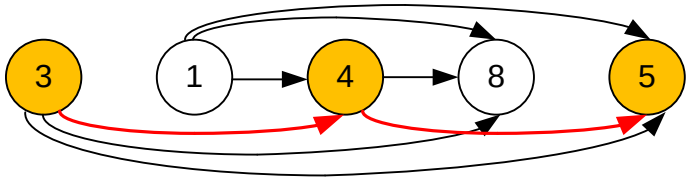
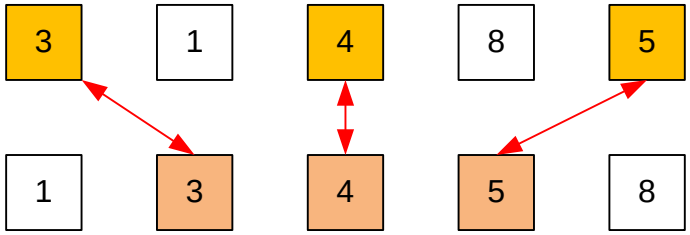
Longest Increasing Subsequence (LIS)

Let there be a sequence x_0, x_1, …, x_{n-1}. It is necessary to find in it a strictly increasing subsequence of the longest length. Formally, one should find a sequence of indices $i_1 < i_2 <\ldots < i_k$ such that $x_{i1} < x_{i2} <_{\ldots} < x_{ik}$, and k is maximum.

Longest path in a directed acyclic graph. The LIS problem can be reduced to finding the longest path in a directed acyclic graph. To each number of the sequence we associate a vertex of the graph, and join the numbers x and y with an edge if x is less than y and in the sequence x occurs before y.

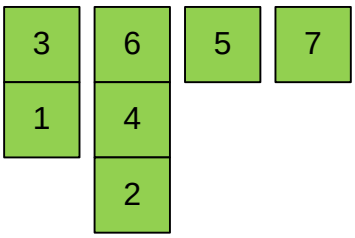


Reduction to the Largest Common Subsequence (LCS). Let's make a copy of the sequence and sort it in ascending order. The LIS of the original sequence will also be the increasing subsequence of sorted. Therefore, it suffices to find the largest common subsequence of the original and sorted sequence, which will be the largest non-decreasing subsequence.



Solitaire Sorting. Let there be a deck of cards with numbers 1, 2,…, n. The deck was shuffled. Then the cards are revealed one at a time and laid out on the table as follows. A card with a lower number can be put on a card with a higher number, or in a new pile to the right of the current ones. We always see the top cards of all piles. If the current card is larger than all the open cards at the top of the piles, then it must be put in a separate pile to the right of the existing ones. The goal of the game is to get *as few stacks as possible* on the table.

Below is the input sequence and the game of solitaire. The length of the LIS is 4 (for example, 3, 4, 5, 7), so in any case you will have to use at least 4 stacks.



Exponential solution for O(2^n).

Iterate over all possible subsequences (subsets) from $x_0, x_1, \ldots, x_{n-1}$. Find out which of them are increasing. Among all increasing subsequences, choose the largest one.

Dynamic programming O($n\log_2 n$). Consider a solution to a problem with time complexity O($n * \log_2 n$). After processing i elements of array x, the tail of the *lis* array (cells from lis[k] to lis[$len - 1$] for some k) contains the tail of the largest increasing subsequence that can be distinguished among the numbers x[0], x[1], … x[i]. Moreover, the tail of this LIS will be lexicographically smallest.

When processing the next element x[i], we will try to insert it into the *lis* array using a binary search. If x[i] is greater than the value of lis [$len - 1$], then the element x[i] increases the length of the LIS by one, so set lis[len] = x[i] and increase len by 1. If lis[k] < x[i] ≤ lis[$k + 1$], then replace lis[$k + 1$] with x[i]. This operation does not increase the length of the LIS, but can lexicographically reduce the tail of the LIS. After processing all elements of the array x, the value len contains the length of the LIS.

E-OLYMP **988. Subsequence** Given a sequence, find the length of the largest strictly increasing subsequence.

► In this problem you must find the length of the longest increasing subsequence (LIS). For the sample given, LIS can be {3, 5, 6} or {3, 5, 28}. The length of the LIS is 3.

3	29	5	5	28	6

3	29	5	5	28	6

Store the input sequence in the array x. Declare an auxiliary array lis.

```
#define MAX 1001
int x[MAX], lis[MAX];
```

The main part of the program. Read the input sequence.

```
scanf("%d",&n);
for(i = 0; i < n; i++) scanf("%d",&x[i]);
```

Carry out n iterations of the algorithm.

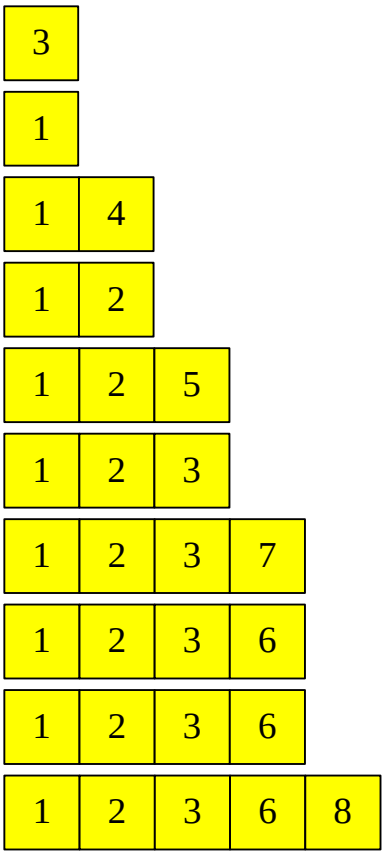
```
for (len = i = 0; i < n; i++)
{
  pos = lower_bound(lis,lis+len,x[i]) - lis;
  if (pos < len) lis[pos] = x[i]; else lis[len++] = x[i];
}
```

Print the length of LIS.

```
printf("%d\n",len);
```

x =

3	1	4	2	5	3	7	6	3	8



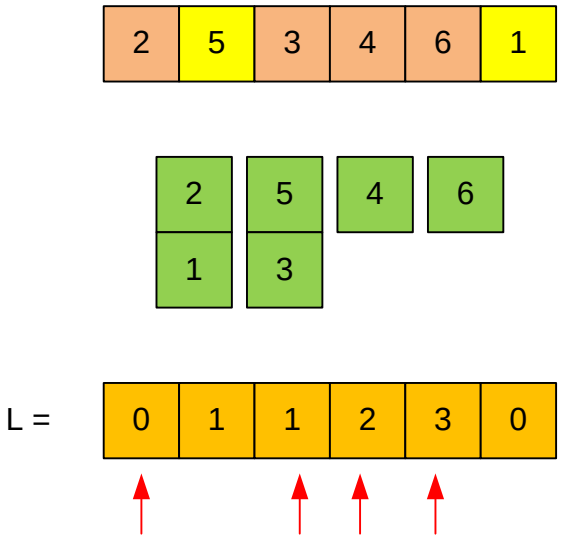
E-OLYMP 832. Increasing subsequence Given n ($1 \le n \le 10^5$) integers x_1, x_2, ..., x_n ($1 \le x_i \le 60\ 000$). Delete from them the least amount of numbers so that the rest were in ascending order.

► The largest increasing subsequence must be found and printed.

For each element x_i ($1 \le i \le n$), find the length of the LIS that ends at x_i. Store these values in array L. This information is enough to find the LIS itself in linear time. The length of the LIS equals to the maximum number k in array L. Let this largest number k be in the index *pos*. Then the LIS ends with the number x_{pos}. Decrease *pos* until L[*pos*] becomes equal to $k - 1$. The current x_{pos} will be the penultimate element in the LIS. And so on, we continue to move the *pos* index to the start of array L, stopping at the first positions for which L[*pos*] = $k - 2$, …, L[*pos*] = 0.

Theorem. Consider the indices $i_1 < i_2 < \ldots < i_m$, for which L[i_1] = L[i_2] = …= L[i_m]. Then the sequence x_{i1}, x_{i2}, …, x_{im} is decreasing.

The solitaire layout looks like:



L[i] contains the number of the pile where the card x_i will be placed (the piles are numbered starting from zero).