**最长递增子序列（LIS）**

最长递增子序列又叫做最长上升子序列；子序列，正如[LCS](http://www.ahathinking.com/archives/115.html)一样，元素不一定要求连续。本节讨论实现三种常见方法，主要是练手。

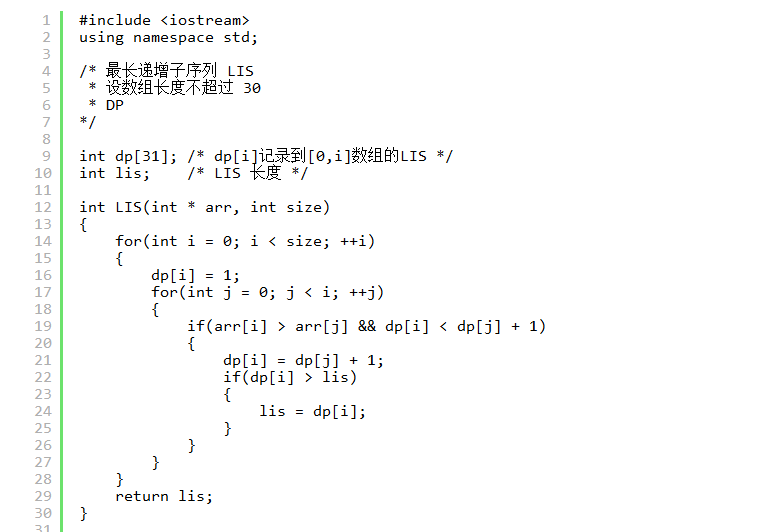
题：求一个一维数组arr[i]中的最长递增子序列的长度，如在序列1，-1，2，-3，4，-5，6，-7中，最长递增子序列长度为4，可以是1，2，4，6，也可以是-1，2，4，6。

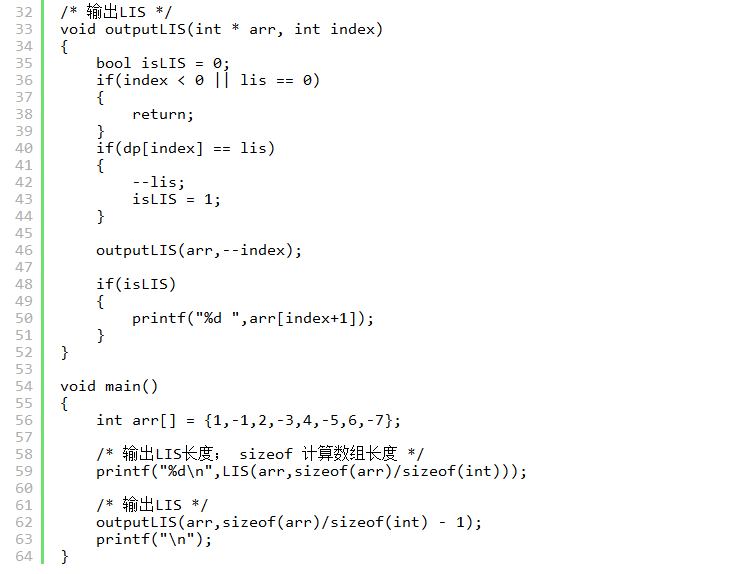
方法一：DP

像LCS一样，从后向前分析，很容易想到，第i个元素之前的最长递增子序列的长度要么是1（单独成一个序列），要么就是第i-1个元素之前的最长递增子序列加1，可以有状态方程：

LIS[i] = max{1,LIS[k]+1}，其中，对于任意的k<=i-1，arr[i] > arr[k]，这样arr[i]才能在arr[k]的基础上构成一个新的递增子序列。

代码如下：在计算好LIS长度之后，output函数递归输出其中的一个最长递增子序列。





这个方法也最容易想到也是最传统的解决方案，对于该方法和LIS，有以下两点说明：

1. 由LIS可以衍生出来最长非递减子序列，最长递减子序列，道理是一样的。
2. 对于输出序列，也是可以再申请一数组pre[i]记录子序列中array[i]的前驱，道理跟本节的实现也是一样的

**方法二：排序+LCS**

这个方法是在Felix’blog（见参考资料）中看到的，因为简单，他在博文中只是提了一句，不过为了练手，虽然懒，还是硬着头皮写一遍吧，正好再写一遍快排，用quicksort + LCS，这个思路还是很巧妙的，因为LIS是单调递增的性质，所以任意一个LIS一定跟排序后的序列有LCS，并且就是LIS本身。

**方法三：DP+二分查找**

**引用**[Felix](http://www.felix021.com/blog/read.php?1587) blog

最长递增子序列，Longest Increasing Subsequence 下面我们简记为 LIS。  
排序+LCS算法 以及 DP算法就忽略了，这两个太容易理解了。  
  
假设存在一个序列d[1..9] = 2 1 5 3 6 4 8 9 7，可以看出来它的LIS长度为5。  
下面一步一步试着找出它。  
我们定义一个序列B，然后令 i = 1 to 9 逐个考察这个序列。  
此外，我们用一个变量Len来记录现在最长算到多少了  
  
首先，把d[1]有序地放到B里，令B[1] = 2，就是说当只有1一个数字2的时候，长度为1的LIS的最小末尾是2。这时Len=1  
  
然后，把d[2]有序地放到B里，令B[1] = 1，就是说长度为1的LIS的最小末尾是1，d[1]=2已经没用了，很容易理解吧。这时Len=1  
  
接着，d[3] = 5，d[3]>B[1]，所以令B[1+1]=B[2]=d[3]=5，就是说长度为2的LIS的最小末尾是5，很容易理解吧。这时候B[1..2] = 1, 5，Len＝2  
  
再来，d[4] = 3，它正好加在1,5之间，放在1的位置显然不合适，因为1小于3，长度为1的LIS最小末尾应该是1，这样很容易推知，长度为2的LIS最小末尾是3，于是可以把5淘汰掉，这时候B[1..2] = 1, 3，Len = 2  
  
继续，d[5] = 6，它在3后面，因为B[2] = 3, 而6在3后面，于是很容易可以推知B[3] = 6, 这时B[1..3] = 1, 3, 6，还是很容易理解吧？ Len = 3 了噢。  
  
第6个, d[6] = 4，你看它在3和6之间，于是我们就可以把6替换掉，得到B[3] = 4。B[1..3] = 1, 3, 4， Len继续等于3  
  
第7个, d[7] = 8，它很大，比4大，嗯。于是B[4] = 8。Len变成4了  
  
第8个, d[8] = 9，得到B[5] = 9，嗯。Len继续增大，到5了。  
  
最后一个, d[9] = 7，它在B[3] = 4和B[4] = 8之间，所以我们知道，最新的B[4] =7，B[1..5] = 1, 3, 4, 7, 9，Len = 5。  
  
于是我们知道了LIS的长度为5。  
  
!!!!! 注意。这个1,3,4,7,9不是LIS，它只是存储的对应长度LIS的最小末尾。有了这个末尾，我们就可以一个一个地插入数据。虽然最后一个d[9] = 7更新进去对于这组数据没有什么意义，但是如果后面再出现两个数字 8 和 9，那么就可以把8更新到d[5], 9更新到d[6]，得出LIS的长度为6。  
  
然后应该发现一件事情了：在B中插入数据是有序的，而且是进行替换而不需要挪动——也就是说，我们可以使用二分查找，将每一个数字的插入时间优化到O(logN)~~~~~于是算法的时间复杂度就降低到了O(NlogN)～！  
  
代码如下:  
**int** LIS(**int** d[], **int** n){  
    **int** \*B = **new** **int**[n];  
    **int** left, right, mid, len = 1;  
    B[0] = d[1]; //为了和上面的一致，我们从1开始计数吧:)  
    **for**(i = 2; i <= n; ++i){  
        left = 0, right = len;  
        **while**(left <= right){  
            mid = (left + right) / 2;  
            **if**(B[mid] < d[i]) left = mid + 1; //二分查找d[i]的插入位置  
            **else** right = mid - 1;  
        }  
        B[left] = d[i]; //插入  
        **if**(left > len) len++; //d[i]比现有的所有数字都大，所以left 才会大于 len。  
    }  
    **delete**[] B;  
    **return** len;  
}