```
01: LIS_DP( s[1..n] )
02: {
03:   for (k=1 to n) {
04:     L[k] = 0
05:     for (j=1 to k-1)
06:        if ( s[j] < s[k] AND L[k] < L[j] ) L[k] = L[j]
07:     L[k] = L[k] + 1
08:   }
09:   return max( L[1..n] );
10: }
```

บรรทัดที่ 3 ถึง 8 เขียนตรงๆ จากความสัมพันธ์เวียนเกิดที่หาได้ เมื่อหาเสร็จก็คืนค่ามากสุดใน L ซึ่งก็คือความยาวของลำดับยาวสุดที่หาได้

❹ แล้วลำดับย่อยเพิ่มขึ้นที่ยาวสุดคืออะไร ถ้าอยากรู้ก็ต้องจำค่า j ของบรรทัดที่ 6 ด้วย จึงจะสามารถสร้างผลเฉลยเหมาะที่สุดได้จากค่าต่างๆ ที่จำมา เขียนเป็นรหัสเทียมได้ดังนี้

```
01: LIS_DP( s[1..n] )
02: {
03:   for (k=1 to n) {
04:     L[k] = 0
05:     prev[k] = 0
06:     for (j=1 to k-1)
07:        if ( s[j] < s[k] AND L[k] < L[j] ) {
08:          L[k] = L[j]
09:          prev[k] = j
10:        }
11:     L[k] = L[k] + 1
12:   }
13:   p = 1
14:   for (k=2 to n) if (L[p] < L[k]) p = k
15:   i = 0
16:   while( p ≠ 0 ) {
17:     tmp[++i] = s[p]
18:     p = prev[p]
19:   }
20:   return  Reverse( tmp[1..i] )
21: }
```

บรรทัดที่ 3 ถึง 12 คล้ายกับที่ได้เขียนมา ต่างกันตรงบรรทัดที่ 9 มีไว้จำตำแหน่งที่ได้ค่ามากสุดเก็บในตาราง prev ตีความได้ว่า prev[k] คือตำแหน่งก่อนหน้าตัวที่ k ในลำดับย่อยเพิ่มขึ้นที่ยาวสุดที่ได้พบมา ถ้า prev[k] มีค่าเป็น 0 หมายความว่าไม่มีตัวก่อนหน้า บรรทัดที่ 16 ถึง 19 วิ่งถอยหลังในตาราง s เพื่อสร้างลำดับย่อยเพิ่มขึ้นที่ยาวสุด (แบบกลับลำดับ) โดยเริ่มที่ช่อง p (บรรทัดที่ 13 ถึง 15 หาค่าของ p ซึ่งคือเลขช่องของ L ที่มีค่ามากที่สุด) หยิบ s[p] ไปต่อท้ายแถวลำดับ tmp แล้วเปลี่ยนค่า p เป็น prev[p] กระทำเช่นนี้ไปจนกระทั่งพบตำแหน่ง p=0 เป็นอันสิ้นสุด นำข้อมูลใน tmp ไปกลับลำดับได้เป็นลำดับย่อยเพิ่มขึ้นที่ยาวสุดที่ต้องการ

บรรทัดที่ 7 เป็นคำสั่งมาตรเวลาของอัลกอริทึมข้างบนนี้ ดังนั้นใช้เวลาการทำงานเป็น