```
// Longest increasing subsequence
// Method: Dynamic Programming
// Space complexity: O(n)
// Time complexity: O(n^2)
// Note:
// 1. Not the best method, there's an O(n lg n) algorithm
// 2. LIS = LCS(original, sorted) - Space: O(n^n), Time: O(n^n)
extension Array where Element: Comparable {
    func longestIncreasingSubsequence() -> [Element] {
        // size of LIS that ends with element at i
        // since LIS at every i is sorted and must include i, checking last element is enough to
         decide membership later on
        var sizeOfLisEndingAt = [Int](repeating: 0, count: self.count)

        // keeps track of previous node for i-th node, -1 if no previous node i.e. if i-th node
         is a single element subsequence
        // essentially the back pointer in a linked list node
        var path = [Int](repeating: -1, count: self.count)

        // overall size of LIS -- used to reserve capacity at end
        var sizeOfLis = 0

        // last node in LIS
        var lisLastIndex = -1
        for i in 0..<self.count {
            var sizeOfPreviousLIS = 0

            // find longest of all previous eligible LIS
            // takes care of single element subsequence as well
            for j in 0..<i {
                // which previous LIS, if any, should i append myself to
                // since LIS is sorted, checking last element is enough
                if self[i] > self[j]  {
                    // is current LIS bigger than what i've seen so far
                    if sizeOfLisEndingAt[j] > sizeOfPreviousLIS {
                        sizeOfPreviousLIS = sizeOfLisEndingAt[j]
                        path[i] = j
                    }
                }
            }
            sizeOfLisEndingAt[i] = sizeOfPreviousLIS + 1

            // update size and last element of the overall LIS
            if sizeOfLisEndingAt[i] > sizeOfLis {
                sizeOfLis = sizeOfLisEndingAt[i]
                lisLastIndex = i
            }
        }

        // start from last element and backtrack to form the LIS
        var lis = [Element]()
        lis.reserveCapacity(sizeOfLis)

        var k = lisLastIndex
        while k >= 0 {
            // elements found in reverse order, hence push to front
            lis.insert(self[k], at: 0)
            k = path[k]
        }

        return lis
    }
}

let a = [10, 22, 9, 33, 21, 50, 41, 60, 80, 1]
print(a)
print(a.longestIncreasingSubsequence())
```