

# Stock Buy & Sell - Max K Transactions Allowed :-

**Stock Buy and Sell - Max K Transactions Allowed**

Difficulty: Hard Accuracy: 68.35% Submissions: 388+ Points: 8

In the stock market, a person buys a stock and sells it on some future date. You are given an array `prices[]` representing stock prices on different days and a positive integer `k`, find out the **maximum** profit a person can make in at-most `k` transactions.

A transaction consists of buying and subsequently selling a stock and new transaction can start only when the previous transaction has been completed.

**Examples:**

Input: `prices[] = [10, 22, 5, 80]`, `k = 2`  
 Output: 87  
 Explanation:  
 1st transaction: Buy at 10 and sell at 22.  
 2nd transaction: Buy at 5 and sell at 80.  
 Total Profit will be  $12 + 75 = 87$ .

Input: `prices[] = [20, 580, 420, 900]`, `k = 3`  
 Output: 1040  
 Explanation:  
 1st transaction: Buy at 20 and sell at 580.  
 2nd transaction: Buy at 420 and sell at 900.  
 Total Profit will be  $560 + 480 = 1040$ .

Input: `prices[] = [100, 90, 80, 50, 25]`, `k = 1`  
 Output: 0  
 Explanation: Selling price is decreasing continuously leading to loss. So seller cannot have any profit.

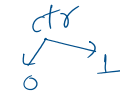
**Constraints:**

- $1 \leq \text{prices.size()} \leq 10^3$
- $1 \leq k \leq 200$
- $1 \leq \text{prices}[i] \leq 10^3$

Brute Force :-

$p = [10, 22, 5, 80], k = 2$

$(idx, k, ctr, profit)$



$\neq (ctr = 0)$   
 $\rightarrow$  buy  
 $\rightarrow$  not buy

else  
 $\rightarrow$  sell  
 $\rightarrow$  not sell

10, 22, 5, 80

int a=0, b=0;

if (ctr == 0)  
 $\rightarrow$  buy  $(idx+1, 1, k) - p[idx] = a$   
 $\rightarrow$  not buy  $(idx+1, 0, k) = b$

else  
 $\rightarrow$  sell  $(idx+1, 0, k-1) + p[idx] = a$   
 $\rightarrow$  not sell  $(idx+1, 1, k) = b$

return max(a, b)

Base Case :-

if ( $k == 0$  ||  $idx == n$ )  
 $\rightarrow$  return 0

Solution Code :-

```

1 // C++ Solution
2
3 class Solution {
4 public:
5     int maxProfit(vector<int> prices, int k) {
6         // code here
7         int n = prices.size();
8         return help(prices, 0, 0, k);
9     }
10
11     int help(vector<int> prices, int idx, int ctr, int k) {
12         if (idx == n || k == 0) return 0;
13         if (ctr == 0) {
14             int a = 0;
15             if (idx < n) a = help(prices, idx+1, 1, k) - prices[idx];
16             int b = 0;
17             b = help(prices, idx+1, 0, k);
18             return max(a, b);
19         }
20         int a = 0;
21         a = help(prices, idx+1, 0, k-1) + prices[idx];
22         int b = 0;
23         if (idx < n) b = help(prices, idx+1, 1, k);
24         return max(a, b);
25     }
26 };
  
```

T.C :  $O(2^n)$   
 S.C :  $O(n)$

Optimized Approach :-

We are just going to memoize our solution to each subproblem.

```

C++ g++ 5.4
1 // Driver Code Ends
7
8 class Solution {
9 public:
10     int maxProfit(vector<int>& prices, int k) {
11         // code here
12         int n=prices.size();
13         vector<vector<int>>> dp(n, vector<vector<int>>>(k+1, vector<int>(2, -1)));
14         return help(prices, 0, 0, k, dp);
15     }
16     int help(vector<int>& prices, int idx, int ctr, int k, vector<vector<vector<int>>>& dp){
17         if(idx==n || k==0) return 0;
18         if(dp[idx][k][ctr]!=-1) return dp[idx][k][ctr];
19         if(ctr==0){
20             int a=0;
21             if(idx==n-1) a=help(prices, idx+1, 1, k, dp)-prices[idx];
22             int b=0;
23             b=help(prices, idx+1, 0, k, dp);
24             return dp[idx][k][ctr]=max(a, b);
25         }
26         int a=0;
27         a=help(prices, idx+1, 0, k-1, dp)-prices[idx];
28         int b=0;
29         b=(idx==n-1)?help(prices, idx+1, 1, k, dp);
30         return dp[idx][k][ctr]=max(a, b);
31     }
32 };
33
34
35 // Driver Code Ends

```

T.C Analysis :-  $O(n \times k)$

S.C :-  $O(n \times k)$