300. Longest Increasing Subsequence

Given an array A of size N

$i \in [0; N-1]$

LIS(i) = length of LIS ending strictly at position i

$$LIS(i) = \begin{cases} 1, & \text{if } \nexists\, j \in [0; i-1] : A[j] < A[i] \\ 1 + \max(LIS(j) \text{ for } j \in [0; i-1] \text{ if } A[j] < A[i]), & \text{else} \end{cases}$$

$LIS(A) = \max(LIS(i) \text{ for } i \in [0; N-1])$

Naive aproach: Recursion (slow)

A = []

```
lenghtOfLIS(nums):
   n = len(nums)
   A = nums
   maxLen = 0
   for i in range(n):
      maxLen = max(maxLen, aux(i))
   return maxLen

aux(i):
 maxLen = 0
 for j in range(i):
  if A[j] < A[i]
     maxLen = max(maxLen, aux(j))
 return 1 + maxLen
```

Exponential time complexity => TLE
Space: O(n) (stack, recursiveness)

Recursion + Memorization (Faster) (Top-down)

```
mem = list()
A = []

lenLIS(nums):
  n = len(nums)
  A = nums
  mem = list(n,-1)
  result = 0

  for i in range(n):
     result = max(result, aux(i))

  return result

aux(i):
  if mem[i] ≠ 1:
     return mem[i]

  maxLen = 0
  for j in range(i):
     if A[j] < A[i]:
        maxLen = max(maxLen, aux(j))

  return mem[i] = 1 + maxLen
```

Time: $O(n^2)$

Space: $O(n)$

Dynamic Programming (Bottom-up)

```
lenLIS(nums):
    n = len(nums)
    dp = list(n, 1)

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

    return max(dp)
```

Time: $O(n^2)$
Space: $O(n)$