**一，**    **最长递增子序列问题的描述**

设L=<a1,a2,…,an>是n个不同的实数的序列，L的递增子序列是这样一个子序列Lin=<aK1,ak2,…,akm>，其中k1<k2<…<km且aK1<ak2<…<akm。求最大的m值。

**二，**    **第一种算法：转化为LCS问题求解**

设序列X=<b1,b2,…,bn>是对序列L=<a1,a2,…,an>按递增排好序的序列。那么显然X与L的最长公共子序列即为L的最长递增子序列。这样就把求最长递增子序列的问题转化为求最长公共子序列问题LCS了。

最长公共子序列问题用动态规划的算法可解。设Li=< a1,a2,…,ai>,Xj=< b1,b2,…,bj>，它们分别为L和X的子序列。令C[i,j]为Li与Xj的最长公共子序列的长度。则有如下的递推方程：

这可以用时间复杂度为O(n2)的算法求解，由于这个算法上课时讲过，所以具体代码在此略去。求最长递增子序列的算法时间复杂度由排序所用的O(nlogn)的时间加上求LCS的O(n2)的时间，算法的最坏时间复杂度为O(nlogn)＋O(n2)＝O(n2)。

**三，**    **第二种算法：动态规划法**

设f(i)表示L中以ai为末元素的最长递增子序列的长度。则有如下的递推方程：

这个递推方程的意思是，在求以ai为末元素的最长递增子序列时，找到所有序号在L前面且小于ai的元素aj，即j<i且aj<ai。如果这样的元素存在，那么对所有aj,都有一个以aj为末元素的最长递增子序列的长度f(j)，把其中最大的f(j)选出来，那么f(i)就等于最大的f(j)加上1，即以ai为末元素的最长递增子序列，等于以使f(j)最大的那个aj为末元素的递增子序列最末再加上ai；如果这样的元素不存在，那么ai自身构成一个长度为1的以ai为末元素的递增子序列。

这个算法由Java实现的代码如下：

public void lis(float[] L)

  {

         int n = L.length;

         int[] f = new int[n];//用于存放f(i)值；

         f[0]=1;//以第a1为末元素的最长递增子序列长度为1；

         for(int i = 1;i<n;i++)//循环n-1次

         {

                f[i]=1;//f[i]的最小值为1；

                for(int j=0;j<i;j++)//循环i 次

                {

                       if(L[j]<L[i]&&f[j]>f[i]-1)

                              f[i]=f[j]+1;//更新f[i]的值。

                }

         }

         System.out.println(f[n-1]);

              }

这个算法有两层循环，外层循环次数为n-1次，内层循环次数为i次，算法的时间复杂度

所以T(n)=O(n2)。这个算法的最坏时间复杂度与第一种算法的阶是相同的。但这个算法没有排序的时间，所以时间复杂度要优于第一种算法。

**四，**    **对第二种算法的改进**

在第二种算法中，在计算每一个f(i)时，都要找出最大的f(j)(j<i)来，由于f(j)没有顺序，只能顺序查找满足aj<ai最大的f(j)，如果能将让f(j)有序，就可以使用二分查找，这样算法的时间复杂度就可能降到O(nlogn)。于是想到用一个数组B来存储“子序列的”最大递增子序列的最末元素，即有

B[f(j)] = aj

在计算f(i)时，在数组B中用二分查找法找到满足j<i且B[f(j)]=aj<ai的最大的j,并将B[f[j]+1]置为ai。下面先写出代码，再证明算法的证明性。用Java实现的代码如下：

lis1(float[] L)

{

    int n = L.length;

    float[] B = new float[n+1];//数组B；

    B[0]=-10000;//把B[0]设为最小，假设任何输入都大于-10000；

    B[1]=L[0];//初始时，最大递增子序列长度为1的最末元素为a1

    int Len = 1;//Len为当前最大递增子序列长度，初始化为1；

    int p,r,m;//p,r,m分别为二分查找的上界，下界和中点；

    for(int i = 1;i<n;i++)

    {

        p=0;r=Len;

        while(p<=r)//二分查找最末元素小于ai+1的长度最大的最大递增子序列；

        {

           m = (p+r)/2;

           if(B[m]<L[i]) p = m+1;

           else r = m-1;

        }

        B[p] = L[i];//将长度为p的最大递增子序列的当前最末元素置为ai+1;

        if(p>Len) Len++;//更新当前最大递增子序列长度；

    }