300. Longest Increasing Subsequence

Given an integer array `nums`, return *the length of the longest* ***strictly increasing subsequence***.

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

Constraints:

- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

Follow up: Can you come up with an algorithm that runs in `O(n log(n))` time complexity?

Longest Increasing Subsequence

```
/*
   Linear search
   Time complexity: O(n^2)
   Space complexity: O(n)
*/
class Solution {
   public:
      /*
         if x in A, return index of x
         otherwise, return index of element > x
         if all elemnts are less than x, return -1
      */
      int index_of(std::vector<int>& A,int x){
         int n=A.size();
         for(int i=0;i<n;++i){
            if(A[i]==x || A[i]>x) return i;
         }
         return -1;
      }

      int lengthOfLIS(vector<int>& nums){
         int n = nums.size();
         std::vector<int> tmp;
         tmp.push_back(nums[0]);
         for(int i=1;i<n;++i){
            if(nums[i]>tmp.back()) tmp.push_back(nums[i]);
            else{
               int j=index_of(tmp,nums[i]);
               tmp[j]=nums[i];
            }
         }
         return tmp.size();
      }
};
```

Longest Increasing Subsequence

```
/*
   Binary search
   Time complexity: O(n log n)
   Space complexity: O(n)
*/
class Solution {
   public:
      int lengthOfLIS(vector<int>& nums) {
         int n = nums.size();
         std::vector<int> tmp;
         tmp.push_back(nums[0]);
         for(int i=1;i<n;++i){
            if(nums[i]>tmp.back()) tmp.push_back(nums[i]);
            else{
               int j=std::lower_bound(tmp.begin(),tmp.end(),nums[i])-tmp.begin();
               tmp[j]=nums[i];
            }
         }
         return tmp.size();
      }
};
```