DP Season

- Tiling DP
- Knapsack DP
- Palindrome DP
- LCS/LIS
- Matrix Chain DP
- Digit DP
- Bitmask DP

Assignment 2 — Fibonacci

Fibonacci	Longest Increasing Subsequence	Coin Change Problem
DP(n)	DP(n)	DP(n)

DP — 1D array

Longest Palindromic Substring — 2D array

Longest Common Subsequence — 2D array

P1. Tiling 2xN

2 × N — 1D array — Clues → Array

Find number of ways to fill the rectangle

Example at N=3

1. VVV
2. VHH
3. HVV

Ex 2: N = 4

1. HHHH
2. VVVV

Define
1. DP state
DP(i) = number of ways to tile 2 × i

DP(i-1) — DP(i-2)

Recipe
Define DP state → RR → DP optimize (write)

- easy to tackle

DP(n) = DP(n-1) + DP(n-2)

Iterative (Bottom-to-Top)

DP[0] = 1, DP[1] = 2
for (i=3; i<=n; i++)
DP[i] = DP[i-1]+DP[i-2]
cout << DP[n]

- Stack
- Faster slightly

Recursive (Top-to-Bottom)

```
int solve(int i) {
    if (i<=2) return i; //base
    if (DP[i] != -1) return DP[i]; //memoization
    DP[i] = solve(i-1) + solve(i-2);
    return DP[i];
}
```

- memoization

P2 Wine Cellar

original price: 2 3 1 4
1 2 3 4

1. every year → we sell exactly 1 bottle

1st →
2nd →
3rd →
4th →

II. either leftmost / rightmost ⇒ shelf always continous subarray ⇒ [L,R]

III. i^{th} year → $SP = i \times OP[j]$
j^{th} bottle

Find max profit?

1st year 3 1 4	2 × 1
2nd year 3 1	4 × 2
3rd year 3	1 × 3
4th year ϕ	3 × 4

(0,1,4) 1 2 3 4: 2 3 1 4
L → (1,2,4): 3 1 4
R → (1,1,3): 2 3 1
L → (2,3,4): 1 4
R → (2,2,3): 3 1

(Y, L, R)

4 — 0
3 — 1
2 — 2

DP state define → RR → space opt.

L, R → Y

state → P
state → SP
SP

DP[Y][L1|R] ≡ Y^{th} year [L,R] max profit

base ✓ ✓ ✓ ans 1,N

f(4)

```
int solve (int Y, int L, int R) {
    if (L > R) return 0; //base
    if (DP[Y][L][R] != -1) return DP[Y][L][R];
    DP[Y][L][R] = max(OP[L]*Y + solve(Y+1, L+1, R) → Left sell
                      , OP[R]*Y + solve(Y+1, L, R-1)) → Right sell
    return DP[Y][L][R];
}
```

1...N 1...N 1...N
O(N³)

1, 1, N
2, 1, N-1 2, 2, N
3, 1, N-2 3, 3, N 3, 2, N-1

int DP[300][300][300];
vector<vector<vector<int>>> DP;

```
int solve (int L, int R) {
    if (L > R) return 0;
    if (DP[L][R] != -1) return DP[L][R];
    Y = N-(R-L+1)+1;
    return DP[L][R] = max(Y*OP[L], solve(L+1,R)
                      , Y*OP[R], solve(L,R-1));
}
```

1...N 1...N
O(N²)

- Recipe
  Define DP state → RR → space optim.

- State definition
  ↳ < 12 ideas / patterns
  Tiling → ending at i → LIS
  wine → [L,R] → LCS? PR
  Dynamic Programming
  classical problem

greedy

Y	
1 × 2	= 2
2 × 3	= 6
3 × 4	= 12
4 × 1	= 4
5 × 5	= 25
	49

Y	
1 × 2	= 2
2 × 4	= 8
3 × 1	= 3
4 × 3	= 12
5 × 5	= 25
	50