设序列长度为 n. 记这 n 个数为 $a_1, a_2, \cdots a_n$，求序列的最长(严格)递增子序列.

解法如下：设 f_i $(i=1,2,\cdots n)$. 表示以 a_i 结尾的所有递增子序列中，最长的一个的长度. 则 f_i 无后效性，且具有最优子结构，转移方程可写成：

$$f_i = \max_{\substack{j<i \\ a_j<a_i}} \{f_j\} + 1.$$

只要从小到大地更新 f_i 即可. 最后答案即为 $Ans = \max_{1\le i\le n} \{f_i\}$.

计算 f_i 的伪代码如下：

```
for i from 1 to n
    f_i ← 1
    for j from 1 to i-1.
        if a_j < a_i
            f_i ← max(f_i, f_j+1)
```

可以看出，以上算法能在 $\Theta(n^2)$ 的时间复杂度内求出答案.

举一个简单的例子：($n=7$).

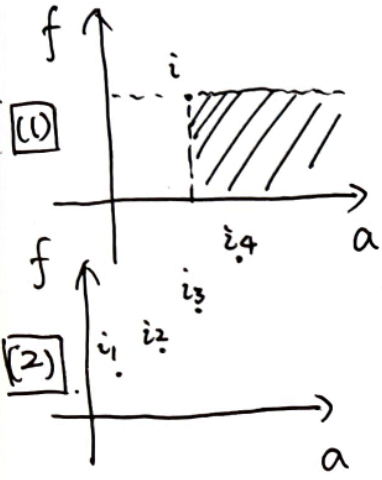
i	1	2	3	4	5	6	7
a_i	1	2	3	1	3	4	1
f_i	1	2	3	1	3	4	1.

箭头表示转移方向. 容易看出，时间复杂度为 $\Theta(n^2)$.

一种优化的方法是使用平衡树(二分查找树),作如下观察:

对于两个已更新完毕的位置 i,j,若 $a_i \geq a_j$,但 $f_i \leq f_j$.

那么 i 在任何情况下均会优于 j. 具体地,画一个 $a_i - f_i$ 轴



当向图中加入 i 点时,其右下角(阴影部分,包括边界)的点在任何情况下均不如 i 点优,从而可以直接排除(图(1))

因此,经转移(可能造成负环)的点必定在图上形如图(2)排布. 也即:如果从小到大排序 a_{i_k},则 f_{i_k} 也会从小到大排布. 于是,只要能在 $\Theta(\log n)$ 时间内从图中获取每一个 f_i 的值,并同时维护这个图,总时间复杂度就能控制在 $\Theta(n \log n)$. 现假设图上的点均为 a_i 为键值存在了 BST 中.

对于 $1 \leq i \leq n$,从小到大地像这样更新 f_i:

① 更新 f_i:找到图中 a_{i_k} 最大的,但 $a_{i_k} < a_i$ 的点,依假设,f_{i_k} 亦是 $a_{i_k} < a_i$ 的元素中 f_{i_k} 最大的,从而 $f_i = f_{i_k} + 1$.(若 i_k 不存在,$f_i = 1$)

由于 BST 的性质,查询是 $O(\log n)$ 的

② 将 (a_i, f_i) 加入图表中:按如下步骤进行:

Step 1. 找到 BST 中 a_{i_k} 最小的,但 $a_{i_k} \geq a_i$ 的点.
如果不存在这样的点,跳出循环.

Step 2. 比较 f_i 和 f_{i_k},若 $f_i \geq f_{i_k}$,则将 i_k 从 BST 中移除,并回到 Step 1. 否则,跳出循环.

Step 3. 将 (a_i, f_i) 插入 BST 中.

注意到每个位置至多一次被插入/寻到/删除,每次插入至多一次失败查找.

每个动作都是 $\Theta(\log n)$ 的,从而总复杂度也是 $\Theta(n \log n)$ 的.

$\Theta(n^2)$ 做法

```cpp
#include <cstdio>
using namespace std;
const int N = 1e3;
inline int max(int a, int b) {return a > b ? a : b;}
int n, Ans, a[N + 5], f[N + 5];

int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);

    a[0] = -0x3f3f3f3f;
    for(int i = 1; i <= n; ++i)
        for(int j = 0; j < i; ++j)
            if(a[j] < a[i])
                f[i] = max(f[i], f[j] + 1);

    for(int i = 1; i <= n; ++i)
        Ans = max(Ans, f[i]);

    printf("%d\n", Ans);
    return 0;
}
```

$\Theta(n \log n)$ 做法

以下是使用 `std::set` 来模拟 BST 的一个示范代码，它的正确性经过了 100 组 $n = 10^3$ 规模下的随机数据和 $\Theta(n^2)$ 做法对拍：

```cpp
#include <cstdio>
#include <set>
using namespace std;
const int N = 1e5;
int n, Ans, a[N + 5], f[N + 5];
struct Node {
    int val, len;   // val 是 a，len 是 f
    Node(int _v, int _l = 0) {val = _v, len = _l;}
    bool operator<(const Node &y) const {
        return val < y.val;
    }
};

set<Node> s;

void insert(Node &&temp) {
    for(auto it = s.lower_bound(std::move(temp)); it != s.end(); )
        if(it->len <= temp.len)
            it = s.erase(it);
        else break;
    s.insert(std::move(temp));
}

int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);

    for(int i = 1; i <= n; ++i) {
        auto it = s.lower_bound(Node(a[i]));
        if(it != s.begin())
            f[i] = (--it)->len + 1;
        else
            f[i] = 1;
        insert(Node(a[i], f[i]));
    }

    for(int i = 1; i <= n; ++i)
        Ans = max(Ans, f[i]);

    printf("%d\n", Ans);
    return 0;
}
```