 Longest 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.

My Solution:-

class Solution {

public int lengthOfLIS(int[] nums) {

int res[]=new int[nums.length];

Arrays.fill(res,1);

for(int j=1;j<nums.length;j++){

for(int i=0;i<j;i++){

if(nums[i]<nums[j]){

if(res[i]+1>res[j])

res[j]=res[i]+1;

}

}

}

int max=1;

for(int i=0;i<nums.length;i++){

if(res[i]>max)

max=res[i];

}

return max;

}

}

Better Solution:-

Java/Python Binary search O(nlogn) time with explanation

[dietpepsi](https://leetcode.com/dietpepsi/)

17167

Nov 03, 2015

tails is an array storing the smallest tail of all increasing subsequences with length i+1 in tails[i].  
For example, say we have nums = [4,5,6,3], then all the available increasing subsequences are:

len = 1 : [4], [5], [6], [3] => tails[0] = 3

len = 2 : [4, 5], [5, 6] => tails[1] = 5

len = 3 : [4, 5, 6] => tails[2] = 6

We can easily prove that tails is a increasing array. Therefore it is possible to do a binary search in tails array to find the one needs update.

Each time we only do one of the two:

(1) if x is larger than all tails, append it, increase the size by 1

(2) if tails[i-1] < x <= tails[i], update tails[i]

Doing so will maintain the tails invariant. The the final answer is just the size.

**Java**

public int lengthOfLIS(int[] nums) {

int[] tails = new int[nums.length];

int size = 0;

for (int x : nums) {

int i = 0, j = size;

while (i != j) {

int m = (i + j) / 2;

if (tails[m] < x)

i = m + 1;

else

j = m;

}

tails[i] = x;

if (i == size) ++size;

}

return size;

}

// Runtime: 2 ms