

Dynamic Programming

- a. Fibonacci Series
- b. Weighted Interval Scheduling
- c. Knapsack
- d. Longest Common Subsequence

Common Subsequences

- Given a string $x \in \Sigma^n$ a **subsequence** is any string obtained by deleting a subset of the symbols

r e c u r a n c e

- Given two strings $x \in \Sigma^n, y \in \Sigma^m$, a **common subsequence** is a **subsequence** of both x and y

r e c u r a n c e

r e c u r r e n c e

Longest Common Subsequence (LCS)

- **Input:** Two strings $x \in \Sigma^n, y \in \Sigma^m$
- **Output:** The longest common subsequence of x and y

Writing the Recurrence

Recurrence:

$$\text{LCS}(i, j) = \begin{cases} 1 + \text{LCS}(i - 1, j - 1) & \text{if } x_i = y_j \\ \max\{\text{LCS}(i - 1, j), \text{LCS}(i, j - 1)\} & \text{if } x_i \neq y_j \end{cases}$$

Base Cases:

$$\text{LCS}(i, 0) = 0, \text{LCS}(0, j) = 0$$

Solving the Recurrence: Bottom-Up

```
// All inputs are global vars
FindOPT(n,m):
    M[i,0] ← 0,    M[0,j] ← 0

    for (i= 1,...,n):
        for (j = 1,...,m):
            if (xi = yj):
                M[i,j] ← 1 + M[i-1,j-1]
            else:
                M[i,j] ← max{M[i-1,j],M[i,j-1]}

    return M[n,m]
```

Ask the Audience

x = **peat**

y = **leapt**

Compute LCS(i,j) for
each subproblem

	j=0	1	2	2	4	5
	-	l	e	a	p	t
i=0	-	0	0	0	0	0
1	p	0				
2	e	0				
3	a	0				
4	t	0				

Ask the Audience

x = **peat**

y = **leapt**

Compute $LCS(i,j)$ for each subproblem

	j=0	1	2	2	4	5	
	-	l	e	a	p	t	
i=0	-	0	0	0	0	0	
1	p	0	0	0	1	1	
2	e	0	0	1	1	1	
3	a	0	0	1	2	2	
4	t	0	0	1	2	2	3

Finding the Solution

```
// All inputs are global vars
FindLCS(i,j):
  if (i = 0 or j = 0)
    return ""
  if (xi = yj):
    return FindLCS(i-1,j-1)+ xi
  else:
    if (M[i-1,j] > M[i,j-1])
      return FindLCS(i-1,j)
    else:
      return FindLCS(i,j-1)
return M[n,m]
```


Dynamic Programming

- a. Fibonacci Series
- b. Weighted Interval Scheduling
- c. Knapsack
- d. Longest Common Subsequence
- e. Longest Increasing Subsequence

Longest Increasing Subsequence (LIS)

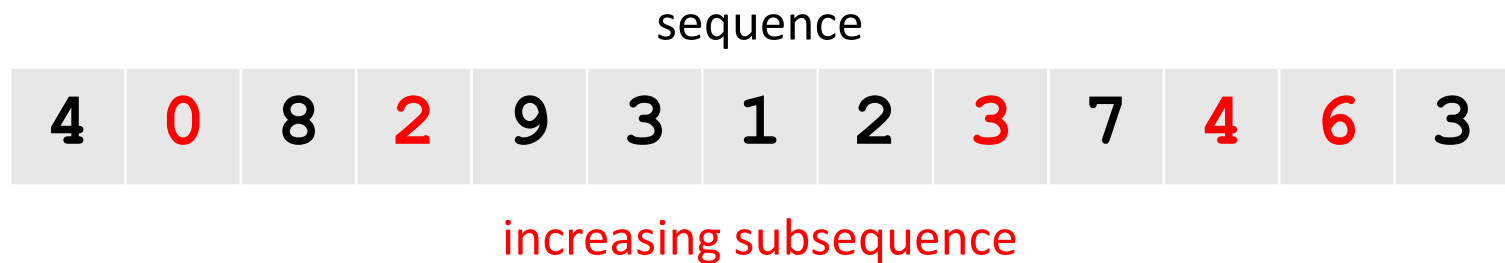
- **Input:** a sequence of numbers x_1, \dots, x_n

sequence

4	0	8	2	9	3	1	2	3	7	4	6	3
---	---	---	---	---	---	---	---	---	---	---	---	---

Longest Increasing Subsequence (LIS)

- **Input:** a sequence of numbers x_1, \dots, x_n



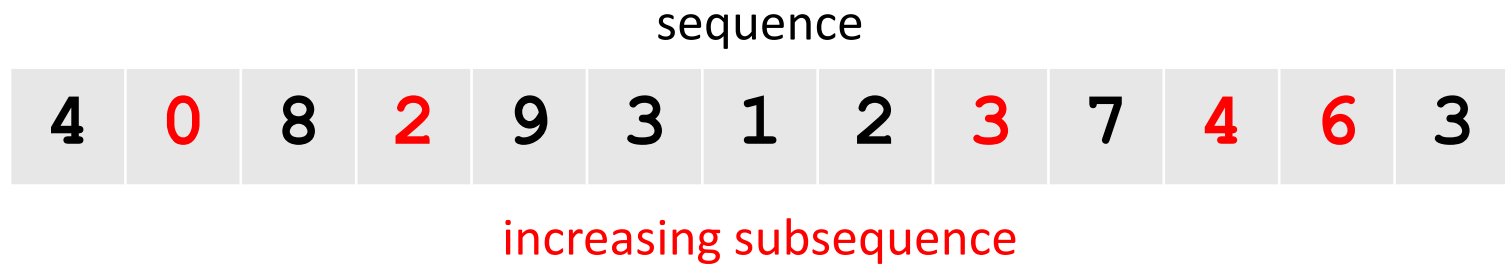
- **Increasing Subsequence:**

indices $1 \leq i_1 \leq i_2 \leq \dots \leq i_k \leq n$

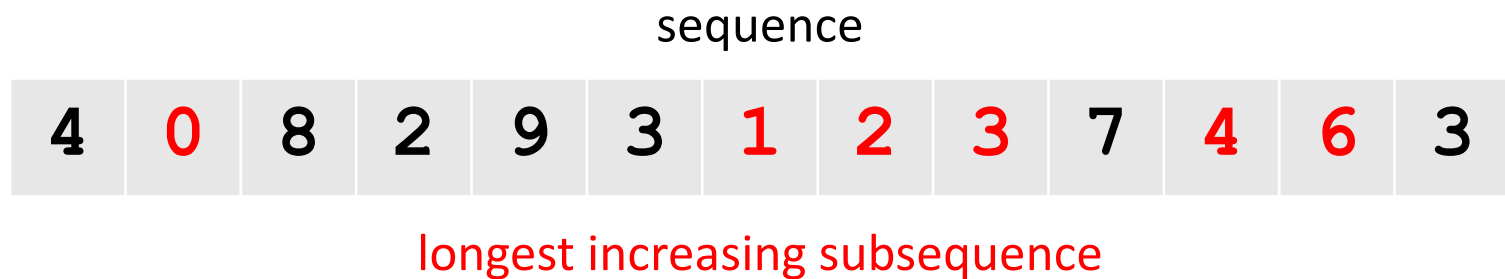
such that $x_{i_1} < x_{i_2} < \dots < x_{i_k}$

Longest Increasing Subsequence (LIS)

- **Input:** a sequence of numbers x_1, \dots, x_n



- **Output:** a longest increasing subsequence



Ask the Audience

- Find a longest increasing subsequence of

14	7	5	6	2	12
----	---	---	---	---	----

Identifying the Subproblems

- Start by finding the value of the optimal solution:
 - In this problem: length of the LIS

Identifying the Subproblems

- Start by finding the value of the optimal solution:
 - In this problem: length of the LIS
- What about defining $LIS(j)$ to be the length of the longest increasing subsequence between the first j elements?

8	9	12	3	6	10
---	---	----	---	---	----

Writing the Recurrence

- Let $LIS(j)$ be the length of the longest increasing subsequence **that ends with** x_j

8	9	12	3	6	10
---	---	----	---	---	----

Writing the Recurrence

- Let $LIS(j)$ be the length of the longest increasing subsequence **that ends with** x_j
- **Case i :** the previous element is x_i

6	7	14	5	12	8
---	---	----	---	----	---

Writing the Recurrence

- Let $LIS(j)$ be the length of the longest increasing subsequence **that ends with** x_j
- **Case i :** the previous element is x_i
 - Some cases are invalid

6	7	14	5	12	8
---	---	----	---	----	---

Writing the Recurrence

- Let $LIS(j)$ be the length of the longest increasing subsequence **that ends with** x_j
- **Case i :** the last two numbers are x_i and x_j

Recurrence:

$$LIS(j) = 1 + \max_{1 \leq i < j \text{ and } x_i < x_j} LIS(i)$$

Base Case:

$$LIS(1) = 1$$

Ask the Audience

- Fill out the values $LIS(j)$ for $j = 1, \dots, 6$

6	10	5	14	8	7
---	----	---	----	---	---

j	1	2	3	4	5	6
LIS(j)	1					

Ask the Audience

Is $LIS(n)$ the length of the optimal solution?

Solving the Recurrence: Bottom-Up

```
// All inputs are global vars
FindOPT(n) :
    M[1] ← 1

    for (j = 2, ..., n) :
        M[j] = 1 + max_{1 ≤ i < j and x_i < x_j} M[i]

    return max_{1 ≤ j ≤ n} M[j]
```

Solving the Recurrence: Bottom-Up

```
FindOPT (n) :
```

```
  M[1] ← 1
```

```
  for (j = 2, ..., n) :
```

```
    M[j] = 1 +  $\max_{1 \leq i < j \text{ and } x_i < x_j} M[i]$ 
```

```
  return  $\max_{1 \leq j \leq n} M[j]$ 
```

Running time:

Recovering the LIS

- Fill out the values $LIS(j)$ for $j = 1, \dots, 6$

6	10	5	14	8	7
---	----	---	----	---	---

j	1	2	3	4	5	6
LIS(j)	1	2	1	3	2	2

Recovering the LIS

```
FindLIS(n) :  
  if (n=1)  
    return x1  
  j = argmax1 ≤ i < n and xi < xn M[i]  
  return FindLIS(j) + {xn}
```

Summary

- Can compute a LIS in time $O(n^2)$
 - Same algorithm works for longest non-decreasing, longest decreasing, longest non-increasing, and more
- Dynamic Programming:
 - Question: What is the final symbol in the LIS?
 - Subproblems represent LIS **with a specific final symbol**
 - The actual optimal value is not always in LIS(n)