Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example,
Given `[10, 9, 2, 5, 3, 7, 101, 18]`,
The longest increasing subsequence is `[2, 3, 7, 101]`, therefore the length is `4`.
Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n^2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

Credits:
Special thanks to @pbrother for adding this problem and creating all test cases.

Solution 1

```
public class Solution {
    public int lengthOfLIS(int[] nums) {
        int[] dp = new int[nums.length];
        int len = 0;

        for(int x : nums) {
            int i = Arrays.binarySearch(dp, 0, len, x);
            if(i < 0) i = -(i + 1);
            dp[i] = x;
            if(i == len) len++;
        }

        return len;
    }
}
```

written by jopiko123 original link here

Solution 2

Inspired by http://www.geeksforgeeks.org/longest-monotonically-increasing-subsequence-size-n-log-n/

```
int lengthOfLIS(vector<int>& nums) {
    vector<int> res;
    for(int i=0; i<nums.size(); i++) {
        auto it = std::lower_bound(res.begin(), res.end(), nums[i]);
        if(it==res.end()) res.push_back(nums[i]);
        else *it = nums[i];
    }
    return res.size();
}
```

written by dtccwl original link here

Solution 3

This is a classic problem and here is a DP solution for reference Please note a NLogN solution can be found in the following link Geek for Geek

```
class Solution {
public:
    // There's a typical DP solution with O(N^2) Time and O(N) space
    // DP[i] means the result ends at i
    // So for dp[i], dp[i] is max(dp[j]+1), for all j < i and nums[j] < nums[i]
    int lengthOfLIS(vector<int>& nums) {
        const int size = nums.size();
        if (size == 0) { return 0; }
        vector<int> dp(size, 1);
        int res = 1;
        for (int i = 1; i < size; ++i) {
            for (int j = 0; j < i; ++j) {
                if (nums[j] < nums[i]) {
                    dp[i] = max(dp[i], dp[j]+1);
                }
            }
            res = max (res, dp[i]);
        }
        return res;
    }
};
```

written by moumoutsay original link here