Design and Analysis of Algorithms

Dynamic Programming III

Seven letters against six: only four survive — in order.
Skipping the right letters is the whole game — DP finds which.

Dr. Golzari
University of Hormozgan

Fall 1405

Where We're Going

1. Longest common subsequence — a twin in disguise
2. Full solve: grid, row by row, walk back
3. Longest increasing subsequence — the early peak trap
4. Full solve: one row, predecessors, the climb
5. Same problem, new angles — LCS glasses, a faster chase
6. Full solve: $O(n \log n)$, two rows, one split

Session 2 → Session 3: The Same Recipe

problem	state	table shape
matrix chain (S2)	$C[i][j]$: cheapest for $A_i..A_j$	upper triangle
knapsack (S2)	$K[i][w]$: best, first i items	$(n{+}1) \times (W{+}1)$
tree set (S2)	$(in[v], out[v])$ per node	climb post-order
LCS	$L[i][j]$: LCS of the two prefixes	$(m{+}1) \times (n{+}1)$ grid
LIS	$D[i]$: best chain ending at A_i	one row
chase	$T[x]$: smallest tail of length x	grows by one

Session 2 → Session 3: The Same Recipe

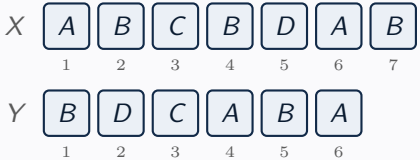
problem	state	table shape
matrix chain (S2)	$C[i][j]$: cheapest for $A_i..A_j$	upper triangle
knapsack (S2)	$K[i][w]$: best, first i items	$(n{+}1) \times (W{+}1)$
tree set (S2)	$(in[v], out[v])$ per node	climb post-order
LCS	$L[i][j]$: LCS of the two prefixes	$(m{+}1) \times (n{+}1)$ grid
LIS	$D[i]$: best chain ending at A_i	one row
chase	$T[x]$: smallest tail of length x	grows by one

The recipe, every time

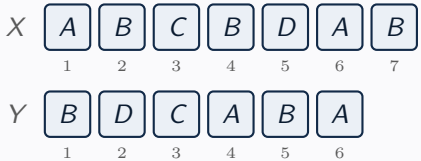
state → recurrence → fill in dependency order → walk back → verify.

The Longest Common Subsequence

The Problem: The Hidden Twin

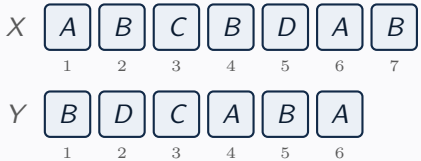


The Problem: The Hidden Twin



A **subsequence** keeps the order, skips freely: $BDAB$ sits inside X (positions $2, 5, 6, 7$) — but it is *not* a substring, the letters need not be adjacent.

The Problem: The Hidden Twin



A **subsequence** keeps the order, skips freely: $BDAB$ sits inside X (positions $2, 5, 6, 7$) — but it is *not* a substring, the letters need not be adjacent.

Goal

Find the **longest** string that is a subsequence of **both** X and Y — the largest hidden twin.

Warm-Up: Brute Force Explodes

Brute force: list every subsequence of X, test each against Y.

Warm-Up: Brute Force Explodes

Brute force: list every subsequence of X, test each against Y.

$m = 7:\quad 2^7 = 128$ candidates — fine. $\qquad m = n = 1000:\quad 2^{1000} \approx 10^{301}$.

Warm-Up: Brute Force Explodes

Brute force: list every subsequence of X, test each against Y.

$m = 7$: $2^7 = 128$ candidates — fine. $m = n = 1000$: $2^{1000} \approx 10^{301}$.

More candidates than atoms in the observable universe ($\sim 10^{80}$).

Warm-Up: Brute Force Explodes

Brute force: list every subsequence of X, test each against Y.

$$m = 7: \quad 2^7 = 128 \text{ candidates — fine.} \qquad m = n = 1000: \quad 2^{1000} \approx 10^{301}.$$

More candidates than atoms in the observable universe ($\sim 10^{80}$).

- Order matters, counts don't: both strings carry $\{A, B, C, D\}$ — yet *DCBA* is a subsequence of *neither*.

Warm-Up: Brute Force Explodes

Brute force: list every subsequence of X, test each against Y.

$$m = 7: \quad 2^7 = 128 \text{ candidates — fine.} \qquad m = n = 1000: \quad 2^{1000} \approx 10^{301}.$$

More candidates than atoms in the observable universe ($\sim 10^{80}$).

- Order matters, counts don't: both strings carry $\{A, B, C, D\}$ — yet *DCBA* is a subsequence of *neither*.

Exponential. DP time.

Subsequences are exponential to list — but their **prefixes** overlap. Share the work.

The State and the Pair

State

$L[i][j]$ = length of the LCS of the prefixes $X_1 X_2 \cdots X_i$ and $Y_1 Y_2 \cdots Y_j$. Base: $L[i][0] = L[0][j] = 0$ (an empty prefix).

The State and the Pair

State

$L[i][j]$ = length of the LCS of the prefixes $X_1 X_2 \cdots X_i$ and $Y_1 Y_2 \cdots Y_j$. Base: $L[i][0] = L[0][j] = 0$ (an empty prefix).

The last decision — what becomes of the final letters x_i and y_j?

The State and the Pair

State

$L[i][j]$ = length of the LCS of the prefixes $X_1 X_2 \cdots X_i$ and $Y_1 Y_2 \cdots Y_j$. Base: $L[i][0] = L[0][j] = 0$ (an empty prefix).

The last decision — what becomes of the final letters x_i and y_j?

$$L[i][j] \;=\; \begin{cases} \underbrace{1 + L[i-1][j-1]}_{\text{match — take it}} & x_i = y_j \\ \max\Big(\underbrace{L[i-1][j]}_{\text{drop } x_i},\; \underbrace{L[i][j-1]}_{\text{drop } y_j} \Big) & x_i \neq y_j \end{cases}$$

The State and the Pair

State

$L[i][j]$ = length of the LCS of the prefixes $X_1 X_2 \cdots X_i$ and $Y_1 Y_2 \cdots Y_j$. Base: $L[i][0] = L[0][j] = 0$ (an empty prefix).

The last decision — what becomes of the final letters x_i and y_j?

$$L[i][j] \;=\; \begin{cases} \underbrace{1 + L[i-1][j-1]}_{\text{match — take it}} & x_i = y_j \\ \max\Big(\underbrace{L[i-1][j]}_{\text{drop } x_i},\; \underbrace{L[i][j-1]}_{\text{drop } y_j} \Big) & x_i \neq y_j \end{cases}$$

- Fill **row by row**: every cell reads only $\nwarrow$, up, left. Answer: $L[7][6]$.
- "Drop both" is never needed — $L[i-1][j-1]$ hides inside the other two.
- $7 \times 6 = 42$ cells: $O(mn)$ time, $O(mn)$ memory (we will shave the memory later).

Fill: Rows 1–2

Two sample cells in full:

Fill: Rows 1–2

Two sample cells in full:

$$L[1][4] = 1 + L[0][3] = 1 \qquad (x_1 = A = y_4 : \text{ first match})$$

$$L[2][2] = \max(L[1][2],\, L[2][1]) = \max(0, 1) = 1 \qquad (x_2 = B \neq y_2 = D)$$

$$L[2][5] = 1 + L[1][4] = 2 \qquad (x_2 = B = y_5)$$

Fill: Rows 1–2

Two sample cells in full:

$$L[1][4] = 1 + L[0][3] = 1 \qquad (x_1 = A = y_4 : \text{ first match})$$

$$L[2][2] = \max(L[1][2],\, L[2][1]) = \max(0, 1) = 1 \qquad (x_2 = B \neq y_2 = D)$$

$$L[2][5] = 1 + L[1][4] = 2 \qquad (x_2 = B = y_5)$$

$i \backslash j$	B	D	C	A	B	A
A_1	0	0	0	1	1	1
B_2	1	1	1	1	2	2

Fill: Rows 1–2

Two sample cells in full:

$$L[1][4] = 1 + L[0][3] = 1 \qquad (x_1 = A = y_4 : \text{ first match})$$

$$L[2][2] = \max(L[1][2],\, L[2][1]) = \max(0, 1) = 1 \qquad (x_2 = B \neq y_2 = D)$$

$$L[2][5] = 1 + L[1][4] = 2 \qquad (x_2 = B = y_5)$$

$i \backslash j$	B	D	C	A	B	A
A_1	0	0	0	1	1	1
B_2	1	1	1	1	2	2

- Row 0 (not shown) is all zeros — the empty prefix shares nothing.

Fill: Rows 3–5

Three sample cells in full:

Fill: Rows 3–5

Three sample cells in full:

$$L[3][3] = 1 + L[2][2] = 2 \qquad L[4][5] = 1 + L[3][4] = 3 \qquad L[5][2] = 1 + L[4][1] = 2$$

Fill: Rows 3–5

Three sample cells in full:

$$L[3][3] = 1 + L[2][2] = 2 \qquad L[4][5] = 1 + L[3][4] = 3 \qquad L[5][2] = 1 + L[4][1] = 2$$

$i \setminus j$	B	D	C	A	B	A
C_3	1	1	2	2	2	2
B_4	1	1	2	2	3	3
D_5	1	2	2	2	3	3

Fill: Rows 3–5

Three sample cells in full:

$$L[3][3] = 1 + L[2][2] = 2 \qquad L[4][5] = 1 + L[3][4] = 3 \qquad L[5][2] = 1 + L[4][1] = 2$$

$i \diagdown j$	B	D	C	A	B	A
C_3	1	1	2	2	2	2
B_4	1	1	2	2	3	3
D_5	1	2	2	2	3	3

- Every match pulls its cell one past the $\nwarrow$ neighbour — the twin is growing.

Fill: Rows 6–7 — The Full Grid

Fill: Rows 6–7 — The Full Grid

$$\underbrace{L[6][4] = 1 + L[5][3] = 3}_{x_6 = A = y_4} \qquad \underbrace{L[6][6] = 1 + L[5][5] = \mathbf{4}}_{x_6 = A = y_6} \qquad \underbrace{L[7][5] = 1 + L[6][4] = 4}_{x_7 = B = y_5}$$

Fill: Rows 6–7 — The Full Grid

$$\underbrace{L[6][4] = 1 + L[5][3] = 3}_{x_6 = A = y_4} \qquad \underbrace{L[6][6] = 1 + L[5][5] = \mathbf{4}}_{x_6 = A = y_6} \qquad \underbrace{L[7][5] = 1 + L[6][4] = 4}_{x_7 = B = y_5}$$

$i \backslash j$	B	D	C	A	B	A
A_1	0	0	0	1	1	1
B_2	1	1	1	1	2	2
C_3	1	1	2	2	2	2
B_4	1	1	2	2	3	3
D_5	1	2	2	2	3	3
A_6	1	2	2	3	3	4
B_7	1	2	2	3	4	**4**

Fill: Rows 6–7 — The Full Grid

$$\underbrace{L[6][4] = 1 + L[5][3] = 3}_{x_6 = A = y_4} \qquad \underbrace{L[6][6] = 1 + L[5][5] = \mathbf{4}}_{x_6 = A = y_6} \qquad \underbrace{L[7][5] = 1 + L[6][4] = 4}_{x_7 = B = y_5}$$

$i \backslash j$	B	D	C	A	B	A
A_1	0	0	0	1	1	1
B_2	1	1	1	1	2	2
C_3	1	1	2	2	2	2
B_4	1	1	2	2	3	3
D_5	1	2	2	2	3	3
A_6	1	2	2	3	3	4
B_7	1	2	2	3	4	**4**

Answer

$L[7][6] = 4$ — now find *which* letters. (*BCAB* and *BDAB* score 4 too — the walk picks one.)

Traceback: The Hidden Twin

Rule: $x_i = y_j \Rightarrow$ the letter **joins**, go $\nwarrow$; otherwise follow the **bigger** neighbour (tie $\Rightarrow$ up).

Traceback: The Hidden Twin

Rule: $x_i = y_j \Rightarrow$ the letter **joins**, go $\nwarrow$; otherwise follow the **bigger** neighbour (tie $\Rightarrow$ up).

$$(7,6)\uparrow\ (6,6)\ \nwarrow\ \boxed{A}\ (5,5)\uparrow\ (4,5)\ \nwarrow\ \boxed{B}\ (3,4)\leftarrow$$

$$(3,3)\ \nwarrow\ \boxed{C}\ (2,2)\leftarrow\ (2,1)\ \nwarrow\ \boxed{B}\ (1,0)\ \text{stop}$$

Traceback: The Hidden Twin

Rule: $x_i = y_j \Rightarrow$ the letter **joins**, go $\nwarrow$; otherwise follow the **bigger** neighbour (tie $\Rightarrow$ up).

$$(7,6)\uparrow\ (6,6)\ \nwarrow\ \boxed{A}\ (5,5)\uparrow\ (4,5)\ \nwarrow\ \boxed{B}\ (3,4)\leftarrow$$

$$(3,3)\ \nwarrow\ \boxed{C}\ (2,2)\leftarrow\ (2,1)\ \nwarrow\ \boxed{B}\ (1,0)\ \text{stop}$$

B	C	B	A
X_2, Y_1	X_3, Y_3	X_4, Y_5	X_6, Y_6

Traceback: The Hidden Twin

Rule: $x_i = y_j \Rightarrow$ the letter **joins**, go $\nwarrow$; otherwise follow the **bigger** neighbour (tie $\Rightarrow$ up).

$$(7,6)\uparrow\ (6,6)\ \nwarrow\ \boxed{A}\ (5,5)\uparrow\ (4,5)\ \nwarrow\ \boxed{B}\ (3,4)\leftarrow$$

$$(3,3)\ \nwarrow\ \boxed{C}\ (2,2)\leftarrow\ (2,1)\ \nwarrow\ \boxed{B}\ (1,0)\ \text{stop}$$



Verify: X: positions $2,3,4,6$; Y: $1,3,5,6$ — both read $B\,C\,B\,A$, and $4 = L[7][6]$. ✓

Traceback: The Hidden Twin

Rule: $x_i = y_j \Rightarrow$ the letter **joins**, go $\nwarrow$; otherwise follow the **bigger** neighbour (tie $\Rightarrow$ up).

$$(7,6)\uparrow\ (6,6)\ \nwarrow\ \boxed{A}\ (5,5)\uparrow\ (4,5)\ \nwarrow\ \boxed{B}\ (3,4)\leftarrow$$

$$(3,3)\ \nwarrow\ \boxed{C}\ (2,2)\leftarrow\ (2,1)\ \nwarrow\ \boxed{B}\ (1,0)\ \text{stop}$$



Verify: X: positions $2,3,4,6$; Y: $1,3,5,6$ — both read $BCBA$, and $4 = L[7][6]$. ✓

- Brute force offered 128 candidates; DP filled 42 cells and walked 8 steps.

The Longest Increasing Subsequence

The Problem: Never Dip



value on top, position below

The Problem: Never Dip



value on top, position below

A **subsequence** keeps the order, skips freely; **increasing** means every kept value is strictly larger than the one before it.

The Problem: Never Dip



value on top, position below

A **subsequence** keeps the order, skips freely; **increasing** means every kept value is strictly larger than the one before it.
Example legal but short: $5, 6, 9$ (positions $1, 4, 7$). The full climb has more.

The Problem: Never Dip



value on top, position below

A **subsequence** keeps the order, skips freely; **increasing** means every kept value is strictly larger than the one before it.
Example legal but short: $5, 6, 9$ (positions $1, 4, 7$). The full climb has more.

Goal

Choose positions $i_1 < i_2 < \cdots < i_k$ with $A[i_1] < A[i_2] < \cdots < A[i_k]$ and **maximum** k.

Warm-Up: The Early Peak

Array: $[1,\ 100,\ 2,\ 3]$.

Warm-Up: The Early Peak

Array: $[1,\ 100,\ 2,\ 3]$.
Grab what extends: take 1, then 100 — stuck: 2 and 3 are smaller than 100. Total 2.

Warm-Up: The Early Peak

Array: $[1,\ 100,\ 2,\ 3]$.
Grab what extends: take 1, then 100 — stuck: 2 and 3 are smaller than 100. Total 2.
Skip the peak: $1, 2, 3$ — total 3.

Warm-Up: The Early Peak

Array: $[1,\ 100,\ 2,\ 3]$.
Grab what extends: take 1, then 100 — stuck: 2 and 3 are smaller than 100. Total 2.
Skip the peak: $1, 2, 3$ — total 3.

The early peak starves the climb

One flashy pick costs the whole tail. DP time.

The State: End Where You Land

State

$D[i]$ = length of the longest increasing subsequence that **ends exactly at** $A[i]$. Base: every chain starts somewhere — $D[i] \geq 1$.

The State: End Where You Land

State

$D[i]$ = length of the longest increasing subsequence that **ends exactly at** $A[i]$. Base: every chain starts somewhere — $D[i] \geq 1$.

The last step jumps into A_i from some earlier, smaller value:

The State: End Where You Land

State

$D[i]$ = length of the longest increasing subsequence that **ends exactly at** $A[i]$. Base: every chain starts somewhere — $D[i] \geq 1$.

The last step jumps into A_i from some earlier, smaller value:

$$D[i] \;=\; 1 \;+\; \max_{\substack{k<i \\ A_k<A_i}} D[k] \qquad (\text{empty max} \equiv 0 \;\Rightarrow\; D[i]=1)$$

The State: End Where You Land

State

$D[i]$ = length of the longest increasing subsequence that **ends exactly at** $A[i]$. Base: every chain starts somewhere — $D[i] \geq 1$.

The last step jumps into A_i from some earlier, smaller value:

$$D[i] \;=\; 1 \;+\; \max_{\substack{k<i \\ A_k<A_i}} D[k] \qquad\qquad (\text{empty max} \equiv 0 \;\Rightarrow\; D[i]=1)$$

- Fill **left to right**: cell i reads all earlier cells — $O(n^2)$ total.
- Answer: $\max_i D[i]$ — **not** $D[8]$: unlike knapsack's $K[4][8]$, the best chain may land anywhere.
- Strictly smaller: $A_k < A_i$ — equal values never chain.

The Full Table

One sample cell in full:

The Full Table

One sample cell in full:
$D[7]$: $A[7] = 9$; earlier smaller values: $5\,(1),\ 2\,(1),\ 8\,(2),\ 6\,(2),\ 3\,(2),\ 6\,(3)$ — best landing pad is $D = 3$ at the second 6:

$$D[7] = 1 + 3 = 4$$

The Full Table

One sample cell in full:
$D[7]$: $A[7] = 9$; earlier smaller values: $5\,(1),\ 2\,(1),\ 8\,(2),\ 6\,(2),\ 3\,(2),\ 6\,(3)$ — best landing pad is $D = 3$ at the second 6:

$$D[7] = 1 + 3 = 4$$

position i	1	2	3	4	5	6	7	8
$A[i]$	5	2	8	6	3	6	9	7
$D[i]$	1	1	2	2	2	3	**4**	**4**

The Full Table

One sample cell in full:
$D[7]$: $A[7] = 9$; earlier smaller values: $5\,(1),\ 2\,(1),\ 8\,(2),\ 6\,(2),\ 3\,(2),\ 6\,(3)$ — best landing pad is $D = 3$ at the second 6:

$$D[7] = 1 + 3 = 4$$

position i	1	2	3	4	5	6	7	8
$A[i]$	5	2	8	6	3	6	9	7
$D[i]$	1	1	2	2	2	3	**4**	**4**

Answer

$\max_i D[i] = 4$ — reached twice: ending at 9 or at 7. Now find *which* chain.

Traceback: The Climb

Rule: start at the **leftmost** max; step to any earlier smaller $A[k]$ with $D[k] = D[i] - 1$ (we take the latest).

Traceback: The Climb

Rule: start at the **leftmost** max; step to any earlier smaller $A[k]$ with $D[k] = D[i] - 1$ (we take the latest).

$$i{=}7:\ 9 \ \Rightarrow\ k{=}6:\ 6 \ \Rightarrow\ k{=}5:\ 3 \ \Rightarrow\ k{=}2:\ 2 \ \Rightarrow\ \text{stop}$$

Read backwards — the chain $2, 3, 6, 9$.

Traceback: The Climb

Rule: start at the **leftmost** max; step to any earlier smaller $A[k]$ with $D[k] = D[i] - 1$ (we take the latest).

$$i{=}7:\ 9 \ \Rightarrow\ k{=}6:\ 6 \ \Rightarrow\ k{=}5:\ 3 \ \Rightarrow\ k{=}2:\ 2 \ \Rightarrow\ \text{stop}$$

Read backwards — the chain $2, 3, 6, 9$.



Traceback: The Climb

Rule: start at the **leftmost** max; step to any earlier smaller $A[k]$ with $D[k] = D[i] - 1$ (we take the latest).

$$i{=}7:\ 9 \ \Rightarrow\ k{=}6:\ 6 \ \Rightarrow\ k{=}5:\ 3 \ \Rightarrow\ k{=}2:\ 2 \ \Rightarrow\ \text{stop}$$

Read backwards — the chain $2, 3, 6, 9$.



Verify: $2 < 3 < 6 < 9$ ✓ positions $2 < 5 < 6 < 7$ ✓ length $4 = \max_i D[i]$. ✓

Traceback: The Climb

Rule: start at the **leftmost** max; step to any earlier smaller $A[k]$ with $D[k] = D[i] - 1$ (we take the latest).

$$i{=}7:\ 9 \ \Rightarrow\ k{=}6:\ 6 \ \Rightarrow\ k{=}5:\ 3 \ \Rightarrow\ k{=}2:\ 2 \ \Rightarrow\ \text{stop}$$

Read backwards — the chain $2, 3, 6, 9$.



Verify: $2 < 3 < 6 < 9$ ✓ positions $2 < 5 < 6 < 7$ ✓ length $4 = \max_i D[i]$. ✓

- $2, 5, 6, 9$ and $2, 3, 6, 7$ also score 4 — and strictness matters: $6 \to 6$ is not a step.

Same Problem, New Angles

The Bridge: LIS Through LCS Glasses

Sort a copy of A:

$$S = [2,\ 3,\ 5,\ 6,\ 6,\ 7,\ 8,\ 9]$$

The Bridge: LIS Through LCS Glasses

Sort a copy of A:

$$S = [2,\ 3,\ 5,\ 6,\ 6,\ 7,\ 8,\ 9]$$

The claim

An increasing subsequence of A is exactly a common subsequence of A and S: $LIS(A) = LCS(A, S)$.

The Bridge: LIS Through LCS Glasses

Sort a copy of A:

$$S = [2,\ 3,\ 5,\ 6,\ 6,\ 7,\ 8,\ 9]$$

The claim

An increasing subsequence of A is exactly a common subsequence of A and S: $LIS(A) = LCS(A, S)$.

- Our array: $2, 3, 6, 9$ lives in A (positions $2, 5, 6, 7$) and in S (positions $1, 2, 4, 8$) — LCS $= 4$. Same answer, no new table.

The Bridge: LIS Through LCS Glasses

Sort a copy of A:

$$S = [2,\ 3,\ 5,\ 6,\ 6,\ 7,\ 8,\ 9]$$

The claim

An increasing subsequence of A is exactly a common subsequence of A and S: $LIS(A) = LCS(A, S)$.

- Our array: $2, 3, 6, 9$ lives in A (positions $2, 5, 6, 7$) and in S (positions $1, 2, 4, 8$) — LCS $= 4$. Same answer, no new table.
- Two doors, one room: whichever you fill, the recipe is identical — **state, recurrence, fill, walk back, verify**.

The Bridge: LIS Through LCS Glasses

Sort a copy of A:

$$S = [2,\ 3,\ 5,\ 6,\ 6,\ 7,\ 8,\ 9]$$

The claim

An increasing subsequence of A is exactly a common subsequence of A and S: $LIS(A) = LCS(A, S)$.

- Our array: $2, 3, 6, 9$ lives in A (positions $2, 5, 6, 7$) and in S (positions $1, 2, 4, 8$) — LCS $= 4$. Same answer, no new table.
- Two doors, one room: whichever you fill, the recipe is identical — **state, recurrence, fill, walk back, verify**.

Fine print

Duplicates can lie: $A = [2, 2]$, $S = [2, 2]$: LCS says 2, strict LIS says 1. Distinct values $\Rightarrow$ safe. (Exercise 4.)

Faster: Chase the Smallest Tail

The chase

$T[x]$ = the **smallest possible tail** among all increasing chains of length x seen so far.

Faster: Chase the Smallest Tail

The chase

$T[x]$ = the **smallest possible tail** among all increasing chains of length x seen so far.

- T stays strictly increasing $\Rightarrow$ each element binary-searches its slot: replace the first $T[x] \geq$ element, or append. $O(\log n)$ each — $O(n \log n)$ total.

Faster: Chase the Smallest Tail

The chase

$T[x]$ = the **smallest possible tail** among all increasing chains of length x seen so far.

- T stays strictly increasing $\Rightarrow$ each element binary-searches its slot: replace the first $T[x] \geq$ element, or append. $O(\log n)$ each — $O(n \log n)$ total.

step	read	action	T after
1	5	append	5
2	2	replace 5	2
3	8	append	2 8
4	6	replace 8	2 6

step	read	action	T after
5	3	replace 6	2 3
6	6	append	2 3 6
7	9	append	2 3 6 9
8	7	replace 9	2 3 6 **7**

Faster: Chase the Smallest Tail

The chase

$T[x]$ = the **smallest possible tail** among all increasing chains of length x seen so far.

- T stays strictly increasing $\Rightarrow$ each element binary-searches its slot: replace the first $T[x] \geq$ element, or append. $O(\log n)$ each — $O(n \log n)$ total.

step	read	action	T after
1	5	append	5
2	2	replace 5	2
3	8	append	2 8
4	6	replace 8	2 6

step	read	action	T after
5	3	replace 6	2 3
6	6	append	2 3 6
7	9	append	2 3 6 9
8	7	replace 9	2 3 6 **7**

Answer

$|T| = 4$ — the same length as the D-table, at $O(n \log n)$ speed. The chase promises the **length** only: 9 was evicted at step 8 even though $2, 3, 6, 9$ was optimal — membership lives in the D-table.

Lean Memory: Two Rows and a Split

Lean Memory: Two Rows and a Split

Two rows suffice

Row i reads only row $i-1$ and its left neighbour — keep two rows: $O(\min(m, n))$ memory, not $O(mn)$.

Lean Memory: Two Rows and a Split

Two rows suffice

Row i reads only row $i-1$ and its left neighbour — keep two rows: $O(\min(m,n))$ memory, not $O(mn)$.

But two rows know the *length*, not the *string*. Hirschberg's fix: fill L forward and a suffix twin $B[i][j]$ = LCS of $X_i..X_7$, $Y_j..Y_6$ backward; the optimal chain must cross the cut:

Lean Memory: Two Rows and a Split

Two rows suffice

Row i reads only row $i-1$ and its left neighbour — keep two rows: $O(\min(m,n))$ memory, not $O(mn)$.

But two rows know the *length*, not the *string*. Hirschberg's fix: fill L forward and a suffix twin $B[i][j]$ = LCS of $X_i..X_7$, $Y_j..Y_6$ backward; the optimal chain must cross the cut:

j	0	1	2	3	4	5	6
$L[4][j]$	0	1	1	2	2	3	3
$B[5][j{+}1]$	3	3	2	2	1	1	0
sum	3	**4**	3	**4**	3	**4**	3

Lean Memory: Two Rows and a Split

Two rows suffice

Row i reads only row $i-1$ and its left neighbour — keep two rows: $O(\min(m,n))$ memory, not $O(mn)$.

But two rows know the *length*, not the *string*. Hirschberg's fix: fill L forward and a suffix twin $B[i][j]$ = LCS of $X_i..X_7$, $Y_j..Y_6$ backward; the optimal chain must cross the cut:

j	0	1	2	3	4	5	6
$L[4][j]$	0	1	1	2	2	3	3
$B[5][j{+}1]$	3	3	2	2	1	1	0
sum	3	**4**	3	**4**	3	**4**	3

First max $= 4$ at $j = 1$: left $=$ **B**, right $=$ **DAB**; stitch: **BDAB**, length 4. ✓

Lean Memory: Two Rows and a Split

Two rows suffice

Row i reads only row $i-1$ and its left neighbour — keep two rows: $O(\min(m,n))$ memory, not $O(mn)$.

But two rows know the *length*, not the *string*. Hirschberg's fix: fill L forward and a suffix twin $B[i][j]$ = LCS of $X_i..X_7$, $Y_j..Y_6$ backward; the optimal chain must cross the cut:

j	0	1	2	3	4	5	6
$L[4][j]$	0	1	1	2	2	3	3
$B[5][j{+}1]$	3	3	2	2	1	1	0
sum	3	**4**	3	**4**	3	**4**	3

First max $= 4$ at $j = 1$: left $=$ **B**, right $=$ **DAB**; stitch: **BDAB**, length 4. ✓

Two witnesses, one length

BCBA from the arrows, *BDAB* from the split — both honest. Time stays $O(mn)$; memory drops to two rows.

Wrap-Up

Summary

- **LCS**: state = grid of prefixes; match $\Rightarrow$ take $+$ $\nwarrow$, else the bigger neighbour; fill row by row; walk back — 128 candidates $\rightarrow$ $4 = BCBA$ (and $BDAB$ by the split).

Summary

- **LCS**: state = grid of prefixes; match $\Rightarrow$ take $+$ $\nwarrow$, else the bigger neighbour; fill row by row; walk back — 128 candidates $\rightarrow 4 =$ *BCBA* (and *BDAB* by the split).
- **LIS**: state = landing pad; look back at earlier, smaller cells; answer $\max_i D[i] = 4$ with $\{2, 3, 6, 9\}$ — the early peak starves the climb.

Summary

- **LCS**: state = grid of prefixes; match $\Rightarrow$ take + $\nwarrow$, else the bigger neighbour; fill row by row; walk back — 128 candidates $\rightarrow 4 = BCBA$ (and $BDAB$ by the split).
- **LIS**: state = landing pad; look back at earlier, smaller cells; answer $\max_i D[i] = 4$ with $\{2, 3, 6, 9\}$ — the early peak starves the climb.
- **New angles**: $LIS = LCS$ with a sorted copy; the chase answers in $O(n \log n)$; two rows and one split rebuild the string in lean memory.

Summary

- **LCS**: state = grid of prefixes; match $\Rightarrow$ take $+$ $\nwarrow$, else the bigger neighbour; fill row by row; walk back — 128 candidates $\rightarrow 4 = BCBA$ (and $BDAB$ by the split).
- **LIS**: state = landing pad; look back at earlier, smaller cells; answer $\max_i D[i] = 4$ with $\{2, 3, 6, 9\}$ — the early peak starves the climb.
- **New angles**: $LIS = LCS$ with a sorted copy; the chase answers in $O(n \log n)$; two rows and one split rebuild the string in lean memory.
- Same recipe as S1–S2: **state, recurrence, fill, walk back, verify**.

Next session

DP continues — no spoilers.

Exercises

1. $X = AGGTAB$, $Y = GXTXAYB$: build L, give the length and one optimal string.
2. LIS of $[3, 1, 4, 1, 5, 9, 2, 6]$: fill D, give the picks. (Bragging rights: the array is π.)
3. Why does the LIS answer live at $\max_i D[i]$, while knapsack's sat at $K[4][8]$? One sentence each.
4. Find a tiny array where $LCS(A,\ \text{sorted}(A))$ **overcounts** the strict LIS — and say why. (Hint: duplicates.)
5. **(Challenge)** Run the chase on $[1, 4, 2, 3, 0, 5]$: it ends with $T = [0, 2, 3, 5]$ — *not* a subsequence of A (0 comes last!). Why is the length still correct?