**Clrs 15.4-6 P212**

**/\*最长递增子序列（LIS）的O（nlogn）解法  
\* 本文档中，递增不包括相等  
\*  
\* 大致思路：  
\* a数组中存放待处理数组，从左往右扫描数组，  
\* c[k] 表示 当前发现的 长度为k的 最长递增子序列 中的最小结尾元素值,   
\* 由c的定义可知 c为递增数组  
\* b[i] 表示以a[i]结尾的最长递增子序列的长度  
\*  
\* 如 a数组为 2 5 3 4， 则  
\* 初始 c[1]=2, b[0]=1,   
\* 从 a[1] 开始扫描，  
\* 5>2 得到c[2]=5， 记录 b[1]=2;  
\* 3<5 对 c[1]--c[2] 进行搜索， 找到位置t，使 c[t-1]<a[i]<= c[t] (此例下，t为2), c[t]=a[2]=3, b[2]= t ;【关键,程序中还有解释】  
\* 4>3 得到 c[3]=4, 记录 b[3]=3;  
\*  
\* 由于c为有序递增数组，查找 t 时可用二分搜索（效率为O(logn)）  
\* 整体效率即为O（nlogn）  
\*/**

**int LIS(int \*a, int \*b, int \*c, int n)  
{  
//n为a数组的长度  
int k=1;  
c[k] = a[0];  
b[0] = 1;  
  
for( int i=1; i<n; ++i)  
{  
   if( a[i] > c[k] )  
   {  
    //若a[i]大于当前已发现 最长递增子序列 的最小结尾元素， 则最长递增子序列的长度可以扩展一位  
    //新扩展位的c值即为 a[i]  
    c[++k] = a[i];  
    b[i] = k;  
   }  
   else   
   {  
    int t= lower\_bound( c+1, c+k+1, a[i] ) - c;  
    //找到满足 c[t-1]<a[i]<= c[t] 的t位， 则由于 c[t-1]<a[i], a[i]可对长度为t-1的最长递增子序列进行扩展，  
    //又由于 a[i]<= c[t]， 扩展后即更新了t位的c值  
    c[t]=a[i];  
    b[i]=t;  
   }   
}**

**return 0;**

**}  
注解：  
1. lower\_bound（）函数为STL algorithm 头文件中包含的二分搜索算法， 对于一个有序序列，lower\_bound( beg, end, val )返回序列中第一个>=val 的位置（迭代器）  
跟其相关的还有upper\_bound()函数**

**2. 此算法在已在BUCTOJ,1079合唱队形题中测试，结果比 常规 O（n\*n）算法效率高，就我的代码来说，新方法为0ms，原方法为15ms。n越大，效果越明显。**