Dynamic Programming

Video - 83

Note :- This playlist is only for explanation of dns & solutions.

See my "DP Concepts & dns" playlist for understanding DP from scratch...

Facebook, Instagram → codestorywithMIK

Twitter → cswithMIK

→ codestorywithMIK

Leetcode - 629 Hard

Recursion + Memo → Bottom UP → Optimal Bottom UP

629. K Inverse Pairs Array

Hard 2017 230 Add to List Share

For an integer array `nums`, an **inverse pair** is a pair of integers `[i, j]` where `0 <= i < j < nums.length` and `nums[i] > nums[j]`.

Given two integers n and k, return the number of different arrays consist of numbers from `1` to `n` such that there are exactly `k` **inverse pairs**. Since the answer can be huge, return it **modulo** `10^9 + 7`.

Example :- n = 3, K = 0

Output = 1

{1, 2, 3}, K=0

n = 3, K = 1

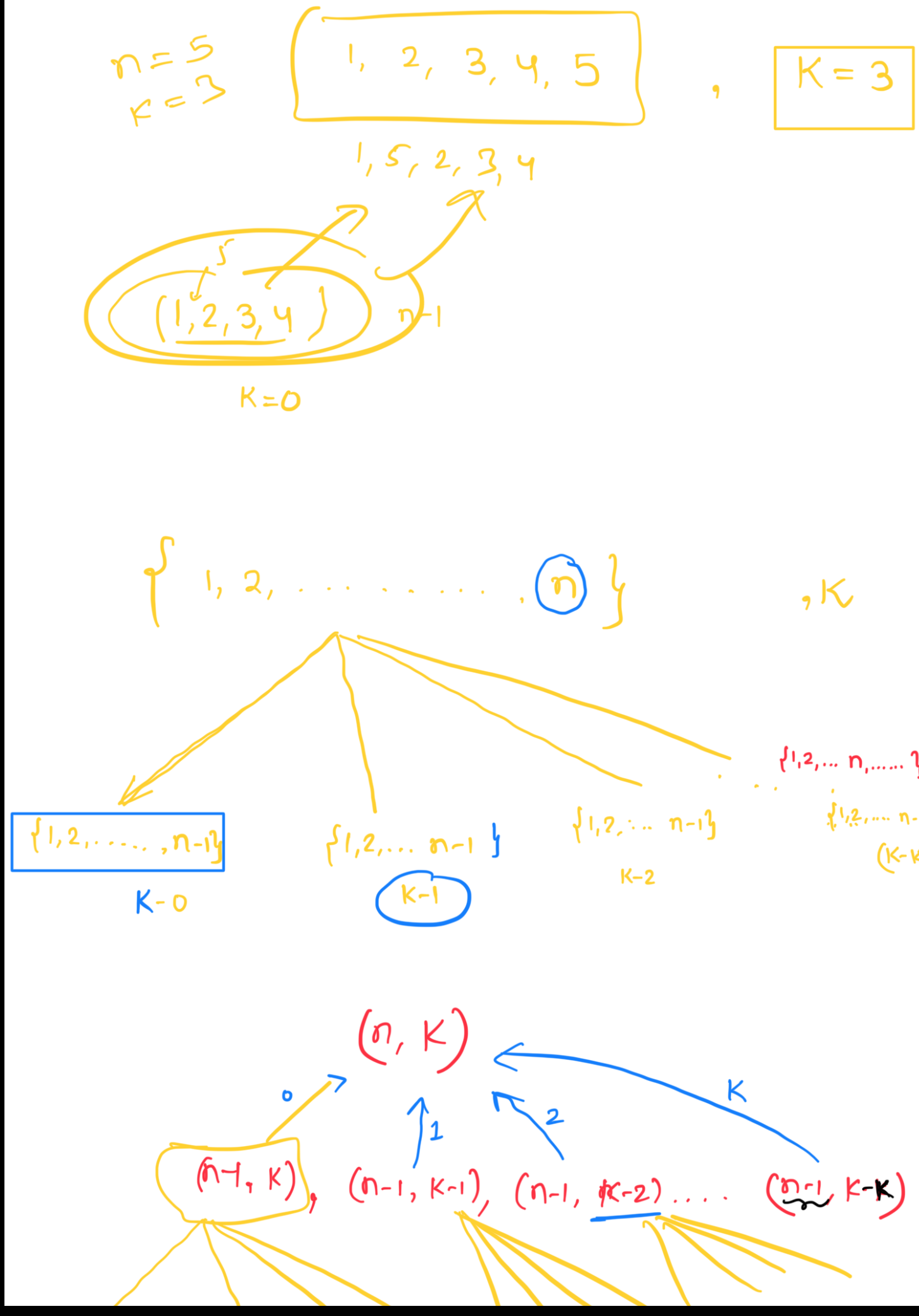
Output = 2

{1, 2, 3},

{1, 3, 2} K=1

{2, 1, 3} K=1

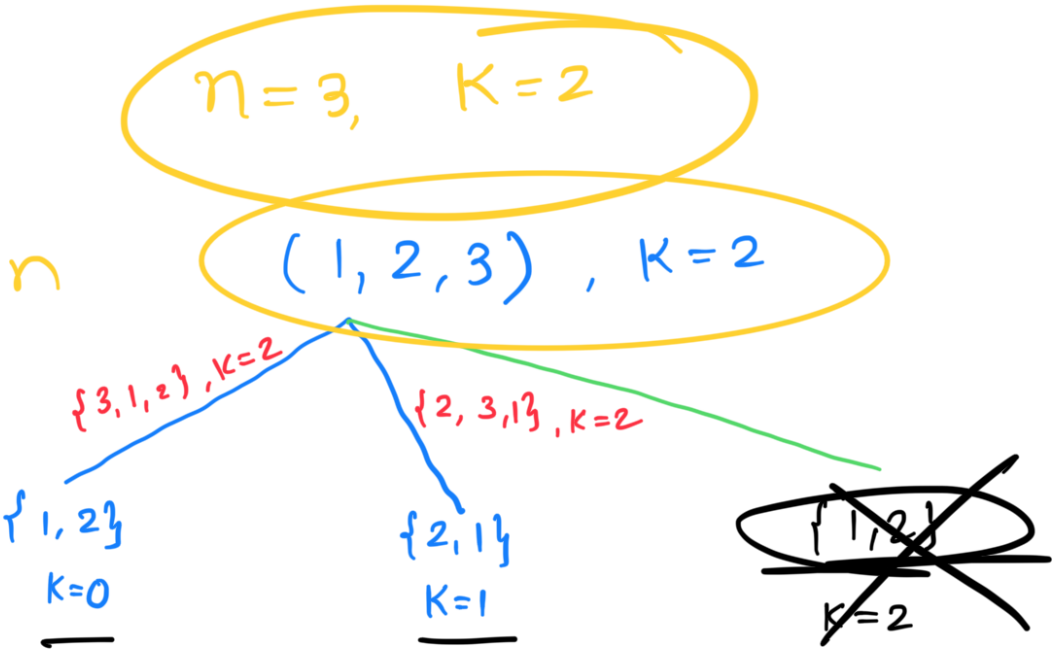
Intuition Building


n = 5
K = 3
1, 2, 3, 4, 5
K = 3
1, 5, 2, 3, 4
s
{1, 2, 3, 4}
n-1
K=0
{ 1, 2, ... n }
, K
{1, 2, ... , n-1}
K-0
{1, 2, . . . n-1 }
K-1
{1, 2, . . . n-1}
K-2
{1, 2, . . . n, ... }
{1, 2, . . . n-1}
(K-K)
(n, K)
0
1
2
K
(n-1, K)
(n-1, K-1)
(n-1, K-2) . . . .
(n-1, K-K)

{1, 2 .... n-2 } K

(n-2, K-1)

Example:-

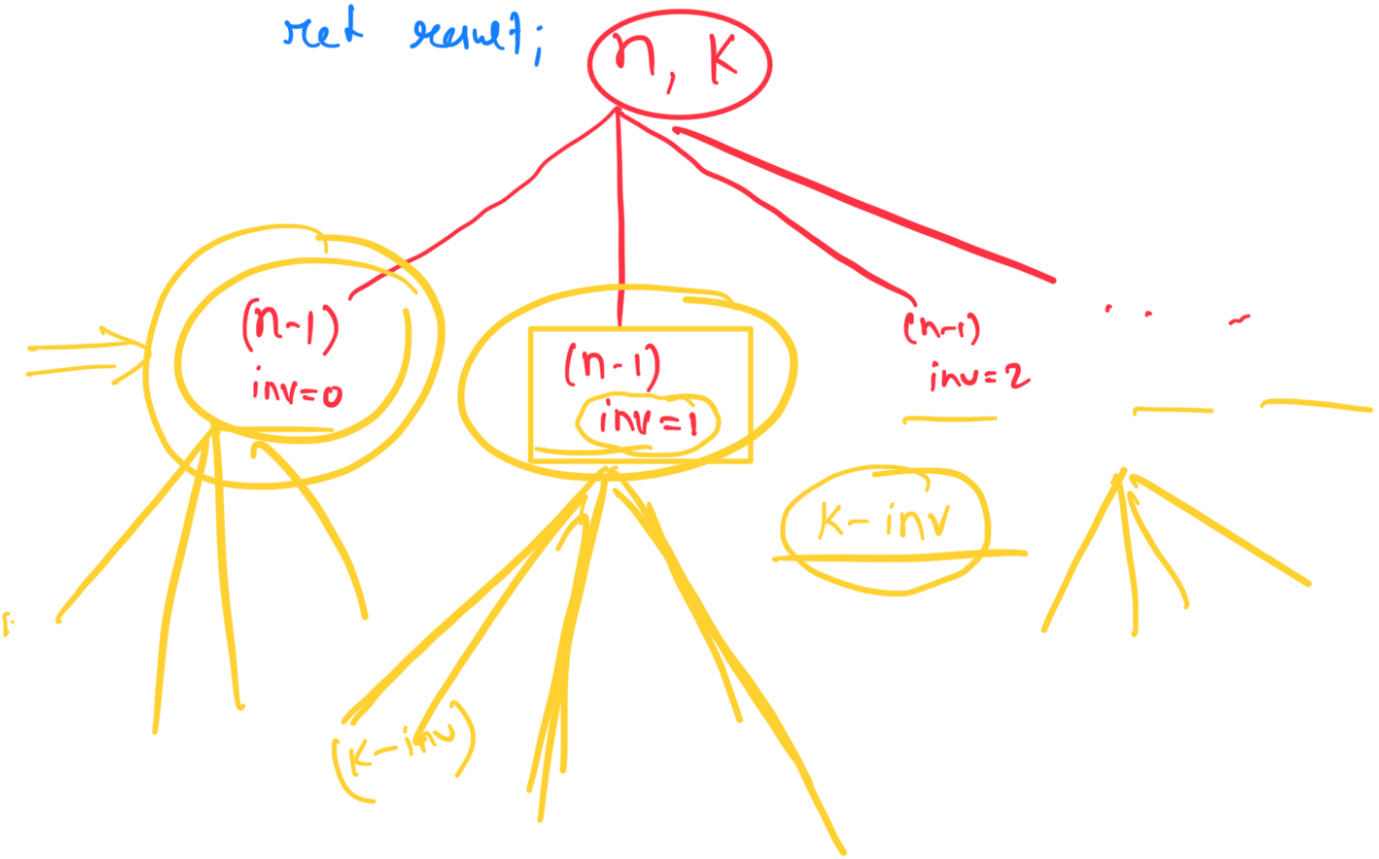



```
solve( n, K) {
    if(n==0) {
        return 0;
    }
    if(K==0) return 1;
result = 0;
for( inv = 0; inv <= min(K, n-1); inv++) {
```

result += Solve(n-1, K-inv);

}

ret result;



Time Complexity :-

① Recursion :- $O(K^n)$ ✗ TLE, space = height of R.T.

② Memoization :- $O(n * K * n)$ ☆, space = $O(n * K)$

$(n * K)$ for inv = 0; inv < 2n

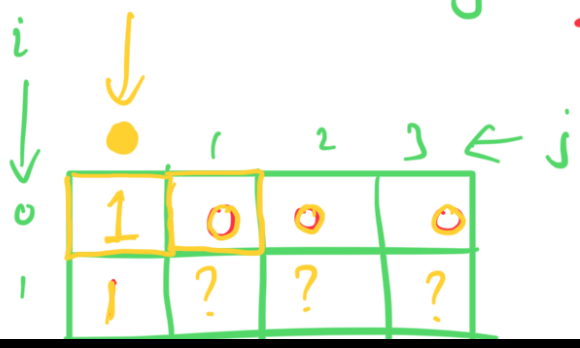
Bottom UP :-

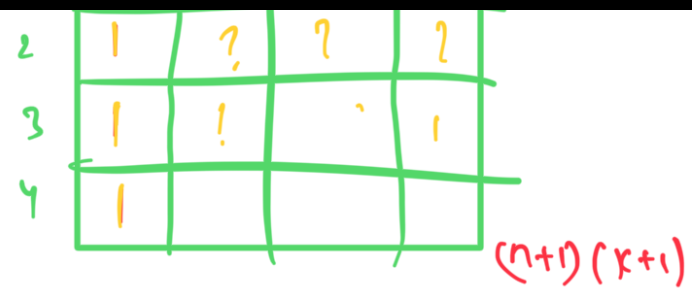
```
int solve(int n, int k) {
    if(n == 0) {
        return 0;
    }

    if(k == 0) {
        return 1; //sorted array
    }

    if(t[n][k] != -1) {
        return t[n][k];
    }

    int result = 0;
    //For an array of length n, you can have at max (n-1) inversions
    for(int inv = 0; inv <= min(k, n-1); inv++) {
        result = (result % MOD + solve(n-1, k-inv) % MOD) % MOD; //I al
need k-inv inversions
    }

    return t[n][k] = result;
}
```

// State definition

t[i][j] = Total no of arrays of size i & exactly j inversions.

return t[n][K];






```
for(i=1 ; i<=n ; i++) {
    for(j=1; j<=k ; j++) {
        for(inv=0 ; inv <= min(j, i-1) ; inv++) {
            t[i][j] = (t[i][j] + t[i-1][j-inv]);
        }
    }
}
return t[n][k];
```

T.C = O(n*k*n);

S.C = O(n*k).

Optimal

Bottom Up

n = 5

K = 5

i \ j	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	1	1	0	0	0	0
3	1	2	2	1	0	0
4	1	3	5	6	5	3
⇒ i = 5	1	4	9	15	20	22

ans = 22

j - i

5 - 5 = 0

i = 5

j = 0

```
cumSum = dt[i-1][j];
dt[i][j] = cumSum;
```

j = 5

```
cumSum += dt[i-1][j];
if (j >= i) {
    cumSum -= dt[i-1][j-i];
}
dt[i][j] = cumSum;
```