

Longest Increasing Subsequence

LIS — the foundation of sequence DP ■

● MEDIUM

■ Hook

Find the length of the longest strictly increasing subsequence in an array. Used in stack sorting, patience sorting, and is a building block for dozens of harder problems. Two solutions: $O(n^2)$ DP and $O(n \log n)$ Binary Search.

■ Core Intuition

$dp[i]$ = length of LIS ending at index i . For every $j < i$ where $nums[j] < nums[i]$: $dp[i] = \max(dp[i], dp[j] + 1)$. Answer = max of all $dp[i]$.

■ DP Table — [10,9,2,5,3,7,101,18]

```
nums: [10, 9, 2, 5, 3, 7, 101, 18]
idx:   0  1  2  3  4  5  6  7
dp[0]=1 (just [10])
dp[1]=1 (just [9], 9<10 can't extend)
dp[2]=1 (just [2])
dp[3]=2 (2<5: dp[2]+1=2) → [2,5]
dp[4]=2 (2<3: dp[2]+1=2) → [2,3]
dp[5]=3 (2<7, 3<7, 5<7: max dp[j]+1=3) → [2,3,7] or [2,5,7]
dp[6]=4 (7<101: dp[5]+1=4) → [2,3,7,101]
dp[7]=4 (7<18: dp[5]+1=4) → [2,3,7,18]
★ ANSWER = max(dp) = 4
```

■ Code — $O(n^2)$ and $O(n \log n)$

```
// O(n²) DP
public int lengthOfLIS(int[] nums) {
    int n=nums.length, maxLen=1;
    int[] dp = new int[n];
    Arrays.fill(dp,1);
    for (int i=1; i<n; i++) {
        for (int j=0; j<i; j++)
            if (nums[j] < nums[i])
                dp[i] = Math.max(dp[i], dp[j]+1);
        maxLen = Math.max(maxLen, dp[i]);
    }
    return maxLen;
}

// O(n log n) – Patience Sorting / Binary Search
public int lengthOfLISFast(int[] nums) {
    List<Integer> tails = new ArrayList<>();
    for (int num : nums) {
        int lo=0, hi=tails.size();
        while (lo < hi) { // binary search for insertion point
            int mid = (lo+hi)/2;
            if (tails.get(mid) < num) lo=mid+1; else hi=mid;
        }
        if (lo == tails.size()) tails.add(num);
        else tails.set(lo, num);
    }
    return tails.size();
}
```

■ TIME

$O(n^2)$ / $O(n \log n)$

■ SPACE

$O(n)$

Practice

#	Problem	Diff	Pattern
1	Longest Increasing Subsequence (LC 300)	Medium	DP / Binary Search
2	Russian Doll Envelopes (LC 354)	Hard	LIS in 2D
3	Longest Bitonic Subsequence	Medium	LIS + LDS
4	Number of Longest Increasing Subsequences (LC 673)	Medium	DP + count
5	Minimum Number of Removals to Make Mountain Array	Hard	LIS variant

- ✓ $dp[i]$ = LIS length ending at index i
- ✓ For each j less than i where $nums[j]$ less than $nums[i]$: $dp[i] = \max(dp[i], dp[j] + 1)$
- ✓ Answer = max of all dp values
- ✓ $O(n^2)$ DP or $O(n \log n)$ with patience sorting + binary search
- ✓ $tails[]$ in $O(n \log n)$ does NOT store the actual LIS — only its length

■ MENTAL MODEL — Remember This Forever

Think of building a tower: each new number either extends an existing tower (if it's bigger than the top) or replaces the top of the shortest tower it can beat (patience sort). The number of towers at the end = LIS length. The towers metaphor makes the $O(n \log n)$ solution intuitive.