下述问题中，请做如下部分回答：（1）用自然语言描述问题的优化解包含哪些子问题的优化解，注意说明用到的符号的含义；（2）写出优化解代价的递归方程，注意说明每个符号的含义；（3）写出算法伪代码；（4）分析算法的时间复杂性。

1、给定一个整数序列 $a_1,\ldots,a_n$。相邻两个整数可以合并，合并之后的结果是两个整数之和（用和替换原来的两个整数），合并两个整数的代价是这两个整数之和。通过不断合并最终可以将整个序列合并成一个整数，整个过程的总代价是每次合并操作代价之和。试设计一个动态规划算法给出 $a_1,\ldots,a_n$ 的一个合并方案使得该方案的总代价最大。

优化子结构：

所有合并操作中的最后一次合并将两个整数x和y合并为一个整数，其中x和y一定是合并输入中的连续整数序列所得，即存在整数k，使得x是 $a_1,\ldots,a_k$ 的合并结果，而y是 $a_{k+1},\ldots,a_n$ 的合并结果。由合并操作的说明可知，$x=\sum_{i=1}^{k} a_i$，$y=\sum_{k+1}^{n} a_i$，而且最后一次合并的代价为$x+y=\sum_{i=1}^{n} a_i$。因此，最优解的总代价是合并 $a_1,\ldots,a_k$ 得到x的最大代价、合并 $a_{k+1},\ldots,a_n$ 得到y的最大代价、以及$x+y=\sum_{i=1}^{n} a_i$的加和。其中包括合并 $a_1,\ldots,a_k$ 和 $a_{k+1},\ldots,a_n$ 对应的两个子问题的优化解代价。

优化解代价的递归方程：

令$M[i,j]$表示合并$a_i \ldots a_j$的优化解代价，$S[i,j]=\sum_{p=i}^{j} a_p$.

边界条件：$M[i,j]=0$，如果$i=j$;

当$i>j$时，$M[i,j]=\min\limits_{i\le k<j}\{M[i,k]+M[k+1,j]\}+S[i,j]$。

算法思路：

逐条对角线计算，首先用边界条件确定满足$i-j=0$的$M[i,j]$，随后增加$i-j$的差值，并计算满足该差值的$M[i,j]$。$M[1,n]$即位优化解的代价。

计算$M[i,j]$时，用$B[i,j]$记录最优的划分点k，并通过B构造优化的合并序列。

伪代码：略

代价：$O(n^3)$.

2、最长增长子序列问题定义如下：

输入：由 n 个数组成的一个序列 S：$a_1,a_2,\ldots,a_n$

输出：S 的子序列 $S'=b_1,b_2,\ldots,b_k$ ，满足：

(1) $b_1\le b_2\le\ldots\le b_k$ ，

(2) $|S'|$最大

使用动态规划技术设计算法求解最长增长子序列问题.

优化子结构：

令 $b_1,b_2,\ldots,b_k$ 是一个优化解，其中 b_k 对应 a_i, b_{k-1} 对应 a_j, 则 $i>j$, $a_i\ge a_j$ 而且 $b_1,b_2,\ldots,b_{k-1}$ 必须是 $a_1,a_2,\ldots,a_i$ 的最长增长子序列。

优化解代价的递归方程：

令 $L[i]$表示以 a_i 结尾的增长子序列的最大长度，最长增长子序列必定以某个元素结尾，因此 $\max\{L[i]\}$即最长增长子序列的长度。$L[i]$的求解如下：

边界条件：$L[0]=0$, $L[1]=1$;

当 i>1 时，$L[i]=\max\limits_{\substack{0\le j\le i-1\\ a[i]\ge a[j]}}\{L[j]+1\}$.

算法思路：

从左向右计算 $L[i]$，计算过程中 a_i 的前序元素保存为使 $L[i]$最大的 a_j;

扫描所有 $L[i]$并取其最大值，即为最长增长子序列的长度，使用该长度计算过程中保存的前序元素构造最长增长子序列。

伪代码：略。

代价：$O(n^2)$。

（选做）3、最长公共增长子序列问题定义如下：

输入：由 n 个数组成的一个序列 S：$a_1,a_2,\ldots,a_n$,

由 m 个数组成的一个序列 T：$b_1,b_2,\ldots,b_m$.

输出：S 和 T 的公共子序列 $X= c_1,c_2,\ldots,c_k$，满足：

(1) $c_1\le c_2\le\ldots\le c_k$,

(2) $|X|$最大

使用动态规划技术设计算法求解最长公共增长子序列问题.

优化子结构：

令 $c_1, c_2, \ldots, c_{k-1}, c_k$ 为是一个优化解，而且 c_k 在 S 和 T 中分别对应 a_i 和 b_j，c_{k-1} 在 S 和 T 中分别对应 a_p 和 b_q，则 $a_i = b_j$，$c_1, c_2, \ldots, c_{k-1}, c_k$ 是 $a_1,a_2,\ldots,a_i$ 和 $b_1,b_2,\ldots,b_j$ 的包含它们各自最后一个字符的最长公共增长子序列；$a_p = b_q$，$c_1, c_2, \ldots, c_{k-1}$ 是 $a_1,a_2,\ldots,a_p$ 和 $b_1,b_2,\ldots,b_q$ 的包含它们各自最后一个字符的最长公共增长子序列。

优化解代价的递归方程：

令$L[i,j]$表示 $a_1,a_2,\ldots,a_i$ 和 $b_1,b_2,\ldots,b_j$ 的包含它们各自最后一个字符的最长公共增长子序列的长度，当$a_i\neq b_j$时，$L[i,j]=0$。$a_1,a_2,\ldots,a_n$ 和 $b_1,b_2,\ldots,b_m$ 的最长公共增长子序列的长度为 $\max\limits_{1\le i\le n,1\le j\le m}\{L[i,j]\}$。$L[i,j]$的求解过程如下：

边界条件：

$$L[i,1]=\begin{cases}1 & \text{如果}a_i=b_1;\\ 0 & \text{如果}a_i\neq b_1;\end{cases},\ 1\le i\le n;$$

$$L[1,j]=\begin{cases}1 & \text{如果}a_1=b_j;\\ 0 & \text{如果}a_1\neq b_j;\end{cases},\ 2\le j\le m.$$

当$i>1$而且$j>1$时，如果$a_i\neq b_j$时，则$L[i,j]=0$；否则，$L[i,j]=\max\limits_{\substack{p<i,q<j,\\ a_p=b_q,a_p\le a_i}}\{L[p,q]\}+1$。

算法思路：

逐行计算$L[i,j]$，并保留计算$L[i,j]$时使用的子问题代价$L[p,q]$。最终，$\max\limits_{1\le i\le n,1\le j\le m}\{L[i,j]\}$为优

化解代价，从优化解代价对应的$L[i,j]$开始，反向便利计算过程中使用的子问题，构建最长公共增长子序列。

以下为基本的方法：

伪代码：

```
LongestIncreasingSubsequenceBasic(a1,a2,...,an, b1,b2,...,bm)
for i =1 to n
    if ai= b1;
        L[i,1] ←1;
    else
        L[i,1] ←0;
for j =2 to m
    if a1= bj;
        L[1,j] ←1;
    else
        L[1,j] ←0;
for i = 2 to n
    for j =2 to m
        if ai = bj
            L[i,j] ← 1;
            for p = 1 to i-1
                for q =1 to j-1
                    if ap ≤ ai and  L[p,q] + 1 > L[i,j]
                        L[i,j] ← L[p,q] + 1;
                        B[i,j] ← (p,q);
    else
        L[i,j] ← 0;
return L, B;
```

代价：$O(n^2m^2)$，该代价可以降低为 $O(n^2m)$或 $O(nm^2)$，并进一步降低为 $O(mn)$，详情见下一页。

以下为改进的方法：计算$L[i,j]$时不必扫描所有的$L[p,q]$，其中$1 \le p < i$，$1 \le q < j$。将L中每列中已经计算的最大值存储在$MAX[1]$~ $MAX[m]$中，计算该最大值所需的子问题存储在$SP[1]$~ $SP[m]$中。这样，计算$L[i,j]$时不必扫描左上角的整个方块中的所有$L[p,q]$，只需扫描$MAX[1]$~$MAX[j-1]$即可，如此可将算法代价降低为 $O(n^2m)$或 $O(nm^2)$。更进一步，可以避免每次计算都扫描$MAX[1]$~$MAX[j-1]$，在计算第 i 行时，用$MaxSPCost$和$MaxSPLoc$维护计算当前元素所需的子问题代价和子问题位置，计算$L[i,j]$时不必扫描$MAX[1]$~$MAX[j-1]$。通过上述方法，在子问题优化解代价已知的情况下，计算$L[i,j]$的代价从$O(ij)$降低为 $O(n)$或 $O(m)$，最后降低为$O(1)$，最终算法代价降低为 $O(mn)$。

伪代码：

```
LongestIncreasingSubsequenceImproved(a1,a2,...,an, b1,b2,...,bm)
    for i =2 to n
        if ai= b1;
            L[i,1] ←1;
        else
            L[i,1] ← 0;
    for j =1 to m
        MAX[j] ← 0;
        SP[j] ← NIL;
        if a1= bj;
            L[1,j] ←1;
        else
            L[1,j] ← 0;
    for i = 2 to n
        for j= 1 to n
            if L[i − 1,j] > MAX[j]
                MAX[j] ← L[i − 1,j];
                SP[j] ← B[i − 1,j];
        MaxSPCost ← 0;
        if ai > b1
            MaxSPCost ← Max[1];
            MaxSPLoc ← SP[j];
        for j =2 to m
            if ai = bj
                L[i,j] ← 1;
                if MaxSPCost > 0
                    L[i,j] ← MaxSPCost + 1;
                    B[i,j] ← MaxSPLoc;
            else
                L[i,j] ← 0;
            if ai > bj and MAX[j] > MaxSPCost
                MaxSPCost ← MAX[j];
                MaxSPLoc ← SP[j];
    return L, B;
```

代价：$O(mn)$。