1.1 算法：是对特定问题求解步骤的一种描述，是指令的有限序列。

程序：当一个算法用某种程序设计语言来描述时，得到的就是程序，也就是说,程序是用某种程序设计语言对算法的具体实现.

算法有输入、输出、确定性、能行性和有限性等特征，当不具备有穷性时，只能叫做计算过程，而不能称之为算法，算法可以终止，而程序没有此限制。

1.2 程序证明和程序测试的目的各是什么？

程序证明是确认一个算法能正确无误的工作.

程序测试的目的是发现错误

1-9 解: n!的递归定义： $n!=\begin{cases}1 & n=0\\ n*(n-1)! & n\geq 1\end{cases}$

求解 n!的递归函数

```
long Factorial (long n)
    {
        if(n<0)
        {
            cout<<"error!";
            exit(0);
        }
        if(n==0)
            return 1;
         else return n *Factorial (n-1);
     }
```

1-10 使用归纳法，证明上题所设计的计算 n！的递归函数的正确性

证明(归纳法证明)：

(1)首先，如果 n=0，那么程序执行

```
if(n==0)
    return 1;
```

返回 1,算法显然正确;

(2)假定函数 Factorial 对 n<k(>1)能正确运行，那么，当 n=k 时，算法必定执行:

```
else return k *Factorial (k-1);
```

因为 Factorial (k-1)正确，所以，当 n=k 时，程序运行正确

综合以上两点，可得程序正确.

证毕.

2-1, 简述衡量一个算法的主要性能标准,说明算法的正确性与健壮性的关系

答: 衡量一个算法的主要性能指标有:

正确性，简单性，高效低存储，最优性

算法的正确性与健壮性的关系:

所谓算法的正确性:是指在合法的输入下，算法应实现预先规定的功能和计算精度要求;所谓算法的健壮性，是指当输入不合法时，算法应该能按某种预定的方式做出适当的处理;

正确的算法不一定的是健壮的，健壮的算法也不一定是完全正确的.正确性和健壮性是相互补充的.一个可靠的算法，要求能在正常情况下正确的工作，而在异常情况下，亦

能做出适当处理.

2-9(1)设计一个 C/C++程序，实现一个 n*m 的矩阵转置，原矩阵与其转置矩阵保存在二维数组中.

```
Void reverse(int **a,int **b,int n,int m)
{
    For(int i=0;i<n;i++)
        For(int j=0;j<m;j++)
            b[j][i]=a[i][j];
}
```

(2)使用全局变量 count,改写矩阵转置程序，并运行修改后的程序以确定此程序所需的程序步

```
Void reverse(int **a,int **b,int n,int m,int &count)
{
    int i=0;
    count++;
    int j=0;
    count++;
    For(;i<n;i++)
        For(;j<m;j++)
        {
            count++;
            b[j][i]=a[i][j];
            count++;
         }
}
```

2-10 试用定义证明下列等式的正确性

(1) $5n^2-8n+2=O(n^2)$

证明: 因为当 $n_0=1,C=6$ 时,当 $n>n_0$ 时，有 $5n^2-8n+2<=6n^2$

2-16 使用递推关系式计算求 n!的递归函数的时间(即分析 1-9 题中的函数的时间复杂度),要求使用替换和迭代两种方法分别计算之.

解: 分析 1-9 题中的函数的时间复杂度:用基本运算乘法的运算次数作为衡量时间复杂度的量

当 n=0 时，程序执行 if(n==0) return 1;,并没有做乘法，故 T(0)=0;当 n>=1 时程序执行 n *Factorial (n-1);此时 T(n)= T(n-1)+1 故:

$$T(n)=\begin{cases}0 & n=0\\ T(n-1)+1 & n\geq 1\end{cases}$$

替换法: T(0)=0,T(1)=1,T(2)=2-----

总结得到:T(n)=n;

归纳法证明:

(1),当 n=0 时,T(0)=0,结论成立;

(2)假设当 k<n 时，有 T(k)=k,那么，当 k=n 时，有 T(n)=T(n-1)+1=(n-1)+1=n

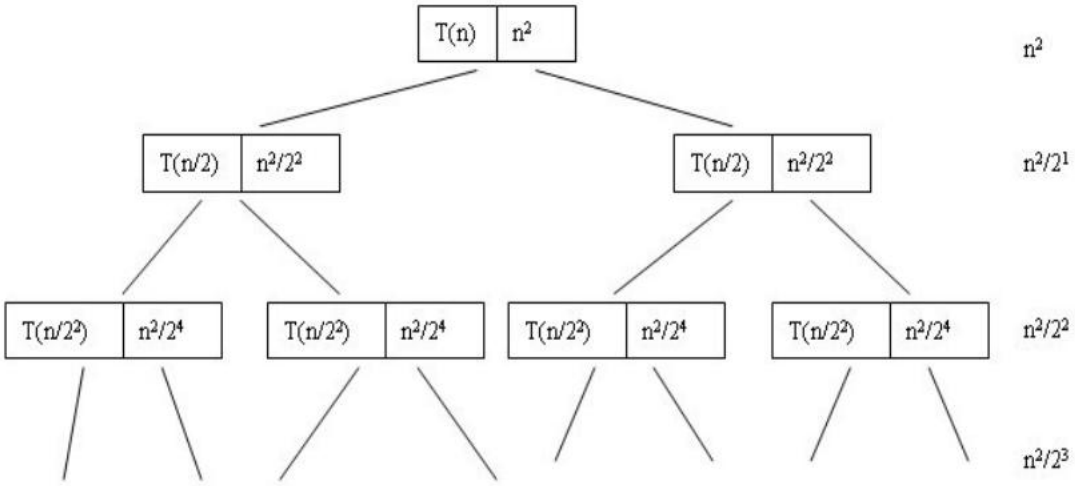
所以，对所有 n>=0 有 T(n)=n;成立.

迭代法:

T(n)=T(n-1)+1

=(T(n-2)+1)+1=((T(n-3)+1)+1)+1=....=T(0)+1+1.....+1(n 个 1)=n

2-19 利用递归树计算递推方程 $T(n)=2T(n/2)+n^2 \quad T(1)=2$



假设 $n=2^k$,那么,总共有 logn+1(即 k+1)层,非递归部分之和为

$n^2+n^2/2^1+n^2/2^2+\ldots+n^2/2^k=(1+1/2+1/2^2+1/2^3+\ldots+1/2^{logn})n^2$

$=2n^2+2n=O(n^2)$

5-8 三分搜索算法的做法是:它先将待查元素 X 与 n/3 处的元素比较，然后将 X 与 2n/3 处的元素比较,比较的结果或者找到 X,或者将范围缩小到原来的 n/3

```
int Search3(int a[],int left,int right,int x)     /*递归算法*/
{
    int l,u;
    if(left<=right)
   {
      l=left+(right-left)/3;
      u=left+(right-left)*2/3;
      if(x==a[u])
          return u;
      else if(x==a[l])
                return l;
            else if(x>a[u])
                      return Search3(a, u+1, right,x);
                 else if(x>a[l])
                           return Search3(a, l+1, u-1,x);
                      else
                              return Search3(a, left, l-1,x);
   }
   return -1;
}
void main()
{
```

```
    int n,*a;
    int x,i;
    cout<<"Please input n:";
    cin>>n;
    a=new int[n];                                   //动态数组
    int location=-1;
    for(i=0;i<n;i++)
    {
        a[i]=i+1;
    }
    cout<<"Please input the search x:";
    cin>>x;
      cout<<endl<<Search3(a, 0, n-1,x)<<endl;
}
void main()                                                     /*非递归算法*/
{
    int a[15];
    int x,i;
    int location=-1;
    for(i=0;i<15;i++)
    {
        a[i]=i+1;
    }
    cin>>x;
    i=0;
    int j=14,l,u;
    while(i<=j)
    {
        l=i+(j-i)/3;
        u=i+(j-i)*2/3;
        if(x==a[u])
        {
            location=u;
            break;
        }
        else if(x==a[l])
        {
            location=l;
            break;
        }
            else if(x>a[u])
                    i=u+1;

                else if(x<a[l])
```

```
                    j=l-1;
                else
                {
                  i=l+1;
                  j=u-1;
                }
    }
    cout<<location<<endl; //x 的位置
}
```

5-12

```
Void stoogesort(nt a[],int left,int right)
{
   if(a[left]>a[right]) swap(a,left,right);
   if(left+1>=right) return;
   int k=(right-left+1)/3;
   stoogesort(a,left,right-k);
   stoogesort(a,left+k,right);
   stoogesort(a,left,right-k);
}
```

证明:

元素个数 n=right-left+1;

(1) 若为空表或只有一个元素(n=1 时,即 left+1==right)时，程序执行 if(a[left]>a[right]) swap(a,left,right);之后，执行 if(left+1>=right) return;即此时程序做了一次元素之间的比较之后，不做任何操作，显然正确.

(2) 假设当 n< right-left+1(n>=2)时，算法正确，即对于所有元素个数小于 n 的元素集,算法能正确排序.

那么,当 n= right-left+1 时，算法执行程序段:

```
int k=(right-left+1)/3;
stoogesort(a,left,right-k);
stoogesort(a,left+k,right);
stoogesort(a,left,right-k);
```

由假设可知:以上三条语句都能正确运行，所以，当 n= right-left+1 时,算法正确.

由以上两点可知，算法正确.

分析算法的时间复杂度:

排序算法，基本运算仍然是元素之间的比较,所以，算法时间复杂度为:

$$T(n)=\begin{cases}1 & n=0,1\\ 3T(2n/3)+1 & n\geq 2\end{cases}$$ （用替换或迭代法计算之即可）

6-1 设有背包问题实例，n=7，(w0,w1,w2,w3,w4,w5,w6)=(2,3,5,7,1,4,1)，(p0,p1,p2,p3,p4,p5,p6)=(10,5,15,7,6,18,3)，M=15。求这一实例的最优解及最大收益.

解:

首先，选择最优量度标准为收益重量比;

其次，依据收益重量比的非增次序对输入(物品)进行排序

(p0/w0,p1/w1,p2/w2,p3/w3,p4/w4,p5/w5,p6/w6)=(5,5/3,3,1,6,4.5,3)

对物品排序结果为:4,0,5,2,6,1,3

最后，进行贪心选择:

	X=(4)	X=(4,0)	X=(4,0,5)
(剩余载重)U=14		U=12	U=8
(收益)	P=6	P=6+10=16	P=16+18=34

	X=(4,0,5,2)	X=(4,0,5,2,6)	X=(4,0,5,2,6,1(2/3))
(剩余载重)U=3		U=2	U=0
(收益)	P=34+15=49	P=49+3=52	P=52+2/3*5=55.33

所以，最优解为 x=(1, 2/3, 1, 0, 1, 1, 1)； 即装入第 0, 2, 4, 5, 6 物品和第 1 个物品的 2/3

最大收益: P=55.33

6-2, 0/1 背包问题是一种特殊的背包问题，装入背包的物品不能分割，只允许或者整个物品装入背包，或者不装入，即 xi=0, 或 1, (0<=i<n)，以题 6-1 的数据作为 0/1 背包的实例，按贪婪法求解，这样求得的解一定是最优解吗?为什么?

解:

首先，选择最优量度标准为收益重量比;

其次，依据收益重量比的非增次序对输入(物品)进行排序

(p0/w0,p1/w1,p2/w2,p3/w3,p4/w4,p5/w5,p6/w6)=(5,5/3,3,1,6,4.5,3)

对物品排序结果为:4,0,5,2,6,1,3

最后，进行贪心选择:

	X=(4)	X=(4,0)	X=(4,0,5)
(剩余载重)U=14		U=12	U=8
(收益)	P=6	P=6+10=16	P=16+18=34

	X=(4,0,5,2)	X=(4,0,5,2,6)	X=(4,0,5,2,6)
(剩余载重)U=3		U=2	继续考察第 1 和第 3 个
(收益)	P=34+15=49	P=49+3=52	物品,都不能装入.

所以，贪心法求得的 0/1 背包问题的最优解为 x=(1, 0, 1, 0, 1, 1, 1)；即装入第 0, 2, 4, 5, 6 物品

最大收益: P=52

但实际上，当 y=(1, 1, 1, 0, 1, 1, 0) 即装入第 0, 1, 2, 4, 5 物品，可获收益为 P=54，所以，贪心法求得的 0/1 背包问题的解 x 一定不是最优解.

原因是: 对于 0/1 背包问题，贪心法并不能保证使其单位载重下的收益最大，因为通常在背包没还装满时，却再也装不下任何物品，这样，就使得单位载重下的物品收益减少，所以， 0/1 背包问题通常不能用贪心法求解.

6-3 设有带时限的作业排序实例 n=7，收益(p0, p1, p2, p3, p4, p5, p6)=(3,5,20,18,1,6,30)，作业的时限(d0, d1, d2, d3, d4, d5, d6)=(1,3,4,3,2,1,2),给出以此实例为输入，执行函数 JS 得到的用最优解和最大收益。

解:X={5,6,3,2} 最大收益为 74

函数 JS 如下:

```
int JS(int *d, int *x, int n)
{ //设 p0≥p1≥…≥pn-1
    int k=0; x[0]=0;
    for (int j=1; j<n; j++){//O(n)
        int r=k;
        while (r>=0 && d[x[r]]>d[j] && d[x[r]]>r+1)r--;        //搜索作业 j 的插入位置
        if((r<0 || d[x[r]]<=d[j]) && d[j]>r+1){                //若条件不满足，选下一个作业
            for (int i=k; i>=r+1; i--) x[i+1]=x[i];            //将 x[r]以后的作业后移
            x[r+1]=j; k++;                                     //将作业 j 插入 r+1 处
        }
    }
    return k;
}
```

在执行 JS 函数之前，必须先对输入(即作业)按作业的收益非增次序排序，结果为:

6,2,3,5,1,0,4

接着执行 JS 函数:

最初，解集合 X 为空. X:

首先，考虑作业 6，假设将其加入集合 X，即 x[0]=6;

X:

考虑 X 中的作业能否均如期完成,因为此时 X 中只有作业 6,其截止时限为 2,故,能如期完成,此时,将作业 6 加入作业子集 X 中,此时，子集 X 中的最大可用下标 k=0;

X:

接着，考虑作业 2.

首先搜索作业 2 在 X 集合中的插入位置，使得 X 集合中的元素按作业的截止时限的非减次序排序,因为 d6=2,而 d2=4,所以，可将作业 2 插在作业 6 的后面，即 x[1]=2,得到 X=(6,2),

X:

考虑 X 中的作业能否均如期完成?因为 d6=2>=1, d2=4>=2,所以，X 中作业均能如期完成，将作业 2 加入子集 X 中. 子集 X 中的最大可用下标 k=k+1=1

X:

6	2					
0	1	2	3	4	5	6
	k					

考虑作业 3.

首先搜索作业 3 在 X 集合中的插入位置，使得 X 集合中的元素按作业的截止时限的非减次序排序,因为 d6=2, d2=4,而 d3=3 所以，可将作业 3 插在作业 6 的后面,作业 2 的前面，得到 X=(6,3,2),

X:

6	3	2				
0	1	2	3	4	5	6
	k					

考虑 X 中的作业能否均如期完成?因为 d6=2>=1, d3=3>=2, d2=4>=3 所以，X 中作业均能如期完成，将作业 2 加入子集 X 中. 子集 X 中的最大可用下标 k=k+1=2

X:

6	3	2				
0	1	2	3	4	5	6
		k				

考虑作业 5.

首先搜索作业 5 在 X 集合中的插入位置，使得 X 集合中的元素按作业的截止时限的非减次序排序,因为 d6=2, d2=4, d3=3 而 d5=1 所以，可将作业 5 插在作业 6 的前面,得到 X=(5,6,3,2),

X:

5	6	3	2			
0	1	2	3	4	5	6
		k				

考虑 X 中的作业能否均如期完成?因为 d5=1>=1,d6=2>=2, d3=3>=3, d2=4>=4 所以，X 中作业均能如期完成，将作业 5 加入子集 X 中. 子集 X 中的最大可用下标 k=k+1=3

X:

5	6	3	2			
0	1	2	3	4	5	6
			k			

考虑作业 1.

首先搜索作业 1 在 X 集合中的插入位置，使得 X 集合中的元素按作业的截止时限的非减次序排序,因为 d5=1,d6=2, d3=3,d2=4,而 d1=3 所以，可将作业 1 插在作业 2 的前面,作业 3 的后面,得到 X=(5,6,3,1,2),

X:

5	6	3	1	2		
0	1	2	3	4	5	6
			k			

考虑 X 中的作业能否均如期完成?因为 d5=1>=1,d6=2>=2, d3=3>=3, d1=3<4 所以，X 中 1 作业不能如期完成，所以，不能将作业 1 加入子集 X.

X:

5	6	3	2			
0	1	2	3	4	5	6

k

接着考虑作业 0,4 均不能加入子集 X,

故，执行 JS 得到的最优解为 X=(5,6,3,2),最大收益为 P=p5+p6+p3+p2=30+20+18+6=74

6-17,最佳装载问题是将一批集装箱装上一艘载重为 C 的轮船，其中集装箱 i 的重量为 wi(0<=i<=n-1),最优装载问题是指在装载体积不受限制的情况下,求使得装箱数目最多的装载方案.

(1)按贪心策略的要求,给出关于上述最优化问题的形式化描述.

(2)给出贪心法求解这一问题的最优量度标准;

(3)讨论其最优解的最优子结构.

(4)编写装箱问题的贪心算法;

(5)设有重量为(4,6,3,5,7,2,9)的 7 个集装箱，轮船载重为 26,求最优解.

解;(1),形式化描述如下：

给定C>0, w_i>0 $0 \le i \le n-1$ 求X=($(x_0, x_1, x_2 ... x_{n-1})$ $x_i \in \{0,1\}$ $0 \le i \le n-1$)

使得 $\sum_{i=0}^{n-1} x_i w_i \le C$ 并且使 $\sum_{i=0}^{n-1} x_i$ 最大

（2）以重量作为最优量度标准，以重量最轻者先装来选择集装箱装上船

（3）设$(x_0,x_1,\text{----}x_{n-1})$是最优装载问题的最优解，则易知$(x_1,x_2,\text{----}x_{n-1})$是轮船载重为 $C-x_0w_0$ 且待装船的集装箱为{1，3----n-1}时相应最优装载问题的一个最优解，即最优装载问题具有最优子结构特性。否则，假设$(x_1,x_2,\text{----}x_{n-1})$不是子问题的最优解，假设有另一个解 Z=$(z_1, z_2,\text{----}z_{n-1})$是子问题的最优解，则有：

$$\sum_{1\le i\le n-1} x_i < \sum_{1\le i\le n-1} z_i \text{且} \sum_{1\le i\le n-1} w_i z_i \le C - w_0 x_0,$$

则：$x_0 + \sum_{1\le i\le n-1} x_i < x_0 + \sum_{1\le i\le n-1} z_i$ 且 $x_0 w_0 + \sum_{1\le i\le n-1} w_i z_i \le C$，即$(x_0, z_1, z_2,\text{--}\ z_{n-1})$是最优装载问题的最优解,与$(x_0,x_1,\text{----}x_{n-1})$是最优装载问题的最优解矛盾，所以, $(x_1,x_2,\text{----}x_{n-1})$是子问题的最优解,故最优装载问题具有最优子结构特性。

(4) 参考程序 1

```
/*箱子信息结构体*/
   struct goodinfo
   {
    float w; /*箱子重量*/
    int X; /*箱子存放的状态*/
    int flag; /*箱子编号*/
```

```
};

/*按物品重量做升序排列*/
void sort(goodinfo goods[],int n)
{
 int j,i;
 for(j=2;j<=n;j++)
 {
        goods[0]=goods[j];
        i=j-1;
        while (goods[0].w<goods[i].w)
        {
              goods[i+1]=goods[i];
              i--;
        }
        goods[i+1]=goods[0];
 }
}

/*用贪心法对物品进行选择装入*/
void loading(goodinfo goods[],float M,int n)
{
 float cu;
 int i,j;
 int A=0;/*对装入箱子进行计数*/
 for(i=1;i<=n;i++)/*赋初值*/
        goods[i].X=0;

 cu=M; /*船的剩余载重*/
 for(i=1;i<n;i++)
 {
        if(goods[i].w>cu)/*当该箱重量大于剩余载重跳出*/
              break;
        goods[i].X=1;
        A++;
        cu=cu-goods[i].w;/*确定船的剩余载重*/
 }

 for(j=2;j<=n;j++)/*对箱子按序号大小作升序排列*/
 {
        goods[0]=goods[j];
        i=j-1;
        while (goods[0].flag<goods[i].flag)
        {
```

```
            goods[i+1]=goods[i];
            i--;
        }
        goods[i+1]=goods[0];
    }

    cout<<"① 最优解为:"<<endl;/*输出*/
    for(i=1;i<=n;i++)
    {
        cout<<"    第"<<i<<"件物的存放状态:";
        cout<<goods[i].X<<endl;
    }
    cout<<"<0：未装入；1：装入>"<<endl;
    cout<<endl;
    cout<<"② 最多能装入的箱子数为:";
    cout<<A<<endl;
  }
```

(5)

首先，选择最优量度标准为重量;

其次，依据集装箱重量的非减次序对输入(物品)进行排序

对集装箱的排序结果为:5,2,0,3,1,4,6

最后，进行贪心选择:

	X=(5)	X=(5,2)	X=(5,2,0)
(剩余载重)U=24		U=21	U=17
	X=(5,2,0,3)	X=(5,2,0,3,1)	X=(5,2,0,3,1)
(剩余载重)U=12		U=6	

所以，最优解为 X=(0,1,2,3, 5),最优解值为 5

参考程序 **2**

```
public static float loading(float c, float [] w, int [] x)
    {
        int n=w.length;
        Element [] d = new Element [n];
        for (int i = 0; i < n; i++)
           d[i] = new Element(w[i],i);
        MergeSort.mergeSort(d);
        float opt=0;
        for (int i = 0; i < n; i++) x[i] = 0;
        for (int i = 0; i < n && d[i].w <= c; i++) {
            x[d[i].i] = 1;
            opt+=d[i].w;
            c -= d[i].w;
            }
        return opt;
```

}

7-5 设有 4 个矩阵连乘积 ABCD：A: 45*8, B: 8*40, C: 40*25, D: 25*10, ,请求出它们的最优计算次序和计算量。

解：p0=45，p1=8，p2=40，p3=25，p4=10

可只给出矩阵形式

计算 m 矩阵为： m[0][1]= p0*p1* p2=45*8*40=14400;

m[1][2]= p1*p2* p3=8*40*25=8000;

m[2][3]= p2*p3* p4=40*25*10=11250;

m[0][2]= m[0][0]+m[1][2]+p0*p1* p3=8000+45*8*25=8000+9000=17000

= m[0][1]+m[2][2]+p0*p2* p3=14400+45*40*25=14400+45000

m[1][3]= m[1][1]+m[2][3]+p1*p2* p4=11250+8*40*10=11250+3200

= m[1][2]+m[3][3]+p1*p3* p4=8000+8*25*10=10000

m[0][3]= m[0][0]+m[1][3]+p0*p1* p4=10000+45*8*10=10000+3600=13600

= m[0][1]+m[2][3]+p0*p2* p4=14400+11250+45*40*10=

=m[0][2]+m[3][3]+p0*p3* p4=17000+45*25*10=17000+11250=28250

$$m=\begin{pmatrix} 0 & 14400 & 17000 & 13600 \\ & 0 & 8000 & 10000 \\ & & 0 & 11250 \\ & & & 0 \end{pmatrix} \qquad s=\begin{pmatrix} 0 & 0 & 0 & 0 \\ & 1 & 1 & 2 \\ & & 2 & 2 \\ & & & 3 \end{pmatrix}$$

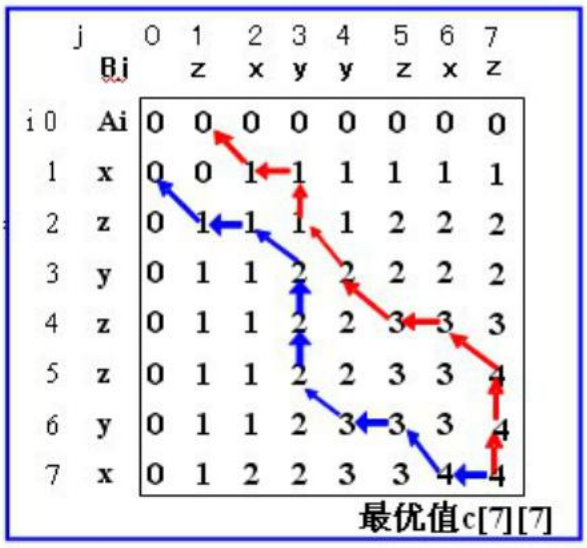
这 4 个矩阵相乘需要的最小数量乘法的次数=13600

最优计算次序 A（（BC）D）

7-9 给定字符串 A=“xzyzzyx”和 B=“zxyyzxz”，使用 LCS 算法求最长公共子串，并给出一个最长公共子串。

提示：从上到下，从左往右计算 C 矩阵，依据 C 矩阵，求得最长公共子序列

解：计算求得 C 矩阵如下，



依矩阵 C 可求得两个最长公共子序列分别为 xyzz 和 zyyx （求一个即可）

7-17 设流水作业调度的实例为 n=7，$(a_0, a_1, a_2, a_3, a_4, a_5, a_6)=(6,2,4,1,7,4,7)$, $(b_0, b_1, b_2, b_3, b_4, b_5, b_6)=(3,9,3,8,1,5,6)$.请使用流水作业调度的 Johnson 算法求使完成时间最小的最优调度,并求该最小完成时间。

提示：依据调度规则，求得最优调度次序

解:；令 mi=min{ai，bi}　0<=i<7

即得:　m0=b0=3,　m1=a1=2,　m2=b2=3,　m3=a3=1,

m4=b4=1,　m5=a5=4　m6=b6=6

考虑 mi,对其从小到大排序得(m3,m4,m1,m0,m2,m5,m6)

考虑 mi 序列(如果序列中下一个数 mi 是 ai,则作业 i 放在最左的空位，否则，作业 i 放在最右的空位)得:

最优调度顺序(3,1,5,6,2,0,4)

依据最优调度在两台处理机上处理作业，最小完成时间:36(画出如教材 P170 的图 7-17 的形式即可求得最小完成时间 36)

8-2

```
#include <iostream.h>
#include <math.h>

int count=0;//记录可行解的个数

//先将书中的递归变为如下非递归函数，再在输出一个可行解之后，加上 break;
int place(int k,int xk,int *x)
{

 for(int j=0;j<k;j++)
      if(x[j]==xk || abs(x[j]-xk)==abs(j-k))//相互冲突,return 0;
           return 0;
 return 1;//互不冲突,return 1
}
void NQueens(int n,int *x)
{
 int k=0;
 x[k]=-1;
 while(k>=0)
 {
        x[k]=x[k]+1;
      while (x[k]<n && !place(k,x[k],x))//找安置第 K 个皇后的合法位置
          x[k]=x[k]+1;
      if(x[k]<n)//找到安置第 K 个皇后合法的位置，
           if (k==n-1)//找到一个可行解，输出
           {
               cout<<"The solution is:  ";
               for(int j=0;j<n;j++)//找到一个可行解，输出
```

```
                    cout<<x[j]<<"    ";
                cout<<endl;
//              break;//删除，则可输出所有可行解。
                count++;
            }
            else //找到安置第 K 个皇后合法的位置，但只是部分向量，所以，继续安置下一个皇后，首先将下个皇后放在棋盘外面
            {
                k=k+1;
                x[k]=-1;
            }
        else k--;//找不到安置第 K 个皇后合法的位置，回溯到 k-1，将 K-1 皇后安置到下一个位置 x[k]=x[k]+1;
    }
}
void main()
{
 int n;
 cout<<"Please input the number n:";
 cin>>n;
 int *x=new int[n];

/*    for(int i=0;i<n;i++)
 {
      x[i]=-1;
 }*/
 NQueens(n,x);
     cout<<"the number of the solutions is:"<<count<<endl;
}
```

8-3

```
#include <iostream.h>
#include <math.h>
int place(int k,int n,int xk,int *x)
{
 for(int i=xk;i<n;i++)
 {
      for(int j=0;j<k;j++)
          if(x[j]==i || abs(x[j]-i)==abs(j-k))//相互冲突，检测下一个位置 i=i+1
              j=k;
         if(j==k)//互不冲突
          return i;
 }
 return -1;
```

```
}
void NQueens(int n,int *x)
{
 int k=0;
 while(k>=0)
 {
      int f=place(k,n,x[k],x);
      if(f!=-1)
      {
           x[k]=f;
           if(k==n-1)
           {
                cout<<"The solution is:    ";
                for(int i=0;i<n;i++)
                     cout<<x[i]<<"    ";
                cout<<endl;
                x[k]=0;
                k--;
                x[k]+=1;
           }
           else k++;
      }
      else
      {
           x[k]=0;
           k--;
           x[k]+=1;
      }
 }
}
void main()
{
 int n;
 cout<<"Please input the number n:";
 cin>>n;
 int *x=new int[n];
 for(int i=0;i<n;i++)
 {
      x[i]=0;
 }
 NQueens(n,x);

}
```