Stock Buy Sell - K Transaction Allowed

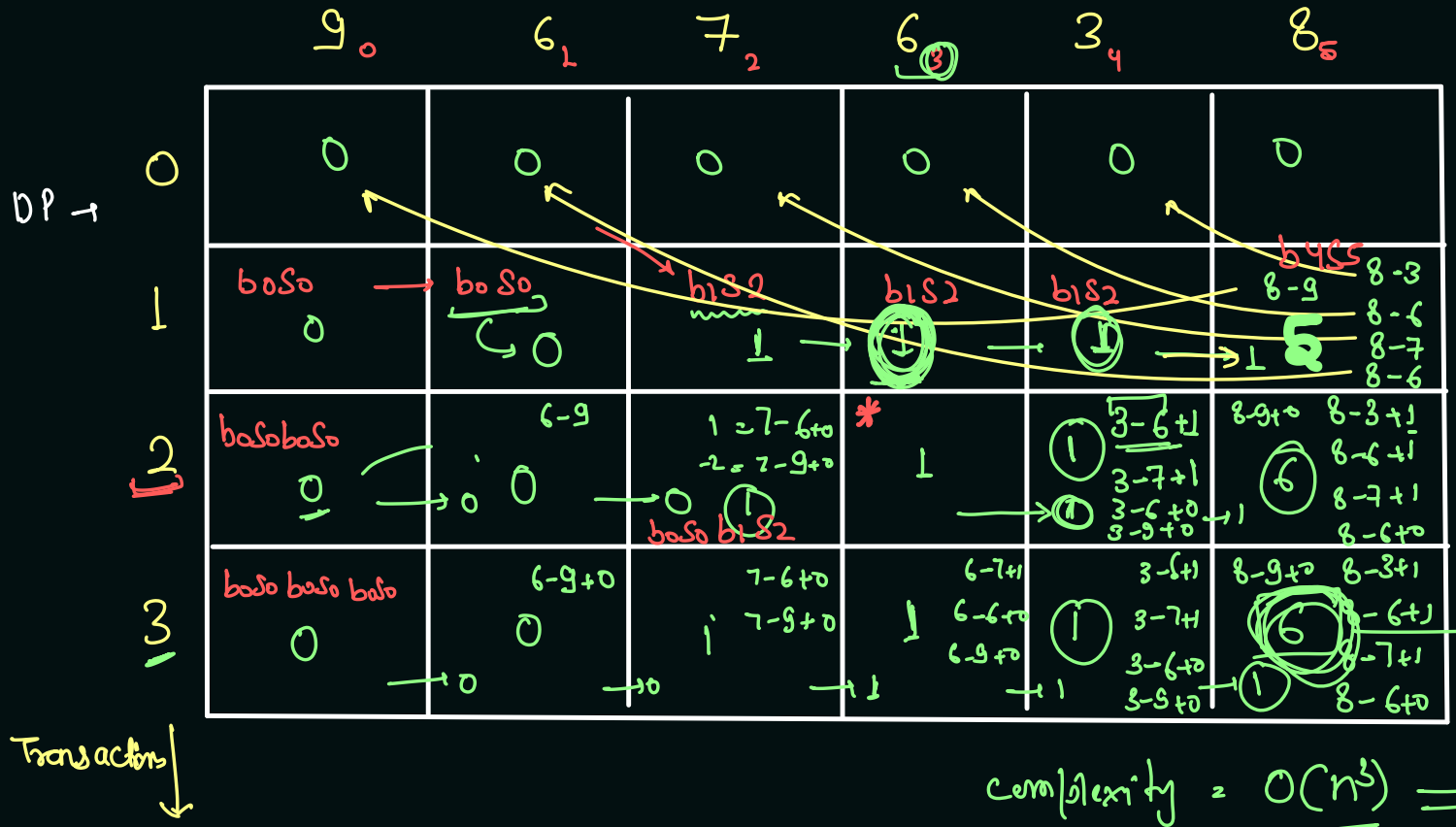
Price :- 9 6 7 6 3 8
0 1 2 3 4 5

K = 3 → transaction allowed

→ Price

Steps in tabulation :

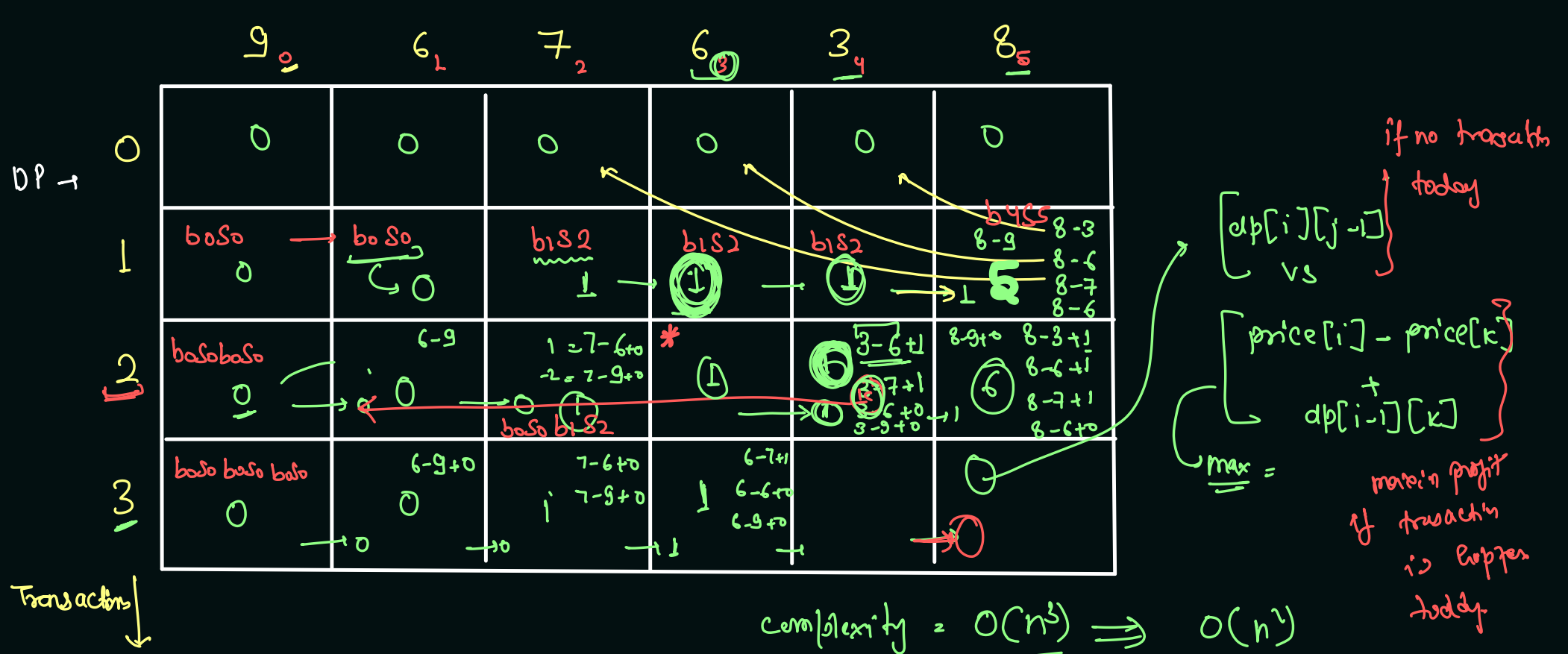
1. Storage
2. Assign memory to cell
3. figure out smallest problem
4. Travel and fill



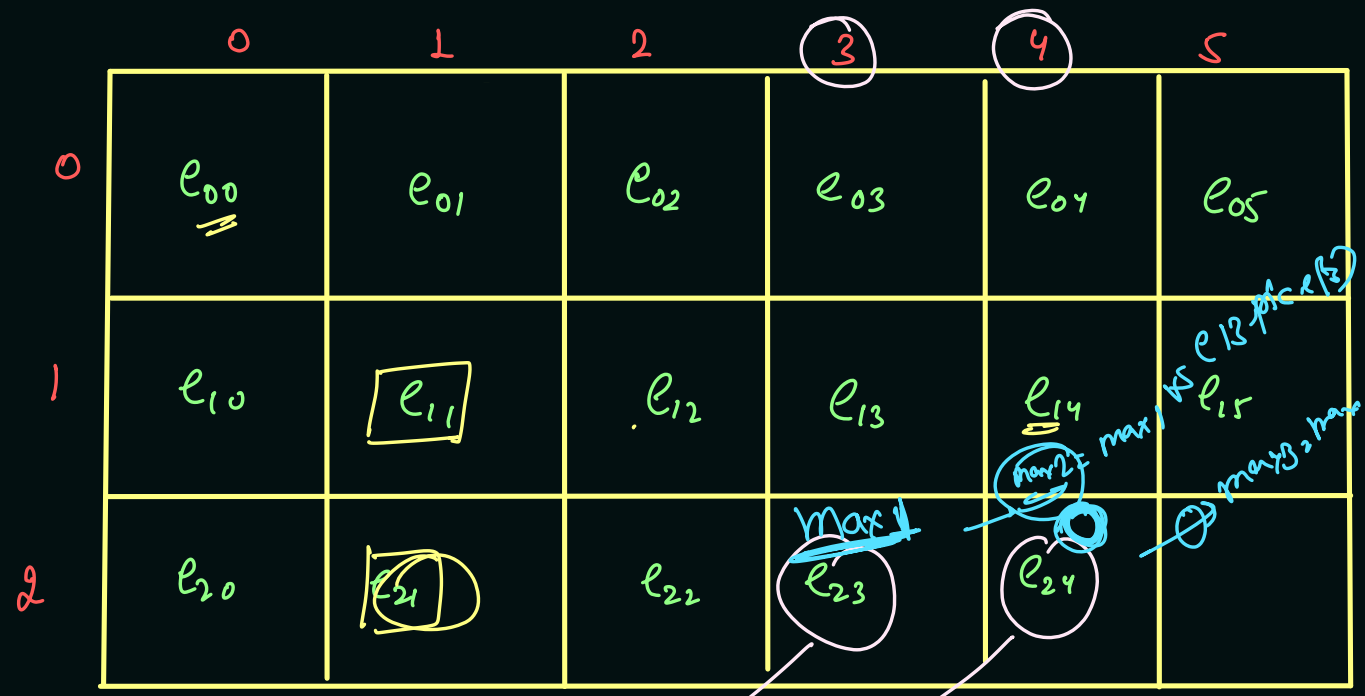
Rows → k+1
cols → arr.length

complexity = $O(n^3) \Rightarrow O(n^2)$

* Max. profit we can generate till 3rd day with 2 transaction.


9 6 7 6 3 8
0 1 2 3 4 5
DP →
Transactions
0 0 0 0 0 0
boSo 0
boSo 0
bIS2 1
bIS2 1
bIS2 1
boSoboSo 0
baSo bIS2
boSo boSo boSo 0
8-9
8-3
8-6
8-7
8-6
6-9
6-9+0
7-6+0
7-9+0
6-7+1
6-6+0
6-9+0
3-6+1
3-7+1
3-6+0
3-9+0
8-9+0
8-3+1
8-6+1
8-7+1
8-6+0
if no transaction today
dp[i][j-1]
vs
price[i] - price[k] + dp[i-1][k]
max =
max in profit if transaction is happen today
complexity = O(n³) ⇒ O(n²)

Case - Where we does transaction today.



price[5] - price[4] + e_{14}

price[5] - price[3] + e_{13}

price[5] - price[2] + e_{12}

price[5] - price[1] + e_{11}

price[5] - price[0] + e_{10}

max

e_{12} - price[2]
e_{11} - price[1]
e_{10} - price[0]
max1

+ price[3]

e_{13} - price[3]
e_{12} - price[2]
e_{11} - price[1]
e_{10} - price[0]
max1

+ price[4]

e_{14} - price[4]
e_{13} - price[3]
e_{12} - price[2]
e_{11} - price[1]
e_{10} - price[0]
max

+ price[5]

for max in current = Solve to use max

max for next day

$O(n^3)$ to $O(n^2)$

max = max vs. price e_{21} - price[1]

Longest Increasing Subsequence LIS :→ $O(n^2)$

Given an array, find longest increasing subset from array.

Subsequence → 2^n in Strg

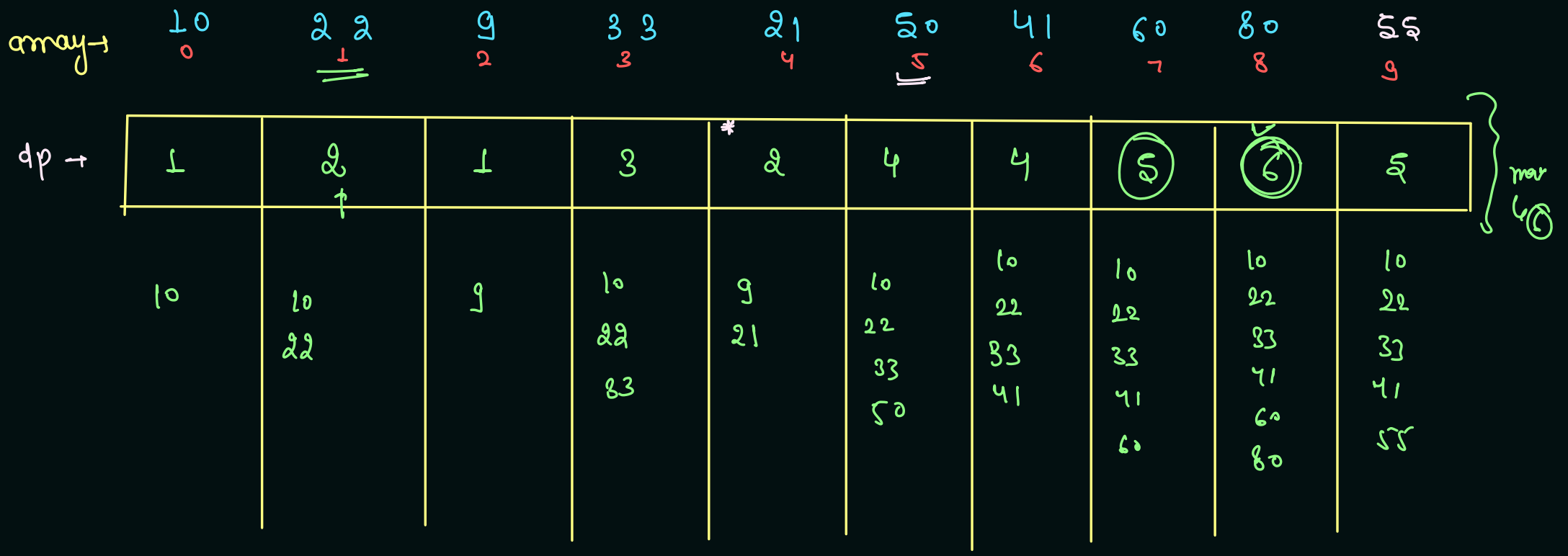
Substring → $\frac{n(n+1)}{2}$ Strg

Subset → 2^n in Array

Subarray → $\frac{n(n+1)}{2}$ in Arr

Strg =

Manig-



* dp[i] → length of LIS which is ending at arr[i]

10	22	9	33	21	50	41	60	80	55
0	1	2	3	4	5	6	7	8	9
1	2	1	3	2	4	4	5	6	5

dp

0	1	2	3	4	5	6	7	8	9
10	10 22	9	10 22 33	9 21	10 22 33 50	10 22 33 41	10 22 33 41 60	10 22 33 41 60 80	10 22 33 41 55

i

max = 0 1

ans = 1 2 3 4 5 6

Complexity - $O(n^2)$

↳ $O(n \log n)$

```
int[] dp = new int[arr.length];
dp[0] = 1;
int ans = 1;
for(int i = 1; i < arr.length; i++) {
    int max = 0;
    for(int j = i - 1; j >= 0; j--) {
        if(arr[j] < arr[i]) {
            max = Math.max(max, dp[j]);
        }
    }
    dp[i] = max + 1;
    ans = Math.max(ans, dp[i]);
}
return ans;
```

Maximum Sum Increasing Subsequence

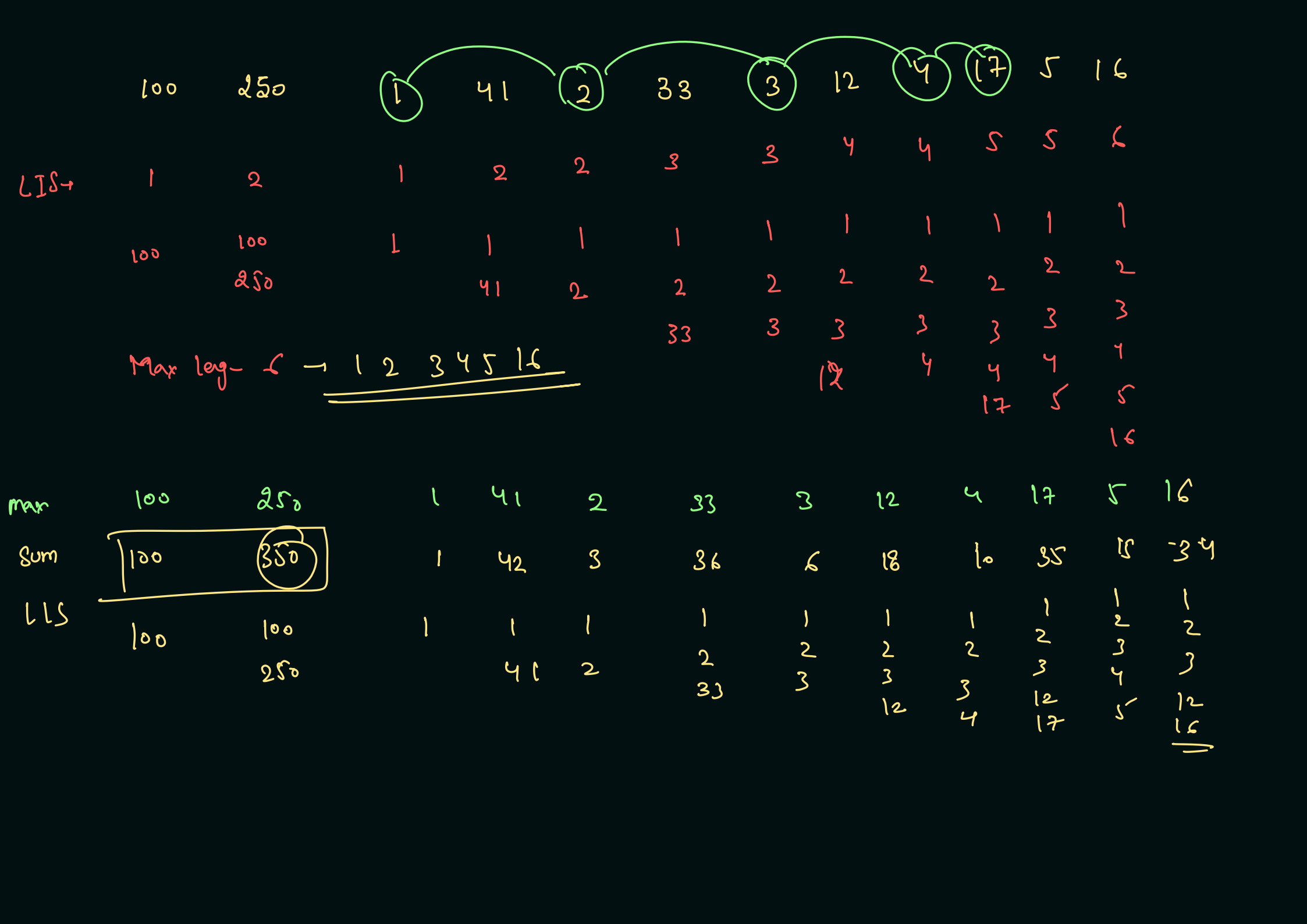
Maximum sum from all possible increasing subsequence, brute-force -

dp →

	10	22	9	33	21	50	41	60	80	1
	0	1	2	3	4	5	6	7	8	9
404	10	32	9	65	* 81	115	106	175	255	1
404	10	10 22	9	10 22 33	10 21	10 22 33 55	10 22 33 41	10 22 33 55 60	10 22 33 55 60 80	1

* → dp[i] → max sum of increasing subseq. which is ending at arr[i]

Major Role is of sum not of length,


100 250 1 41 2 33 3 12 4 17 5 16
LIS+
1 2 1 2 2 3 3 4 4 5 5 6
100 100 1 1 1 1 1 1 1 1 1 1
250 41 2 2 2 2 2 2 2 2
33 3 3 3 3 3 3
4 4 4 4
17 5 5
16
Max leng → 1 2 3 4 5 16
max 100 250 1 41 2 33 3 12 4 17 5 16
Sum 100 350 1 42 3 36 6 18 10 35 15 34
LIS 100 100 1 1 1 1 1 1 1 1 1 1
250 41 2 2 2 2 2 2 2 2
33 3 3 3 3 3 3
12 4 4 4 4
17 5 5
16

Longest Decreasing Subsequence :

LDS + Bitonic

→ Left to Right LDS

10	22	9	33	21	50	41	60	80	1
0	1	2	3	4	5	6	7	8	9
3	3	2	3	2	3	2	2	2	1

10	22	9	33	21	50	41	60	80	1
9	21	1	21	1	41	1	1	1	
1	1		1		1				

LDS [Right to left] — length of longest decreasing subseq. starting at i^{th} index

Meaning in LIS → ending at i

Meaning in LDS → starting at i

Bitonic subseq. = LIS (increasing, left) + LDS (Dec, right)

LIS	left to Right	Right to left
LDS	left to Right	Right to left