Input $\{a_1, a_2, \cdots a_n\}$

type Comparable

同样使用 Dynamical Programming

Def LIS[i] 为末尾为 a_i 的递增子列

∴ 若 $a_i < a_j$ $(i, j \in \{1, 2, \cdots n\})$

⇒ LIS[j] ≥ LIS[i]+1

```
for (i=1; i<=n; i++)
    LIS[i] = 1;          O(n)
Max_LIS = 1;
for (i=1; i<=n; i++)
{   for (j=1; j<i; j++1
    {   if (aj < ai)
            LIS[i] = max(LIS[j]+1, LIS[i])
    }
    Max_LIS = max(Max_LIS, LIS[i])
}                        O(n²)
for (i=n; i>=1; i++)     O(n²)
{   if (LIS[i] = Max_LIS)
    {
        Comparable array Out[n];
```

```
                Out[Max_LIS-1] = ai;
                Find_next ( Max_LIS-1, i, Out);
            }
    }
Find_next ( Max_LIS, i, Out [n])
 {      for(j = i-1;  j>=1, j--)
          { if ( Max_LIS == 1 && LIS[j] == 1)
             {
                 Out [0] = aj;
                 for ( k=0; k <n; k++1
                       cout<<  Out [k]; }
            if ( LIS[j] == Max_LIS && LIS[j]>
             {
                 Out[Max_LIS-1] = aj;
                 Find_ next (Max_LIS-1, j, Out) ;
             }
          }
}
```

總效率 $O(n^2)$

ex: (1, 5, 2, 6, 9, 10, 3, 15, 14)

LIS[1] = 1;

LIS[2] = 1 → 2;

LIS[3] = 1 → 2;

LIS[4] = 1 → 3 → 3;

LIS[5] = 1 → 3 → 3 → 4;

LIS[6] = 1 → 3 → 3 → 5;

LIS[7] = 3;

LIS[8] = 6;

LIS[9] = 6;

LIS	1	2	3	4	5	6	7	8	9
	1	2	2	3	4	5	3	6	6

Max_LIS = 6;

⇒ Out[5] = a_9 = 14;

or

⇒ Out[5] = a_8 = 15;

⇒ Out[4] = a_6 = 10;

⇒ Out[3] = a_5 = 9

⇒ Out[2] = a_4 = 6

⇒ Out[1] = a_2 = 5

or

Out[1] = a_3 = 2

⇒ Out[0] = a_1 = 1

Out Put:

1 → 2 → 6 → 9 → 10 → 14

1 → 2 → 6 → 9 → 10 → 15

1 → 5 → 6 → 9 → 10 → 15

1 → 5 → 6 → 9 → 10 → 14

为了实现 O(n log n)

第一个 $O(n^2)$ 中 for(j=1;j<i;j++) max

可用树状数组查找

由 $O(n)$ 优化为 $O(\log n)$

则第一个优化为 $O(n\log n)$

第二个 $O(n^2)$ 中

依旧采用树状数组查找优化

LIS 最大元素 $O(\log n)$

并遍历 Max_LIS $O(n)$

总耗时 $O(n\log n)$

从而实现优化