It's not difficult to get the DP recursive formula:

dp[k, i] = max(dp[k, i-1], prices[i] - prices[j] + dp[k-1, j-1]), j=[0..i-1]

For k transactions, on i-th day,  
if we don't trade then the profit is same as previous day dp[k, i-1];  
and if we bought the share on j-th day where j=[0..i-1], then sell the share on i-th day then the profit is prices[i] - prices[j] + dp[k-1, j-1] .  
Actually j can be i as well. When j is i, the one more extra item prices[i] - prices[j] + dp[k-1, j] = dp[k-1, i] looks like we just lose one chance of transaction.

I see someone else use the formula dp[k, i] = max(dp[k, i-1], prices[i] - prices[j] + dp[k-1, j]), where the last one is dp[k-1, j] instead of dp[k-1, j-1]. It's not the direct sense, as if the share was bought on j-th day, then the total profit of previous transactions should be done on (j-1)th day. However, the result based on that formula is also correct, because if the share was sold on j-th day and then bought again, it is the same if we didn't trade on that day.

So the straigtforward implementation is:

**public** **int** MaxProfitDp(**int**[] prices) {

if (prices.Length == 0) return 0;

var dp = **new** **int**[3, prices.Length];

for (**int** k = 1; k <= 2; k++) {

for (**int** i = 1; i < prices.Length; i++) {

**int** min = prices[0];

for (**int** j = 1; j <= i; j++)

min = Math.Min(min, prices[j] - dp[k-1, j-1]);

dp[k, i] = Math.Max(dp[k, i-1], prices[i] - min);

}

}

return dp[2, prices.Length - 1];

}

Time complexity is O(kn^2), space complexity is O(kn).  
In the above code, min is repeated calculated. It can be easily improved as:

**public** **int** MaxProfitDpCompact1(**int**[] prices) {

if (prices.Length == 0) return 0;

var dp = **new** **int**[3, prices.Length];

for (**int** k = 1; k <= 2; k++) {

**int** min = prices[0];

for (**int** i = 1; i < prices.Length; i++) {

min = Math.Min(min, prices[i] - dp[k-1, i-1]);

dp[k, i] = Math.Max(dp[k, i-1], prices[i] - min);

}

}

return dp[2, prices.Length - 1];

}

Time complexity is O(kn), space complexity is O(kn).  
If we slight swap the two 'for' loops:

**public** int MaxProfitDpCompact1T(int[] prices) {

**if** (prices.Length == 0) **return** 0;

var dp = **new** int[3, prices.Length];

var min = **new** int[3];

**Array**.Fill(min, prices[0]);

**for** (int i = 1; i < prices.Length; i++) {

**for** (int k = 1; k <= 2; k++) {

min[k] = Math.Min(min[k], prices[i] - dp[k-1, i-1]);

dp[k, i] = Math.Max(dp[k, i-1], prices[i] - min[k]);

}

}

**return** dp[2, prices.Length - 1];

}

We need to save min for each transaction, so there are k 'min'.  
We can find the second dimension (variable i) is only dependent on the previous one (i-1), so we can compact this dimension. (We can choose the first dimension (variable k) as well since it is also only dependent on its previous one k-1, but can't compact both.)

**public** int MaxProfitDpCompact2(int[] prices) {

**if** (prices.Length == 0) **return** 0;

var dp = **new** int[3];

var min = **new** int[3];

**Array**.Fill(min, prices[0]);

**for** (int i = 1; i < prices.Length; i++) {

**for** (int k = 1; k <= 2; k++) {

min[k] = Math.Min(min[k], prices[i] - dp[k-1]);

dp[k] = Math.Max(dp[k], prices[i] - min[k]);

}

}

**return** dp[2];

}

So time complexity is O(kn), space complexity becomes O(k).  
In this case, K is 2. We can expand the array to all named variables:

public int MaxProfitDpCompactFinal(int[] prices) {

int **buy1** = int.MaxValue, **buy2** = int.MaxValue;

int **sell1** = 0, **sell2** = 0;

for (int i = 0; i < prices.Length; i++) {

**buy1** = Math.Min(**buy1,** prices[i]);

**sell1** = Math.Max(**sell1,** prices[i] - **buy1);**

**buy2** = Math.Min(**buy2,** prices[i] - **sell1);**

**sell2** = Math.Max(**sell2,** prices[i] - **buy2);**

}

return **sell2;**

}

We can also explain the above codes in other words. On every day, we buy the share with the price as low as we can, and sell the share with price as high as we can. For the second transaction, we integrate the profit of first transaction into the cost of the second buy, then the profit of the second sell will be the total profit of two transactions.