3-1 设计一个 $O(n^2)$ 时间的算法，找出由 n 个数组成的序列的最长单调递增子序列。

解：

用数组 b[0:n-1] 记录以 a[i] ($0 \le i \le n$) 为结尾的最长递增子序列的长度，

序列 a 的最长递增子序列长度为 $\max\{b[i]\}$

满足递归式：

$$\begin{cases} b[0]=1 \\ b[i]=\max\limits_{\substack{0<k<i \\ a[k]<a[i]}}\{b[k]\}+1 \end{cases}$$

将 b[i] 转换成一个规模更小的问题

```
int lisdyna() {
  int i, j, k;
  for( i=1, b[0]=1; i<n; i++ ) {
     for (j=0, k=0; j<i; j++)
        if( a[j]<=a[i] && k<b[j] )
           k=b[j];
        b[i]=k+1
     }
```

然后再求 b[0], b[1] ··· b[n-1] 的最大值，即为最长的单调递增子序列的长度。

lisdyna() 二重循环，∴复杂度为 $O(n^2)$。

32. 最长单调递增序列的O(nlogn)算法. 一个长度为i的候选子序列的最后一个元素为一个长度为i-1的候选子序列的最后一个元素一样大，通过指向输入序列中元素的指针来维持候选子序列.

解:

已知计算序列 a[0:i-1] (i<n) 的最长递增子序列的长度为k, 以及序列 a[0:i-1] 中所有长度为k的递增子序列中的最小结尾元素值 b[k].

① a[i] ≥ b[k]. ⇒ k=k+1. b[k]=a[i].

② a[i] = b[k] ⇒ k=k. b[k]=b[k].

③ a[i] < b[k]. { a[i] < b[i]. ⇒ b[i]=a[i].
b[i] < a[i] < b[k] ⇒ b有序. 通过二分搜索法.

找到j. 使得 b[j-1] ≤ a[i] ≤ b[j]. 此时. b[j]=a[i].

b[1:j-1] 和 b[j+1, k] 的值不变.

算法:
```
int LIS( ){
    b[1]= a[0].
    for( int i=1, k=1; k<n; i++ ){
      if ( a[i] ≥ b[k] )
          b[++k]= a[i].          // 操作数+1. b[k+1]= a[i].
      else                       // a[i] < b[k]
          b[ binary(i,k)] = a[i]; // 使用二分搜索 用a[i] 替代 a[j]
    }
    return k.
```

```
int binary ( int i , int k ) {
    if ( a[i] < b[1] )                    // a[i] < b[1] 时.  b[1] = a[i] .
        return 1 ;
    for ( int h=1 , j=k ; h != j-1 ; ) {
      if ( b[ k = (h+j)/2 ] <= a[i] )
          h=k ;
      else
          j= k ;                          // 二分查找
      }
    return j ;
}
```

二分查找的 binary(i,k) 的计算时间为 O(logk). 在最坏情况下, 上述算法所需的计算时间为 O(nlogn)

3-4. 独立作务调度问题.

用两台处理机 A 和 B 处理 n 个作业. 设第 i 个作业交给机器 A 处理时需要时间 a_i, 若由机器 B 来处理, 则需要时间 b_i. 由于各作业的特点和机器的性能关系, 可能对某些 i 有 $a_i > b_i$, 而对于某些 j ($j \neq i$) 有 $a_j < b_j$. 既不能将一个作业分开由两台机器处理, 也没有一台机器能够同时处理两个作业. 设计一个动态算法, 使两台机器处理完这 n 个作业的时间最短.

解：dp[i][j] 为完成前 i 个任务，A机器运行 j 分钟，B机器运行的最少时间

递归式：dp[0][0]=0.　　第i个任务给A做　给B做

当 j>=a[i]时，dp[i][j]= min(dp[i-1][j-a[i]], dp[i-1][j]+b[i])

当 j<a[i]时，dp[i][j]= dp[i-1][j]+b[i]. (只能给B做).

最后，机器A花费时间 j，和，机器B花费时间 dp[i][j] 中的较大值，为AB机器完成最后任务的最终时间.

时间复杂度 for循环 O(n*sum) 线性，数组赋值 O(n²)

∴复杂度为 O(n²)

实现：

```
def xiaopu(a, b, n):
    sum=0.
    dp=np.zeros(202, 10000)
    for i in range(1, n+1):
        sum+=a[i].
    for j in range(sum)
        if(j>=a[i].):
            dp[i][j]=min(dp[i-1][j]+b[i], dp[i-1][j-a[i]])
        else:
            dp[i][j]=dp[i-1][j]+b[i]
    ans=9999
```

```
for i in range(sum):
    ans = min(ans, max(dp[n][i], i))
return ans.
```