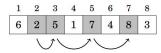
# Longest Increasing Subsequence

#### The LIS problem

- Given an array that contains n numbers  $x_1, x_2 ... x_n$  we need to find the longest subsequence such that the elements of the subsequence are in the sorted order.
- For example,

the longest increasing subsequence contains 4 elements:



#### The substructure

- Let f(k) be the length of LIS that ends at index k.
- To calculate the value of f(k), there are two possibilities
  - **1** The subsequence contains only one element  $x_k$ , Then f(k) = 1
  - ② The subsequence is constructed by adding the element  $x_k$  to a subsequence that ends at position i < k and  $x_i < x_K$ . In this case f(k) = f(i) + 1.
- In other words, Case 2 shows that, From the LIS of the array that ends at an index *i* < *k*, we can construct the LIS of the array that ends at *k* - *Optimal substructure Property*.

### Compute values of f(k)

- To compute f(k) we use two indices i and j, such that for each  $x_i$  we take all elements  $x_j$  from the beginning of the array to  $x_i$  and checks whether any one of them forms a longer subsequence with  $x_i$ .
- We initialize all f(k) with 1. (case 1). The two indices i and j are also initially at index 1. See the following table.

i, j

index	1	2	3	4	5	6	7	8
element	6	2	5	1	7	4	8	3
f(k)	1	1	1	1	1	1	1	1

- We increase the value of i by 1 (becomes 2) and then we compute f(i) using the values of j = 1 and j = 2.
- We update f(i) using the following condition.

if 
$$x_i > x_j$$
 and  $f(i) < f(j) + 1$   
 $f(i) = f(j) + 1$ 

The values of different values of i and j are shown. Initially i = 2 and j = 1

	,							
index	1	2	3	4	5	6	7	8
x	6	2	5	1	7	4	8	3
f(k)	1	1	1	1	1	1	1	1

• Increase *i* to 3 and reset j = 1, No change in f(i)

	j		i					
index	1	2	3	4	5	6	7	8
x	6	2	5	1	7	4	8	3
f(k)	1	1	1	1	1	1	1	1

• Increase j to 2. Since the elements  $x_2 = 2$  and  $x_3 = 5$  forms an increasing sequence, f(i) is increased by 1

		j	i					
index	1	2	3	4	5	6	7	8
х	6	2	5	1	7	4	8	3
f(k)	1	1	2	1	1	1	1	1

• Increase i to 4 and reset j = 1, It can be noted that none of the j values satisfy the conditions to increase f(k). So No change.

	j			i				
index	1	2	3	4	5	6	7	8
х	6	2	5	1	7	4	8	3
f(k)	1	1	2	1	1	1	1	1

• Increase i = 5 and reset j = 1, The pair  $x_1 = 6$  and  $x_5 = 7$  forms an increasing sequence and f(4) becomes 2.

	j				i			
index	1	2	3	4	5	6	7	8
x	6	2	5	1	7	4	8	3
f(k)	1	1	2	1	2	1	1	1

• For j = 2, the pair  $x_2 = 2$  and  $x_5 = 7$  form a new pair but the length is same, so no change.

		j			i			
index	1	2	3	4	5	6	7	8
x	6	2	5	1	7	4	8	3
f(k)	1	1	2	1	2	1	1	1

• For j = 3, the pair  $x_3 = 5$  and  $x_5 = 7$  form a new pair and the new length is greater that f(i), so we update f(5) = 3.

			j		i			
index	1	2	3	4	5	6	7	8
x	6	2	5	1	7	4	8	3
f(k)	1	1	2	1	3	1	1	1

#### Complexity of the algorithm

- This process repeats for all elements until i = 8. Then the largest f(k) gives the LIS.
- The complexity of the f(k) updating algorithm is  $\mathcal{O}(n^2)$  and the searching largest f(k) has a complexity  $\mathcal{O}(n)$ . So the total complexity is  $\mathcal{O}(n^2)$ .

#### LIS DP algorithm

```
Algorithm 1 Dynamic programming method for LIS
Require: A is an array of length n.
Ensure: LIS is the length of the longest increasing subsequence of A.
  procedure LongestIncreasingSubsequence(A)
     LIS = 0
     for i in \{1, 2, \dots, n\} do
        a[i] = 1
        for j in \{1, 2, \dots, i-1\} do
           if A[j] < A[i] and a[j] + 1 > a[i] then
               a[i] = a[j] + 1
           end if
        end for
        if a[i] > LIS then
           LIS = a[i]
        end if
     end for
     return LIS
  end procedure
```