prices array ⇒ {7, 1, 5, 3, 6, 4, 11, 9} → n days

you are allowed to have k transactions

whenever we buy, no. of transaction increases

you

dp[i][k][x] → number of stocks you are holding

(0,0) (0,1) (0,2)

days

maximum profit i can make on 2nd day with k transaction and with x stocks in my hand.

dp(2)(0)(1) = -3

dp(2)(

and dp(n-1)(k)(0)

dp states:

dp(i)(k)(0) = max { rest, sell }

dp(i-1)(k)(0) → rest

(i-1)(k)(0)

dp(i)(k)(1) = max { rest, buy }

hold

buy → no. of transaction increase

dp(i-1)(k)(1)

dp(i)(k)(0) ⇒ max { dp(i-1)(k)(0), dp(i-1)(k)(1) + prices(i) }

dp(i)(k)(1) ⇒ max { dp(i-1)(k)(1), dp(i-1)(k-1)(0) - prices(i) }

dp0(i) = max { dp0(i-1), dp1(i-1) + p(i) }

dp1(i) = max (dp1(i-1), dp0(i-1) - p(i) }

base cases:

dp[-1][k][0] = 0 → not possible

dp[-1][k][1] = -∞

dp(i)(-1)(0) = 0 → not possible

dp(i)(0)(1) = -∞

	0	1
-1	0	-∞
0	0	-7
1	0	-1
2	0	-1
3	0	1
4	0	1
5	0	3

x = 0 x = 1

{7, 1, 5, 3, 6, 4}

0 1 2 3

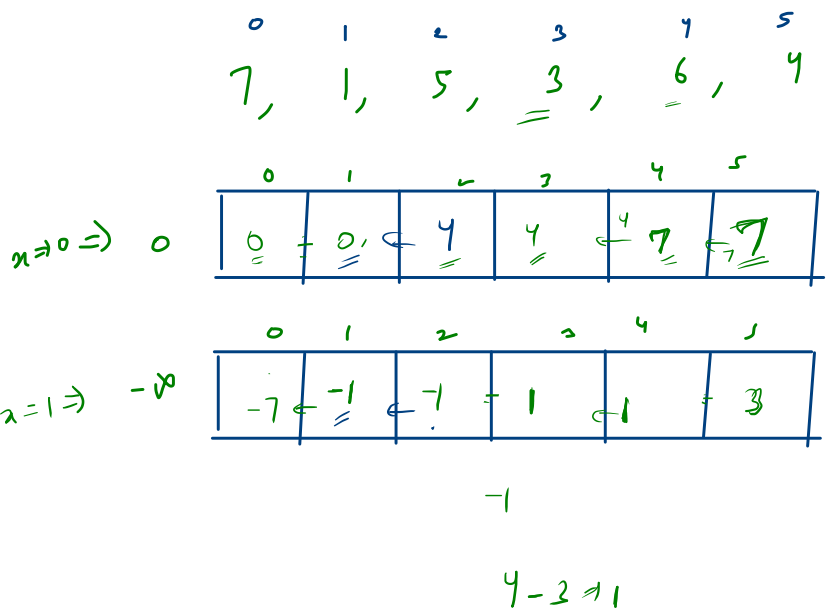
```java
public int maxProfit(int[] prices) {
    int n = prices.length;

    int[] dp0 = new int[n];
    int[] dp1 = new int[n];

    for(int i=0; i<n; i++){
        if(i==0){
            dp0[i] = 0;
            dp1[i] = - prices[i];
            continue;
        }

        dp0[i] = Math.max(dp0[i-1],dp1[i-1]+prices[i]);
        dp1[i] = Math.max(dp1[i-1],dp0[i-1]-prices[i]);
    }

    return dp0[n-1];
}
```

dp0(i) = max (dp0[i], dp1[i] + prices[i])

dp1[i] = max (dp1[i], dp0[i-1] - p(i))

prev

dp0[i] → dp1[i]




```java
public int maxProfit(int[] prices) {
    int n = prices.length;

    int[] dp0 = new int[n];
    int[] dp1 = new int[n];

    for(int i=0; i<n; i++){
        if(i==0){
            dp0[i] = 0;
            dp1[i] = - prices[i];
            continue;
        }

        dp0[i] = Math.max(dp0[i-1],dp1[i-1]+prices[i]);
        dp1[i] = Math.max(dp1[i-1],dp0[i-1]-prices[i]);
    }

    return dp0[n-1];
}
```