一个结点的时间复杂度为 O(n)。

15．下列哪中数据结构常用于系统程序的作业调度（　　）

A）栈　　B）队列　　C）链表　　D）数组

【答案】B

【解析】作业调度采用先到先服务的方式，因此最适合的数据结构为队列。

3.2　填空题

1．栈是_____的线性表，其运算遵循_____的原则。

【答案】（1）操作受限（或限定仅在表尾进行插入和删除操作）　（2）后进先出

2．设有一个空栈，栈顶指针为 1000H(十六进制)，现有输入序列为　1，2，3，4，5，经过 PUSH,PUSH,POP,PUSH,POP,PUSH,PUSH 之后，输出序列是_____，而栈顶指针值是_____H。设栈为顺序栈，每个元素占 4 个字节。

【答案】（1）23　（2）100CH

【解析】PUSH 为入栈操作，POP 为出栈操作。根据栈的性质，经过 PUSH,PUSH,POP 运算之后，栈中存在元素 1，输出数据为 2，然后经过 PUSH,POP，3 入栈，3 出栈，然后经过 PUSH,PUSH 之后 4，5 入栈，此时出栈序列为　2，3，栈中元素为　1，4，5；每个元素占　4 个字节，所以栈顶指针的值为 1000H+3*4=100CH（十六进制数）

3．循环队列的引入，目的是为了克服_____。

【答案】假溢出时大量移动数据元素。

4．队列是限制插入只能在表的一端，而删除在表的另一端进行的线性表，其特点是_____。

【答案】先进先出

5．已知链队列的头尾指针分别是 f 和 r，则将值 x 入队的操作序列是_____。

【答案】

```
s=(LinkList)malloc(sizeof(LNode));
s->data=x;
s->next=r->next;
r->next=s;
r=s;
```

【解析】根据队列的性质，新插入的元素永远插在队尾。

6．区分循环队列的满与空，只有两种方法，它们是_____和_____。

【答案】（1）牺牲一个存储单元　（2）设标记

7．表达式 23+((12*3-2)/4+34*5/7)+108/9 的后缀表达式是_____。

【答案】23.12.3*2-4/34.5*7/++108.9/+（注：表达式中的点(.)表示将数隔开，如 23.12.3 是三个数）

【解析】表达式的后缀表达式是将表达式中的运算符写在两个操作数之后。转换原则如下：

（1）从左到右扫描表达式，若读到的是操作数，则直接把它输出

（2）若读到的是运算符：

①该运算符为左括号“（”，则直接入栈

②该运算符为右括号“）”，则输出栈中运算符，直到遇到左括号为止

③该运算符为非括号运算符，则与栈顶元素做优先级比较：若比栈顶元素的优先级高或相等，则直接入栈；若比栈顶元素的优先级低，则输出栈顶元素。

（3）当表达式扫描完后栈中还有运算符，则依次输出运算符，直到栈空。

8．用下标 0 开始的 N 元数组实现循环队列时，为实现下标变量 M 加 1 后在数组有效下标范围内循环，可采用的表达式是：M=_____。

【答案】(M+1)% N;

【解析】循环队列是解决假溢出的问题，通常把一维数组看成首尾相接。在循环意义下的加 1 运算通常用求模运算来实现。

9．当两个栈共享一存储区时，栈利用一维数组 stack[1..n]表示，两栈顶指针为 top[1]与 top[2]，则当栈 1 空时，top[1]为_____，栈 2 空时，top[2]为_____，栈满时为_____。

【答案】（1）0　（2）n+1　（3）top[1]+1=top[2]

【解析】为了增加内存空间的利用率和减少溢出的可能性 ,由两个栈共享一片连续的内存空间时 ,应将两栈的栈顶分别设在这片内存空间的两端 ,这样,当两个栈的栈顶在栈空间的某一位置相遇时，才产生上溢，即 top[1[+1=top[2]。

10．在作进栈运算时应先判别栈是否_____；在作退栈运算时应先判别栈是否_____；当栈中元素为 n 个，作进栈运算时发生上溢，则说明该栈的最大容量为_____。

为了增加内存空间的利用率和减少溢出的可能性，由两个栈共享一片连续的空间时，应将两栈的_____分别设在内存空间的两端，这样只有当_____时才产生溢出。

【答案】（1）满　（2）空　（3）n　（4）栈底　（5）两栈顶指针相邻

11．在 Q 的链队列中，判断只有一个结点的条件是_____。

【答案】Q->front!=NULL&&Q->front==Q->rear

【解析】只有一个结点时，队列不为空并且队头指针和队尾指针指向同一结点。

12．若用不带头结点的单链表来表示链栈 S，则创建一个空栈所需要执行的操作是_____。

【答案】S=NULL

13．无论对于顺序存储还是链式存储的栈和队列来说，进行插入和删除运算的时间复杂度均相同为_____。

【答案】O(1)

【解析】对于栈用栈顶指针表示栈顶，而栈的插入和删除操作均在栈顶进行。对于队列用队头和队尾指针分别表示允许插入和删除的一端。

14．在顺序队列中，当尾指针等于数组的上界，即使队列不满，再作入队操作也会产生溢出，这种现象称为_____。

【答案】假溢出

【解析】产生该现象的原因是，被删元素空间在该元素被删除后就永远得不到使用。为了克服这种现象，采用循环队列来实现。

15．设元素 1，2，3，4，5 依次入栈， 若要得到 输出序列 34251，则应进行的操 作序列为 PUSH(S,1)， PUSH(S,2)， _____， POP(S)， PUSH(S,4)， POP(S)， _____， _____，POP(S)，POP(S)。

【答案】（1）PUSH(S,3)　　（2）POP(S)　（3）PUSH(S,5)

3.3　判断题

1．即使对不含相同元素的同一输入序列进行两组不同的合法的入栈和出栈组合操作，所得的输出序列也一定相同（　　）

【答案】×

【解析】栈的性质为后进先出，不同的入栈出栈组合得到的输出序列不一定相同。例如：对于序列 123，进行不同的入栈出栈操作，可能得到的输出序列有：123，213，321 等。

2. 链式队列队满条件是尾指针加一等于头指针（　　）

【答案】×

【解析】链队列本身没有容量限制，所以在用户内存空间的允许范围内不会出现队满的情况。

3. 栈和队列都是线性表，只是在插入和删除时受到了一些限制（　　）

【答案】√

4. 循环队列也存在空间溢出问题（　　）

【答案】√

5. 循环队列通常用指针来实现队列的头尾相接（　　）

【答案】×

【解析】循环队列是解决假溢出的问题，通常把一维数组看成首尾相接。在循环意义下的加 1 运算通常用求模运算来实现。

3.4　应用题

1．名词解释：栈和队列

栈是只允许在一端进行插入和删除操作的线性表，允许插入和删除的一端叫栈顶，另一端叫栈底。最后插入的元素最先删除，故栈也称后进先出（LIFO）表。

队列是允许在一端插入而在另一端删除的线性表，允许插入的一端叫队尾，允许删除的一端叫队头。最先插入队的元素最先离开（删除），故队列也常称先进先出（FIFO）表。

2. 假设以 S 和 X 分别表示入栈和出栈操作，则对初态和终态均为空的栈操作可由 S 和 X 组成的序列表示（如 SXSX）。

（1）试指出判别给定序列是否合法的一般规则。

（2）两个不同合法序列（对同一输入序列）能否得到相同的输出元素序列？如能得到，请举列说明。

【答案】（1）通常有两条规则。第一是给定序列中 S 的个数和 X 的个数相等；第二是从给定序列的开始，到给定序列中的任一位置，S 的个数要大于或等于 X 的个数。（2）可以得到相同的输出元素序列。例如，输入元素为 A，B，C，则两个输入的合法序列 ABC 和 BAC 均可得到输出元素序列 ABC。对于合法序列 ABC，我们使用本题约定的 SXSXSX 操作序列；对于合法序列 BAC，我们使用 SSXXSX 操作序列。

3. 如果输入序列为 123456，试问能否通过栈结构得到以下两个序列：435612 和 13542 6，请说明为什么不能或如何才能得到。

【答案】输入序列为 123456，不能得出 435612，其理由是，输出序列最后两元素是 12，前面 4 个元素（4356）得到后，栈中元素剩 12，且 2 在栈顶，不可能栈底元素 1 在栈顶元素 2 之前出栈。得到 135426 的过程如下：1 入栈并出栈，得到部分输出序列 1；然后 2 和 3 入栈，3 出栈，部分输出序列变为：13；接着 4 和 5 入栈，5，4 和 2 依次出栈，部分输出序列变为 13542；最后 6 入栈并退栈，得最终结果 135426。

4. 简述顺序存储队列的假溢出的避免方法及队列满和空的条件。

【答案】设顺序存储队列用一维数组 q[m]表示，其中 m 为队列中元素个数，队列中元素在向量中的下标从 0 到 m-1。设队头指针为 front，队尾指针是 rear，约定 front 指向队头元素的前一位置，rear 指向队尾元素。当 front 等于-1 时队空，rear 等于 m-1 时为队满。由于队列的性质（“删除”在队头而“插入”在队尾），所以当队尾指针 rear 等于 m-1 时，若 front 不等于-1，则队列中仍有空闲单元，所以队列并不是真满。这时若再有入队操作，会造成假“溢出”。其解决办法有二，一是将队列元素向前“平移”（占用 0 至 rear-front-1）；二是将队列看成首尾相连，即循环队列（0..m-1）。在循环队列下，仍定义 front=rear 时为队空，而判断队满则用两种办法，一是用“牺牲一个单元”，即

rear+1=front（准确记是（ rear+1）%m=front，m 是队列容量）时为队满。另一种解法是“设标记”方法，如设标记 tag，tag 等于 0 情况下，若删除时导致 front=rear 为队空；tag=1 情况下，若因插入导致 front=rear 则为队满。

5. 若以 1、2、3、4 作为双端队列的输入序列，试分别求出以下条件的输出序列：

（1）能由输入受限的双端队列得到，但不能由输出受限的双端队列得到的输出序列；

（2）能由输出受限的双端队列得到，但不能由输入受限的双端队列得到的输出序列；

（3）既不能由输入受限双端队列得到，也不能由输出受限双端队列得到的输出序列。

【答案】（1）4132　（2）4213　（3）4231

3.5　程序设计题

1．设表达式以字符形式已存入数组 E[n]中，‘#’为表达式的结束符，试写出判断表达式中括号（‘（’和‘）’）是否配对的 C 语言描述算法：EXYX(E); (注：算法中可调用栈操作的基本算法。)

【算法分析】判断表达式中括号是否匹配，可通过栈，简单说是左括号时进栈，右括号时退栈。退栈时，若栈顶元素是左括号，则新读入的右括号与栈顶左括号就可消去。如此下去，输入表达式结束时，栈为空则正确，否则括号不匹配。

【算法源代码】

```
int EXYX (char E[]){
/*E[]存放字符串表达式，以‘#’结束*/
 char s[30];        /*s 是一维数组，容量足够大，用作存放括号的栈*/
 int top=0,i;        /*top 用作栈顶指针*/
 s[top]='#';      /*‘#’先入栈，用于和表达式结束符号‘#’匹配*/
 i=0;           /*字符数组 E 的工作指针*/
 while(E[i]!='#')  /*逐字符处理字符表达式的数组*/
  switch (E[i])
   {case '(': s[++top]= '('; i++ ;  break ;
    case ')':  if(s[top]=='('){top--; i++; break;}
               else{printf("括号不配对");exit(0);}
    case '#':  if(s[top]=='#'){printf("括号配对\n");return (1);}
               else {printf(" 括号不配对\n");return (0);} /*括号不配对*/
    default :  i++;   /*读入其它字符，不作处理*/
   }
}/*算法结束*/
```

2．假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾结点，但不设头指针，请写出相应的入队列和出队列算法。

【算法分析】

根据队列的先进先出的性质，队列的入队操作在队尾进行，出队操作在队头进行。而题目所采用的数据结构是只设一个尾指针的循环链表。我们可以根据循环链表的特点找到头指针。

【算法源代码 1】

```
void EnQueue (LinkList rear, ElemType x)
/* rear 是带头结点的循环链队列的尾指针，本算法将元素 x 插入到队尾*/
{ s=(LinkList)malloc(sizeof(LNode)); /*申请结点空间*/
  s->data=x;  s->next=rear->next;        /*将 s 结点链入队尾*/
  rear->next=s;  rear=s;                 /*rear 指向新队尾*/
}
```

【算法源代码 2】

```
void DeQueue (LinkList  rear)
/* rear 是带头结点的循环链队列的尾指针，本算法执行出队操作，操作成功输出队头元
素；否则给出出错信息*/
{ if(rear->next==rear)  {printf("队空\n"); exit(0);}
  s=rear->next->next;            /*s 指向队头元素*/
  rear->next->next=s->next;      /*队头元素出队*/
  printf ("出队元素是:%d",s->data);
  if(s==rear) rear=rear->next;   /*空队列*/
```

```
        free(s);
    }
```

3．设整数序列 a_1，a_2，…，a_n，给出求解最大值的递归程序。

【算法分析】根据题意，本题的函数定义为：

$$maxvalue(a,n)=\begin{cases} a[1] & n=1 \\ a[n] & a[n]>maxvalue(a,n-1) \\ maxvalue(a,n-1) & a[n]<maxvalue(a,n-1) \end{cases}$$

【算法源代码】

```
int MaxValue (int a[],int n)
/*设整数序列存于数组 a 中，共有 n 个，本算法求解其最大值*/
{int max;
 if (n==1) max=a[1];
 else if (a[n]>MaxValue(a,n-1)) max=a[n];
     else max=MaxValue(a,n-1);
 return(max);
 }
```

4．试将下列递归函数改写为非递归函数。

```
void  test(int  *sum)
{
int  x;
scanf("%d",&x);
if(x==0) *sum=0 ;
        else {test(&sum); (*sum)+=x;}
printf("%5d",*sum);
}
```

【算法分析】

该函数是以读入数据的顺序为相反顺序进行累加问题，可将读入数据放入栈中，等输入结束时，将栈中数据退出进行累加。累加的初值为 0。

【算法源代码】

```
int test()
{
int x,sum=0,top=0,s[30];
scanf("%d",&x);
while (x!=0)
       {s[++top]=a; scanf("%d",&x); }
printf("%5d",sum);
while (top)
        {sum+=s[top--]; printf("%5d",sum); }
 }
```

5．编写一个算法，利用栈的基本运算将指定栈中的内容进行逆转。

【算法分析】

利用两个临时栈 s1 和 s2。先将 s 栈中的内容移到 s1 栈中，再将 s1 栈中的内容移到 s2 栈中，最后将 s2 栈中的内容移到 s 栈中，即可实现。

【算法源代码】

```
reverse(SqStack *s)
{SqStack *s1,*s2;  /*s,s1,s2 均为栈类型
 ElemType x;  /*栈中元素的类型，用于存储从栈中取出元素的临时变量*/
 initstack(s1); /*栈的初始化*/
 initstack(s2);
while(!stackempty(s))  /*如果栈不空，将 s 栈中的内容移到 s1 栈中*/
 {pop(s,x);   /*取栈顶元素放入变量 x 中*/
  push(s1,x);  /*将变量 x 入栈*/
 }
 while(!stackempty(s1)) /*如果栈不空，将 s1 栈中的内容移到 s2 栈中*/
 {pop(s1,x);
```

```
 push(s2,x);
 }
 while(!stackempty(s2)) /*如果栈不空，将 s2 栈中的内容移到 s 栈中*/
 {pop(s2,x);
  push(s,x);
 }
}
```

6．假设循环队列中只设 rear 和 length 来分别指示队尾元素的位置和队中元素的个数，试给出判别此循环队列的队满条件，并写出相应的入队和出队算法，要求出队时需返回队头元素。

【算法分析】

该题的关键问题是如何确定头指针，根据为指针 rear 和元素个数 length 很容易确定头指针。front=(rear-length+MAXSIZE)%MAXSIZE

【算法源代码】

```
#define  MAXQSIZE 100                    //最大队列长度
typedef  int  ElemType;
typedef  struct {
      ElemType data[MAXSIZE];            //队列存储空间
      int  rear;                         //尾指针，若队列不空，指向队列尾元素
      int  length;                       //队列内含元素的个数
}CyQueue;
int FullQueue( CyQueue *Q)
{/*判队满,队中元素个数等于空间大小*/
 return Q->length==Maxsize;
}
void EnQueue( CyQueue  *Q, ElemType x)
{/* 入队
 if(FullQueue( Q)) {printf("队已满，无法入队");return;} */
 Q->Data[Q->rear]=x;
 Q->rear=(Q->rear+1)%MAXSIZE /*在循环意义上的加 1*/
 Q->length++;
}
ElemType DeQueue( CyQueue *Q)
{/*出队*/
 int front; /*设一个临时队头指针*/
if(Q->length==0)
 Error("队已空，无元素可出队");
front=(Q->rear + MAXSIZE - Q->length)%MAXSIZE;
 Q->length --;
 return  Q->Data[front];
}
```

7．一个双向栈 S 是在同一向量空间内实现的两个栈，它们的栈底分别设在向量空间的两端。 试为此双向栈设计初始化 InitStack (S) 、入栈 Push(S , i , x) 和出栈 Pop(S , i) 等算法， 其中 i 为 0 或 1， 用以表示栈号。

【算法分析】

双向栈其实和单向栈原理相同，只是在一个向量空间内，好比是两个头对头的栈放在一起，中间的空间可以充分利用。

【算法源代码】

```
void InitStack( DuStack *S )/*初始化双向栈*/
{S->top[0] = -1;
S->top[1]  = STACKSIZE;
 }
int EmptyStack( DuStack *S, int  i )
/*判栈空(栈号 i) */
{return  (i == 0 && S->top[0] == -1|| i == 1 && S->top[1] == STACKSIZE)  ;
}
```

```
int FullStack( DuStack *S)
/*判栈满,满时肯定两头相遇*/
{return (S->top[0] == S-top1-1);
}
void Push(DuStack *S, int i, ElemType x)
/*进栈(栈号 i) */
{if (FullStack( S ))
 Error("Stack overflow");/*上溢、退出运行*/
if ( i == 0) S->Data[ ++ S->top0]= x;  /*栈 0 入栈*/
if ( i == 1) S->Data[ -- S->top[1] ]= x;  /* 栈 1 入栈*/
}
ElemType Pop(DuStack *S,  int i)
/*出栈(栈号 i) */
{if (EmptyStack ( S,i) )
 Error("Stack underflow");/*下溢退出*/
if( i==0 )
return ( S->Data[ S->top0--] );/*返回栈顶元素，指针值减 1*/
if( i==1 )
return ( S->Data[ S->top[1] ++] ); /*该栈是以另一端为底的，所以指针加 1*/
}
```

8．回文是指正读反读均相同的字符序列，如 "abba"和"abdba"均是回文，但 "good"不是回文。设计一个算法判定给定的字符向量是否为回文。(提示：将一半字符入栈)

【算法源代码】

```
void  sympthy(LinkList head, stack *s)/*判断长为 n 的字符串是否中心对称*/
{ int i=1;
LinkList p=head->next;
while(i<=n/2)      /* 前一半字符进栈*/
        { push(s,p->data); p=p->next;  }
 if(n%2!==0) p=p->next;/* 奇数个结点时跳过中心结点*/
 while(p&&p->data==pop(s)) p=p->next;
 if (p==NULL)
        printf("链表中心对称");
 else printf("链表不是中心对称");
} /* 算法结束*/
```

9．用标志位方式设计出在循环队列中进行插入和删除运算的算法。

【算法分析】

可引入标志位 flag，且规定当 flag=0 时表示队列空，当 flag=1 时表示队列非空。同时设 front，rear 和 MAXSIZE 分别为队头指针，队尾指针和队列的长度。其中，队头指针指向队头元素所在的实际存储单元的前一个位置，队尾指针指向队尾元素所在的位置。从而可得队满的条件是：(rear==front)&&(flag==1)

【算法源代码】

```
int flag; /*设置全局变量 flag 作为标志位*/
enqueue(SqQueue*sq,ElemType x) {
/*将 x 插入循环队列 sq 中，sq 具有队头和队尾指针*/
if((flag==1)&&(sq->rear==sq->front))  /*判断队满*/
 printf("queue is full!\n");
 else
 {sq->rear=(sq->rear+1)%MAXSIZE;
  sq->data[sq->rear]=x;
  }
  if(flag==0) flag=1;
}
ElemType dequeue(SqQueue*sq) /*删除队列 sq 的队头元素，并返回该元素*/
{ElemType x;
 if(flag==0) printf("queue is empty!\n");  /*判断队空*/
```

```
else
{sq->front=(sq->front+1)%MAXSIZE;
 x=sq->data[sq->front];
}
if(sq->front==sq->rear)  flag=0;
return x;
}
```

第 4 章　串

4.1　选择题

1．下面关于串的的叙述中，哪一个是不正确的？（　　）

A）串是字符的有限序列

B）空串是由空格构成的串

C）模式匹配是串的一种重要运算

D）串既可以采用顺序存储，也可以采用链式存储

【答案】B

【解析】空串是不含任何字符的串，即空串的长度是零。空格串是由空格组成的串，其长度等于空格的个数。

2．设有两个串 p 和 q，其中 q 是 p 的子串，求 q 在 p 中首次出现的位置的算法称为（　　）

A）求子串　　B）联接　　C）匹配　　D）求串长

【答案】C

3．若串 s="software"，其子串个数是（　　）

A）8　　B）37　　C）36　　D）9

【答案】C

【解析】s 的长度为 8，长度为 8 的子串有 1 个，长度为 7 的子串有 2 个，长度为 6 的子串有 3 个，长度为 5 的子串有 4 个，…，长度为 1 的子串有 8 个，共有(1+8)*8/2=36 个。

4．串的长度是指（　　）

A）串中所含不同字母的个数

B）串中所含字符的个数

C）串中所含不同字符的个数

D）串中所含非空格字符的个数

【答案】B

5．若串 S1="ABCDEFG"，S2="9898"，S3="###"，S4="012345"，则执行 concat(replace(S1,substr(S1,length(S2),length(S3)),S3),substr(S4,index(S2, '8'),length(S2)))其结果为（　　）

A）ABC###G0123　　B）ABCD###2345

C）ABC###G2345　　D）ABC###G1234

【答案】D

【解析】函数 concat(x,y)返回 x 和 y 的连接串，substr(s,i,j)返回串 s 的从序号 i 的字符开始的 j 个字符组成的子串，length(s)返回串 s 的长度。replase(s,t,v)用 v 替换 s 中出现的所有与 t 相等的子串,index(s,t,i)当 s 中存在与 t 值相同的子串时，返回它在 s 中的第 i 个字符之后第一次出现的位置。

substr(S1,length(S2),length(S3))=substr(S1,4,3)= "DEF";

replase(S1,substr(S1,length(S2),length(S3)),S3)=replase(S1, "DEF",S3)= "ABC###G";

substr(S4,index(S2, '8'),length(S2))=substr(S4,2,4)= "1234";

concat(replace(S1,substr(S1,length(S2),length(S3)),S3),substr(S4,index(S2, '8'),length(S2)))=concat("ABC###G", "1234")= "ABC###G1234"

4.2　填空题

1．空格串是指_____，其长度等于_____。

【答案】（1）由空格字符（ASCII 值 32）所组成的字符串　（2）空格个数

2．组成串的数据元素只能是_____。

【答案】字符

【解析】串是一种特殊的线性表，其特殊性在于串中的元素只能是字符型数据。

3．设正文串长度为 n，模式串长度为 m，则串匹配的 KMP 算法的时间复杂度为_____。

【答案】O(m+n)

【解析】朴素的模式匹配（Brute－Force）时间复杂度是 O（m*n），KMP 算法有一定改进，时间复杂度达到 O（m＋n）。

4．两个字符串相等的充分必要条件是_____。

【答案】两串的长度相等且两串中对应位置上的字符也相等。

5．一个字符串中_____称为该串的子串 。

【答案】任意个连续的字符组成的子序列

4.3 **判断题**

1．KMP 算法的特点是在模式匹配时指示主串的指针不会变小（ ）

【答案】√

【解析】KMP 算法的改进在于：每当一趟匹配过程中出现字符比较不相等时，不需回溯主串指针，而是利用已经得到的“部分匹配”的结果将模式向右“滑动”尽可能远的一段距离后，继续进行比较。

2．串是一种数据对象和操作都特殊的线性表（ ）

【答案】√

【解析】串是一种特殊的线性表，其特殊性在于串中的元素只能是字符型数据。字符型数据的操作符合字符型数据的操作规范，具有它的特殊性。

3．如果一个串中的所有字符均在另一串中出现，那么说明前者是后者的子串（ ）

【答案】×

【解析】一个字符串中任意个连续的字符组成的子序列称为该串的子串，注意其中字符的连续性。

4.4 **应用题**

1．描述以下概念的区别：空格串与空串。

【答案】空格是一个字符，其 ASCII 码值是 32。空格串是由空格组成的串，其长度等于空格的个数。空串是不含任何字符的串，即空串的长度是零。

2．设 S1，S2 为串，请给出使 S1/*S2=S2/*S1 成立的所有可能的条件（/*为连接符）。

【答案】

（1）s1 和 s2 均为空串；

（2）两串之一为空串；

（3）两串串值相等（即两串长度相等且对应位置上的字符相同）。

（4）两串中一个串长是另一个串长（包括串长为 1 仅有一个字符的情况）的数倍，而且长串就好象是由数个短串经过连接操作得到的。

3．已知：s ＝" (xyz)＋*"，t ＝" (x＋z)*y"。试利用联结、求子串和置换等基本运算，将 s 转化为 t 。

【答案】本题有多种解法，下面是其中的一种：

（1） s1=substr（s,3,1） /*取出子串："y"

（2） s2=substr（s,6,1） /*取出子串："+"

（3） s3=substr（s,1,5） /*取出子串：" (xyz) "

（4） s4=substr（s,7,1） /*取出子串："*"

（5） s5=replace（s3,3,1,s2）/*形成部分串：" (x+z) "

（6） s=s5/*s4/*s1 /*形成串 t 即" (x+z)*y"

【解析】题中所给操作的含义如下：

/*：连接函数，将两个串连接成一个串

substr（s,i,j）：取子串函数，从串 s 的第 i 个字符开始，取连续 j 个字符形成子串

replace（s1,i,j,s2）：置换函数，用 s2 串替换 s1 串中从第 i 个字符开始的连续 j 个字符

4.5 **算法设计题**

1．设 s、t 为两个字符串，分别放在两个一维数组中，m、n 分别为其长度，判断 t 是否为 s 的子串。如果是，输出子串所在位置（第一个字符），否则输出 0。

【算法分析】

判断字符串 t 是否是字符串 s 的子串，称为串的模式匹配，其基本思想是对串 s 和 t 各设一个指针 i 和 j，i 的值域是 0..m-n，j 的值域是 0..n-1。初始值 i 和 j 均为 0。模式匹配从 s_0 和 t_0 开始，若 s_0==t_0，则 i 和 j 指针增加 1，若在某个位置 s_i!=t_j，则主串指针 i 回溯到 i=i-j+1，j 仍从 0 开始，进行下一轮的比较，直到匹配成功（j>n-1），返回子串在主串的位置（i-j）。否则，当 i>m-n 则为匹配失败。

【算法源代码】

```
int index(char s[ ], char t[ ],int m,int n)
{int i=0,j=0;
 while (i<=m-n && j<=n-1)
   if (s[i]==t[j]){i++;j++;}  /*对应字符相等，指针后移*/
        else {i=i-j+1;j=0;}     /*对应字符不相等，i 回溯，j 仍为 0*/
if(i<=m-n && j==n)
     {printf("t 在 s 串中位置是%d",i-n+1);
      return(i-n+1);
      }/*匹配成功*/
else  return(0); /*匹配失败*/
}
```

2．函数 void insert(char*s,char*t,int pos)将字符串 t 插入到字符串 s 中，插入位置为 pos。请用 c 语言实现该函数。假设分配给字符串 s 的空间足够让字符串 t 插入。（说明：不得使用任何库函数）

【算法分析】

本题是字符串的插入问题，要求在字符串 s 的 pos 位置，插入字符串 t。首先应查找字符串 s 的 pos 位置，将第 pos 个字符到字符串 s 尾的子串向后移动字符串 t 的长度，然后将字符串 t 复制到字符串 s 的第 pos 位置后。

对插入位置 pos 要验证其合法性，小于 1 或大于串 s 的长度均为非法，因题目假设给字符串 s 的空间足够大，故对插入不必判溢出。

【算法源代码】

```
void  insert(char *s,char *t,int pos)
/*将字符串 t 插入字符串 s 的第 pos 个位置*/
{
 int i=1,x=0,j;  char *p=s,*q=t;  /*p，q 分别为字符串 s 和 t 的工作指针*/
 if(pos<1)
       {printf("pos 参数位置非法\n");exit(0);}
while(*p!='\0'&&i<pos) {p++;i++;} /*查 pos 位置*/
 if(*p=='/0')
       {printf("%d 位置大于字符串 s 的长度",pos);exit(0);}
 else     /*查找字符串的尾*/
while(*p!= '/0')
       {p++; i++;} /*查到尾时，i 为字符'\0'的下标，p 也指向'\0'*/
  while(*q!= '\0')
       {q++; x++; }  /*查找字符串 t 的长度 x，循环结束时 q 指向'\0'*/
   for(j=i;j>=pos ;j--)
         {*(p+x)=*p; p--;}/*串 s 的 pos 后的子串右移，空出串 t 的位置*/
  q--;  /*指针 q 回退到串 t 的最后一个字符
  for(j=1;j<=x;j++) *p--=*q--;  /*将 t 串插入到 s 的 pos 位置上*/
}
```

3．设计一个算法，统计在输入字符串中各个不同字符出现的频度。（字符串中的合法字符为 'A'-'Z'这 26 个字母和'0'-'9'这 10 个数字）。

【算法分析】

由于字母共 26 个，加上数字符号 10 个共 36 个，所以设一长 36 的整型数组，前 10 个分量存放数字字符出现的次数，余下存放字母出现的次数。从字符串中读出数字字符时，字符的 ASCII 代码值减去数字字符'0'的 ASCII 代码值，得出其数值(0..9)，字母的 ASCII 代码值减去字符'A'的 ASCII 代码值加上 10，存入其数组的对应下标分量中。遇其它符号不作处理，直至输入字符串结束。

【算法源代码】

```
void Count()
/*统计输入字符串中数字字符和字母字符的个数*/
{int i,num[36];
char ch;
for(i=0;i<36;i++)num[i]=0;/* 初始化*/
```

```
while((ch=getchar())!='#')  /*‘#表示输入字符串结束*/
if(('0'<=ch)&&(ch<='9'))
{i=ch-'0';num[i]++;}        /* 数字字符*/
else if(('A'<= ch)&&(ch <='Z'))
{i=ch-'A'+10;num[i]++;}/* 字母字符*/
for(i=0;i<10;i++)  /* 输出数字字符的个数*/
printf("数字%d 的个数＝%d\n",i,num[i]);
for(i=10;i<36;i++)/* 求出字母字符的个数*/
printf("字母字符%c 的个数＝%d\n",i+55,num[i]);
}/* 算法结束*/
```

4．若 S 和 T 是用结点大小为 1 的单链表存储的两个串，试设计一个算法找出 S 中第一个不在 T 中出现的字符。

【算法分析】

查找过程是这样的，取 S 中的一个字符(结点)，然后和 T 中所有的字符一一比较，直到比完仍没有相同的字符时，查找过程结束，否则再取 S 中下一个字符，重新进行上述过程。

【算法源代码】

```
char SearchNo( LinkString S, LinkString T)
/*查找不在 T 中出现的字符*/
{ LinkString p,q;
  p=S;
  q=T;
  while (p)
  {/*取 S 中结点字符*/
   while(q&&p->data!=q->data)/*进行字符比较*/
   q->next;
   if(q==NULL)return p->data;/*找到并返回字符值*/
   q=T;        /*指针恢复串 T 的开始结点*/
   p=p->next;
   }
   printf("there's no such character.");
   return NULL;
}
```

5．如果一个字符串的一个子串（其长度大于 1）的各个字符均相同，则称之为等值子串。试设计一个算法，输入字符串 s，以“！”作为结束标志。如果串 s 中不存在等值子串，则输出信息“无等值子串”，否则求出（输出）一个长度最大的等值子串。

【算法分析】

用字符数组 s 接受用户输入的字符串。设 head 指向当前发现的最长等值子串的串头，max 记录此子串的长度。对 s 进行扫描，若发现新的等值子串，用 count 变量统计其长度，若他的长度大于原有的 max，则对 head 和 max 进行更新。重复上述过程直到 s 末尾。

【算法源代码】

```
#define MAXSIZE 100
{int i,j,k,head,max,count;
 char s[MAXSIZE];
 printf("输入字符串： ");
 k=0;
 scanf("%c",&s[k]);
 while(s[k]!='!')
 scanf("%d",&s[++k]);
 i=0,j=1,head=0,max=1;
 for(;s[i]!='!'&&s[j]!='!';i=j,j++)
 {count=1;
  while(s[i]==s[j])
  {j++;
   count++;
  }
```

```
 if(count>max)
 {head=i;
  max=count;
 }
}
if(max>1)
{printf("最大等值子串： ");
for(k=head;k<(head+max);k++)
printf("%c",s[k]);
}
else printf("无等值子串");
printf("\n");
}
```

6．采用顺序存储结构存储的串，编写一个程序，将两个字符串进行比较，若 s>t 时返回 1，s=t 时返回 0，s<t 时返回-1。不能用 strcmp 库函数。

【算法分析】

从两个字符串的第一个字符开始逐个进行比较（按字符的 ASCII 码大小比较），直到出现不同的字符或遇到'\0'为止。如果全部字符都相同，就认为两个字符串相等，返回 0。若出现了不相同的字符，则以第一个不相同的字符的比较结果为准。若前者字符大于后者字符，则返回 1，否则返回-1。

【算法源代码】

```
int comp(SString *s1,SString *s2)
{
int i=0,minlen;
minlen=s1->len>s2->len?s1->len:s2->len;
/*minlen 存放 s1 与 s2 中的较短的字符串的长度*/
while(i<=minlen)
{
if(s1->data[i]==s2->data[i]) /*如果 s1 与 s2 的当前字符相等，则比较下一个*/
    i++;
else  if(s1->data[i]<s2->data[i])  /*s1 的当前值小于 s2 的当前值，返回-1*/
    retrun -1;
else return 1;  /*s1 的当前值大于 s2 的当前值，返回 1*/
}
if(s1->len==s2->len)
    return 0;  /*s1 与 s2 所有字符均相等，且长度相等，则返回 0*/
}
```

7．输入一个由若干单词组成的文本行，每个单词之间用若干个空格隔开，统计其中的单词数。

【算法分析】

单词的数目可以有空格出现的次数来决定（连续的多个空格作为出现一次空格；不包含一行开头的空格）。如果当前字符为非空格，而他前面的字符是空格，则表示新的单词出现，此时让 num（单词数）累加 1。如果当前字符为非空格，而他前面的字符也是非空格，则表示此字符仍然是原单词的继续， num 不应累加。前面一个单词是否为空格，可以设置一个变量 word，若 word=0，则表示前一个字符时空格，如果 word=1，表示前一个字符为非空格。

【算法源代码】

```
int count(s)
char s[80];
{char c;
 int i,num=0,word=0;
 for(i=0;s[i]!='\0';i++)
 {if(s[i]==' ') word=0;
  else if(word==0)
      {word=1;
       num++;
      }
 }
```

```
return num;
}
```

8．一个仅由字母组成的字符串 s，长度为 n，其结构为单链表，每个结点的 data 字段只存放一个字母。试设计一个函数，去掉字符串中所有的 X 字母，并将串中的一个最小字母排列到串尾。

【算法分析】

从链表的表头开始查找每一个结点，如果该结点的数据值为 X，则删除该结点。同时在查找的过程中，顺便比较该结点与前驱结点的大小，如果该结点的值闭其前驱结点的值大，则顺便交换，直到整个链表结束。

【算法源代码】

```
search(LinkList s,char x)
/*在带头结点的单链表 s 中查找数据值为 x 的结点*/
{
 LinkList p,q,r;
 char temp;
 p=s->next;
 q=s;/*p 表示当前操作的结点，q 表示 p 的前驱结点*/
 while(p!=NULL)
 {if(p->data==x)
   {r=p;
    q-next=p->next;
    p=p->next;
    free(r);
     } /*找到释放结点*/
  else
  {q=p;  p=p->next; }
  if(q!=s&&p->data>q->data) /*将结点值最小的结点移到表尾*/
   {
     temp=p->data;
     p->data=q->data;
     q->data=temp;
    }
  }
 }
```

9．设计一个算法，将字符串 s 的全部字符复制到字符串 t 中，不能利用 strcpy 函数。

【算法分析】

要实现两个字符串的复制，实质为两个字符数组之间的复制，在复制时，一个字符一个字符的复制，直到遇到'\0'，'\0'一同复制过去，'\0'之后的字符不复制。

【算法源代码】

```
copy(char s[],char t[])
{
  int i;
 for(i=0;s[i]!='\0';i++)  t[i]=s[i];
 t[i]=s[i];
}
```

10．编写一个实现串通配符匹配的函数，其中的通配符只有'?'，它可以和任何一个字符匹配成功。

【算法分析】

本题基本思想与模式匹配的基本思想相似，只是增加了 '?'的处理功能。因为'?'可以与任何字符匹配成功，所以当字串中遇到'?'时，可以直接让主串和子串同时后移。

【算法源代码】

```
int pattern(SString s,SString t)
{
    int i=1,j=1;
     while(i<=s[0]&&j<=t[0])
     {if(s[i]==t[j])
      {i++;j++;
```

```
        }
        else if(t[j]=='?')  /*若字串中遇到'?'时，主串和子串同时后移*/
        {i++;j++;
        }
        else
        {i=i-j+2;j=1;
        }
       if(j>t[0]) return(i-t[0]);
          else return 0;
}
```

第 5 章　多维数组和广义表

5.1　选择题

1．数组 A 中，每个元素的长度为 3 个字节，行下标 I 从 1 到 8，列下标 J 从 1 到 10，从首地址 SA 开始连续存放在存储器内，该数组占用的字节数为（　　）

A）80　　B）100　　C）240　　D）270

【答案】C

2．数组 A 中，每个元素的长度为 3 个字节，行下标 I 从 1 到 8，列下标 J 从 1 到 10，从首地址 SA 开始连续存放在存储器内，该数组按行存放时，元素 A[8][5]的起始地址为（　　）

A）SA＋141　　B）SA＋144　　C）SA＋222　　D）SA＋225

【答案】C

【解析】数组 A 有 8 行 10 列，按行存放时， LOC（A[8][5]）=SA+（（8-1）*10+（5-1））*3=SA+222。

3．一个 n*n 的对称矩阵，如果以行或列为主序放入内存，则其容量为（　　）

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

【答案】C

【解析】对称矩阵可用上（或下）三角矩阵存储，第一行存 1 个，第二行存 2 个，…，第 n 行存 n 个，共 1+2+…+n=（n+1）*n/2。

4．稀疏矩阵一般的压缩存储方法有两种，即（　　）

A）二维数组和三维数组　　B）三元组和散列

C）三元组和十字链表　　D）散列和十字链表

【答案】C

5．设有广义表 D=（a,b,D），则其长度为（　　），深度为（　　）

A）1　　B）3　　C）∞　　D）5

【答案】B　C

6．广义表运算式（Tail（（a，B），（c，d））的操作结果是（　　）

A）（c，d）　B）c,d　C）（（c，d））　D）d

【答案】C

【解析】由于表中共 2 个元素，分别是两个广义表（a,b），（c,d），（a,B）是表头，因此 Tail 求得除表头外的元素构成的表即（（c,d））。

5.2　填空题

1．一维数组的逻辑结构是_____，存储结构是_____。

【答案】（1）线性结构　（2）顺序结构

2．对于二维数组或多维数组，分为按_____和按_____两种不同的存储方式存储。

【答案】（1）以行为主序　　（2）以列为主序

3．二维数组 A[c1..d1,c2..d2]共含有_____个元素。

【答案】（d1-c1+1）*（d2-c2+1）

4．二维数组 A[10][20]采用列序为主方式存储，每个元素占一个存储单元，且 A[0][0]的地址是 200，则 A[6][12]的地址是_____。

【答案】326

【解析】采用列主序时，LOC（A[6][12]）=LOC（A[0][0]+（12*10+6）*1=326

5．有一个 10 阶对称矩阵 A，采用以行为主序的压缩存储方式，A[0][0]的地址为 1，则 A[8][5]的地址是_____。

【答案】42

【解析】A[8][5]前有 8 行，第 0 行 1 个元素，第 1 行 2 个元素，…，第 7 行 8 个元素，共（1+8）*8/2=36 个元素，第 8 行前有 5 个元素，所以 A[8][5]的地址为 36+5+1=42。

6．广义表运算式 HEAD（TAIL（（a，b，c），（x，y，z））的结果为_____。

【答案】（x,y,z）

5.3 判断题

1．数组中存储的数可是任意类型的任何数据（ ）

【答案】×

【解析】同一数组中数据元素的类型应该相同（ ）

2．N*N 对称矩阵的经过压缩存储后占用的存储单元是原先的 1/2。

【答案】×

【解析】应为（N＋1）*N/2 个存储单元。

3．稀疏矩阵在用三元组表示法时，可节省空间，但对矩阵的操作会增加算法的难度及耗费更多的时间（ ）

【答案】√

4．广义表不是线性表（ ）

【答案】×

【解析】广义表是特殊的线性表，其特殊性在于表中的数据元素还可以是广义表。

5．tail（a,b,c,d）得到的是（b,c,d）（ ）

【答案】√

5.4 应用题

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？

【答案】设数组元素 A[i][j]存放在起始地址为 Loc （ i, j ） 的存储单元中。

∵ Loc （ 2, 2 ） = Loc （ 0, 0 ） + 2 * n + 2 = 644 + 2 * n + 2 = 676.

∴ n = （ 676 - 2 - 644 ） / 2 = 15

∴ Loc （ 3, 3 ） = Loc （ 0, 0 ） + 3 * 15 + 3 = 644 + 45 + 3 = 692.

2．设有一个 $n\times n$ 的对称矩阵 *A*，为了节约存储，可以只存对角线及对角线以上的元素，或者只存对角线或对角线以下的元素。前者称为上三角矩阵，后者称为下三角矩阵。我们把它们按行存放于一个一维数组 *B* 中，称之为对称矩阵 *A* 的压缩存储方式。试问：

（1） 存放对称矩阵 *A* 上三角部分或下三角部分的一维数组 *B* 有多少元素？

（2） 若在一维数组 *B* 中从 0 号位置开始存放，则对称矩阵中的任一元素 a_{ij}在只存下三角部分的情形下应存于一维数组的什么下标位置？给出计算公式。

【答案】

（1） 数组 B 共有 1＋2 + 3 +…… + n= （ n+1 ）*n / 2 个元素。

（2） 只存下三角部分时，若 i ≥ j，则数组元素 A[i][j]前面有 i-1 行（1~i-1，第 0 行第 0 列不算），第 1 行有 1 个元素，第 2 行有 2 个元素，……，第 i-1 行有 i-1 个元素。在第 i 行中，第 j 号元素排在第 j 个元素位置，因此，数组元素 A[i][j]在数组 B 中的存放位置为：

1 + 2 + …… + （i-1） + j = （ i-1）*i / 2 + j

若 i < j，数组元素 A[i][j]在数组 B 中没有存放，可以找它的对称元素 A[j][i]。在数组 B 的第 （j-1）*j / 2 + i 位置中找到。

如果第 0 行第 0 列也计入，数组 B 从 0 号位置开始存放，则数组元素 A[i][j]在数组 B 中的存放位置可以改为：

当 i ≥ j 时，= i*（i+1） / 2 + j

当 i < j 时，= j*（j+1） / 2 + i

3．利用广义表的 head 和 tail 操作写出函数表达式，把以下各题中的单元素 banana 从广义表中分离出来：

（1） L1（apple, pear, banana, orange）

（2） L2（（apple, pear）, （banana, orange））

（3） L3（（（apple）, （pear）, （bananA）, （orange））

（4） L4（（（（apple））, （（pear））, （bananA）, orange）

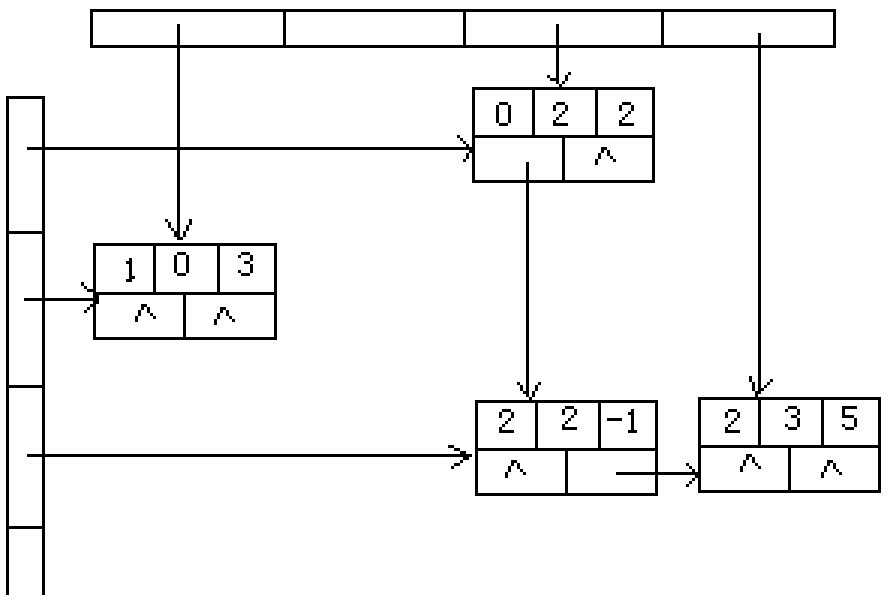
（5） L5（（（（apple）, pear）, bananA）, orange）
（6） L6（apple, （pear, （bananA）, orange））
【答案】
（1） Head （Tail （Tail （L1） ） ）
（2） Head （Head （Tail （L2） ） ）
（3） Head （Head （Tail （Tail （Head （L3） ） ） ） ）
（4） Head （Head （Tail （Tail （L4） ） ） ）
（5） Head （Tail （Head（L5） ） ）
（6） Head （Head （Tail （Head （Tail （L6） ） ） ） ）

3．一个稀疏矩阵为 $\begin{pmatrix} 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}$，则对应的三元组线性表是什么？其对应的十字链表是什么？

【答案】由于线性表中的每个结点对应稀疏矩阵的一个非零元素，其中包括 3 个字段，分别为该元素的行下标、列下标和值，结点间的次序按矩阵的行优先顺序排列，这个线性表用顺序的方法存储在连续的存储区，则对应的三元组为：

0	2	2
1	0	3
2	2	-1
2	3	5

其十字链表形式为：



5.5　算法设计题

1．假定数组 A[n]的 n 个元素中有多个零元素，编写算法将 A 中所有的非零元素依次移到 A 的前端。

【算法分析】从前向后找零元素 A[i]，从后向前找非零元素 A[j]，将 A[i]与 A[j]交换。

【算法源代码】

```
void move(int A[],int n)
{int i=0,j=n-1;
int temp;
while(i<j)
  {
   while(A[i]!=0)  i++;
   while(A[j]==0)  j--;
   if(i<j)
     {temp=A[i];A[i]=A[j];A[j]=temp;}
   }
}
```

2．给定有 m 个整数的递增有序数组 A[1..m]和有 n 个整数的递减有序数组 B[1..n]。试写出算法：将数组 A 和 B 归并为递增有序数组 C[1..m+n]。（要求：算法的时间复杂度为 O（m+n）。）

【算法分析】为保证算法的时间复杂度为 O（m+n），即需要对数组 A 和 B 的数据元素仅扫描一次就能生成 C 数组，我们可采用设三个下标指针 i,j,k 初始时分别指向 A 数组的最后一个元素（A 数组的最大

值）、B 数组的第一个元素（B 数组的最大值）、C 数组将存放最大值的位置，然后比较 A 与 B 数组的最大值大者放 C 数组 k 所指单元；在上述比较中若 A 数组 i 所指单元数大，则送完后 i 前移，否则 j 所指单元数送完后 j 后移，同时 k 前移，直到把 A 与 B 数组的所有元素扫描完。

【算法源代码】

```
#define m 3
#define n 4
void  Merge(int  A[],int B[],int  C[])
 {int  i,j,k;
 i=m-1;   j=0;   k=m+n-1;
 while((i>=0)&&(j<=n-1))
 {if(A[i]>B[j])
     {C[k]=A[i]; i--;}
 else
     {C[k]=B[j]; j++; }
 k--; }
 while(i>=0)            {C[k]=A[i];i--;k--;}
 while(j<=n-1)         {C[k]=B[j];j++;k--; }
 }
```

3．假设稀疏矩阵 A 采用三元组表示，编写一个函数计算其转置矩阵 B，要求 B 也用三元组表示。

【算法分析】三元组表示中要求按行的顺序存放，所有转置过程不能直接将行下标和列下标转换，还必须使得列按顺序存放。因此在 A 中首先找出第一列中的所有元素，它们是转置矩阵中第一行非 0 元素，并把它们依次放在转置矩阵三元组数组 B 中；然后依次找出第二列中的所有元素，把它们依次放在数组 B 中；按照同样的方法逐列进行，直到找出第 n 列的所有元素，并把它们依次放在数组 B 中。

【算法源代码】

```
void transpose(TSMatrix A,TSMatrix *B)
/*A 是稀疏矩阵的三元组形式，B 是存放 A 的转置矩阵的三元组数组*/
{int i,j,k;
B->mu=A.nu;
B->nu=A.mu;
B->tu=A.tu;
if(B->tu>0)
 {j=1;
 for(k=1;k<=A.nu;k++)
   for(i=1;i<=A.tu;i++)
    if(A.data[i].col==k)
    {B->data[j].row=A.data[i].col;
     B->data[j].col=A.data[i].row;
     B->data[j].e=A.data[i].e;
     j++;
     }
  }
 }
```

4．求广义表深度的递归算法。

【算法分析】在求广义表深度的递归算法中，若结点为原子则深度为 0，若是空表深度为 1，否则返回头指针与尾指针所指广义表的深度最大值。

【算法源代码】

```
int Glist_Getdeph(Glist L)
{int m,n;
 if(!L->tag) return 0;
 else if(!L) return 1;
 m=Glist_Getdeph(L->ptr.hp)+1;
 n=Glist_Getdeph(L->ptr.tp);
 return m>n?n:n;
 }
```

5．按层序输出广义表 A 中的所有元素。

【算法分析】层次遍历的问题，一般都是借助队列来完成的，每次从队头中取出一个元素的同时把它

的下一层的孩子插入队尾，这就是层序遍历的基本思想。

【算法源代码】

```
void Glist_printf(Glist L)
{InitQueue(Q);
for(p=L;p;p=p->ptr.tp)   EnQueue(Q,p);
while(!QueueEmpty(Q))
 {DeQueue(Q,r);
  if(!r->tag)
    printf("%d",r->atom);
  else
    for(r=r->ptr.hp;r;r=r->ptr.tp)      EnQueue(Q,r);
 }
}
```

第 6 章　树

6.1　选择题

1．一棵具有 n 个结点的完全二叉树的树高度（深度）是（　）

A）$\lfloor \log_2 n \rfloor+1$　　B）$\log_2 n+1$　　C）$\lfloor \log_2 n \rfloor$　　D）$\log_2 n-1$

【答案】A

2．有关二叉树下列说法正确的是（　）

A）二叉树的度为 2　　B）一棵二叉树的度可以小于 2

C）二叉树中至少有一个结点的度为 2　D）二叉树中任何一个结点的度都为 2

【答案】B

3．二叉树的第 I 层上最多含有结点数为（　　）

A）2^I　　B）$2^{I-1}-1$ C）2^{I-1}　　D）2^I-1

【答案】C

4．具有 10 个叶结点的二叉树中有（　）个度为 2 的结点

A）8　　B）9　　C）10　　D）11

【答案】B

5．在下述结论中，正确的是（　）

①只有一个结点的二叉树的度为 0；

②二叉树的度为 2；

③二叉树的左右子树可任意交换；

④深度为 K 的完全二叉树的结点个数小于或等于深度相同的满二叉树。

A）①②③　B）②③④　　C）②④　　D）①④

【答案】D

6．由 3 个结点可以构造出多少种不同的二叉树？（　）

A）2　　B）3　　C）4　　D）5

【答案】D

7．引入二叉线索树的目的是（　）

A）加快查找结点的前驱或后继的速度

B）为了能在二叉树中方便的进行插入与删除

C）为了能方便的找到双亲

D）使二叉树的遍历结果惟一

【答案】A

8．有 n 个叶子的哈夫曼树的结点总数为（　）

A）不确定　　B）2n　　C）2n+1 D）2n-1

【答案】D

9．一棵非空的二叉树的先序遍历序列与后序遍历序列正好相反，则该二叉树一定满足（　）

A）所有的结点均无左孩子　　B）所有的结点均无右孩子

C）只有一个叶子结点　　D）是任意一棵二叉树

【答案】C

【解析】先序序列是“根左右”，后序序列是“左右根”，若要这两个序列相反，只有单支树，单支树的特点是只有一个叶子结点或高度等于其结点数，故选 C。

10．一棵完全二叉树上有 1001 个结点，其中叶子结点的个数是（　）

A）250　　B）500　　C）505　　D）以上答案都不对

【答案】D

【解析】若每个结点均已经编号，则最大的编号为 1001，其父亲结点的编号为 500，那么从 501 到 1001 均为叶子结点。因此，叶子结点数为 1001-500=501。故答案为 D。

11．已知一棵二叉树的后序遍历序列为 DABEC，中序遍历序列为 DEBAC，则它的先序遍历序列为（　）

A）ACBED B）DECAB　　C）DEABC　　D）CEDBA

【答案】D

【解析】依据后序遍历序列可确定根结点为 C；再依据中序遍历序列可知其左子树由 DEBA 构成，右子树为空；又由左子树的后序遍历序列可知其根结点为 E，由中序遍历序列可知 E 的左子树为 D，右子树由 BA 构成，所以求得该二叉树的先序遍历序列为选项 D）。

12．若一棵二叉树具有 10 个度为 2 的结点，5 个度为 1 的结点，则度为 0 的结点个数是（　）

A）9　　B）11　　C）15　　D）不确定

【答案】B

13．利用二叉链表存储树时，根结点的右指针是（　）

A）指向最左孩子　　B）指向最右孩子 C）空　　D）非空

【答案】C

【解析】利用二叉链表存储树时，即用孩子兄弟链表存储树，根结点的左指针指向其第一子女，根结点的右指针指向其下一兄弟，所以为空。

14．设森林 F 中有三棵树，第一，第二，第三棵树的结点个数分别为 M_1，M_2 和 M_3。与森林 F 对应的二叉树根结点的右子树上的结点个数是（　）

A）M_1　　B）M_1+M_2　　C）M_3　　D）M_2+M_3

【答案】D

【解析】当森林转化为对应的二叉树时，二叉树的根结点及其左子树是由森林的第一棵树转化而来，二叉树的右子树是由森林的其余树转化而来。

15．若 X 是中序线索二叉树中一个有左孩子的结点，且 X 不为根，则 X 的前驱为（　）

A）X 的双亲　　B）X 的右子树中最左的结点

C）X 的左子树中最右结点　　D）X 的左子树中最右叶结点

【答案】C

16．n 个结点的线索二叉树上含有的线索数为（　）

A）2n　　B）n-l　　C）n+l　　D）n

【答案】C

【解析】线索二叉树是利用二叉树的空链域加上线索，n 个结点的二叉树有 n+1 个空链域。

17．在一棵高度为 k 的满二叉树中，结点总数为（　）

A）2^{k-1}　　B）2^k　　C）2^k-1　　D）$\lfloor \log 2^k \rfloor+1$

【答案】C

18．一棵树高为 K 的完全二叉树至少有（　）个结点

A）2^k-1　　B）$2^{k-1}-1$ C）2^{k-1}　　D）2^k

【答案】C

6.2　填空题

1．在二叉树中，指针 p 所指结点为叶子结点的条件是_____。

【答案】p->lchild==NULL && p->rchlid==NULL

2．深度为 H 的完全二叉树至少有_____个结点；至多有_____个结点；H 和结点总数 N 之间的关系是_____。

【答案】（1）2^{H-1}　　（2）2^H-1　　（3）$H=\lfloor \log_2 N \rfloor+1$

3．一棵有 n 个结点的满二叉树有_____个度为 1 的结点，有_____个分支（非终端）结点和_____个叶子，该满二叉树的深度为_____。

【答案】（1）0　　（2）(n-1）/2　　（3）(n+1）/2　　（4）$\lfloor \log_2 n \rfloor+1$

4．对于一个具有 n 个结点的二叉树，当它为一棵_____时，具有最小高度，当它为一棵_____时，具有最大高度。

【答案】（1）完全二叉树　　（2）单支树，树中任一结点（除最后一个结点是叶子外），只有左子女或只有右子女。

5．在一棵二叉树中，度为 0 的结点的个数为 N_0，度为 2 的结点的个数为 N_2，则有 N_0=_____

【答案】N_2+1

6．已知二叉树有 50 个叶子结点，则该二叉树的总结点数至少是_____。

【答案】99

【解析】在二叉树中，$N_0 = N_2+1$，所以，有 50 个叶子结点的二叉树，有 49 个度为 2 的结点。若要使该二叉树的结点数最少，度为 1 的结点应为 0 个，即总结点数 $N= N_0 +N_1+ N_2 =99$。

7．具有 n 个结点的满二叉树，其叶结点的个数是_____。

【答案】(n+1）/2

8．每一棵树都能惟一的转换为它所对应的二叉树。若已知一棵二叉树的先序序列是 BEFCGDH，中序序列是 FEBGCHD，则它的后序序列是_____。设上述二叉树是由某森林转换而成，则其第一棵的先根次序序列是_____。

【答案】

（1）FEGHDCB

（2）BEF（该二叉树转换成森林，含三棵树，其第一棵树的先根次序是 BEF）

9．已知一棵二叉树的先序序列为　ABDECFHG，中序序列为　DBEAHFCG，则该二叉树的根为_____，左子树中有_____，右子树中有_____。

【答案】（1）A　（2）DBE　（3）HFCG

10．先根次序周游树林正好等同于按_____周游对应的二叉树；后根次序周游树林正好等同于_____周游对应的二叉树。

【答案】（1）先根次序　　（2）中根次序

11．一个深度为 k 的，具有最少结点数的完全二叉树按层次，（同层次从左到右）用自然数依此对结点编号，则编号最小的叶子的序号是_____；编号是 i 的结点所在的层次号是_____（根所在的层次号规定为 1 层）。

【答案】（1）$2^{k-2}+1$　（2）$\lfloor \log_2 i \rfloor+1$

【解析】第 k 层 1 个结点，总结点个数是 2^{k-1}，其双亲是 $2^{k-1}/2=2^{k-2}$。

12．某二叉树有 20 个叶子结点，有 30 个结点仅有一个孩子，则该二叉树的总结点数为_____。

【答案】69

【解析】在二叉树中，$N_0 = N_2+1$，所以，有 20 个叶子结点的二叉树，有 19 个度为 2 的结点。又已知该二叉树中度为 1 的结点有 30 个，则总结点数 $N= N_0 +N_1+ N_2 =69$。

13．有数据 WG={7，19，2，6，32，3，21，10}，则所建 Huffman 树的树高是_____，带权路径长度 WPL 为_____。

【答案】（1）6　　（2）261

14．有一份电文中共使用 6 个字符：a，b，c，d，e，f，它们的出现频率依次为 2，3，4，7，8，9，试构造一棵哈夫曼树，则其加权路径长度 WPL 为_____，字符 c 的编码是_____。

【答案】（1）80　　（2）001（不惟一）

15．具有 N 个结点的二叉树，采用二叉链表存储，共有_____个空链域。

【答案】N+1

【解析】在二叉树中，$N= N_0 +N_1+ N_2$，$N_0 = N_2+1$，空分支数为 $2N_0+N_1 = N_0 +N_1+ (N_2+1) = N+1$。

16．8 层完全二叉树至少有_____个结点，拥有　100 个结点的完全二叉树的最大层数为_____。

【答案】（1）128（第 7 层满，加第 8 层 1 个）　　（2）7

17．一棵树 T 中，包括一个度为 1 的结点，两个度为 2 的结点，三个度为 3 的结点，四个度为 4 的结点和若干叶子结点，则 T 的叶结点数为_____。

【答案】21

【解析】已知该树中结点数和分支数的关系分别如下：

$N= N_0 +N_1+ N_2+N_3+ N_4$　　（1）

$N-1= N_1+ 2N_2+3N_3+ 4N_4$　　（2）

由（2）式求得 N=（1+2*2+3*3+4*4）+1=31，再由（1）式求得 N_0=21。

18． n（n 大于 1）个结点的各棵树中，其深度最小的那棵树的深度是 ＿＿＿＿＿＿。它共有＿＿＿＿＿＿个叶子结点和＿＿＿＿＿＿个非叶子结点，其中深度最大的那棵树的深度是＿＿＿＿＿＿，它共有＿＿＿＿＿＿个叶子结点和＿＿＿＿＿＿个非叶子结点。

【答案】（1）2 （2）n-1 （3）1 （4）n （5）1 （6）n-1

19．设 y 指向中序二叉线索树的一叶子，x 指向一待插入结点，现 x 作为 y 的左孩子插入，树中标志域为 ltag 和 rtag，并规定标志为 1 是线索，则下面的一段算法将 x 插入并修改相应的线索，试补充完整：(lchild，rchild 分别代表左，右孩子)

x->ltag= ＿＿＿＿＿＿;
x->lchild=＿＿＿＿＿＿;
y->ltag= ＿＿＿＿＿＿;
y->lchild= ＿＿＿＿＿＿;
x->rtag= ＿＿＿＿＿＿;
x->rchild= ＿＿＿＿＿＿;
if (x->lchild!=NULL) && (x->lchild->rtag==1) x->lchild->rchild=＿＿＿＿＿＿;

【答案】（1）1 （2）y->lchild （3）0 （4）x
（5）1 （6）y （7）x

6.3 判断题

1．二叉树是度为 2 的有序树（ ）

【答案】×

2．完全二叉树一定存在度为 1 的结点（ ）

【答案】×

3．深度为 K 的二叉树中结点总数≤2^k-1（ ）

【答案】√

4．由一棵二叉树的先序序列和后序序列可以惟一确定它（ ）

【答案】×

5．完全二叉树中，若一个结点没有左孩子，则它必是树叶（ ）

【答案】√

6．用二叉链表存储 n 个结点的二叉树时，结点的 2n 个指针中有 n+1 个空指针（ ）

【答案】√

7．完全二叉树的存储结构通常采用顺序存储结构（ ）

【答案】√

8．哈夫曼树是带权路径长度最短的树，路径上权值较大的结点离根较近（ ）

【答案】√

9．在中序线索二叉树中，每一非空的线索均指向其祖先结点（ ）

【答案】√

【解析】在二叉树上，对有左右子女的结点，其中序前驱是其左子树上按中序遍历的最右边的结点（该结点的后继指针指向祖先），中序后继是其右子树上按中序遍历的最左边的结点（该结点的前驱指针指向祖先）。

10．二叉树中序线索化后，不存在空指针域（ ）

【答案】×

【解析】非空二叉树中序遍历第一个结点无前驱，最后一个结点无后继，这两个结点的前驱线索和后继线索为空指针。

6.4 应用题

1．从概念上讲，树，森林和二叉树是三种不同的数据结构，将树，森林转化为二叉树的基本目的是什么，并指出树和二叉树的主要区别。

【答案】树的孩子兄弟链表表示法和二叉树二叉链表表示法，本质是一样的，只是解释不同，也就是说树（树是森林的特例，即森林中只有一棵树的特殊情况）可用二叉树惟一表示，并可使用二叉树的一些算法去解决树和森林中的问题。

树和二叉树的区别有 3：一是二叉树的度至多为 2，树无此限制；二是二叉树有左右子树之分，即使在只有一个分支的情况下， 也必须指出是左子树还是右子树，树无此限制；三是二叉树允许为空，树一般不允许为空（个别书上允许为空）。

2．若在内存中存放一个完全二叉树，在二叉树上只进行下面两个操作：

（1）寻找某个结点双亲；　　（2）寻找某个结点的子女；

请问应该用何种结构来存储该二叉树？

【答案】用顺序存储结构存储 n 个结点的完全二叉树。编号为 i 的结点，其双亲编号是⌊i/2⌋(i=1 时无双亲)，其左子女是 2i(若 2i≤n，否则 i 无左子女)，右子女是 2i+1(若 2i+1≤n，否则无右子女)。

3．求含有 n 个结点、采用顺序存储结构的完全二叉树中的序号最小的叶子结点的下标。要求写出简要步骤。

【答案】根据完全二叉树的性质，最后一个结点（编号为 n）的双亲结点的编号是⌊n/2⌋，这是最后一个分支结点，在它之后是第一个终端（叶子）结点，故序号最小的叶子结点的下标是⌊n/2⌋+1。

4．试证明，同一棵二叉树的所有叶子结点，在先序序列、中序序列以及后序序列中都按相同的相对位置出现（即先后顺序相同），例如先序 abc，后序 bca，中序 bac。

【答案】先序遍历是“根左右”，中序遍历是“左根右”，后序遍历是“左右根”。三种遍历中只是访问“根”结点的时机不同，对左右子树均是按左右顺序来遍历的，因此所有叶子都按相同的相对位置出现。

5．试找出满足下列条件的二叉树：

1）先序序列与后序序列相同；　2）中序序列与后序序列相同；

3）先序序列与中序序列相同；　4）中序序列与层次序列相同；

【答案】先序遍历二叉树的顺序是“根—左子树—右子树”，中序遍历“左子树—根—右子树”，后序遍历顺序是：“左子树—右子树－根”，根据以上原则，解答如下：

1）若先序序列与后序序列相同，则或为空树，或为只有根结点的二叉树。

2）若中序序列与后序序列相同，则或为空树，或为任一结点至多只有左子树的二叉树。

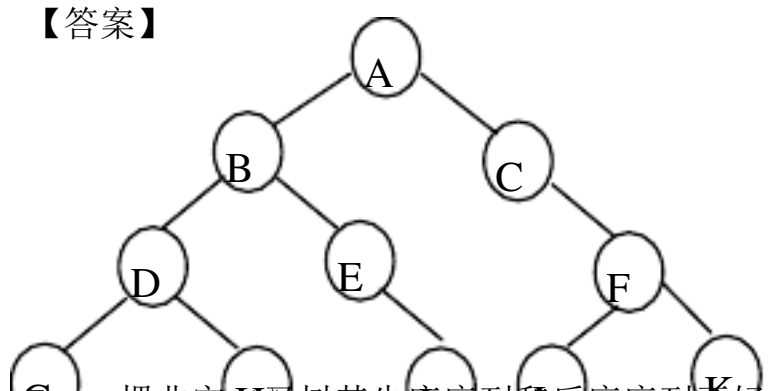
（3）若先序序列与中序序列相同，则或为空树，或为任一结点至多只有右子树的二叉树。

（4）若中序序列与层次遍历序列相同，则或为空树，或为任一结点至多只有右子树的二叉树

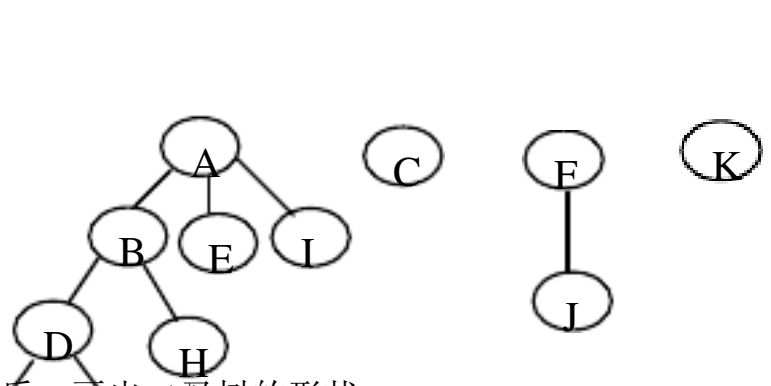
6．已知一棵二叉树的中序序列和后序序列分别为 GLDHBEIACJFK 和 LGHDIEBJKFCA

（1）给出这棵二叉树；（2）转换为对应的森林。

【答案】



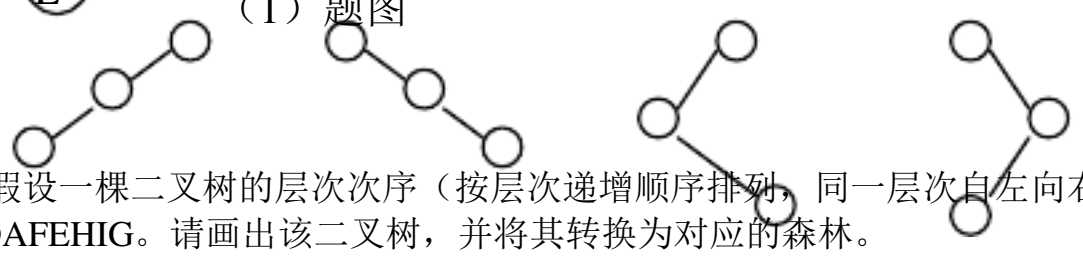
（1）题图



（2）题图

7．一棵非空二叉树其先序序列和后序序列正好相反，画出二叉树的形状。

【答案】先序序列是“根左右” 后序序列是“左右根”，可见对任意结点，若至多只有左子女或至多只有右子女，均可使先序序列与后序序列相反，图示如下：



8．假设一棵二叉树的层次次序（按层次递增顺序排列，同一层次自左向右）为 ABECFGDHI,中序序列为 BCDAFEHIG。请画出该二叉树，并将其转换为对应的森林。

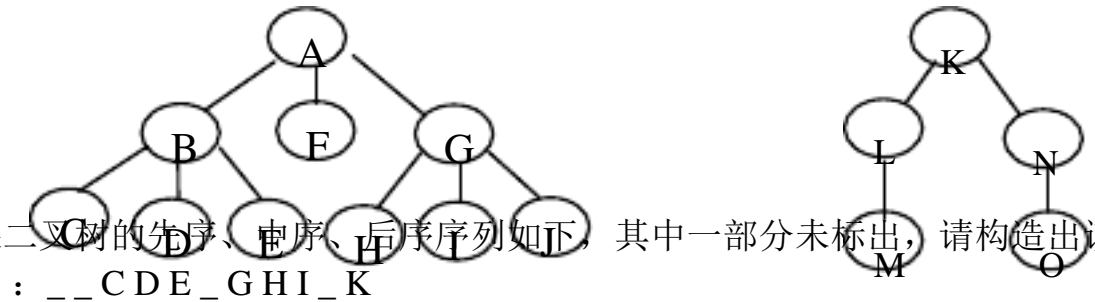
【答案】按层次遍历，第一个结点（若树不空）为根，该结点在中序序列中把序列分成左右两部分：左子树和右子树。若左子树不空，层次序列中第二个结点为左子树的根；若右子树为空，则层次序列中第三个结点为右子树的根。对右子树也作类似的分析。层次序列的特点是，从左到右每个结点或是当前情况下子树的根或是叶子。

9．已知一个森林的先序序列和后序序列如下，请构造出该森林。

先序序列：ABCDEFGHIJKLMNO

后序序列：CDEBFHIJGAMLONK

【答案】森林的先序序列和后序序列对应其转换的二叉树的先序序列和中序序列，应先据此构造二叉树，再构造出森林。



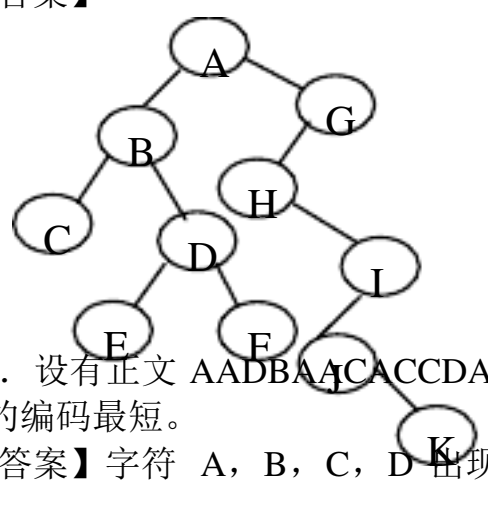
10．一棵二叉树的先序、中序、后序序列如下，其中一部分未标出，请构造出该二叉树。

先序序列：_ _ C D E _ G H I _ K

中序序列：C B _ _ F A _ J K I G

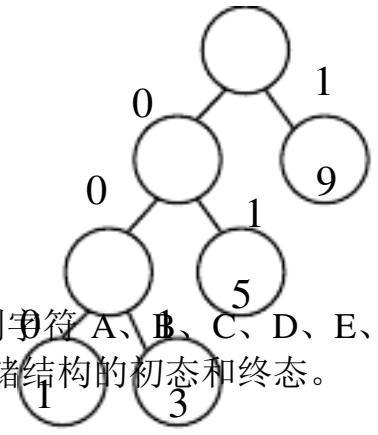
后序序列：_ E F D B _ J I H _ A

【答案】



11．设有正文 AADBAACACCDACACAAD，字符集为 A，B，C，D，设计一套二进制编码，使得上述正文的编码最短。

【答案】字符 A，B，C，D 出现的次数为 9，1，5，3。其哈夫曼编码如下： A:1，B:000，C:01，D:001。



12．已知下列字符 A、B、C、D、E、F、G 的权值分别为 3、12、7、4、2、8，11，试填写出其对应哈夫曼树 HT 的存储结构的初态和终态。

【答案】

初态图

	字符	weight	parent	lch	rch
1	A	3	0	0	0
2	B	12	0	0	0
3	C	7	0	0	0
4	D	4	0	0	0
5	E	2	0	0	0
6	F	8	0	0	0
7	G	11	0	0	0
8			0	0	0
9			0	0	0
10			0	0	0
11			0	0	0
12			0	0	0
13			0	0	0

终态图

	字符	weight	parent	lch	rch
1	A	3	8	0	0
2	B	12	12	0	0
3	C	7	10	0	0
4	D	4	9	0	0
5	E	2	8	0	0
6	F	8	10	0	0
7	G	11	11	0	0
8		5	9	5	1
9		9	11	4	8
10		15	12	3	6
11		20	13	9	7
12		27	13	2	10
13		47	0	11	12

6.5　算法设计题

1．已知一棵二叉树按顺序方式存储在数组 A[1..n]中。设计算法，求出下标分别为 i 和 j 的两个结点的最近的公共祖先结点的值。

【算法分析】

二叉树顺序存储，是按完全二叉树的格式存储，利用完全二叉树双亲结点与子女结点编号间的关系，求下标为 i 和 j 的两结点的双亲，双亲的双亲…等等，直至找到最近的公共祖先。

【算法源代码】

```
typedef  int  ElemType;
void Ancestor(ElemType A[],int n,int i,int j)
{while(i!=j)
if(i>j) i=i/2;              /*下标为 i 的结点的双亲结点的下标*/
else j=j/2;                 /*下标为 j 的结点的双亲结点的下标*/
printf("所查结点的最近公共祖先的下标是%d,值是%d",i,A[i]);
}/* Ancestor*/
```

2．编程求以孩子兄弟表示法存储的森林的叶子结点数，要求描述结构。

【算法分析】

当森林（树）以孩子兄弟表示法存储时，若结点没有左子树（ fch=NULL），则它必是叶子，否则，总的叶子结点个数是左子树（fch）上的叶子数和右子树（nsib）上叶结点个数之和。

【算法源代码】

```
typedef  struct  node{
  ElemType  data;                          /*数据域*/
  struct  node *fch, *nsib;                /*孩子与兄弟域 */
}*Tree;
int Leaves (Tree  t){/*计算以孩子-兄弟表示法存储的森林的叶子数*/
  if(t)
   if(t->fch==NULL)     /*若结点无孩子，则该结点必是叶子*/
       return(1+Leaves(t->nsib));
   else
       return (Leaves(t->fch)+Leaves(t->nsib));
}/*结束 Leaves*/
```

3．假定用两个一维数组 L[N]和 R[N]作为有 N 个结点 1，2,…， N 的二叉树的存储结构。L[i]和 R[i]分别指示结点 i 的左子女和右子女；L[i]=0（R[i]=0）表示 i 的左（右）子女为空。设计一个算法，由 L 和 R 建立一个一维数组 T[n]，使 T[i]存放结点 i 的父亲；然后再写一个判别结点 U 是否为结点 V 的后代的算法。

【算法分析】

由指示结点 i 左子女和右子女的两个一维数组 L[i]和 R[i]，很容易建立指示结点 i 的双亲的一维数组 T[i]，根据 T 数组，判断结点 U 是否是结点 V 后代的算法，转为判断结点 V 是否是结点 U 的祖先的问题。

【算法源代码】

```
int generation (int u, int v, int n, int l[],int r[],int t[]){
/*l[]和 r[]是含有 n 个元素且指示二叉树结点 i 左子女和右子女的一维数组本算法据此建立
结点 i 的双亲数组 t，并判断结点 u 是否是结点 v 的后代*/
  int parent,i;
  for(i=1;i<=n;i++)  t[i]=0;  /*t 数组初始化*/
  for (i=1;i<=n;i++){        /*根据 l 和 r 填写 t*/
     if(l[i]!=0) t[l[i]]=i;   /*若结点 i 的左子女是 l，则结点 l 的双亲是结点 i*/
     if (r[i]!=0) t[r[i]]=i;  /*i 的右子女是 r，则 r 的双亲是 i*/
    parent=u;          /*判断 u 是否是 v 的后代*/
    while (parent!=v && parent!=0) parent=t[parent];
    if (parent==v)
          {printf("结点 u 是结点 v 的后代");return (1);}
   else
         { printf("结点 u 不是结点 v 的后代");return(0);}
    }
```

```
}/*结束 generation*/
```

4．要求二叉树按二叉链表形式存储：

（1）写一个建立二叉树的算法。

（2）写一个判别给定的二叉树是否是完全二叉树的算法。

【算法分析】

二叉树是递归定义的，以递归方式建立最简单。判定是否是完全二叉树，可以使用队列，在遍历中利用完全二叉树“若某结点无左子女就不应有右子女”的原则进行判断。

【算法源代码】

```
BiTree Creat() { /*建立二叉树的二叉链表形式的存储结构*/
      ElemType x;
      BiTree bt;
      scanf("%d",&x);
      if(x==0) bt=NULL;
      else if(x>0)
        {bt=(BiTNode *)malloc(sizeof(BiTNode));
         bt->data=x; bt->lchild=creat(); bt->rchild=creat(); }
      return(bt);
}/*结束 creat*/
int JudgeComplete(BiTree bt){
/*判断二叉树是否是完全二叉树，如是，返回 1，否则，返回 0*/
int tag=0;
BiTree p=bt, Q[50]; /* Q 是队列，元素是二叉树结点指针，容量足够大*/
if(p==NULL) return 1;
QueueInit(Q);
QueueIn(Q,p); /*初始化队列，根结点指针入队*/
while(!QueueEmpty(Q)){
  p=QueueOut(Q);
  if (p->lchild && !tag) QueueIn(Q,p->lchild); /*左子女入队*/
    else  if (p->lchild) return 0;   /*前边已有结点为空，本结点不空*/
        else tag=1;                                    /*首次出现结点为空*/
  if (p->rchild && !tag) QueueIn(Q,p->rchild); /*右子女入队*/
    else if (p->rchild) return 0;
        else tag=1;
  }/*while*/
 return 1;
} /*JudgeComplete*/
```

5．有 n 个结点的完全二叉树存放在一维数组 A[1..n]中，试据此建立一棵用二叉链表表示的二叉树，根由 tree 指向。

【算法源代码】

```
typedef  int  ElemType;
BiTree Creat(ElemType A[],int i){
 BiTree tree;
 if (i<=n) {
   tree=(BiTree)malloc(sizeof(BiNode));
   tree->data=A[i];
   if(2*i>n) tree->lchild=NULL;
        else    tree->lchild=Creat(A,2*i);
   if(2*i+1>n) tree->rchild=NULL;
        else    tree->rchild=Creat(A,2*i+1);
 }
 return (tree);
}/*Creat*/
```

6．编写递归算法，在二叉树中求位于先序序列中第 k 个位置的结点的值。

【算法源代码】

```
int c,k; /*这里把 k 和计数器 c 作为全局变量处理 */
```

```
void Get_PreSeq(BiTree T) /*求先序序列为 k 的结点的值 */
{ if(T)
  { c++; /*每访问一个子树的根都会使前序序号计数器加 1 */
    if(c==k)    { printf("Value is %d\n",T->data);   return;    }
    else
    { Get_PreSeq(T->lchild); /*在左子树中查找 */
      Get_PreSeq(T->rchild); /*在右子树中查找 */
    }
  }/*if */
}/*Get_PreSeq */
```

7．假设以双亲表示法作树的存储结构，写出双亲表示的类型说明，并编写求给定的树的深度的算法。（注：已知树中结点数）

【算法分析】

由于以双亲表示法作树的存储结构，找结点的双亲容易。因此我们可求出每一结点的层次，取其最大层次就是树的深度。对每一结点，找其双亲，双亲的双亲，直至（根）结点双亲为 0 为止。

【算法源代码】

```
int Depth(PTree t){
int maxdepth=0;   int i,f,temp;
for(i=1;i<=t.n;i++){
   temp=0; f=i;
   while(f>0)  {temp++; f=t.nodes[f].parent; }    /* 深度加 1，并取新的双亲*/
   if(temp>maxdepth)  maxdepth=temp; } /*最大深度更新*/
   }
return(maxdepth);/*返回树的深度*/
} /*结束 Depth*/
```

8．二叉树采用二叉链表存储

（1）编写计算整个二叉树高度的算法（二叉树的高度也叫二叉树的深度）。

（2）编写计算二叉树最大宽度的算法（二叉树的最大宽度是指二叉树所有层中结点个数的最大值）。

【算法分析】

二叉树是递归定义的，其运算最好采取递归方式。求最大宽度可采用层次遍历的方法，记下各层结点数，每层遍历完毕，若结点数大于原先最大宽度，则修改最大宽度。

【算法源代码】

```
int Height(BiTree bt)  /*求二叉树 bt 的深度*/
{int hl,hr;
 if (bt==NULL) return(0);
  else {
   hl=Height(bt->lchild);
   hr=Height(bt->rchild);
   if(hl>hr) return (hl+1);
        else return(hr+1);}
 }
int Width(BiTree bt)/*求二叉树 bt 的最大宽度*/
{if (bt==NULL) return (0);  /*空二叉树宽度为 0*/
else
{BiTree p,Q[50];/*Q 是队列，元素为二叉树结点指针，容量足够大*/
 int front=1,rear=1,last=1;
 int temp=0, maxw=0;      /*temp 记局部宽度, maxw 记最大宽度*/
 Q[rear]=bt;              /*根结点入队列*/
 while(front<=last)
  {p=Q[front++]; temp++; /*同层元素数加 1*/
    if (p->lchild!=NULL)  Q[++rear]=p->lchild;   /*左子女入队*/
   if (p->rchild!=NULL)  Q[++rear]=p->rchild;   /*右子女入队*
    if (front>last)     /*一层结束*/
```

```
    {last=rear;       /*last 指向下层最右元素, 更新当前最大宽度*/
    if(temp>maxw) maxw=temp;
     temp=0;}/*if*/
  }/*while*/
  return (maxw);
  }
  }/*结束 width*/
```

9．设一棵二叉树以二叉链表为存贮结构，设计一个算法将二叉树中所有结点的左，右子树相互交换。

【算法源代码】

```
void exchange(BiTree bt)/*将二叉树 bt 所有结点的左右子树交换
{ BiTree p;
 if(bt){
      p=bt->lchild;    bt->lchild=bt->rchild; bt->rchild=p; /*左右子女交换
      exchange(bt->lchild);  /*交换左子树上所有结点的左右子树
      exchange(bt->rchild);  /*交换右子树上所有结点的左右子树
     }
 }
```

【算法讨论】

将上述算法中两个递归调用语句放在前面，将交换语句放在最后，则是以后序遍历方式交换所有结点的左右子树。中序遍历不适合本题。

10．已知一棵高度为 K 具有 n 个结点的二叉树，按顺序方式存储。

（1）编写用先根遍历二叉树中每个结点的递归算法；

（2）编写将树中最大序号叶子结点的祖先结点全部打印输出的算法。

【算法分析】

高度为 K 的二叉树，按顺序方式存储，要占用 2^K-1 个存储单元，与实际结点个数 n 关系不大，对不是完全二叉树的二叉树，要增加“虚结点”，使其在形态上成为完全二叉树。注意：二叉树中最大序号的叶子结点，是在顺序存储方式下编号最大的结点。

【算法源代码】

```
int m=2^K-1;  /*全局变量*/
void PreOrder(ElemType  bt[], int i )
{if (i<=m)     /*设虚结点以 0 表示*/
 {printf("%3c",bt[i]);  /*访问根结点*/
  if(2*i<=m && bt[2*i]!=0) PreOrder(bt,2*i);     /*先序遍历左子树*/
  if(2*i+1<=m && bt[2*i+1]!=0) PreOrder(bt,2*i+1);/*先序遍历右子树*/
  }
}/*结束 PreOrder*/
void Ancesstor(ElemType bt[]){
/*打印最大序号叶子结点的全部祖先*/
  c=m;
  while(bt[c]==0) c--;  /*找最大序号叶子结点，该结点存储时在最后*/
  f=c/2;      /*c 的双亲结点 f*/
  while(f!=0){
   /*从结点 c 的双亲结点直到根结点，路径上所有结点均为祖先结点*/
   printf("%3c",bt[f]);
   f=f/2;
   }/*逆序输出，最老的祖先最后输出*/
}/*结束*/
```

11．编写递归算法，对于二叉树中每一个元素值为 X 的结点，删去以它为根的子树，并释放相应的空间。

【算法源代码】

```
void Del_Sub(BiTree T)/*删除子树 T */
{ if(T->lchild) Del_Sub(T->lchild);
```

```
  if(T->rchild) Del_Sub(T->rchild);
  free(T);
 }/*Del_Sub */
 void Del_Sub_x(BiTree T,int x)/*删除所有以元素 x 为根的子树*/
 { if(T->data==x) Del_Sub(T); /*删除该子树 */
   else
   {if(T->lchild) Del_Sub_x(T->lchild,x);
    if(T->rchild) Del_Sub_x(T->rchild,x); /*在左右子树中继续查找 */
   }/*else */
 }/*Del_Sub_x */
```

12．设计一个算法，在中序全线索二叉树中，查找给定结点 * p 在中序序列中的后继（二叉树的根结点指针并未给出），并中序遍历该线索二叉树。

【算法源代码】

```
BiThrTree  insucc(BiThrTree p)/*找 p 结点的后继*/
{ if(p->rtag==1) /*后继线索*/
      return p->rchild;
  else /*右子树的最左下结点*/
     {  p=p->rchild;
        while(p->ltag==0)  p=p->lchild;
        return p; }
}
void  inorder_thr(BiThrTree  T)
{  BiThrTree  p=T;
   if(p)
   { while(p->ltag==0)  p=p->lchild;/*找中序遍历的第 1 个结点*/
     do{   printf("%3c", p->data);
           p=insucc(p);/*找当前结点的后继结点*/
       }while(p);
   }
}
```

第 7 章　图

7.1　选择题

1．对于一个具有 n 个顶点和 e 条边的有向图，在用邻接表表示图时，拓扑排序算法时间复杂度为（　　）

A） O(n)　　B） O(n+e)　　C） O(n*n)　　D） O(n*n*n)

【答案】B

2．设无向图的顶点个数为 n，则该图最多有（　　）条边。

A）n-1　　B）n(n-1)/2　　C） n(n+1)/2　　D）n^2

【答案】B

3．连通分量指的是（　　）

A） 无向图中的极小连通子图

B） 无向图中的极大连通子图

C） 有向图中的极小连通子图

D） 有向图中的极大连通子图

【答案】B

4．n 个结点的完全有向图含有边的数目（　　）

A）n*n　　B）n（n+1）　　C）n/2　　D）n*（n-1）

【答案】D

5．关键路径是（　　）

A） AOE 网中从源点到汇点的最长路径

B） AOE 网中从源点到汇点的最短路径

C） AOV 网中从源点到汇点的最长路径

D） AOV 网中从源点到汇点的最短路径

【答案】A

6．有向图中一个顶点的度是该顶点的（　　）
A）入度　　B） 出度　　C） 入度与出度之和　　D）（入度+出度）/2
【答案】C
7．有 e 条边的无向图，若用邻接表存储，表中有（　　）边结点。
A） e　　B） 2e　　C） e-1　　D） 2(e-1)
【答案】B
8．实现图的广度优先搜索算法需使用的辅助数据结构为（　　）
A） 栈　　B） 队列　　C） 二叉树　　D） 树
【答案】B
9．实现图的非递归深度优先搜索算法需使用的辅助数据结构为（　　）
A） 栈　　B） 队列　　C） 二叉树　　D） 树
【答案】A
10．存储无向图的邻接矩阵一定是一个（　　）
A） 上三角矩阵　　B）稀疏矩阵　　C） 对称矩阵　　D） 对角矩阵
【答案】C
11．在一个有向图中所有顶点的入度之和等于出度之和的（　　）倍
A） 1/2　　B）1　　C） 2　　D） 4
【答案】B
12．在图采用邻接表存储时，求最小生成树的 Prim 算法的时间复杂度为（　　）
A） O(n)　　B） O(n+e)　　C） $O(n^2)$　　D） $O(n^3)$
【答案】B
13．下列关于 AOE 网的叙述中，不正确的是（　　）
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）所有的关键活动提前完成，那么整个工程将会提前完成
D）某些关键活动提前完成，那么整个工程将会提前完成
【答案】B
14．具有 10 个顶点的无向图至少有多少条边才能保证连通（　　）
A） 9　　B）10　　C） 11　　D） 12
【答案】A
15．在含 n 个顶点和 e 条边的无向图的邻接矩阵中，零元素的个数为（　　）
A） e　　B）2e　　C） n^2-e　　D）n^2-2e
【答案】D

7.2　填空题

1．无向图中所有顶点的度数之和等于所有边数的_____倍。
【答案】2
2．具有 n 个顶点的无向完全图中包含有_____条边，具有 n 个顶点的有向完全图中包含有_____条边。
【答案】（1）n(n-1)/2 （2） n(n-1)
3．一个具有 n 个顶点的无向图中，要连通所有顶点则至少需要_____条边。
【答案】n-1
4．假定一个图具有 n 个顶点和 e 条边，则采用邻接矩阵、邻接表表示时，其相应的空间复杂度分别为_____和_____。
【答案】（1）$O(n^2)$ （2） O(n+e)
5．对用邻接矩阵表示的图进行任一种遍历时，其时间复杂度为_____，对用邻接表表示的图进行任一种遍历时，其时间复杂度为_____。
【答案】（1）$O(n^2)$ （2） O(e)
6．对于一个具有 n 个顶点和 e 条边的有向图和无向图，在其对应的邻接表中，所含边结点分别为_____和_____条。
【答案】（1）e （2）2e
7．在有向图的邻接表和逆邻接表表示中，每个顶点的边链表中分别链接着该顶点的所有

_____和_____结点。

【答案】（1）出边　（2） 入边

8． 对于一个具有 n 个顶点和 e 条边的无向图，当分别采用邻接矩阵、邻接表表示时，求任一顶点度数的时间复杂度依次为_____和_____。

【答案】（1）O(n)　（2）O(e+n)

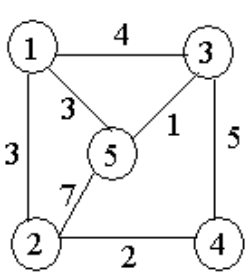
9．对于一个具有 n 个顶点和 e 条边的连通图，其生成树中的顶点数和边数分别为 _____和_____。

【答案】（1）n　（2） n-1

10．Prim 算法和 Kruscal 算法的时间复杂度分别为_____和_____。

【答案】（1）$O(n^2)$　（2）O(eloge)

11．针对右图所示的连通网络，试按如下格式给出在 Kruscal 算法构造最小生成树过程中顺序选出的各条边。



【答案】设边的信息表示为（始点，终点，权值），则在 Kruscal 算法构造最小生成树过程中顺序选出的各条边为：（3，5，1），（2，4，2），（1，5，3），（1，2，3）。

7.3　判断题

1．图是一种非线性结构，所以只能用链式存储。（　　）

【答案】×

2．图的最小生成树是唯一的。（　　）

【答案】×

3．如果一个图有 n 个顶点和小于 n-1 条边，则一定是非连通图。（　　）

【答案】√

4．有 n-1 条边的图一定是生成树。（　　）

【答案】×

5．用邻接矩阵表示图时，矩阵元素的个数与顶点个数相关，与边数无关。（　　）

【答案】√

6．用邻接表表示图时，顶点个数设为 n，边的条数设为 e，在邻接表上执行有关图的遍历操作时，时间代价为 O(n+e)。（　　）

【答案】√

7．逆邻接表只能用于有向图，邻接表对于有向图和无向图的存储都适用。（　　）

【答案】√

8．任何一个关键活动提前完成，那么整个工程将会提前完成。（　　）

【答案】×

9．在 AOE 网络中关键路径只有一条。（　　）

【答案】×

10．在 AOV 网络中如果存在环，则拓扑排序不能完成。（　　）

【答案】√

11．图的邻接矩阵存储是唯一的，邻接表存储也是唯一的。（　　）

【答案】×

12．假设一个有 n 个顶点和 e 条弧的有向图用邻接表表示，则删除与某个顶点 vi 相关的所有弧的时间复杂度是 O(n*e) 。（　　）

【答案】×

13．任意一个图都是其自身的子图。（　　）

【答案】√

14．一个无向连通图的生成树是含有该连通图的全部顶点的极大连通子图。（　　）

【答案】×

7.4　应用题

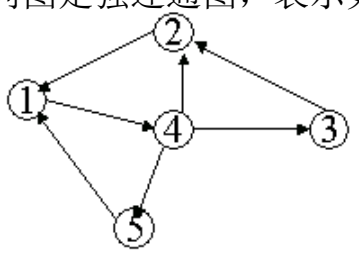
1．设有一有向图为 G=(V，E)。其中，V={ v1, v2, v3, v4, v5} ，E={<v2, v1>, <v3, v2>, <v4, v3>, <v4, v2>, <v1, v4>, <v4, v5>, <v5, v1>}，请画出该有向图并判断是否是强连通图。

分析：作该题的关键是弄清楚以下两点

（1）边集 E 中<vi, vj>表示一条以 vi 为弧尾，vj 为弧头的有向弧。

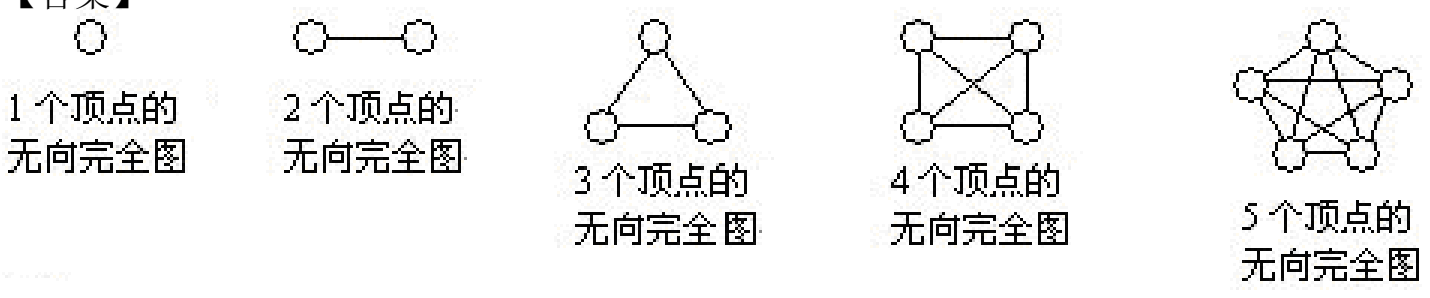
（2）强连通图是任意两顶点间都存在路径的有向图。

【答案】该有向图是强连通图，表示如下：



2．画出 1 个顶点、2 个顶点、3 个顶点、4 个顶点和 5 个顶点的无向完全图。并说明在 n 个顶点的无向完全图中，边的条数为 n(n-1)/2。

【答案】



【解析】因为在有 n 个顶点的无向完全图中，每一个顶点与其它任一顶点都有一条边相连，所以每一个顶点有 n-1 条边与其他顶点相连，则 n 个顶点有 n(n-1)条边。但在无向图中，顶点 i 到顶点 j 与顶点 j 到顶点 i 是同一条边，所以总共有 n(n-1)/2 条边。

3．对 n 个顶点的无向图 G，采用邻接矩阵表示，如何判别下列有关问题：

（1）图中有多少条边?

（2）任意两个顶点 i 和 j 是否有边相连?

（3）任意一个顶点的度是多少?

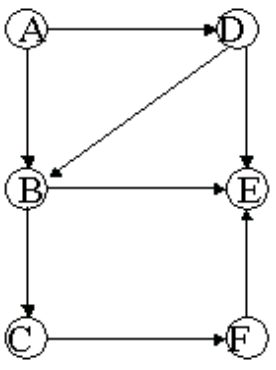
【答案】

（1）无向图的邻接矩阵是对称的，故它的边数应是上三角或下三角的非 0 元个数。

（2）邻接矩阵中如果第 i 行第 j 列的元素非 0 则表示顶点 i 与顶点 j 相连。

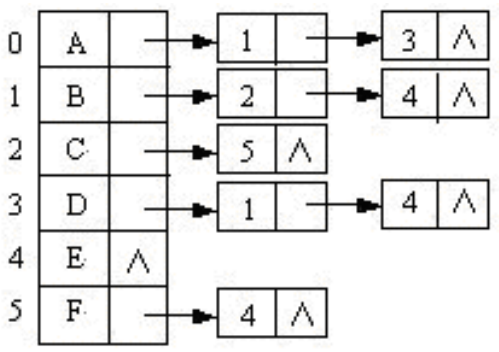
（3）任意一个顶点 vi 的度是第 i 行或第 i 列上非 0 元的个数。

4．熟悉图的存储结构，画出下面有向图的邻接矩阵、邻接表、逆邻接表、十字链表。写出邻接表表示的图从顶点 A 出发的深度优先遍历序列和广度优先遍历序列。

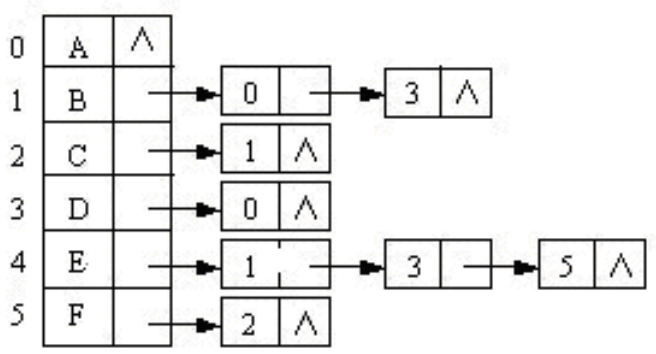


【答案】邻接矩阵如下：　　邻接表如下：

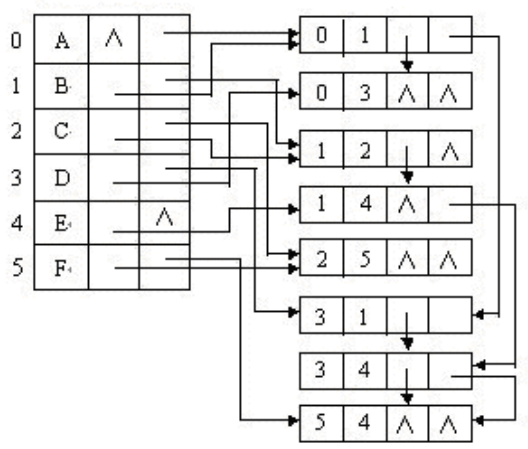
$$\begin{vmatrix} 0&1&0&1&0&0 \\ 0&0&1&0&1&0 \\ 0&0&0&0&0&1 \\ 0&1&0&0&1&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&1&0 \end{vmatrix}$$



逆邻接表如下：

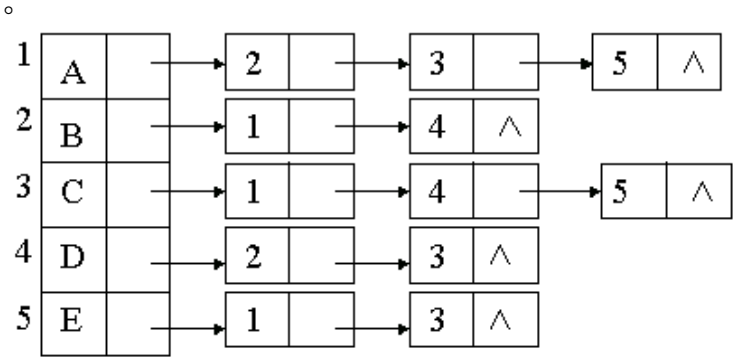


十字链表如下：



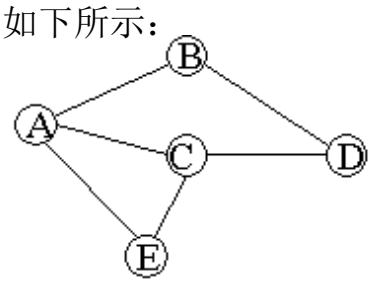
深度优先遍历序列为 ABCFED，广度优先遍历序列为 ABDCEF

5．已知下面是某无向图的邻接表，画出该无向图，并分别给出从 A 出发的深度优先搜索生成树和广度优先搜索生成树。

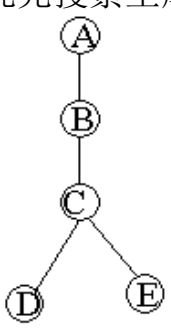


【解析】作该题的关键是弄清楚邻接表的概念，理解深度优先搜索和广度优先搜索的全过程以及二者的区别。

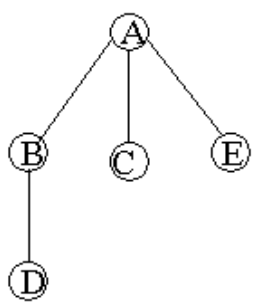
【答案】该无向图如下所示：



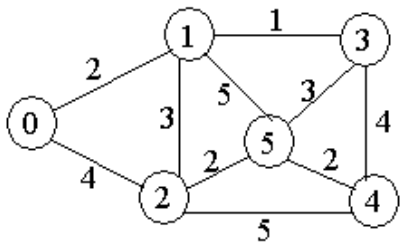
深度优先搜索生成树为：



广度优先搜索生成树为：

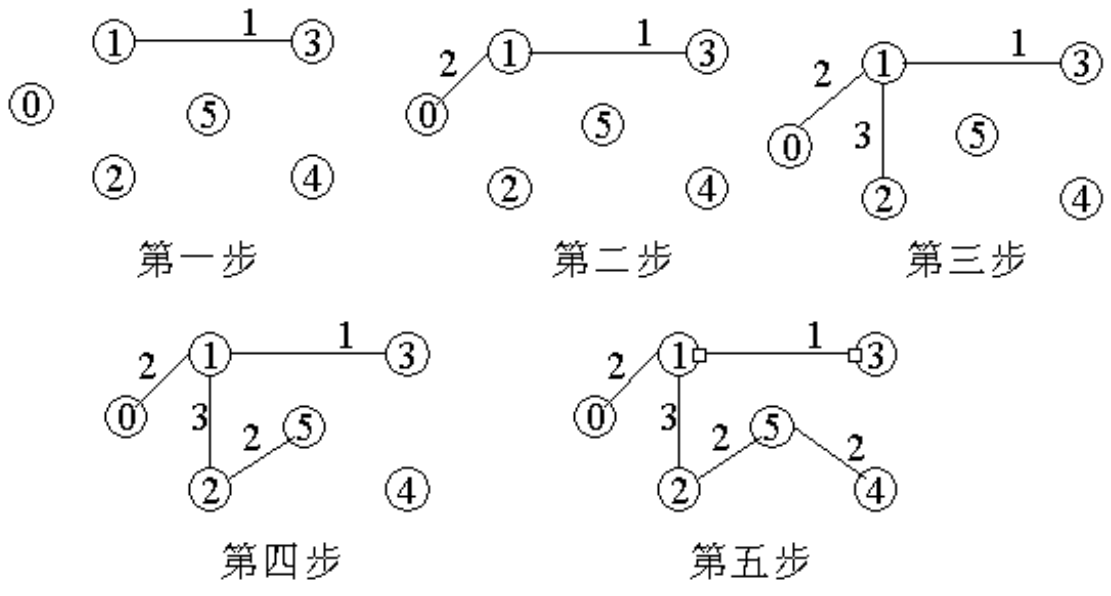


6．请分别用 Prim 算法和 Kruskal 算法构造以下网络的最小生成树，并求出该树的代价。



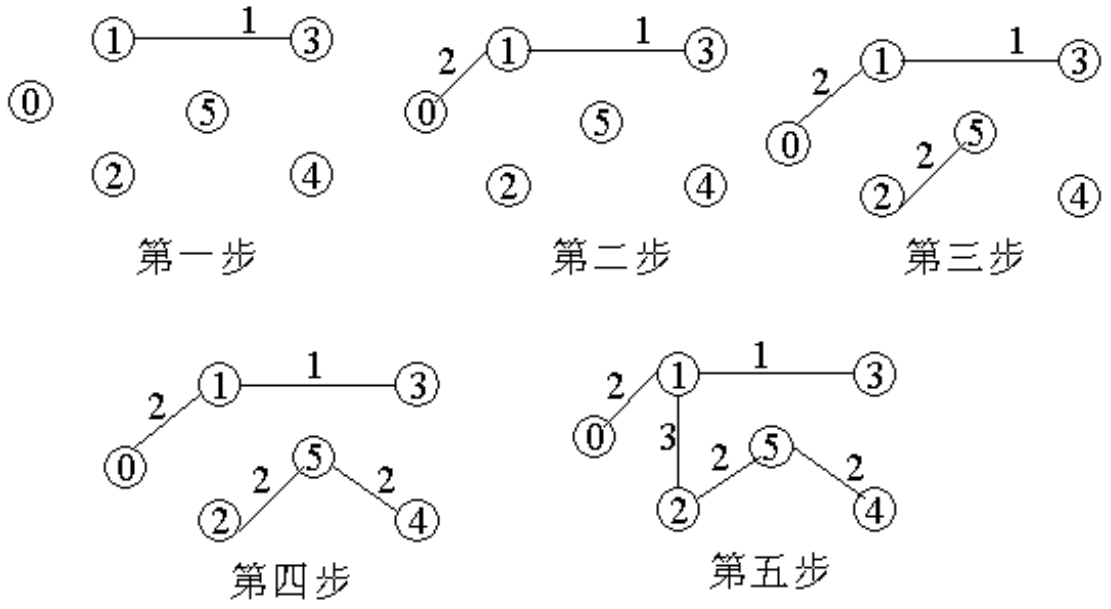
【解析】Prim 算法的操作步骤：首先从一个只有一个顶点的集合开始，通过加入与其中顶点相关联的最小代价的边来扩充顶点集，直到所有顶点都在一个集合中。

【答案】

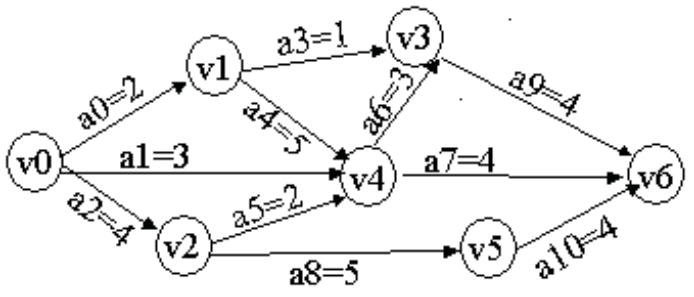


【解析】Kruscal 算法的操作步骤： 首先将 n 个顶点看成 n 个互不连通的分量，从边集中找最小代价的边，如果落在不同连通分量上，则将其加入最小生成树，直到所有顶点都在同一连通分量上。

【答案】



7． 写出求以下 AOE 网的关键路径的过程。要求：给出每一个事件和每一个活动的最早开始时间和最晚开始时间。



【解析】求关键路径首先求关键活动，关键活动 a_i 的求解过程如下

（1）求事件的最早发生时间 ve(j), 最晚发生时间 vl(j);

（2）最早发生时间从 ve(0)开始按拓扑排序向前递推到 ve(6), 最晚发生时间从 vl(6)按逆拓扑排序向后递推到 vl(0);

（3）计算 e(i),l(i)：设 a_i 由弧<j,k>表示，持续时间记为 dut<j,k>，则有下式成立

e(i)=ve(j)
l(i)=vl(k)-dut(<j,k>)

（4）找出 e(i)-l(i)=0 的活动既是关键活动。

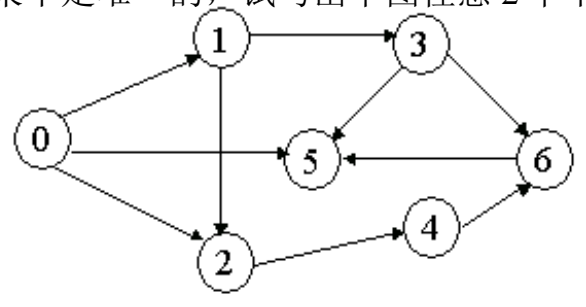
【答案】

顶点	ve	vl
V0	0	0
v1	2	2
v2	4	5
v3	10	10
v4	7	7
v5	9	10
v6	14	14

活动	e	l	l-e
a0	0	0	0
a1	0	4	4
a2	0	1	1
a3	2	9	7
a4	2	2	0
a5	4	5	1
a6	7	7	0
a7	7	10	3
a8	4	5	1
a9	10	10	0
a10	9	10	1

关键路径为：a0->a4->a6->a9

8． 拓扑排序的结果不是唯一的，试写出下图任意 2 个不同的拓扑序列。

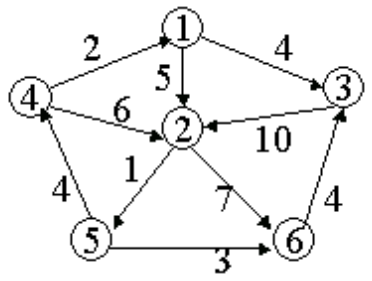


【解析】解题关键是弄清拓扑排序的步骤

（1）在 AOV 网中，选一个没有前驱的结点且输出；（2）删除该顶点和以它为尾的弧；（3）重复上述步骤直至全部顶点均输出或不再有无前驱的顶点。

【答案】（1）0132465　　（2）0123465

9．给定带权有向图 G 和源点 v1,利用迪杰斯特拉（Dijkstra）算法求从 v1 到其余各顶点的最短路径。



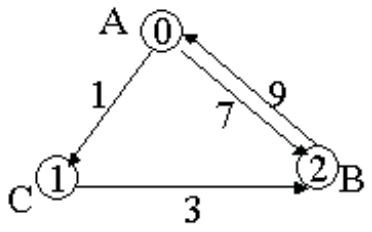
【解析】求解该题的关键是掌握迪杰斯特拉（Dijkstra）算法的设计原理----从一个顶点 v 到另一顶点 v_k 的最短路径或者是(v,v_k)或者是(v,v_j,v_k),它的长度或者是从 v 到 v_k 弧上的权值，或者是 D[j]与 v_j 到 v_k 弧上的权值之和，其中 D[j]是已经找到的从 v 到 v_j 的最短路径。

【答案】S 是已找到最短路径的终点的集合。

终点	从 v1 到各终点的 D 值和最短路径的求解				
	i=2	i=3	i=4	i=5	i=6
v2	5 (v1,v2)	5 (v1,v2)			
v3	4 (v1,v3)				
v4	32767	32767	32767	9 (v1,v2,v5,v4)	
v5	32767	32767	6 (v1,v2,v5)		

v6	32767	32767	12 (v1,v2,v6)	9 (v1,v2,v5,v6)	9 (v1,v2,v5,v6)
vj	v3	v2	v5	v4	v6
S	{v1,v3}	{v1,v2}	{v1,v2,v5}	{v1,v2,v5,v4}	{v1,v2,v5,v6}

10．利用 Floyd 算法求下图中各对顶点之间的路径。



【解析】Floyd 算法是依次添加顶点来修改相应路径，也就是说,若（vi，．．．，vk）和(vk，．．．，vj)分别是从 vi 到 vk 和从 vk 到 vj 的中间顶点的序号不大于 k-1 的最短路径，则将（vi，．．．vk，．．．，vj）和已经得到的从 vi 到 vj 且中间顶点序号不大于 k- 1 的最短路径相比较，其长度较短者便是从 vi 到 vj 的中间顶点的序号不大于 k 的最短路径。经过 n 次比较后必求得 vi 到 vj 的最短路径，依次，可求得各对顶点间的最短路径。

求解的关键是求解如下的一个 n 阶方阵序列：

$D^{(-1)}$，$D^{(0)}$，$D^{(1)}$，．．．，$D^{(k)}$，$D^{(n-1)}$

其中

$D^{(-1)}[i][j]=G.arcs[i][j]$

$D^{(k)}=Min\{D^{(k-1)}[i][j]，D^{(k-1)}[i][k]+D^{(k-1)}[k][j]\}$ $0\leqslant k\leqslant n-1$

【答案】

D	D(-1)			D(0)			D(1)			D(2)		
	0	1	2	0	1	2	0	1	2	0	1	2
0	0	1	7	0	1	7	0	1	4	0	1	4
1	∞	0	3	∞	0	3	∞	0	3	12	0	3
2	9	∞	0	9	10	0	9	10	0	9	10	0
P	P(-1)			P(0)			P(1)			P (2)		
	0	1	2	0	1	2	0	1	2	0	1	2
0		AC	AB		AC	AB		AC	ACB		AC	ACB
1			CB			CB			CB	CBA		CB
2	BA			BA	BAC		BA	BAC		BA	BAC	

每对顶点之间的最短路径及长度总结如下：

顶点 A 到顶点 C 最短路径为：A->C，长度为： 1

顶点 A 到顶点 B 最短路径为：A->C->B，长度为：4

顶点 C 到顶点 A 最短路径为：C->B->A，长度为：12

顶点 C 到顶点 B 最短路径为：C->B，长度为：3

顶点 B 到顶点 A 最短路径为：B->A，长度为：9

顶点 B 到顶点 C 最短路径为：B->A->C，长度为：10

7.5 算法设计题

1．设计一个算法，删除无向图的邻接矩阵中给定顶点。

【算法分析】

要在邻接矩阵中删除某顶点 i 主要操作有以下三步：（1）图的边数要减去与顶点 i 相关联的边的数目；（2）在邻接矩阵中删除第 i 行与 i 列,即把第 i+ 1 行到第 n 行依次前移，第 i+ 1 列到第 n 列依次前移。（3）图中顶点的个数-1。

【算法源代码】

```
void Delvi(MGraph *G,int i)/*在图　G 中删除顶点 I*/
{int num,j,k;
if(i<1||i>G->vexnum)
      {printf("error");  exit(0);}
else
{num=0;
 for(j=1;j<=G->vexnum;j++)
     {if(G->arcs[i][j]) num++;   if(G->arcs[j][i]) num++; }
 G->arcnum-=num;
 for(j=i+1;j<=G->vexnum;j++)
  for(k=1;k<=G->vexnum;k++)
       G->arcs[j-1][k]=G->arcs[j][k];
 for(j=i+1;j<=G->vexnum;j++)
  for(k=1;k<=G->vexnum-1;k++)
      G->arcs[k][j-1]=G->arcs[k][j];
 G->vexnum--;
 }
}
```

2．设计一个算法，求出分别用邻接矩阵和邻接表表示的有向图中顶点的最大出度值。

【算法分析】

用邻接矩阵表示的有向图中顶点的最大出度是该顶点所在行上非零元的个数。邻接表表示的有向图中顶点的最大出度值是该顶点所连接的单链表中结点的个数。

【算法源代码 1】

```
void Max_du(MGraph *G)
{int num,i,j; int max=0;
 for(i=1;i<=G->vexnum;i++)
  {num=0;
   for(j=1;j<=G->vexnum;j++)   if(G->arcs[i][j])  num++;
   if(max<num) max=num;
  }
  printf("The max degree is:%d\n",max);
}
```

【算法源代码 2】用邻接表表示

```
void Max_du(ALGraph *G)
{ArcPtr p; int num,i,j; int max=0;
 for(i=1;i<=G->vexnum;i++)
  {num=0;
   p=G->ag[i].firstarc;
   while(p)   {p=p->nextarc;    num++;   }
   if (max<num) max=num;
  }
  printf("The max degree is:%d\n",max);
}
```

3．已知某有向图用邻接表表示，设计一个算法，求出给定两顶点间的简单路径。

【算法分析】

因为在遍历的过程中每个顶点仅被访问一次，所以从顶点 u 到顶点 v 遍历的过程中走过的路径就是一条简单路径。我们只需在遍历算法中稍作修改，就可实现该算法。为了记录路径中访问过的顶点，只需设一数组存储走过的顶点即可。

【算法源代码】

```
int  visited[MAX_VERTEX_NUM];
int found;
void DFSpath(ALGraph *G,int u,int v){
```

```
int i;
for(i=1;i<=G->vexnum;i++)visited[i]=0;
found=0;
DFS(G,u,v);
}
int path[MAX_VERTEX_NUM];
void DFS(ALGraph *G,int u,int v)
{ /*用深度优先遍历算法实现简单路径的求解*/
ArcPtr p;
if(found) return;
if(G->ag[u].firstarc==NULL){printf("no path\n");return;}
visited[u]=1;
for(p=G->ag[u].firstarc;p;p=p->nextarc)
{
 if (v==p->adjvex)
{path[u]=v; found=1; Print(u,v); return;}
else if(!visited[p->adjvex])
{path[u]=p->adjvex; DFS(G,p->adjvex,v);}
 }
}/*DFS*/
void Print(int u,int v){
 int m;
 printf("%d->",u);
 for(m=path[u];m!=v;m=path[m])
 printf("%d->",m);
 printf("%d\n",v);
 }
```

4．设计一个深度优先遍历图的非递归算法。（图用邻接矩阵存储）

【算法分析】

在相应的深度优先遍历的非递归算法中，使用一个辅助数组 visited[ ]，在 visited[i]中记忆第 i 个顶点是否访问过。使用了一个栈 s，存储回退的路径。

【算法源代码】

```
void  DFS ( int v ,Mgraph  *G) {/*从顶点 v 开始进行深度优先搜索*/
int visited[MAX_VERTEX_NUM];
int s[MAX_VERTEX_NUM];
int i, j, k,top;
top=0;s[++top]=v;
for ( i = 0; i < G->vexnum; i++ ) visited[i] = 0;
while (!top ) {
k = s[top];
top--;          /*栈中退出一个顶点*/
if (!visited[k] ) {
    printf("%d",k);
    visited[k] = 1;        /*访问，并作访问标记*/
    for ( j =Vertex_num-1; j >= 0; j-- )       /*检查 k 的所有邻接顶点*/
     if ( k!= j &&!G->arcs[k][j]) s[++top]=j;/*所有邻接顶点进栈*/
    }
   }
  }
```

5．设计一个算法，输出距离顶点 v0 的最短路径长度为 k 的所有顶点，其中路径长度指的是弧或边的数目。

【算法分析】

本题用广度优先遍历的层次遍历法求解比较简单，要输出到 v0 的路径长度为 k 的所有顶点，也就是由 v0 开始作为第一层，它的邻接点作为第二层，依次…，输出第 k+1 层上的所有元素。故在访问每一个顶点的同时，记录下它所在的层次数。为此需设置两个队列分别存放被访问的顶点及层次，且两个队列要

同步操作。

【算法源代码】

```
#define MAX_VERTEX_NUM   20
int  visited[MAX_VERTEX_NUM];
/*visited[i]=0 表示 i 未被访问, visited[i]=1 表示 i 已被访问*/
/*****************************************/
 void Init(SqQueue *q){
    q->front=q->rear=0;
 }
 /*****************************************/
 int Empty(SqQueue *q){
    if (q->rear==q->front)
       return 1;
    return 0;
 }
 /*****************************************/
 void Inqueue(SqQueue *q,int x){
 if((q->rear+1) % MAX_VERTEX_NUM==q->front)
    {printf("queue is full");
     return;
     }
 else{
              q->rear=(q->rear+1) %MAX_VERTEX_NUM;
              q->data[q->rear]=x;
              }
 }
/*****************************************/
int Outqueue(SqQueue *q){
 if(Empty(q)){
    printf("queue is empty\n");
    return   -1;
  }
 else{
              q->front=(q->front+1) % MAX_VERTEX_NUM;
              return(q->data[q->front]);
  }
}
/*****************************************/
void bfsk(int v0,int k,ALGraph *G){
/*在图 G 中查找距离 v0 的路径长度为 k 的所有顶点*/
   int i,level,v,visited[MAX_VERTEX_NUM]; /*定义访问数组*/
   ArcPtr w;
   SqQueue  Q1,Q2;/*定义两个队列*/
   Init(&Q1);
   Init(&Q2);/*两个队列初始化*/
   for(i=1;i<=G->vexnum;i++)
     visited[i]=0;
   visited[v0]=1;
   level=1;
   Inqueue(&Q1,v0);
   Inqueue(&Q2,level); /*v0 及层次入队列*/
   while(!Empty(&Q1)&&(level<k+1)){
              v=Outqueue(&Q1);/*顶点出队列*/
              level=Outqueue(&Q2); /*层数出队列*/
      w=G->ag[v].firstarc;/*取 v 的第一个邻接点*/
      while(w){
              if(!visited[w->adjvex]){
```

```
                        if(level==k)
                           printf("%d ",w->adjvex); /*输出满足条件的邻接点*/
                        visited[w->adjvex]=1;
                        Inqueue(&Q1,w->adjvex);
                        Inqueue(&Q2,level+1);
                    }
                    w=w->nextarc; /*取 v 的下一个邻接点*/
                }
            }
        }
```

6．设计一个算法，用深度优先遍历法对 AOV 网进行拓扑排序，检测其中是否存在环。

【算法分析】

如果从有向图中的某个顶点 v 出发开始遍历，在 dfs(v)结束之前出现一条从顶点 u 到顶点 v 的回边，由于 u 在生成树上是 v 的子孙，则有向图中必定存在包含顶点 v 和 u 的环。本算法以邻接表作为存储结构，函数返回 1 表示无环，返回 0 表示有环.

【算法源代码】

```
#define MAX_VERTEX_NUM   20
int  visited[MAX_VERTEX_NUM];
/*visited[i]=0 表示 i 未被访问, visited[i]=1 表示 i 已被访问*/
int finished[MAX_VERTEX_NUM];
/*finished[i]=1 表示 i 已被访问，finished[i]= 0 表示 i 未被访问*/
int flag=1;
void dfs(ALGraph *G,VertexType v){
/*遍历时若不存在环，输出的全部顶点序列即为拓扑序列，*/
/*否则 flag 置为 0,表示存在环*/
  ArcPtr p;
  finished[v]=0;
  visited[v]=1;
  p=G->ag[v].firstarc;
  while(p){
     if(visited[p->adjvex]&&(!finished[p->adjvex]))
        flag=0;
     else if(!finished[p->adjvex]){
        dfs(G,p->adjvex);
        finished[p->adjvex]=1;
     }
     p=p->nextarc;
  }
}
void dfs_sort(ALGraph *G){
  int i;
  for(i=0;i<G->vexnum;i++)
              visited[i]=0;
  for(i=0;i<G->vexnum;i++)
              finished[i]=0;
  i=1;
  while(flag&&i<G->vexnum){
              if(!visited[i])
       dfs(G,i);
     finished[i]=1;
  }
}
```

第 8 章　查找

8.1　选择题

1．顺序查找法适合于存储结构为（　　）的线性表。

A）散列存储　　B）顺序存储或链接存储　C）压缩存储　　D）索引存储

【答案】B

2．下面哪些操作不属于静态查找表（　　）

A）查询某个特定元素是否在表中　　B）检索某个特定元素的属性

C）插入一个数据元素　　D）建立一个查找表

【答案】C

3．下面描述不正确的是（　　）

A）顺序查找对表中元素存放位置无任何要求，当 n 较大时，效率低。

B）静态查找表中关键字有序时，可用二分查找。

C）分块查找也是一种静态查找表。

D）经常进行插入和删除操作时可以采用二分查找。

【答案】D

4．散列查找时，解决冲突的方法有（　　）

A）除留余数法　　B）数字分析法　C）直接定址法　D）链地址法

【答案】D

5．若表中的记录顺序存放在一个一维数组中，在等概率情况下顺序查找的平均查找长度为（　　）

A）O（1）　　B）$O(\log_2 n)$　　C）O(n)　　D）$O(n^2)$

【答案】C

6．对长度为 4 的顺序表进行查找，若第一个元素的概率为 1/8，第二个元素的概率为 1/4，第三个元素的概率为 3/8，第四个元素的概率为 1/4，则查找任一个元素的平均查找长度为（　　）

A）11/8　　B）7/4　　C）9/4　　D）11/4

【答案】C

【解析】对顺序表查找，$ASL=\sum_{i=1}^{n} PiCi$，代入题目得：

ASL=4*(1/8)+3*(1/4)+2*(3/8)+1*(1/4)=9/4

7．静态查找表与动态查找表二者的根本差别在于（　　）

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

【答案】B

8．若查找表中的记录按关键字的大小顺序存放在一个一维数组中，在等概率情况下二分法查找的平均检索长度是（　　）

A）O(n)　　B）$O(\log_2 n)$　　C）$O(n\log_2 n)$　　D）$O((\log_2 n)^2)$

【答案】B

9．对有 14 个数据元素的有序表 R[14]（假设下标从 1 开始）进行二分查找，搜索到 R[4]的关键码等于给定值，此时元素比较顺序依次为（　　）。

A）R[1]，R[2]，R[3]，R[4]　　B）R[1]，R[13]，R[2]，R[3]

C）R[7]，R[3]，R[5]，R[4]　　D）R[7]，R[4]，R[2]，R[3]

【答案】C

10．设有一个长度为 100 的已排好序的表，用二分查找进行查找，若查找不成功，至少比较（　　）次。

A）9　　B）8　　C）7　　D）6

【答案】B

【解析】二分查找不成功时和给定值进行比较的关键字个数最多不超过二叉判定树的深度。100 个元素查找表的判定树深为 8（64<100<128）。

11．请指出在顺序表{2,5,7,10,14,15,18,23,35,41,52}中，用二分法查找关键码 12 需做（　　）次关键码比较。

A）2　　B）3　　C）4　　D）5

【答案】C

12．从具有 n 个结点的二叉排序树中查找一个元素时，在最坏情况下的时间复杂度为 (　　)。

A)O (n)　　B） O（1）　　C） O ($\log_2$ n)　　D）O (n^2)

【答案】C

13．分块查找时确定块的查找可以用顺序查找，也可以用（　　），而在块中只能是（　　）

A）静态查找，顺序查找　　B）二分查找，顺序查找

C）二分查找，二分查找　　D）散列查找，顺序查找

【答案】B

14．采用分块查找时，若线性表中共有 625 个元素，查找每个元素的概率相同，假设采用顺序查找来确定结点所在的块时，每块应分（　　）个结点最佳。

A）10　　B）25　　C）6　　D）625

【答案】B

15．采用分块查找法（块长为 s，以二分查找确定块）查找长度为 n 的线性表时，每个元素的平均查找长度为（　　）

A）s+n　　B）$\log_2 n$＋s/2　　C）$\log_2$ (n/s+1)+s/2　　D）(n+s)/2

【答案】C

16．对一棵二叉排序树根结点而言，左子树中所有结点与右子树中所有结点的关键字大小关系是（　　）

A）小于　　B）大于　　C）等于　　D）不小于

【答案】A

17．若二叉排序树中关键字互不相同，则下面命题中不正确的是（　　）

A）最小元和最大元一定是叶子　　B）最大元必无右孩子

C）最小元必无左孩子　　D）新结点总是作为叶结点插入二叉排序树

【答案】A

18．设二叉排序树中关键字由 1 至 1000 的整数构成，现要查找关键字为 363 的结点，下述关键字序列（　　）不可能是在二叉排序树上查找到的序列?

A）2,252,401,398,330, 344,397,363

B）924, 220, 911, 244, 898, 258, 362, 363

C）2, 399, 387, 219, 266, 382, 381, 278, 363

D）925, 202, 911, 240, 912, 245, 363

【答案】D

19．在初始为空的散列表中依次插入关键字序列　(MON,TUE,WED,THU,FRI,SAT,SUN),散列函数为 H(k)=i MOD 7，其中，i 为关键字 k 的第一个字母在英文字母表中的序号，地址值域为 [0:6] ，采用线性再散列法处理冲突。插入后的散列表应该如（　　）所示。

A）0　1　2　3　4　5　6
　THU TUE WED FRI SUN SAT MON

B）0　1　2　3　4　5　6
　TUE THU WED FRI SUN SAT MON

C）0　1　2　3　4　5　6
　TUE THU WED FRI SAT SUN MON

D）0　1　2　3　4　5　6
　TUE THU WED SUN SAT FRI MON

【答案】B

20．若根据查找表建立长度为 m 的散列表，采用线性探测法处理冲突，假定对一个元素第一次计算的散列地址为 d，则下一次的散列地址为 (　　)。

A） d　　B） (d+1)%m　　C） (d+1)/m　　D） d+1

【答案】B

21．若根据查找表建立长度为 m 的散列表，采用二次探测法处理冲突，假定对一个元素第一次计算的散列地址为 d，则第四次计算的散列地址为 (　　)。

A)(d+1)%m　　B） (d-1)%m　　C） (d+4)%m　　D） (d-4)%m

【答案】D

22．下面有关散列查找的说法中正确的是（　　）

A）直接定址法所得地址集合和关键字集合的大小不一定相同。

B）除留余数法构造的哈希函数 H(key)=key MOD p，其中 P 必须选择素数。

C）构造哈希函数时不需要考虑记录的查找频率。

D）数字分析法适用于对哈希表中出现的关键字事先知道的情况。

【答案】D

23．下面有关散列冲突解决的说法中不正确的是（　　）

A）处理冲突即当某关键字得到的哈希地址已经存在时，为其寻找另一个空地址。

B）使用链地址法在链表中插入元素的位置随意，即可以是表头表尾，也可以在中间。

C）二次探测能够保证只要哈希表未填满，总能找到一个不冲突的地址。

D）线性探测能够保证只要哈希表未填满，总能找到一个不冲突的地址。

【答案】C

24．设哈希表长 m=14，哈希函数 H(key)=key%11。表中已有 4 个结点：addr(15)=4，addr(38)=5，addr(61)=6，addr(84)=7 其余地址为空，如用二次探测处理冲突，关键字为 49 的结点的地址是（　　）

A）8　　B）3　　C）5　　D）9

【答案】D

8.2　填空题

1．在散列函数 H(key)=key%p 中，p 应取＿＿＿＿＿＿。

【答案】素数

2．采用分块查找法（块长为 s，以顺序查找确定块）查找长度为 n 的线性表时的平均查找长度为＿＿＿＿＿＿。

【答案】(n/s+1) /2+1

3．已知一个有序表为(12,18,20,25,29,32,40,62,83,90,95,98)，当二分查找值为 29 和 90 的元素时，分别需要＿＿＿＿＿＿次和＿＿＿＿＿＿次比较才能查找成功；若采用顺序查找时，分别需要＿＿＿＿＿＿次和＿＿＿＿＿＿次比较才能查找成功。

【答案】（1）4　　（2）4　　（3）5　　（4）10

4．从一棵二叉排序树中查找一个元素时，若元素的值等于根结点的值，则表明＿＿＿＿＿＿，若元素的值小于根结点的值，则继续向＿＿＿＿＿＿查找，若元素的值大于根结点的值，则继续向＿＿＿＿＿＿查找。

【答案】（1）查找成功　　（2）左子树　　（3）右子树

5．二分查找的存储结构仅限于＿＿＿＿＿＿，且是＿＿＿＿＿＿。

【答案】（1）顺序存储结构　　（2）有序

6．假设在有序线性表 A[1..20]上进行二分查找，则比较一次查找成功的结点数为＿＿＿＿＿＿个，比较二次查找成功的结点数为＿＿＿＿＿＿，比较三次查找成功的结点数为＿＿＿＿＿＿，比较四次查找成功的结点数为＿＿＿＿＿＿，比较五次查找成功的结点数为＿＿＿＿＿＿，平均查找长度为＿＿＿＿＿＿。

【答案】（1）1　　（2）2　（3）4　　（4）8　（5）5　（6）3.7

7．在对有 20 个元素的递增有序表作二分查找时，查找长度为 5 的元素的下标从小到大依次为＿＿＿＿＿＿。（设下标从 1 开始）

【答案】4，9，14，17，20

8．对于线性表（70,34,55,23,65,41,20,100）进行散列存储时，若选用 H(K)=K%9 作为散列函数，则散列地址为 1 的元素有＿＿＿＿＿＿个，散列地址为 7 的元素有＿＿＿＿＿＿个。

【答案】（1）2　　（2）2

9．索引顺序表上的查找分两个阶段：＿＿＿＿＿＿、＿＿＿＿＿＿。

【答案】（1）确定待查元素所在的块　（2）在块内查找待查的元素

10．分块查找中，要得到最好的平均查找长度，应对 256 个元素的线性查找表分成＿＿＿＿＿＿块，每块的最佳长度是＿＿＿＿＿＿。若每块的长度为 8，则等概率下平均查找长度为＿＿＿＿＿＿。

【答案】（1）16　　（2）16　　（3）21

【解析】分块查找的平均查找长度由两部分组成——查找索引表确定所在块的平均查找长度 Lb 和在块中查找元素的平均查找长度 Lw，即 ASLbs=Lb+Lw=(b+s)/2+1，其中 s 为每块的长度，b 为所分的快数。由数学知识可知当 s=$\sqrt{n}$ 时，ASLbs 可取得最小值 $\sqrt{n}$ +1。因此，可得每块的最佳长度是 16，应将查找表分为 16 块。若每块的长度为 8，则 b=32,因此 ASLbs=Lb+Lw=(b+s)/2+1=21。

11．＿＿＿＿＿＿是一棵二叉树，如果不为空，则它必须满足下面的条件：

A）若左子树不空，则左子树上所有结点的值均小于根的值。

B）若右子树不空，则右子树上所有结点的值均大于根的值。

C）其左右子树均为二叉排序树。

【答案】二叉排序树

13．假定有 k 个关键字互为同义词，若用线性探测法把这些同义词存入散列表中，至少要进行_____次探测。

【答案】1+2+3...+(k-1)+k=k(k+1)/2

【解析】在散列表的一连串连续空间内，第一个关键字只需探测一次，第二个就要探测 2 次，如此这般，第 k 个关键字就要探测 k 次才能找到位置存放。

8.3 判断题

1．对查找进行时间分析时，只需要考虑查找成功的平均情况。（ ）

【答案】×

【解析】大多数情况下，特别查找表中记录数 n 很大时，查找不成功的概率可以忽略不计。但是，当查找不成功的情况不能忽视时，查找算法的平均查找长度应是查找成功时的平均查找长度与查找不成功时的平均查找长度之和。

2．在索引顺序表上实现分块查找，在等概率查找情况下，其平均查找长度不仅与表的个数有关，而且与每一块中的元素个数有关。（ ）

【答案】√

3．构造一个好的哈希函数必须均匀，即没有冲突。（ ）

【答案】×

【解析】一个好的哈希函数必须均匀，并不代表完全没有冲突，而是尽量减少冲突。

4．在一定情况下，有可能设计出无冲突的散列函数 H。（ ）

【答案】√

5. 二分查找只适用于有序表，包括有序的顺序表和有序的链表。（ ）

【答案】×

【解析】二分查找只适用于顺序表，而不能在链表结构中采用。因为链表查找都是从头指针开始。

6．对给定的关键字集合，以不同的次序插入初始为空的树中，有可能得到同一棵二叉排序树。（ ）

【答案】√

7．分块查找适用于任何有序表或者无序表。（ ）

【答案】×

【解析】分块查找适用于任何有序表或者分块有序表，而不适用于任意的无序表。

8．在用线性探测法解决冲突所构造的散列表中，每组同义词中至少有一个元素的地址正好等于其散列地址。（ ）

【答案】×

【解析】当存在堆积的冲突时，可能没有一个元素地址等于其计算所得的散列地址。

9．对一棵二叉排序树中序遍历一定得到一个关键字的有序序列。（ ）

【答案】√

10．所谓冲突即是两个关键字的值相同的元素，其散列地址相同。（ ）

【答案】×

【解析】冲突是指两个关键字的值不相同的元素，计算得到的散列地址相同。

11．二叉判定树和二叉排序树一样，都不是唯一的。（ ）

【答案】×

【解析】对于同一组结点，由于建立二叉排序树时插入结点的先后次序不同，所构成的二叉排序树的形态及深度也不同，所以含有 n 个结点的二叉排序树不唯一。但二叉判定树却是唯一的。

12．若二叉树中每个结点的值均大于其左孩子的值，小于其右孩子的值，则该二叉树一定是二叉排序树。（ ）

【答案】×

【解析】判定一棵二叉树是否是二叉排序除上面两个条件外，还必须满足第三个条件，即其左右子树也是二叉排序树。

13．分块查找中，每一块的大小是相同的。（ ）

【答案】×

【解析】最末一块，可以不是整块，前面块的大小必须相同。

14．对一个有序表作二分查找，　查找每个元素所需的查找次数均比用顺序查找所需的查找次数要少。（　　）

【答案】×

【解析】顺序查找时最少的比较次数为 1，它的比较次数小于位于二叉判定树第二层以上的结点。二分查找时最多的比较次数为二叉判定树的深度。

15．散列表的查找效率主要取决于所选择的散列函数与处理冲突的方法。（　　）

【答案】✓

8.4　应用题

1．顺序查找时间为 O(n)，二分法查找时间为 O(log2n)，散列法为 O(1)，为什么有高效率的查找方法而低效率的方法不被放弃?

【答案】不同的查找方法适用的范围不同，高效率的查找方法并不是在所有情况下都比其他查找方法效率要高，而且也不是在所有情况下都可以采用。

2．对含有 n 个互不相同元素的集合，同时找最大元和最小元至少需进行多少次比较?

【答案】n-1 次

【解析】设变量 max 和 min 用于存放最大元和最小元(的位置)，第一次取两个元素进行比较，大的放入 max，小的放入 min。从第 2 次开始，每次取一个元素先和 max 比较，如果大于 max 则以它替换 max，并结束本次比较；若小于 max 则再与 min 相比较，在最好的情况下，比较下去都不用和 min 相比较，所以这种情况下，至少要进行 n-1 次比较就能找到最大元和最小元。

3．若对具有 n 个元素的有序的顺序表和无序的顺序表分别进行顺序查找，试在下述两种情况下分别讨论两者在等概率时的平均查找长度：

（1）查找不成功，即表中无关键字等于给定值 K 的记录；

（2）查找成功，即表中有关键字等于给定值 K 的记录。

【答案】

（1）不成功时需要 n+1 次比较

（2）成功时平均为(n+1)/2 次

【解析】有序表和无序表顺序查找时，都需要进行 n+1 次比较才能确定查找失败。因此平均查找长度都为 n+1。查找成功时，平均查找长度都为(n+1)/2，有序表和无序表也是一样的。因为顺序查找与表的初始序列状态无关。

4．设有序表为(a, b, c, d, e, f, g, h, i, j, k, p, q)，请分别画出对给定值 a, g 和 n 进行折半查找的过程。

【答案】

（1）查找 a 的过程如下(圆括号表示当前比较的关键字)，经过三次比较，查找成功。

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	区间
第一次比较	a	b	c	d	e	f	(g)	h	i	j	k	p	q	[a..q]
第二次比较	a	b	(c)	d	e	f	g	h	i	j	k	p	q	[a..f]
第三次比较	(a)	b	c	d	e	f	g	h	i	j	k	p	q	[a..b]

（2）g 的查找过程如下，一次比较成功。

[a　b　c　　d　e　f　(g)　h　i　　j　k　p　　q　]

（3）n 的查找过程如下，经过四次比较，查找失败。

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	区间
第一次比较	a	b	c	d	e	f	(g)	h	i	j	k	p	q	[a..q]
第二次比较	a	b	c	d	e	f	g	h	i	(j)	k	p	q	[h..q]
第三次比较	a	b	c	d	e	f	g	h	i	j	k	(p)	q	[k..q]
第四次比较	a	b	c	d	e	f	g	h	i	j	(k)	p	q	[k]

5．为什么有序的单链表不能进行折半查找?

【答案】因为链表无法进行随机访问，如果要访问链表的中间结点，就必须先从头结点开始进行依次访问，这就要浪费很多时间，还不如进行顺序查找，而且，用链存储结构将无法判定二分的过程是否结束，因此无法用链表实现二分查找。

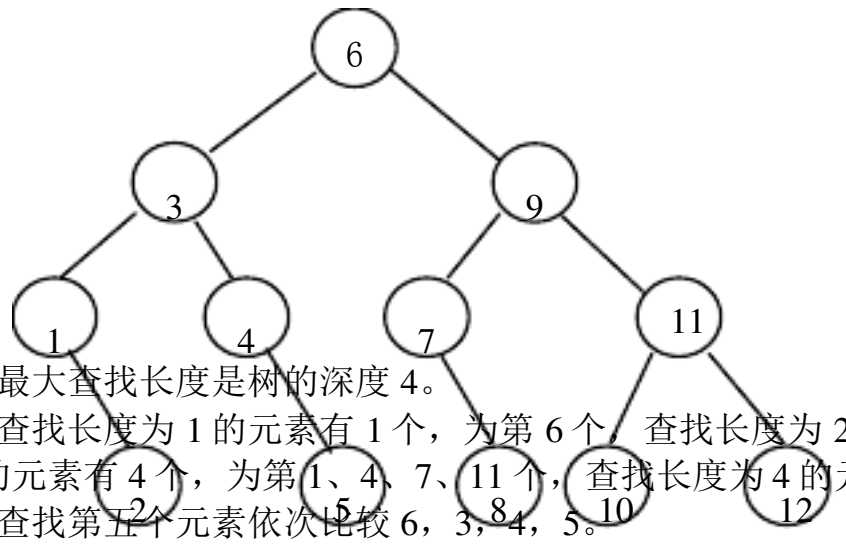
6．构造有 12 个元素的二分查找的判定树，并求解下列问题：

（1）各元素的查找长度最大是多少？

（2）查找长度为 1、2、3、4 的元素各有多少？具体是哪些元素？

（3）查找第 5 个元素依次要比较哪些元素？

【答案】12 个元素的判断树如下图所示：



（1）最大查找长度是树的深度 4。

（2）查找长度为 1 的元素有 1 个，为第 6 个，查找长度为 2 的元素有 2 个，为第 3 个和第 9 个，查找长度为 3 的元素有 4 个，为第 1、4、7、11 个，查找长度为 4 的元素有 5 个，为第 2、5、8、10、12 个。

（3）查找第五个元素依次比较 6，3，4，5。

7．以数据集合{1,2,3,4,5,6}的不同序列为输入，构造 4 棵高度为 4 的二叉排序树。

【答案】

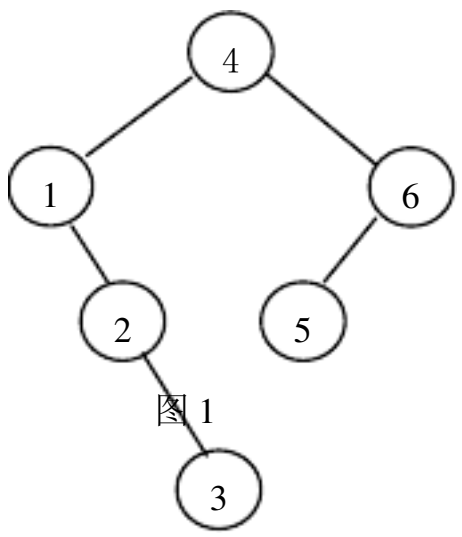


图 1

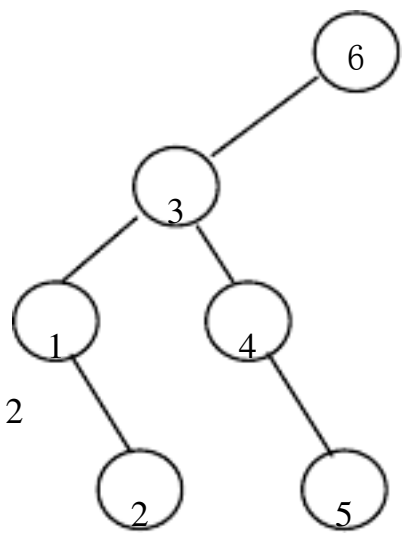


图 2

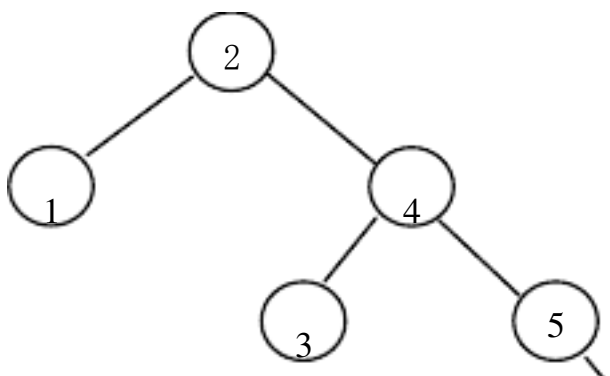


图 3

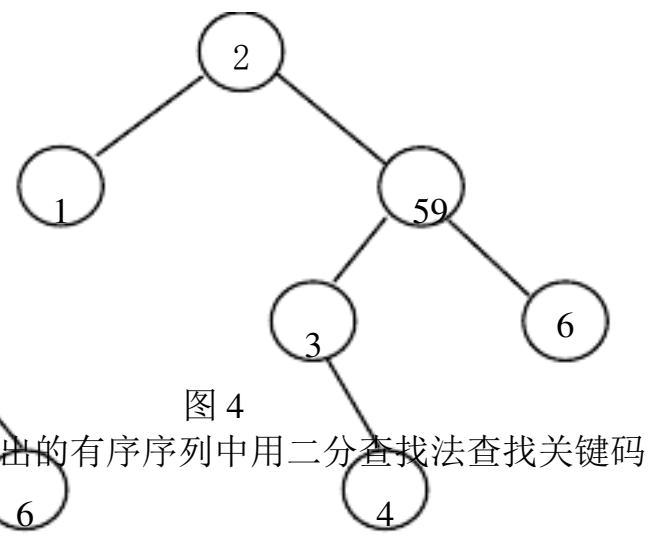


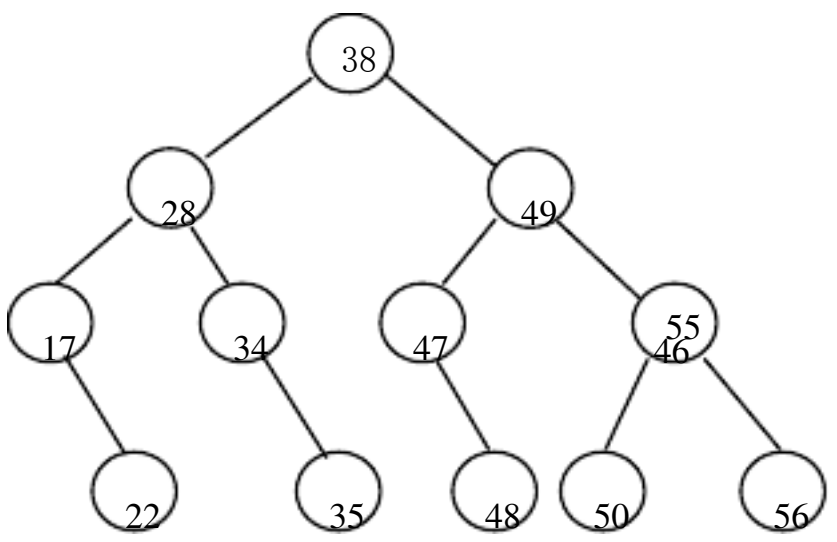
图 4

8．直接在二叉排序树中查找关键码 K 与从中序遍历输出的有序序列中用二分查找法查找关键码 K，其数据比较次数是否相同？

【答案】不相同。

【解析】因为二分查找得到的判定树和二叉排序树的形状不一定相同。

9．已知一棵二叉排序树如下：

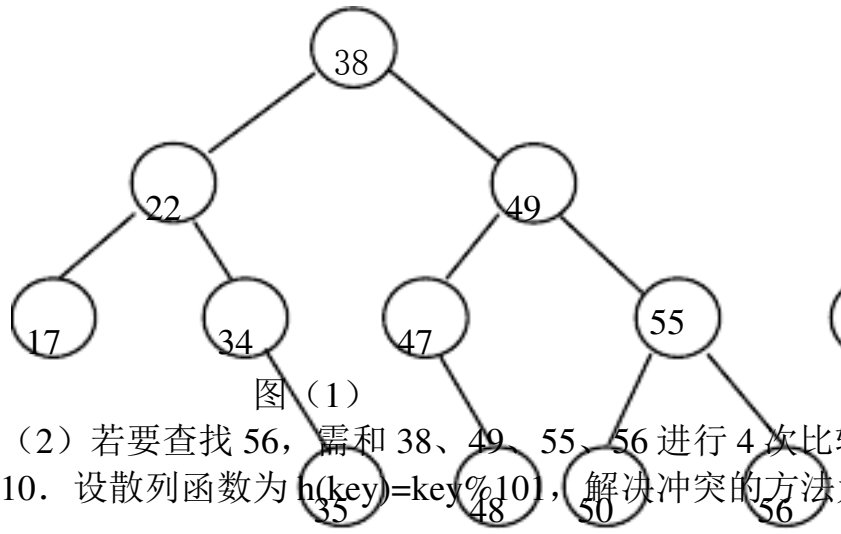


（1）假如删除关键字 28，画出新二叉树。

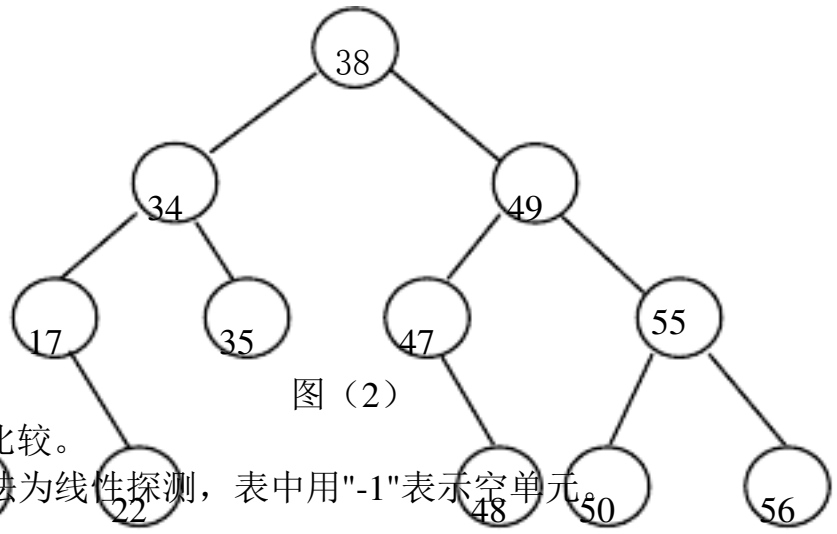
（2）若查找 56，需和哪些关键字比较。

【答案】

（1）删除元素 28 后，需修改二叉排序树的形态，可用结点 28 左子树上最大的结点代替它如图（1），也可用其右子树上最小的结点代替它，如图（2）。



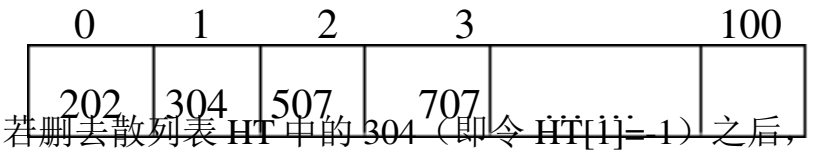
图（1）



图（2）

（2）若要查找 56，需和 38、49、55、56 进行 4 次比较。

10．设散列函数为 h(key)=key%101，解决冲突的方法为线性探测，表中用"-1"表示空单元。



（1）若删去散列表 HT 中的 304（即令 HT[1]=-1）之后，在表 HT 中查找 707 将会发生什么？

（2）若将删去的表项标记为"-2"，查找时探测到"-2"继续向前搜索，探测到"-1"时终止搜索。请问用这种方法删去 304 后能否正确地查找到 707？

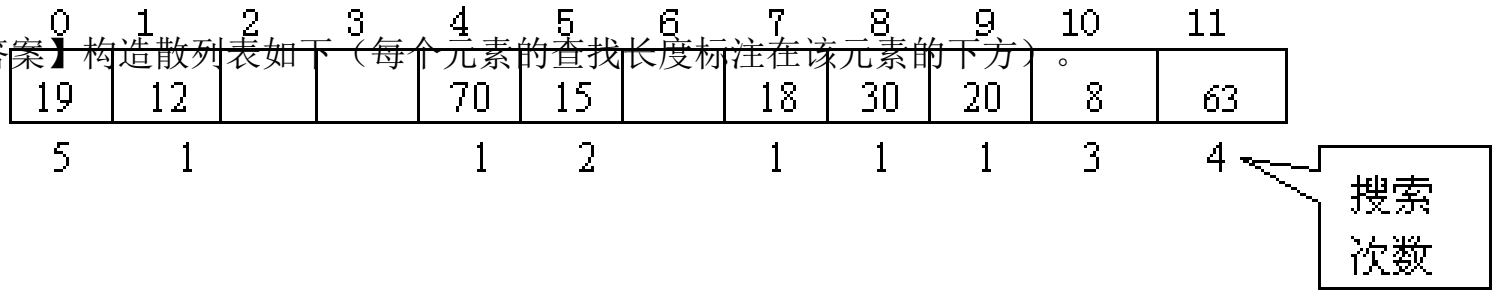
【答案】

（1）查找 707 时，首先根据散列函数计算得出该元素应在散列表中的 0 单元，但是在 0 单元没有找到，因此将向下一单元探测，结果发现该单元是-1(为空单元)，所以结束查找，这将导致 707 无法找到。

（2）如果改用"-2"作为删除标记，则可以正确找到 707 所在的结点。

11．已知散列表的地址区间为 0～11，散列函数为 H(k)=k % 11，采用线性探测法处理冲突，将关键字序列 20,30,70,15,8,12,18,63,19 依次存储到散列表中，试构造出该散列表，并求出在等概率情况下的平均查找长度。

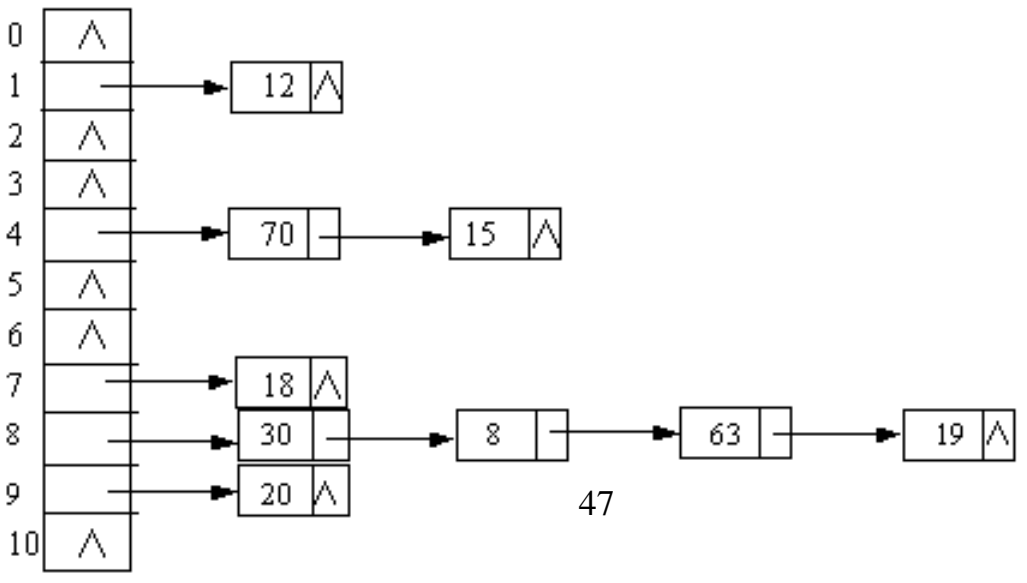
【答案】构造散列表如下（每个元素的查找长度标注在该元素的下方）。



等概率情况下成功时的平均查找长度为（1×5＋2＋3＋4＋5）/9＝19/9

12．设散列函数为 H(k)=k % 11，采用拉链法处理冲突，将上例中关键字序列依次存储到散列表中，并求出在等概率情况下的平均查找长度。

【答案】



在等概率情况下成功的平均查找长度为：

(1*5+2*2+3*1+4*1)/9=16/9

13．假定一个待散列存储的线性表为(32,75,29,63,48,94,25,46,18,70)，散列地址空间为 HT[13]，若采用除留余数法构造散列函数和线性探测法处理冲突，试求出每一元素的初始散列地址和最终散列地址，画出最后得到的散列表，求出平均查找长度。

【答案】

序号	1	2	3	4	5	6	7	8	9	10
元素值	32	75	29	63	48	94	25	46	18	70
初始地址	6	10	3	11	9	3	12	7	5	5
最终地址	6	10	3	11	9	4	12	7	5	8

构造的散列表如下：

0	1	2	3	4	5	6	7	8	9	10	11	12
			29	94	18	32	46	70	48	75	63	25

在等概率情况下成功的平均查找长度为(1*7+2*5+3*1+4*1)/14=24/14

14．散列表的地址区间为 0～15，散列函数为 H(key)=key%13。设有一组关键字{19,01,23,14,55,20,84}, 采用线性探测法解决冲突，依次存放在散列表中。问：

（1）元素 84 存放在散列表中的地址是多少？

（2）搜索元素 84 需要的比较次数是多少？

【答案】构造的散列表如下：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	14	55			19	20	84		23					

（1）元素 84 存放在散列表中的地址是 8。

（2）搜索元素 84 需要的比较 3 次。

8.5 算法设计题

1．已知顺序表 A 长度为 n，试写出将监视哨设在高端的顺序查找算法。

【算法分析】

将监视哨放在高端，即元素从下标为 0 的位置开始存放，将 A[n]设置为待查键值，作为监视哨。查找成功时返回元素下标，否则返回 n。

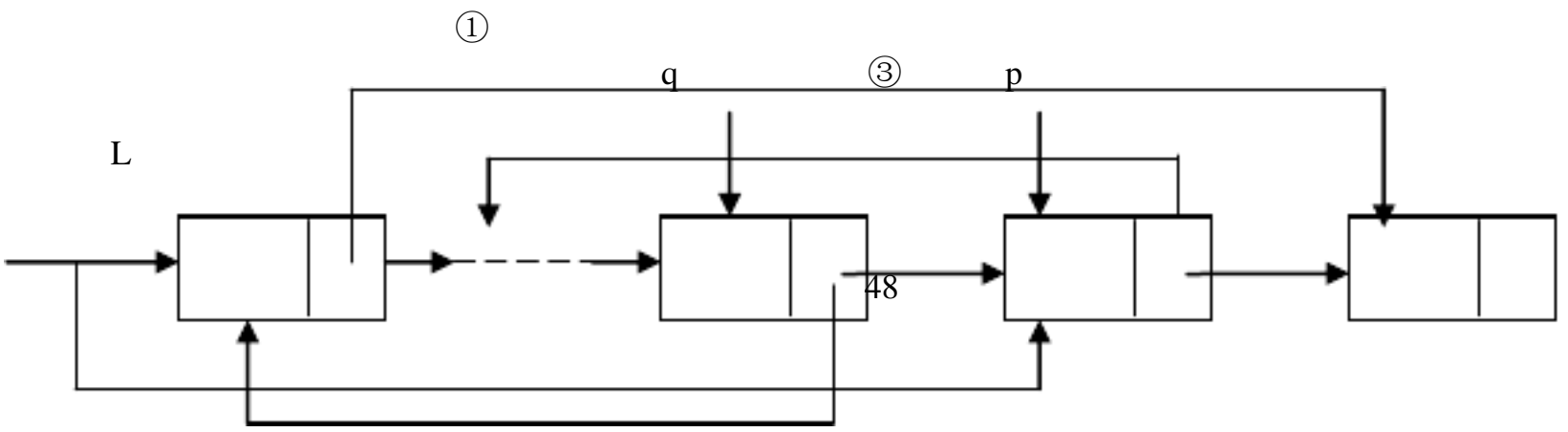
【算法源代码】

```
#include "type.h"
int seach_seq(SSTable A,ElemType key)
{int i,n;
n=A.length;
A.elem[n].key=key;
for(i=0;A.elem[i].key!=key;++i)
return i;
}
```

2．若线性表中各结点的查找概率不等，则可用如下策略提高顺序查找的效率：若找到指定的结点，则将该结点和其前驱结点交换，使得经常被查找的结点尽量位于表的前端。试对线性表的链式存储结构写出实现上述策略的顺序查找算法（查找时必须从表头开始向后扫描）。

【算法分析】

设指针变量 p 指向当前结点，q 指向其前驱结点。过程如下图所示。若要将当前结点和第一个结点交换，通过指针的变化需要经过 4 步（如图所示）：①第一个结点的 next 域指向 q 的后继结点。②将头指针指向当前结点。③当前结点的 next 域等于第一个结点的 next 域。④其前驱结点的 next 指向原来的第一个结点。



② 前驱结点　　当前结点　　后继结点

④

为简化算法，今采用交换当前结点和其前驱结点数据部分的算法，保持指针不变，仅仅改变两个结点的数据部分。

【算法源代码】

```
#include "type.h"
LNode *search_Slist(LNode *L,int key)
 {   LNode *q,*p;    int t;
    p=L;
    q=NULL;
    while(p->next!=NULL)
    if(p->data!=key)   /*当前结点不是要查找结点，继续向后查找*/
          {q=p;  p=p->next;}
   else  /*当前结点是要查找结点*/
          if(q!=NULL)    /*当前结点有前驱*/
          { t=q->data;q->data=p->data;          p->data=t;return p;}
}
```

3．已知关键字序列为{PAL,LAP,PAM,MAP,PAT,PET,SET,SAT,TAT,BAT}，试为它们设计一个散列函数，将其映射到区间[0..n-1]上，要求碰撞尽可能的少。这里 n=11,13,17,19.

【算法分析】

设计的散列函数：把关键字串中的每一个字符按其所在位置分别将其 ASCII 值乘以一个不同的数，然后把这些值相加的和去对 n 求余，余数即为散列表中的位置。

【算法源代码】

```
#include "stdio.h"
#define n 11/*也可以选用 13，17，19*/
int Hash (char key[])
{return (int)(key[0]+key[1]*0.618+key[2]*10)%n;}
main()
{ int i;
char s[10][4]={"PAL","LAP","PAM","MAP","PAT","PET","SET","SAT",
            "TAT","BAT"};       /*用一个二维数组来存放关键字序列*/
   for(i=0; i<10;i++)    printf("%d  ",Hash(s[i]));
}
```

4．有递增排序的顺序线性表 A[n]，写出利用二分查找算法查找元素 K 的递归算法。若找到则给出其位置序号，若找不到则其位置号为 0。

【算法源代码】

```
#include "type.h"
int Binsch(SSElement A[],int low,int high,ElemType K)
{int mid;
if(low<=high )
{ int mid=(low+high)/2;
if(K==A[mid].key)
            return mid;    /* 查找成功，返回元素的下标*/
else if(K<A[mid].key)
            return Binsch(A,low,mid-1,K);/*在左子表上继续查找*/
else
           return Binsch(A,mid+1,high,K);/*在右子表上继续查找*/
}
else   return 0;    /* 查找失败，返回 0*/
}
```

5．设计一个算法，求出指定结点在给定的二叉排序树中所在的层数。

【算法分析】查找成功时的比较次数即为结点所在层数。可设置查找时计数，比较一次计数器加 1。如查找成功时返回计数器累加数字，不成功时，返回 0。

【算法源代码】

```
#include "stdio.h"
#include "type.h"
int search_depth(BiTree T,ElemType key)
/*求当前结点所在层数 */
{
BiTNode *p;
int dep=0;
p=T;
while(p)
{ if(key==T->data)
    {dep++;break;}
  else
   if(key>T->data) {dep++;  p=p->rchild;}
                else     { dep++;  p=p->lchild;}
}
 if(p) return dep;
 else return 0;
}
```

6．设计一个算法，以求出给定二叉排序树中值为最大的结点。

【算法分析】二叉排序树上最大的结点肯定在右子树上。因此，首先从根结点开始查找，然后顺着右子树查找，直到结点没有右子树为止。

【算法源代码】

```
#include "stdio.h"
#include "type.h"
ElemType search_max(BiTree T)  /*求二叉排序树最大值*/
{ BiTNode *p;
  ElemType max;
  p=T;
  if (T==NULL)
    {p=NULL;  printf("树为空\n");  return 0;}
  while(p)
    {max=p->data;   p=p->rchild;}  /*访问右孩子，直到空为止*/
  return max;
}
```

7．设计一个递归算法，向以 BT 为树根的二叉排序树上插入值为 x 的结点。

【算法源代码】

```
#include "type.h"
#include "stdio.h"
BiTree  insort(BiTree  bt, ElemType x)/*函数返回二叉排序树头指针*/
{  BiTree  p,q;
   p=( BiTree)malloc(sizeof(BiTNode)); /*生成新结点*/
   p->data=x;
   p->lchild=NULL;
   p->rchild=NULL;
   q=bt;
   if(q==NULL) bt=p;   /*二叉排序树为空*/
   else        /*二叉排序树不空*/
    {while((q->lchild!=p)&&( q->rchild!=p))
    { if(x<q->data)                  /*插入到左子树*/
            {  if(q->lchild!=NULL) q=q->lchild; else   q->lchild=p;  }
      else                           /*插入到右子树*/
            {if(q->rchild!=NULL)  q=q->rchild;  else q->rchild=p;}
     }
   }
     return(bt);          /*返回二叉排序树的头指针*/
```

```
}
```

8．假设二叉排序树采用链表结构存储，设计一个算法，从大到小输出该二叉排序树中所有关键字不小于 X 的数据元素。

【算法分析】

若对二叉排序树中序遍历，则遍历结果的序列是递增有序的，若遍历先左后右，则输出递增序列。若要输出递减序列，采取先遍历右子树，遍历根结点，再遍历左子树的策略，直到小于 X 为止。

【算法源代码】

```
#include "type.h"
void inorder(BiTree  bst,ElemType x)
{if(bst)
{  inorder(bst->rchild, x);  /*遍历右子树*/
   if(bst->data>=x)       printf("%c",bst->data);   /*访问结点*/
          else return;   /*找到小于 x 的结点停止*/
   inorder(bst->lchild,x);   /*遍历左子树*/
   }
}
```

9．假设散列函数为 H(k)=k % 11，采用链地址法处理冲突。设计算法：

（1）输入一组关键字（09,31,26,19,01,13,02,11,27,16,05,21）构造散列表。

（2）查找值为 x 的元素。若查找成功，返回其所在结点的指针，否则返回 NULL。

【算法分析】

构造散列表时，先把输入序列存入数组，然后顺序填入相应的链表。填入算法类似于单链表的建立，只是链表头指针存放的位置由散列函数计算得到。

查找散列表时，也首先根据散列函数计算链表头指针所在位置，然后顺序查找。找到返回结点地址，否则返回 NULL。

【算法源代码】

```
#include "type.h"
#define M 11                    /*也可以选用 13，17，19*/
int Hash (int key)   /*散列函数*/
{     return (int)(key%M);    }
hashcreat(HashNode *h[],int k)  /*构造散列表函数*/
{int i;
HashNode *p;
i=Hash(k);          /*确定链表头指针存放位置*/
p=(struct node *)malloc(sizeof(HashNode));/*给结点分配空间*/
p->key=k;
p->next=NULL;
h[i-1]=p;
}
HashNode *hashfind(HashNode *h[],int k)  /*查找函数*/
{  int i;
HashNode *p;
i=Hash(k);
p=h[i-1];    /*链表头指针*/
while(p!=NULL)
    {   if(p->key==k) return p;
        else p=p->next;
    }
return NULL;
}
main()
{HashNode *p,*h[12];
 int k[12]={ 9,31,26,19,1,13,2,11,27,16,5,21};/*数据存入数组*/
 int m,n=12;
for(m=0;m<n;m++) h[m]=NULL;    /*初始化顺序表，链表头指针全为空*/
for(m=0;m<n;m++) hashcreat(h,k[m]);
```

```
/*从数组中取数，依次加入散列表*/
for(m=0;m<n;m++)  /*输出散列表中元素*/
   {    p=h[m];
        while(p!=NULL)  {printf("%5d",p->key);p=p->next;}
       printf("\n");
}
printf("input k:");
scanf("%d",&m);  /*输入待查数据*/
p=hashfind(h,m);  /*查找*/
if(p) printf("找到元素%d",p->key);
else  printf("未找到元素!");
}
```

10．试编写利用二分查找法确定记录的所在块的分块查找算法。

【算法分析】

采用分块查找时，除了顺序表之外，还要有索引表。其中索引表中含有各块索引。在各块中进行顺序查找时，监视哨可设在本块的表尾，即将下一块的第一个记录暂时移走（若本块内记录没有填满，则监视哨的位置仍在本块的尾部），待块内顺序查找完成后再移回来。此时增加了赋值运算，但免去了判断下标变量是否越界的比较。注意，最后一块需进行特殊处理。在块内进行顺序查找时，如果需要设置监视哨，则必须先保存相邻块的相邻元素，以免数据丢失。

【算法源代码】

```
#include "type.h"
int Search_Idx(IdxSqlist L,int key)
{  /*分块查找，二分查找确定块，块内顺序查找*/
   int i,j,k,low,high,mid,found,temp;
   if(key>L.idx[L.blknum].maxkey) return -1;  /*超过最大元素，返回-1*/
   low=1;high=L.blknum;
   found=0;
   while(low<=high&&!found)
   {
   mid=(low+high)/2;
   if(key<=L.idx[mid].maxkey&&key>L.idx[mid-1].maxkey)
       found=1;
   else if(key>L.idx[mid].maxkey)              low=mid+1;
              else high=mid-1;
   }
   i=L.idx[mid].firstloc;           /*块的下界*/
   j=i+blksize-1;
   temp=L.elem[i-1];                  /*保存相邻元素*/
   L.elem[i-1]=key;                  /*设置监视哨*/
   for(k=j;L.elem[k]!=key;k--);              /*顺序查找*/
   L.elem[i-1]=temp;               /*恢复元素*/
 if(k<i) return -1;               /*未找到，返回-1*/
   return k;
}
```

第 9 章　　排序

9.1　选择题

1．从未排序的序列中依次取出一个元素与已排序序列中的元素依次进行比较，然后将其放在排序序列的合适位置，该排序方法称为（　　）排序法。

A）插入　B）选择　C）希尔　D）二路归并

【答案】A

2．下面各种排序方法中，最好情况下时间复杂度为 O(n)的是（　　）

A）快速排序　B）直接插入排序　C）堆排序　D）归并排序

【答案】B

3．用某种排序方法对线性表（25,84,21,47,15,27,68,35,20）进行排序时，无序序列的变化情况如下：

25 84 21 47 15 27 68 35 20
20 15 21 25 47 27 68 35 84
15 20 21 25 35 27 47 68 84
15 20 21 25 27 35 47 68 84

则所采用的排序方法是（　　）

A）选择排序　　B）希尔排序　　C）归并排序　　D）快速排序

【答案】D

4．下面给出的四种排序法中，（　　）排序是不稳定排序法。

A）插入　B）冒泡　C）二路归并　D）堆

【答案】D

5．快速排序方法在（　　）情况下最不利于发挥其长处。

A）要排序的数据量太大

B）要排序的数据中含有多个相同值

C）要排序的数据已基本有序

D）要排序的数据个数为奇数

【答案】C

6．一组记录的关键码为（46,79,56,38,40,84），则利用快速排序的方法，以第一个记录为基准得到的一次划分结果为（　　）

A）38,40,46,56,79,84

B）40,38,46,79,56,84

C）40,38,46,56,79,84

D）40,38,46,84,56,79

【答案】C

7．对记录的关键码 {50，26，38，80，70，90，8，30，40，20}进行排序，各趟排序结束时的结果为：

50,26,38,80,70,90 ,8,30,40,20
50,8,30,40,20,90,26,38,80,70
26,8,30,40,20,80,50,38,90,70
8,20,26,30,38,40,50,70,80,90

其使用的排序方法是（　　）

A）快速排序　B）基数排序　C）希尔排序　D）归并排序

【答案】C

8．在文件“局部有序”或文件长度较小的情况下，最佳内部排序方法是（　　）

A）直接插入排序　B）冒泡排序　C）简单选择排序　D）归并排序

【答案】A

【解析】当待排序列基本有序时，对冒泡排序来说，若最大关键字位于序列首部，则每趟排序仅能使其“下沉”一个位置，要使其下沉到底部仍需 n-1 趟排序，也即时间复杂度仍为 $0(n^2)$。而对简单选择排序来说，其比较次数与待排序列的初始状态无关；归并排序要求待排序列已经部分有序，而部分有序的含义是待排序列由若干有序的子序列组成，即每个子序列必须有序，并且其时间复杂度为 $0(n\ \log_2 n)$；直接插入排序在待排序列基本有序时，每趟的比较次数大为降低，也即 n-1 趟比较的时间复杂度由 $0(n^2)$降至 0(n)。

9．在下列算法中，（　　）算法可能出现下列情况：在最后一趟开始之前，所有的元素都不在其最终的位置上。

A）堆排序　B）冒泡排序　C）插入排序　D）快速排序

【答案】C

【解析】在插入排序中，如果待排序列中的最后一个元素其关键字值为最小，则在最后一趟开始之前，前 n-1 个排好序的元素都不在其最终位置上，与排好序后的位置相差一个位置。因此，选 C。

10．设有 5000 个无序的元素，希望用最快速度挑选出其中前 10 个最大的元素，在以下的排序方法中，采用（　　）方法最好

A）快速排序　B）堆排序　C）基数排序

【答案】B

【解析】用堆排序最好，因为堆排序不需要等整个排序结束就可挑出前 10 个最大元素，而快速排序

和基数排序都需等待整个排序结束才能知道前 10 个最大元素。

11．对给出的一组关键字 {14，5，19，20，11，19}。若按关键字非递减排序，第一趟排序结果为{14，5，19，20，11，19}，问采用的排序算法是（　　）

A）简单选择排序 B）快速排序　C）希尔排序　D）二路归并排序

【答案】C

12．以下序列不是堆的是（　　）

A）100,85,98,77,80,60,82,40,20,10,66

B）100,98,85,82,80,77,66,60,40,20,10

C）10,20,40,60,66,77,80,82,85,98,100

D）100,85,40,77,80,60,66,98,82,10,20

【答案】D

【解析】根据堆采用完全二叉树的顺序存储形式及堆的特点，因第一个结点即根结点关键字值最大，则应建立一个大根堆，但依据此数据序列建立起堆后关键字值为 40 的左右孩子结点分别为 60、66，不符合大根堆特点。

13．下面排序方法中，关键字比较次数与记录的初始排列无关的是（　　）

A）希尔排序 B）冒泡排序 C）直接插入排序 D）直接选择排序

【答案】D

【解析】如果初始排列基本有序，则对希尔排序来说，前几趟的插入工作大为减少。冒泡排序和直接插入排序都与初始排序序列有关，只有直接选择排序与初始序列无关．故选 D。

14．一组记录的关键字为{45，80，55，40，42，85}，则利用堆排序的方法建立的初始堆为（　　）

A）80,45,50,40,42,85

B）85,80,55,40,42, 45

C）85,80,55,45,42,40

D）85,55,80,42,45,40

【答案】B

15．一组记录的关键字为{25，50，15，35，80，85，20，40，36，70}，其中含有 5 个长度为 2 的有序表，用归并排序方法对该序列进行一趟归并后的结果为（　　）

A）15,25,35,50,20,40,80,85,36,70

B）15,25,35,50,80,20,85,40,70,36

C）15,25,50,35,80,85,20,36,40,70

D）15,25,35,50,80,20,36,40,70,85

【答案】A

【解析】对 5 个长度为 2 的有序表一趟归并后得到前两个长度为 4 的有序表和最后一个长度为 2 的有序表，故选 A。

16．n 个元素进行冒泡排序的过程中，最好情况下的时间复杂度为（　　）

A）O(1)　B）$O(\log_2^n)$　C）$O(n^2)$　D）O(n)

【答案】D

【解析】最好情况下至少需要一趟排序，即比较 n-1 次，故选 D。

17．n 个元素进行快速排序的过程中，第一次划分最多需要移动（　　）次元素(包括开始将基准元素移到临时变量的那一次)。

A）n / 2　B）n-1　C）n　D）n+l

【答案】D

【解析】移动次数最多的情况是对 n-1 个元素比较时都需移动，加上开始将基准元素移到临时变量以及由临时变量移至正确位置的二次，即共需 n+1 次，故选 D。

18．下述几种排序方法中，要求内存量最大的是（　　）

A）插入排序　B）选择排序　C）快速排序　D）归并排序

【答案】D

【解析】插入排序和选择排序需要的辅助空间为 O(1)，快速排序需要的辅助空间为 $O(\log_2^n)$，归并排序需要的辅助空间为 O(n)，因此选 D。

19．下面排序方法中，时间复杂度不是 $O(n^2)$的是（　　）

A）直接插入排序 B）二路归并排序 C）冒泡排序　D）直接选择排序

【答案】B

【解析】直接插入排序、冒泡排序和直接选择排序的时间复杂度为 $O(n^2)$，而二路归并排序的时间复杂度为 $O(n \log_2 n)$，故选 B。

20．对下列 4 个序列用快速排序方法进行排序，以序列的第 1 个元素为基准进行划分。在第 1 趟划分过程中，元素移动次数最多的是序列（　　）

A）70,75,82,90, 23,16,10,68

B）70,75,68,23,10,16,90,82

C）82,75,70,16,10,90,68,23

D）23,10,16,70,82,75,68,90

【答案】A

【解析】快速排序第一趟划分的方法是：将第 1 个元素放入最终排好序序列中的正确位置上，则在这个位置右边小于该元素值的元素都将移到其左边，在这个位置左边大于该元素值的元素都将其移到其右边。由此得到 A 需移动的元素最多，故选 A。

9.2　**填空题**

1．当数据量特别大需借助外部存储器对数据进行排序，则这种排序称为_____。

【答案】外部排序

2．在堆排序、快速排序和归并排序中，若从节省存储空间考虑，则应首先选取_____方法，其次选取_____方法；若只从排序结果的稳定性考虑，则应先择 _____方法；若只从平均情况下排序的速度来考虑，则选择 _____方法；若只从最坏情况下排序最快并且要节省内存考虑，则应选取_____方法。

【答案】（1）堆排序　　（2）快速排序　　（3）归并排序　（4）快速　（5）堆

3．对 n 个元素的序列进行冒泡排序，最少的比较次数是 _____，此时元素的排列情况为_____，在_____情况下比较次数最多，其比较次数为_____（4）_____。

【答案】

（1）n-1　（2）从小到大排序　（3）元素从大到小排列　（4）n(n-1)/2

【解析】初始元素正序时，第一趟比较 n-1 次，并无数据交换，则不再比较，故只比较 n-1 次。若反序，则比较(n-1)+(n-2)+(n-3)+…. . +2+1共 n(n-1)/2 次。

4．希尔排序是把记录按下标的一定增量分组，对每组记录进行直接插入排序，随着增量_____，所分成的组包含的记录越来越多，当增量的值为_____时，整个数组合为一组。

【答案】（1）减少　　（2）1

5．直接插入排序需借助的存储单元个数（空间复杂度）为_____，最好情况下直接插入排序的算法时间复杂度为_____，最坏情况下该算法的时间复杂度为_____。

【答案】（1）1　（2）O(n)　（3）$O(n^2)$

6．对 n 个数据进行简单选择排序，所需进行的关键字间的比较次数为_____，时间复杂度为_____。

【答案】（1）n(n-1)/2　（2）$O(n^2)$

7．对于关键字序列 (12，13，11，18，60，15，7，20，25，100)，用筛选法建堆，必须从键值为_____的关键字开始。

【答案】60

【解析】建堆必须从 n / 2 结点开始，而 10 / 2=5 位置的结点值为 60，故填 60。

8．对一组记录(54，38，96，23，15，72，60，45，83)进行直接插入排序时，当把第 7 个记录 60 插入到已排序的有序表时，为寻找其插入位置需比较_____次。

【答案】3

【解析】当把第 7 个记录 60 插入到有序表时，则前 6 个记录已经有序，此时记录 60 由后向前与有序表中的元素进行比较，直到遇到值小于 60 的记录为止，也即在有序表(15，23，38，54，72，96)中共需比较 3 次，因此填 3。

9．若对顺序存储在 A[l]～A[9]的记录(76，38，62，53，80，74，83，65，85)进行堆排序，已知除第一个元素 76 外，以其余元素为根的结点都已是堆，则对第一个元素进行筛运算时，它将最终被筛到 A 数组下标为_____的位置上。

【答案】8

【解析】从树结构关键字值看，除根外是小根堆。对第一元素进行筛运算时，得到的数据序列为：

38,53,62,65,80,74,83,76,85。

11．在时间复杂度为 $O(\log_2 n)$的排序方法中，_____排序方法是稳定的；在时间复杂度为 O(n)的排序方法中，_____排序方法是不稳定的。

【答案】（1）归并　　（2）直接选择

12．设表中元素的初态是按键值递增的，若分别用堆排序、快速排序、冒泡排序和归并排序方法对其仍按递增顺序进行排序，则_____最省时间，_____最费时间。

【答案】（1）冒泡排序　　（2）快速排序

【解析】若初始序列已经有序，则冒泡排序仅需一趟(比较 n-1 次)；而快速排序则需 n-1 趟，其时间复杂度升至 $O(n^2)$。因此填：冒泡排序，快速排序。

13．从一个无序序列建立一个堆的方法是：首先将要排序的 n 个键值分放到一棵_____的各个结点中，然后从 i=_____的结点 K_i 开始，逐步把以 K_{i-1}、K_{i-2}、…、K_1 为根的子树排成堆，直到以 K_1 为根的树排成堆，就完成了建堆的过程。

【答案】（1）完全二叉树　（2）n / 2 下取整。

14．在归并排序中，若待排序记录的个数为 20，则共需要进行_____趟归并，在第三趟归并中，是把长度为_____的有序表归并为长度为_____的有序表。

【答案】（1）5　（2）4　（3）8

【解析】第一次把长度为 1 的归并为长度的 2 的子表共 10 个，第二次把长度为 2 的归并成长度为 4 的子表共 5 个，第三次把长度为 4 的归并为长度为 8 的共 3 个，第四次长度为 8 归并为长度为 16 的，第 5 次归并成一个有序表。

9.3　判断题

1．对一个堆，按二叉树层次进行遍历可以得到一个有序序列（　）

【答案】×

【解析】堆的定义只规定了结点与其左、右孩子结点间的大小关系，而同一层上属不同父母的结点之间并无明确的大小关系，所以堆的层次遍历并不能得到一个有序序列。

2．内部排序就是整个排序过程完全在内存中进行的排序（　）

【答案】√

3．在数据基本有序时，直接插入排序法一定是性能最好的算法（　）

【答案】×

【解析】在数据量较少且数据基本有序时，直接插入法性能较好，但当数据量大时，则该算法的性能会大大降低。

4．当数据序列已有序时，若采用冒泡排序法，数据比较 n-1 次（　）

【答案】√

5．内排序中的快速排序方法，在任何情况下均可得到最快的排序效果（　）

【答案】×

【解析】快速排序在待排序记录为随机分布时效果最好，基本有序时效果最差。

6．用希尔方法排序时，若关键字的初始排序杂乱无序，则排序效率就低（　）

【答案】×

【解析】希尔排序又称“缩小增量排序”，即每趟只对相同增量距离的关键字进行比较，这与关键字序列初始有序或无序无关。

7．有一小根堆，堆中任意结点的关键字均小于它的左、右孩子关键字。则其具有最大值的结点一定是一个叶结点并可能在堆的最后两层中（　）

【答案】√

8．对 n 个记录的集合进行归并排序，在最坏情况下所需要的时间是 $O(n^2)$（　）

【答案】×

【解析】归并排序不受记录初始序列的影响，即所谓的最坏情况，其所需时间也是 $O(n\log_2 n)$。

9．对 n 个记录的集合进行冒泡排序，在最坏情况下所需要的时间是 $O(n^2)$（　）

【答案】√

9.4　应用题

1．什么是内排序? 什么是外排序? 什么排序方法是稳定的? 什么排序方法是不稳定的?

【答案】内排序是排序过程中参与排序的数据全部在内存中所做的排序，排序过程中无需进行内外存数据传送，决定排序方法时间性能的主要是数据排序码的比较次数和数据对象的移动次数。

外排序是在排序的过程中参与排序的数据太多，在内存中容纳不下，因此在排序过程中需要不断进行内外存的信息传送的排序。

假设在待排序的文件中存在两个或两个以上的记录具有相同的关键字，若采用某种排序方法后，使得这些具有相同关键字的记录在排序前后相对次序依然保持不变，则认为该排序方法是稳定的，否则就认为排序方法是不稳定的。

2．冒泡排序算法是否稳定？为什么？

【答案】冒泡排序算法是稳定的。因为依据该排序算法的基本思想，排序过程只比较相邻两个记录的关键字，若交换记录也只在相邻二个记录之间进行，从而可知在交换过程中不会出现跨越多个记录的情形。即使是相邻两个记录关键字相同时，经过比较也不会产生相邻记录的交换。所以冒泡排序法不会改变相同关键字记录的相对次序，故是稳定的。

3．在起泡排序过程中，什么情况下排序码会朝向与排序相反的方向移动，试举例说明。在快速排序过程中有这种现象吗？

【答案】如果在待排序序列的后面的若干排序码比前面的排序码小，则在起泡排序的过程中，排序码可能向与最终它应移向的位置相反的方向移动。例如：

初始关键字：59 45 10 90

第一趟排序：45 10 59 90

第二趟排序：10 45 59 90

其中 45 在第一趟排序中移向了与最终位置相反的方向。但在快速排序中不会出现这种情况，因为在每趟排序中，比基准元素大的都交换到右边，而比基准元素小的都交换到左边。

4．设待排序的排序码序列为{12, 2, 16, 30, 28, 10, 16*, 20, 6, 18 }, 试分别写出使用以下排序方法每趟排序后的结果。并说明做了多少次排序码比较。

（1） 直接插入排序　　（2） 希尔排序（增量为 5,2,1）

（3） 起泡排序　　（4） 快速排序

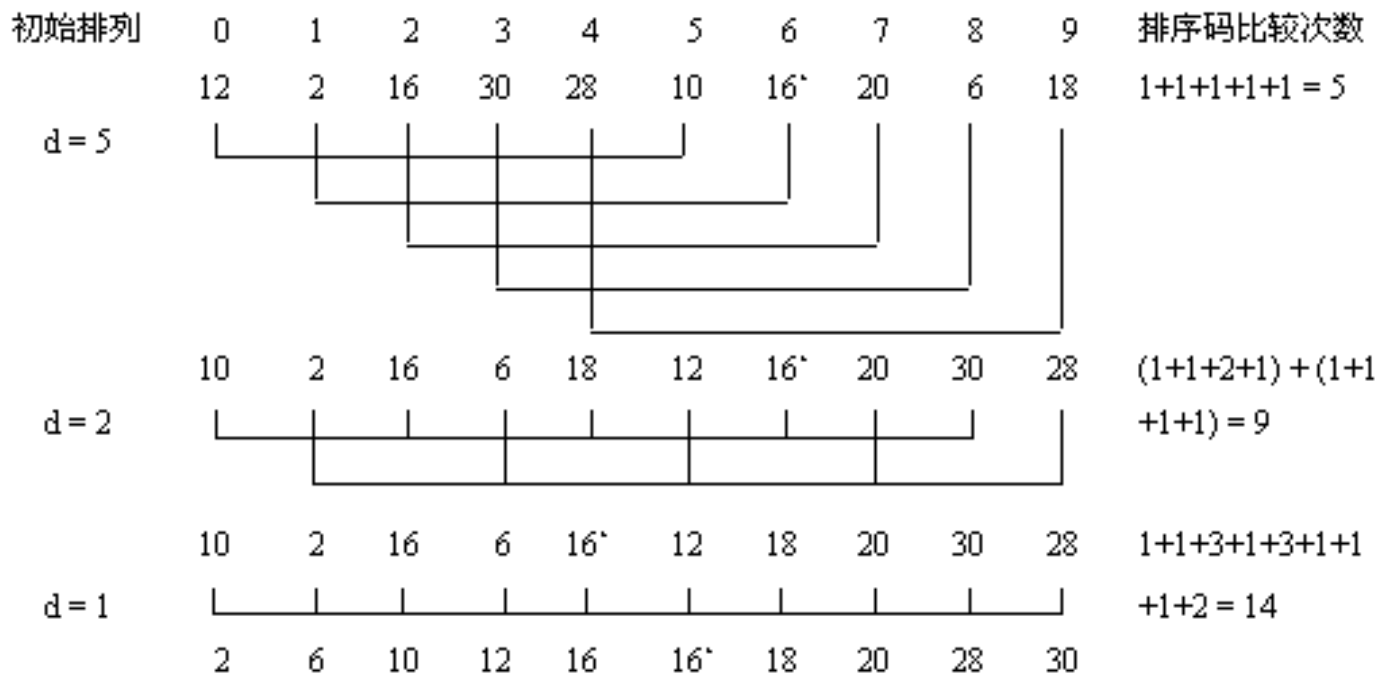
（5） 基数排序　　（6） 堆排序

【答案】

（1） 直接插入排序

初始排列	0	1	2	3	4	5	6	7	8	9	排序码比较次数
i=1	[12]	2	16	30	28	10	16`	20	6	18	1
i=2	[2	12]	16	30	28	10	16`	20	6	18	1
i=3	[2	12	16]	30	28	10	16`	20	6	18	1
i=4	[2	12	16	30]	28	10	16`	20	6	18	2
i=5	[2	12	16	28	30]	10	16`	20	6	18	5
i=6	[2	10	12	16	28	30]	16`	20	6	18	3
i=7	[2	10	12	16	16`	28	30]	20	6	18	3
i=8	[2	10	12	16	16`	20	28	30]	6	18	3
i=9	[2	6	10	12	16	16`	20	28	30]	18	8
	[2	6	10	12	16	16`	18	20	28	30]	

（2） 希尔排序(增量为 5,2,1)



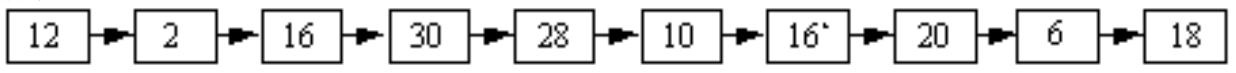
（3）起泡排序

初始排列	0	1	2	3	4	5	6	7	8	9	排序码比较次数
i = 0	[12	2	16	30	28	10	16`	20	6	18]	9
i = 1	2	[12	6	16	30	28	10	16`	20	18]	8
i = 2	2	6	[12	10	16	30	28	16`	18	20]	7
i = 3	2	6	10	[12	16	16`	30	28	18	20]	6
i = 4	2	6	10	12	[16	16`	18	30	28	20]	5
i = 5	2	6	10	12	16	[16`	18	20	30	28]	4
i = 6	2	6	10	12	16	16`	[18	20	28	30]	3
	2	6	10	12	16	16`	18	20	28	30	

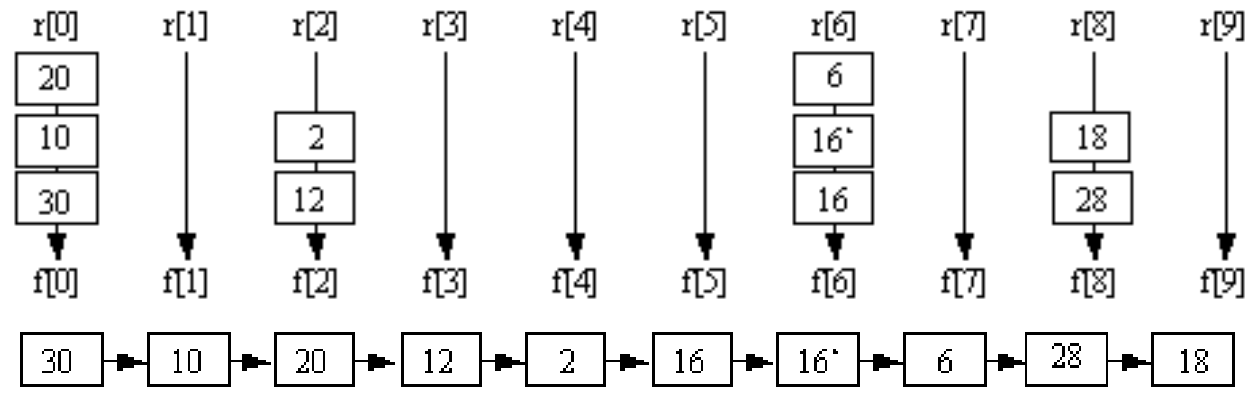
（4） 快速排序

Pivot	Pvtpos	0	1	2	3	4	5	6	7	8	9	排序码比较次数
12	0,1,2,3	**[12**	**2**	**16**	**30**	**28**	**10**	**16`**	**20**	**6**	**18]**	9
		↑pos	↑pos	↑pos	↑pos							
6	0,1	**[6**	**2**	**10]**	12	[28	16	16`	20	30	18]	2
		↑pos	↑pos									
28	4,5,6,7,8	[2]	6	[10]	12	**[28**	**16**	**16`**	**20**	**30**	**18]**	5
						↑pos	↑pos	↑pos	↑pos	↑pos		
18	4,5,6	2	6	10	12	**[18**	**16**	**16`**	**20]**	28	[30]	3
						↑pos	↑pos	↑pos				
16`	4	2	6	10	12	**[16**	**16]**	18	[20]	28	30	1
						· ↑pos						
		2	6	10	12	16`	[16]	18	20	28	30	

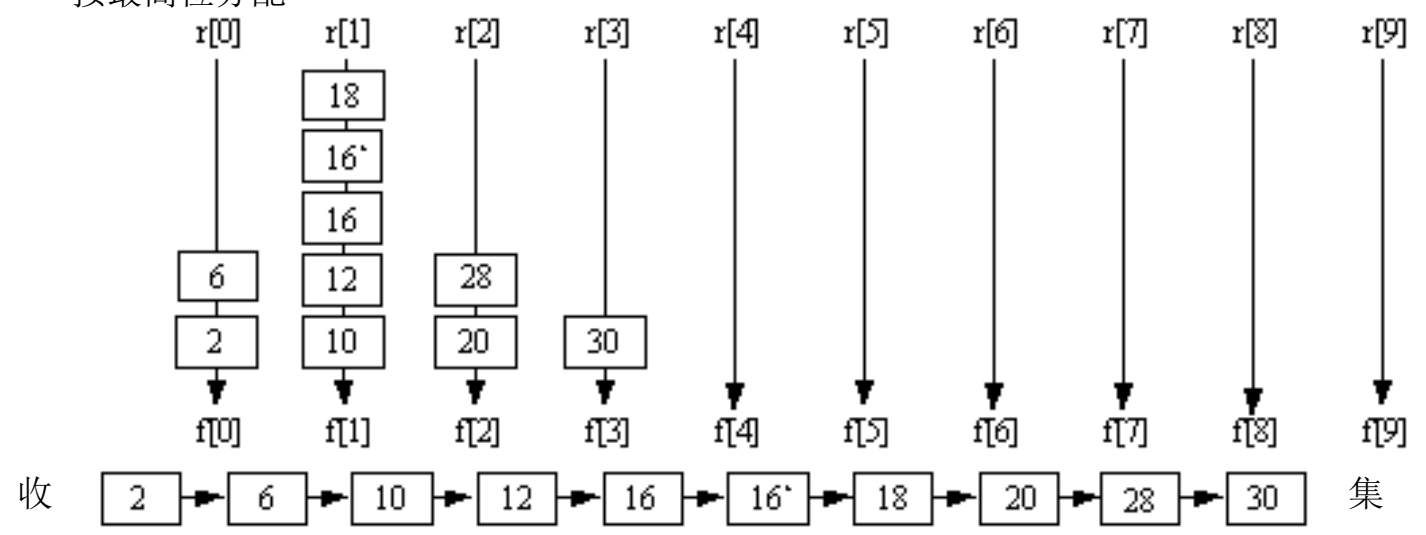
（5）基数排序



按最低位分配

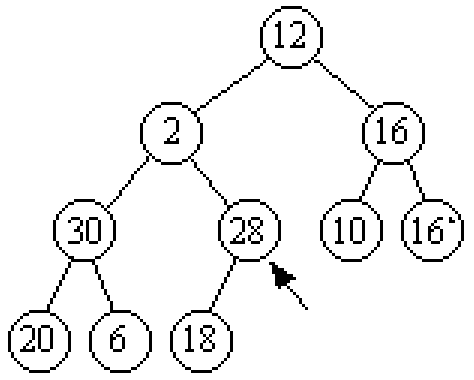


按最高位分配

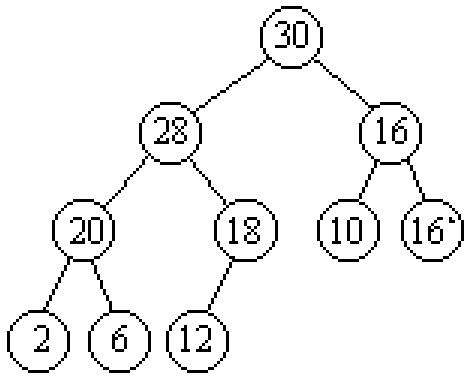


（6） 堆排序

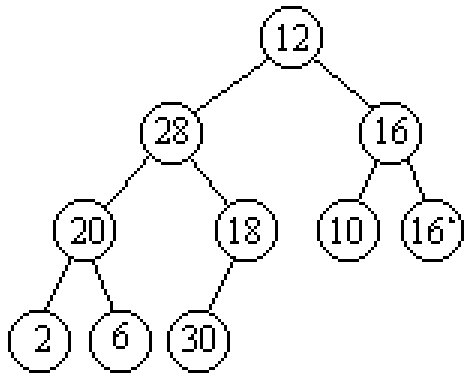
第一步，形成初始的最大堆 (略)，第二步，做堆排序。



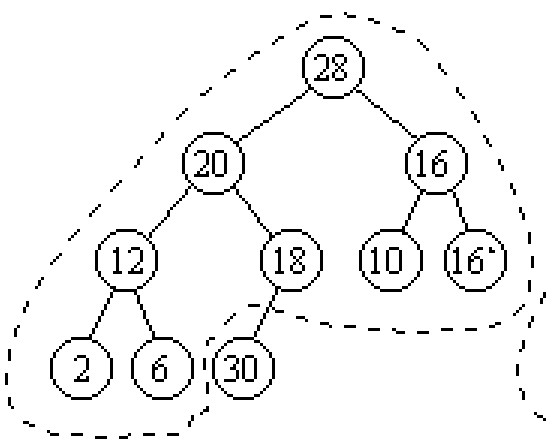
初始排列，不是最大堆



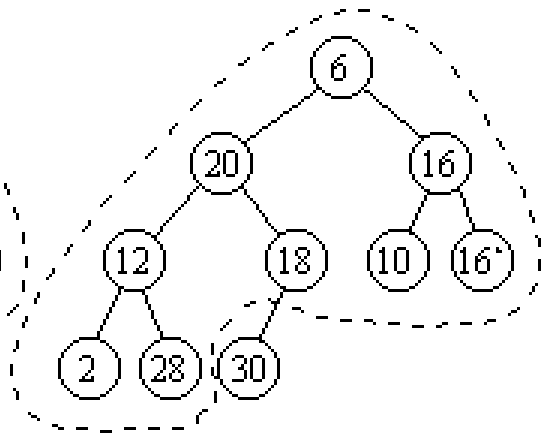
形成初始最大堆



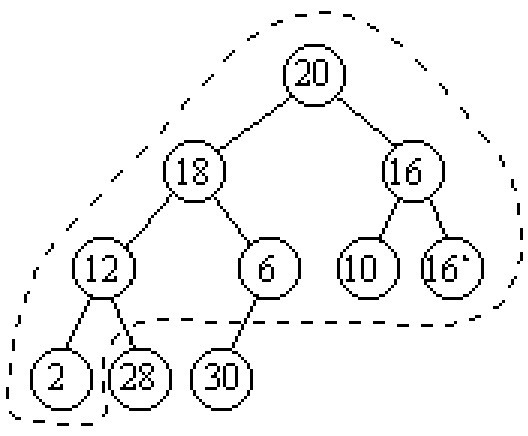
交换 0# 与 9# 对象



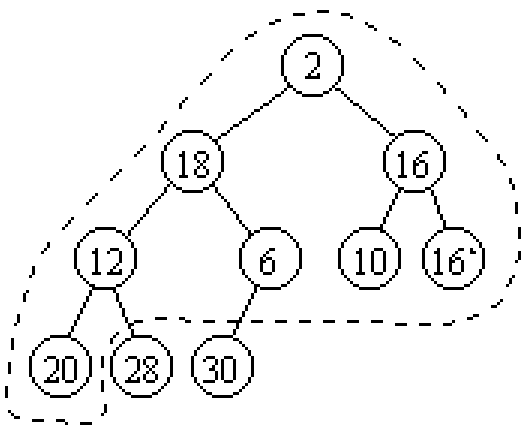
从 0# 到 8# 重新形成堆



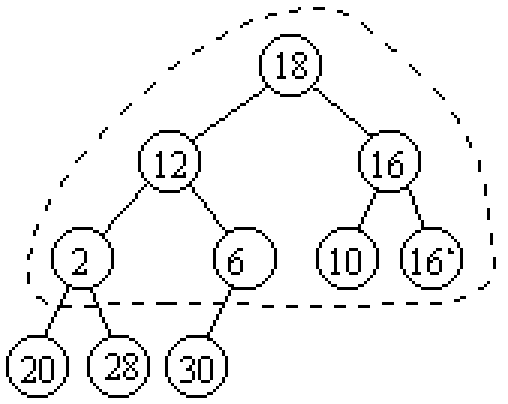
交换 0# 与 8# 对象



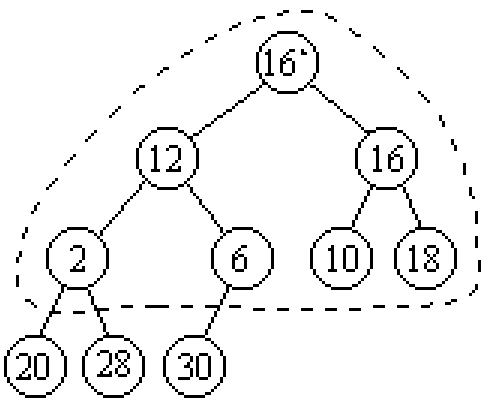
从 0# 到 7# 重新形成堆



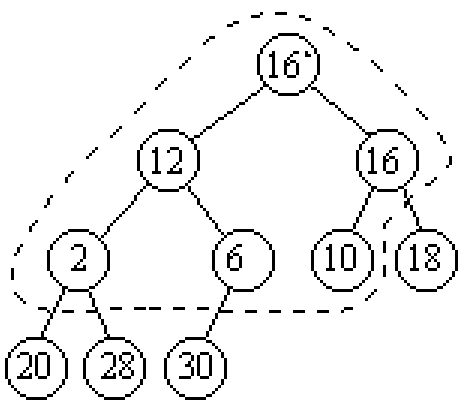
交换 0# 与 7# 对象



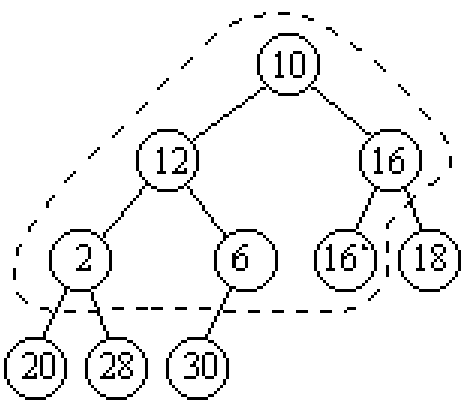
从 0# 到 6# 重新形成堆



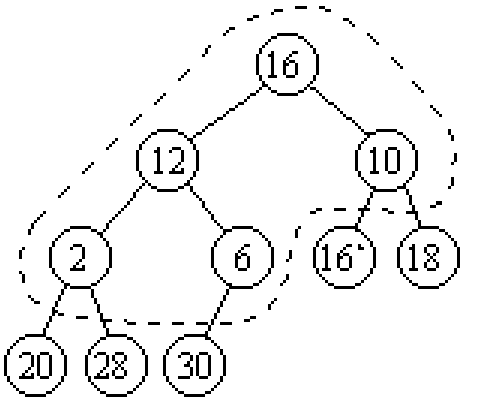
交换 0# 与 6# 对象



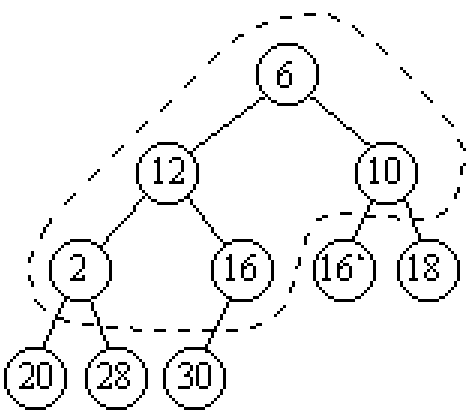
从 0# 到 5# 重新形成堆



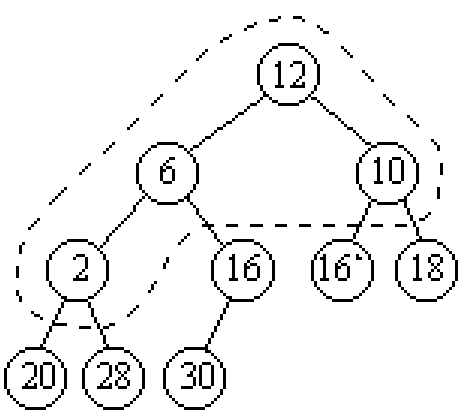
交换 0# 与 5# 对象



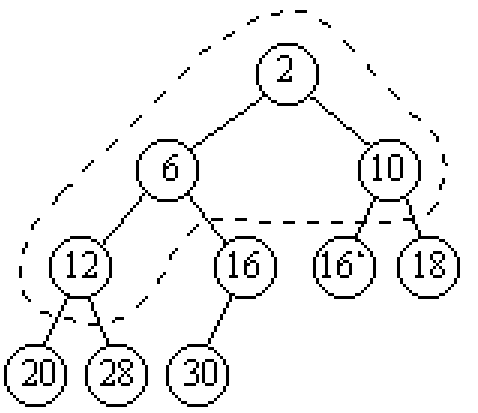
从 0# 到 4# 重新形成堆



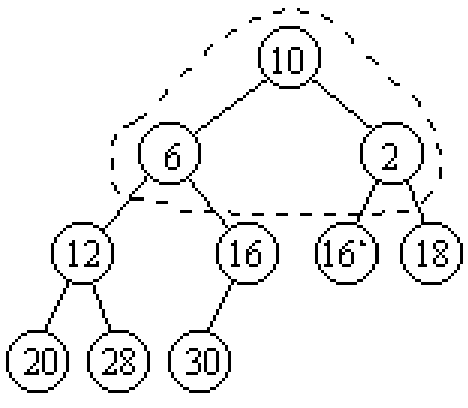
交换 0# 与 4# 对象



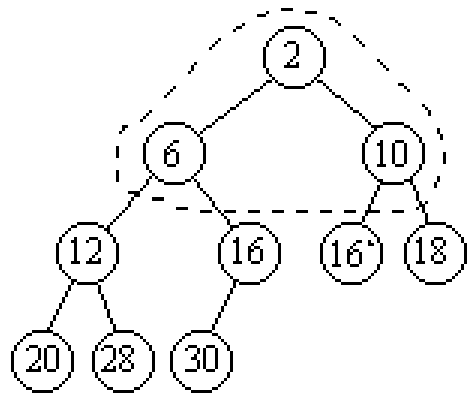
从 0# 到 3# 重新形成堆



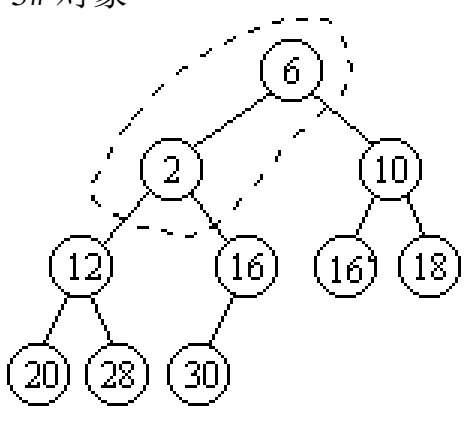
交换 0# 与 3# 对象



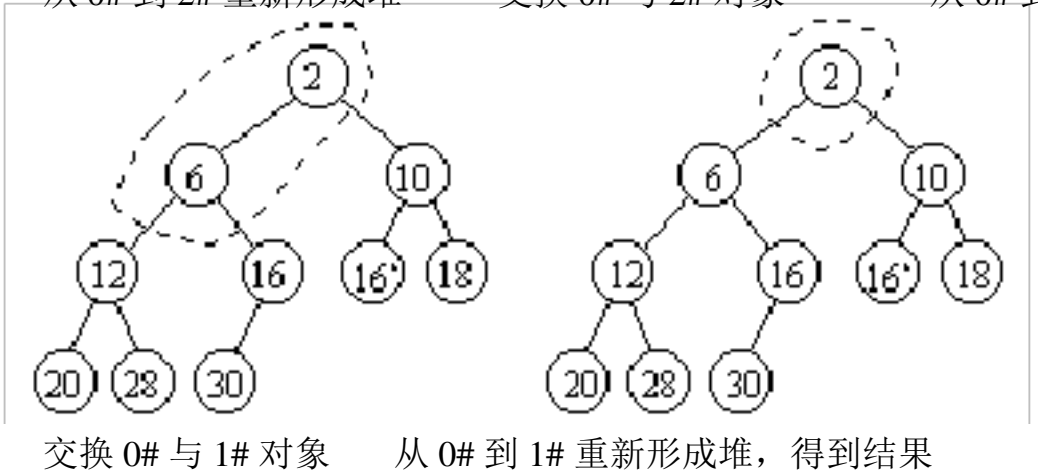
从 0# 到 2# 重新形成堆



交换 0# 与 2# 对象



从 0# 到 1# 重新形成堆



交换 0# 与 1# 对象　　从 0# 到 1# 重新形成堆，得到结果

5．试构造对 5 个整数元素进行排序，最多只用 7 次比较的算法思想。

【答案】设 5 个元素分别用 a，b，c，d，e 表示，取 a 与 b，c 与 d 进行比较，若 a>b，c>d（也可能

是 a<b，c<d，此时情况类似），显然此时进行了 2 次比较，取 b 与 d 再比较，若 b>d，则 a>b>d，若 b<d，则有 c>d>b，此时已进行了 3 次比较，要使排序比较最多 7 次，可把另外两个元素按折半检索排序插入到上面所得的有序序列中，此时共需要 4 次比较。从而经过 7 次比较对 5 个数进行排序。

6．对一个具有 7 个记录的文件进行快速排序，请问：

（1）在最好情况下需进行多少次比较？并给出一个最好情况初始排列的实例。

（2）在最坏情况下需进行多少次比较？为什么？并给出此时的实例。

【答案】

（1）在最好情况下，由于快速排序是一个划分子区间的排序，每次划分最好能得到两个长度相等的子表，设表的长度为 $n=2^k-1$，显然有，第一遍划分得到两个长度均为 $\lfloor n/2 \rfloor$ 的子表。第二遍划分得到 4 个长度均为 $\lfloor n/4 \rfloor$ 的子表，以此类推，总共进行 $k=\log_2(n+1)$遍划分，各子表的长度均为 1 时，此时排序结束。

由于 n=7,k=3，在最好情况下，第一遍经过 6 次，可找到一个其基准是正中间的元素，第二遍分别对两个子表（此时长度为 3）进行排序，各需要 2 次，这样就可将整个数据序列排序完毕，从而知 7 个数据的最好情况下需进行 10 次比较。如：4，7，5，6，3，1，2。

（2）在最坏情况下，若每次划分时用的基准，它的关键字值是当前记录中最大（或最小值），那么每次划分只能得到左子表（或右子表），子表长度只比原表减少了一个。因此，若初始排列的记录是按关键字递增或递减的，而所得的结果须为递减或递增排列的，此时快速排序就退化为与冒泡排序相似，而且时间复杂度为 $O(n^2)$，此时反而不快了。对于 n=7 的数据序列，显然最坏情况下的比较次数为 21。例如：7，6，5，4，3，2，1。

7．如果某个文件经内排序得到 80 个初始归并段，试问：

（1）若使用多路归并执行 3 趟完成排序，那么应取的归并路数至少应为多少？

（2）如果操作系统要求一个程序同时可用的输入/输出文件的总数不超过 15 个，则按多路归并至少需要几趟可以完成排序？如果限定这个趟数，可取的最低路数是多少？

【答案】

（1）设归并路数为 k，初始归并段个数 m = 80，根据归并趟数计算公式 $S = \lceil \log_k m \rceil = \lceil \log_k 80 \rceil = 3$ 得：$k^3 \geq 80$。由此解得 $k \geq 3$，即应取的归并路数至少为 3。

（2）设多路归并的归并路数为 k，需要 k 个输入缓冲区和 1 个输出缓冲区。1 个缓冲区对应 1 个文件，有 k +1 = 15，因此 k = 14，可做 14 路归并。由 $S = \lceil \log_k m \rceil = \lceil \log_{14} 80 \rceil = 2$。即至少需 2 趟归并可完成排序。若限定这个趟数，由 $S = \lceil \log_k 80 \rceil = 2$，有 $80 \leqslant k^2$，可取的最低路数为 9。即要在 2 趟内完成排序，进行 9 路排序即可。

9.5　算法设计题

1．试设计一个算法，使得在 O(n)的时间内重排数组，将所有取负值的排序码排在所有取正值(非负值)的排序码之前。

【算法分析】此题算法较简单，即开始时设头、尾两个下标指针分别指向第一和最后一个元素，头指针逐一向后移动直到遇到非负数为止，尾指针逐一向前移动直到遇到负数为止，此时交换头、尾指针所指的数，即：负数交换到前面而非负数交换到了后面。然后头、尾指针继续按刚才的规律移动并交换数据直到头、尾指针相遇为止。

【算法源代码】

```
void reArrange (int L[],int n)
{  int i = 0, j = n-1,temp;
   while ( i != j )
   { while ( L[i] < 0 )  i++;
     while ( L[j] >= 0 )  j--;
     temp=L[i];L[i]=L[j];L[j]=temp;
     i++;  j--; }
 }
```

2．奇偶交换排序是另一种交换排序。它的第一趟对序列中的所有奇数项 i 扫描，第二趟对序列中的所有偶数项 i 扫描。若 A[i] > A[i+1]，则交换它们。第三趟有对所有的奇数项，第四趟对所有的偶数项，…，如此反复，直到整个序列全部排好序为止。

【算法分析】根据题目要求，可设一个布尔变量 exchange，判断在每一次做过一趟奇数项扫描和一趟偶数项扫描后是否有过交换。若 exchange 为 1，表示刚才有过交换，还需继续做下一趟奇数项扫描和一趟

偶数项扫描；若 exchange 为 0，表示刚才没有交换，可以结束排序。

【算法源代码】

```
OddEvenSort ( int Vector[ ],int n)
{int i, exchange,temp;
do
{   exchange = 0;
    for ( i = 1; i < n-1; i += 2 )/*扫描所有奇数项*/
     if ( Vector[i] > Vector[i+1] )   /*相邻两项比较, 发生逆序*/
      { exchange = 1;   /*作交换标记*/
        temp=Vector[i]; Vector[i]=Vector[i+1]; Vector[i+1]=temp;/*交换*/
       }
    for ( i = 0; i < n-1; i += 2 )    /*扫描所有偶数项*/
      if ( Vector[i] > Vector[i+1] )      /*相邻两项比较, 发生逆序*/
       {   exchange = 1;              /*作交换标记*/
         temp=Vector[i]; Vector[i]=Vector[i+1]; Vector[i+1]=temp;       /*交换*/
       }
   } while ( exchange != 0 );
}
```

3．设计一个算法，实现双向冒泡排序。

【算法分析】冒泡排序从最下面的记录开始，对每两个相邻的关键字进行比较，且使关键字较小的记录换至关键字较大的记录之上，使得经过一趟冒泡排序后，关键字最小的记录到达最上端，接着，再在剩下的记录中找关键字最小的记录，并把它换到第二位置上。依次类推，一直到所有记录都有序为止。双向冒泡排序则是每一趟通过每两个相邻的关键字进行比较，产生最小和最大的元素。

【算法源代码】

```
void dbSort(int r[ ],int n)
{int i=1,j,t,b=1;
while(b)
   {b=0;
    for(j=n-i;j>=i;j--)  /*找最小元素*/
    if (r[j]<r[j-1])
      {b=1;   t=r[j];r[j]=r[j-1];r[j-1]=t;       }
    for(j=i;j<n-i;j++)   /*找最大元素*/
    if (r[j]>r[j+1])
      {b=1;   t=r[j];r[j]=r[j+1];    r[j+1]=t;}
    i++;
    }
}
```

4．写出快速排序的非递归算法。

【算法分析】设对记录空间 R[1..n]进行快速排序，要求用非递归算法，可以利用一个栈 s 来进行，其类型类型为 SqStack，每个栈元素含两个域：一个是 top 域，即栈顶指针；另一个为 data 域，用于存放元素，其中 data 数组元素含两个域，一个为 low，一个为 high，分别指示某个子文件的首、尾记录的首、尾地址，设栈空间最大容量为 MAXSIZE，而且假定在整个排序过程中不会发生溢出。

【算法源代码】

```
QuikSort(int R[ ],int n)
{int i,j,lw,hg,temp;
SqStack *s;
s->top=1;
s->data[s->top].low=1;s->data[s->top].high=n;
while(s->top!=0)    /*栈非空，则取出一个子文件进行划分*/
   {
   lw=s->data[s->top].low;        hg= s->data[s->top].high;
   s->top--;
   i=lw;  j=hg;  temp=R[i];
   do
   {while(i<j&&R[i]>temp)j--;
```

```
    if(i<j){R[i]=R[j]; i++;}
    while(i<j&&R[i]<temp)i++;
    if(i<j){R[j]=R[i];j--;}
   }while(i<j);  /*划分结束*/
   R[i]=temp;  /*基准元素插入*/
   if (i+1<hg)  /*右子文件有两个以上记录，子文件首、尾记录的地址进栈*/
            {s->top++;
            s->data[s->top].low=i+1;          s->data[s->top].high=hg;
            }
   if (lw<i-1)     /*左子文件有两个以上记录，子文件首、尾记录的地址进栈*/
            { s->top++;
            s->data[s->top].low=lw;          s->data[s->top].high=i-1;
            }
   }
}
```

5．对给定的 j(1≤j≤n)，要求在无序的记录区 R[1..n]中找到按关键字自小到大排在第 j 个位置上的记录(即在无序集合中找到第 j 个最小元)，试利用快速排序的划分思想编写算法实现上述的查找操作。

【算法分析】利用快速排序方法排序时，若 low 小于 high，此时应找出基准位置，若基准位置恰是要找的第 j 个位置，则直接返回该位置的数，若刚才的基准位置大于要找位置，则查找区域的上限改为刚才划分的基准位置-1，否则查找区域的下限改为刚才划分的基准位置+1。

【算法源代码】

```
int QuickSort(SqList R,int j,int low,int high)
 { int pivotpos;
 if(low<high)
 { pivotpos=Partition(R,low,high)  /*对 R[low..high]做划分*/
  if (pivotpos==j)  return R[j];
  else if (pivotpos>j)
   return quicksort(R,j,low,pivotpos-1);
  else
   return quicksort(R,j,pivotpos+1,high);
  }
}
```

6．将哨兵放在 R[n]中，被排序的记录放在 R[0..n-1]中，重新编写直接插入排序算法。

【算法分析】

用 R[n]作哨兵，则在插入数据时则是由后向前递推，即来一个待插入的数，把该数插入到其后的序列是有序的数据序列中，此时把待插入的数放到 R[n]中，然后找到插入其后序列的合适位置，此时需要把后续数据中的部分逐个前移，空出适当位置后，把 R[n]中保存的插入值直接放到空位置中去。

【算法源代码】

```
void InsertSort(SqList R)
{ int i,j;
for(i=n-2;i>=0;i--)
if(R[i].key>R[i+1].key)
{ R[n]=R[i];j=i+1;/*R[n]是哨兵*/
do{
R[j-1]=R[j];/*将关键字小于 R[i].key 的记录向右移*/
j++;
}while(R[j].key<R[n].key);
R[j-1]=R[n]; /*将 R[i]插入到正确位置上*/
}
}
```