Session 5: Dynamic Programming

Saravana Senthilkumar - 3122247001057

February 5, 2026

Contents

1 1. Longest Increasing Subsequence (Exhaustive Search)

1.1 1. Design and Implement LengthES(i)

This block defines the recursive Exhaustive Search function and calls it immediately with the sample arrays.

```
def LengthES(A, i):
    max_len = 0
    n = len(A)
    for j in range(i + 1, n):
        if A[j] > A[i]:
            res = LengthES(A, j)
            if res > max_len:
                max_len = res
    return 1 + max_len
```

```
# Function Calls for LengthES
arr1 = [2, 4, 3, 5, 1, 7, 6, 9, 8]
print(LengthES([float('-inf')] + arr1, 0) - 1)

arr2 = [5, 1, 5, 7, 2, 4, 9, 8]
print(LengthES([float('-inf')] + arr2, 0) - 1)

arr3 = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6, 2, 6]
print(LengthES([float('-inf')] + arr3, 0) - 1)
```

2 2. Memoized Exhaustive Search

2.1 1. Construct and Implement LengthM(i)

This block defines the Memoized function and calls it immediately with the sample arrays.

```
def LengthM(A, i, memo):
    if memo[i] != -1:
        return memo[i]
    max_len = 0
    n = len(A)
    for j in range(i + 1, n):
        if A[j] > A[i]:
            res = LengthM(A, j, memo)
            if res > max_len:
                max_len = res
    memo[i] = 1 + max_len
    return memo[i]

# Function Calls for LengthM
A_prime1 = [float('-inf')] + arr1
memo1 = [-1] * len(A_prime1)
print(LengthM(A_prime1, 0, memo1) - 1)

A_prime2 = [float('-inf')] + arr2
memo2 = [-1] * len(A_prime2)
print(LengthM(A_prime2, 0, memo2) - 1)

A_prime3 = [float('-inf')] + arr3
```

```
memo3 = [-1] * len(A_prime3)
print(LengthM(A_prime3, 0, memo3) - 1)
```

3 3. Dynamic Programming

3.1 1. Design LengthDP(A, 1, n)

This block defines the iterative Dynamic Programming function. It calls 'LengthDP' to display the computed Length (L) and Successor (S) arrays.

```
def LengthDP(arr):
    A = [float('-inf')] + arr
    n = len(A)
    L = [0] * n
    S = [-1] * n
    for i in range(n - 1, -1, -1):
        max_sub_len = 0
        successor = -1
        for j in range(i + 1, n):
            if A[j] > A[i]:
                if L[j] > max_sub_len:
                    max_sub_len = L[j]
                    successor = j
        L[i] = 1 + max_sub_len
        S[i] = successor
    return L, S, A

# Function Call for LengthDP (Viewing the tables for arr1)
L_table, S_table, A_prime = LengthDP(arr1)
print(L_table)
print(S_table)
```

3.2 2. Trace the Solution (TraceLIS)

This block defines the tracing function and calls it to reconstruct the actual subsequences for all three test arrays.

```
def TraceLIS(A, S, start_index):
    result_sequence = []
    curr = S[start_index]
    while curr != -1:
```

```
        result_sequence.append(A[curr])
        curr = S[curr]
    return result_sequence

# Function Calls for TraceLIS
L1, S1, A1 = LengthDP(arr1)
print(TraceLIS(A1, S1, 0))

L2, S2, A2 = LengthDP(arr2)
print(TraceLIS(A2, S2, 0))

L3, S3, A3 = LengthDP(arr3)
print(TraceLIS(A3, S3, 0))
```