# Minimum Deletions to Make a Sequence Sorted

We'll cover the following



## Problem Statement #

Given a number sequence, find the minimum number of elements that should be deleted to make the remaining sequence sorted.

**Example 1:**

```
Input: {4,2,3,6,10,1,12}
Output: 2
Explanation: We need to delete {4,1} to make the remaing sequence sorted {2,3,6,10
,12}.
```

**Example 2:**

```
Input: {-4,10,3,7,15}
Output: 1
Explanation: We need to delete {10} to make the remaing sequence sorted {-4,3,7,15
}.
```

**Example 3:**

```
Input: {3,2,1,0}
Output: 3
Explanation: Since the elements are in reverse order, we have to delete all excep
t one to get a
sorted sequence. Sorted sequences are {3}, {2}, {1}, and {0}
```

## Basic Solution #

A basic brute-force solution could be to try deleting all combinations of elements, one by one, and checking if that makes the subsequence sorted.

Alternately, we can convert this problem into a Longest Increasing Subsequence (https://www.educative.io/collection/page/5668639101419520/5633779737559040/5733679603122176/) (LIS) problem. As we know that LIS will give us the length of the longest increasing subsequence (in the sorted order!), which means that the elements which are not part of the

LIS should be removed to make the sequence sorted. This is exactly what we need. So we'll get our solution by subtracting the length of LIS from the length of the input array: `Length-of-input-array - LIS()`

Let's jump directly to the bottom-up dynamic programming solution.

## Bottom-up Dynamic Programming #

Here is the code for our bottom-up dynamic programming approach:

Java | JS | Python3 | C++

```python
def find_minimum_deletions(nums):
  # subtracting the length of LIS from the length of the input array to get minimum number
  return len(nums) - find_LIS_length(nums)


def find_LIS_length(nums):
  n = len(nums)
  dp = [0 for _ in range(n)]
  dp[0] = 1

  maxLength = 1
  for i in range(1, n):
    dp[i] = 1
    for j in range(i):
      if nums[i] > nums[j] and dp[i] <= dp[j]:
        dp[i] = dp[j] + 1
        maxLength = max(maxLength, dp[i])

  return maxLength


def main():
  print(find_minimum_deletions([4, 2, 3, 6, 10, 1, 12]))
  print(find_minimum_deletions([-4, 10, 3, 7, 15]))
  print(find_minimum_deletions([3, 2, 1, 0]))


main()

```

The time complexity of the above algorithm is $O(n^2)$ and the space complexity is $O(n)$.


Back

Shortest Common Super-sequence

Next

Longest Repeating Subsequence

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/minimum-deletions-to-make-a-sequence-sorted__pattern-5-longest-common-substring__grokking-dynamic-programming-patterns-for-coding-interviews)

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.