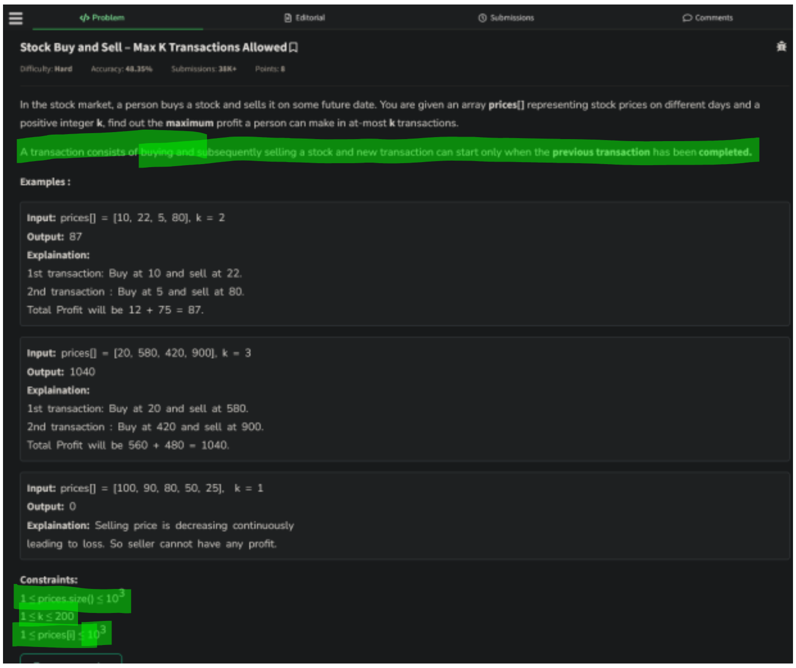
Stock Buy & Sell - Max K Transactions Allowed :-



</> Problem   Editorial   Submissions   Comments

Stock Buy and Sell – Max K Transactions Allowed

Difficulty: **Hard** Accuracy: **48.35%** Submissions: **38K+** 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
Explaination:
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
Explaination:
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
Explaination: Selling price is decreasing continuously leading to loss. So seller cannot have any profit.

Constraints:
$1 \le prices.size() \le 10^3$
$1 \le k \le 200$
$1 \le prices[i] \le 10^3$

Brute Force :-

p=[10, 22, 5, 80], k=2
↑ ↑ ↑ ↑

(idx, k, ctr, profit)

$$\begin{tikzcd} & ctr \arrow[dl] \arrow[dr] & \\ 0 & & 1 \end{tikzcd}$$

```
if (ctr==0)
    ↳ buy
    ↳ not buy
else
    ↳ sell
    ↳ not sell
```

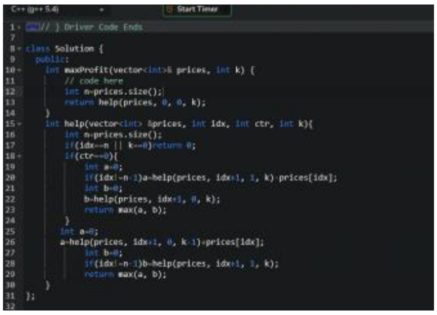
10, 22, 5, 80

```
int a=0, b=0;
if (ctr==0)
    ↳ buy (idx+1, 1, k) - p(idx) = a
    ↳ not buy (idx+1, 0, k) = b
else
    ↳ sell (idx+1, 0, k-1) + p(idx) = a
    ↳ not sell (idx+1, 1, k) = b
return max(a, b)
```

Base Case :-

```
if (k==0 || idx==n)
    ↳ return 0
```

Solution Code :-



C++ (g++ 5.4)   Start Timer

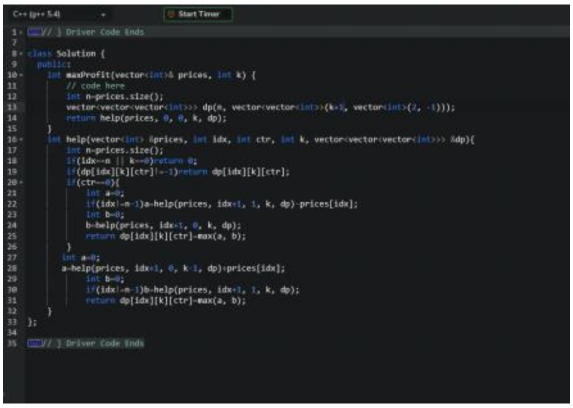
```cpp
// } Driver Code Ends

class Solution {
  public:
    int maxProfit(vector<int>& prices, int k) {
        // code here
        int n=prices.size();
        return help(prices, 0, 0, k);
    }
    int help(vector<int> &prices, int idx, int ctr, int k){
        int n=prices.size();
        if(idx==n || k==0)return 0;
        if(ctr==0){
            int a=0;
            if(idx!=n-1)a=help(prices, idx+1, 1, k)-prices[idx];
            int b=0;
            b=help(prices, idx+1, 0, k);
            return max(a, b);
        }
       int a=0;
       a=help(prices, idx+1, 0, k-1)+prices[idx];
            int b=0;
            if(idx!=n-1)b=help(prices, idx+1, 1, k);
            return max(a, b);
    }
};
```

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

Optimized Approach :-

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



```cpp
// } Driver Code Ends

class Solution {
  public:
    int maxProfit(vector<int>& prices, int k) {
        // code here
        int n=prices.size();
        vector<vector<vector<int>>> dp(n, vector<vector<int>>(k+1, vector<int>(2, -1)));
        return help(prices, 0, 0, k, dp);
    }
    int help(vector<int> &prices, int idx, int ctr, int k, vector<vector<vector<int>>> &dp){
        int n=prices.size();
        if(idx==n || k==0)return 0;
        if(dp[idx][k][ctr]!=-1)return dp[idx][k][ctr];
        if(ctr==0){
            int a=0;
            if(idx!=n-1)a=help(prices, idx+1, 1, k, dp)-prices[idx];
            int b=0;
            b=help(prices, idx+1, 0, k, dp);
            return dp[idx][k][ctr]=max(a, b);
        }
       int a=0;
       a=help(prices, idx+1, 0, k-1, dp)+prices[idx];
            int b=0;
            if(idx!=n-1)b=help(prices, idx+1, 1, k, dp);
            return dp[idx][k][ctr]=max(a, b);
    }
};

// } Driver Code Ends
```

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

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

(Better Approach) Using Bottom Up DP - O$(n \times k)$ time & O$(n \times k)$ space :-

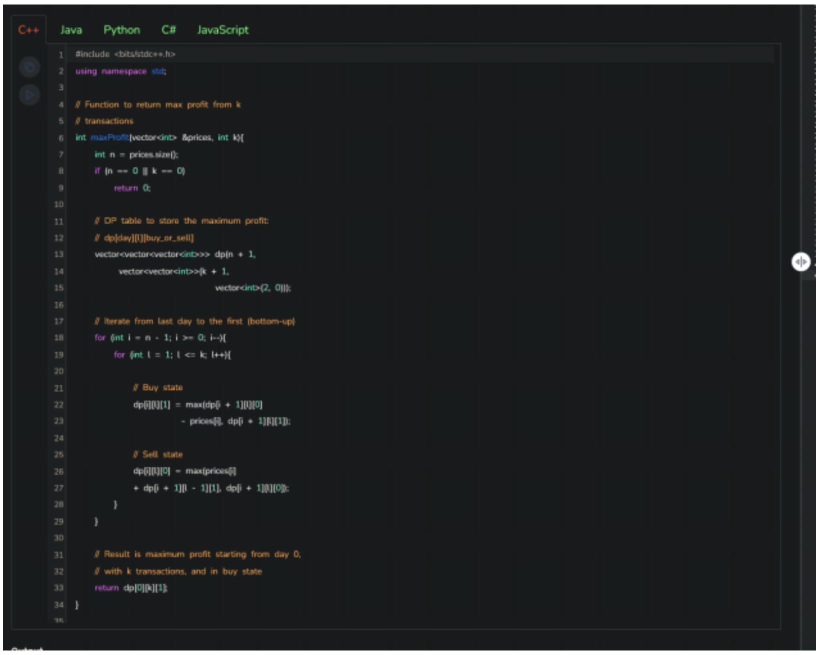
Let dp[i][l][buy] represent the maximum profit achievable starting from day i, with l transactions remaining, & a state indicating whether we can buy or sell.

- buy = 1 → we are allowed to buy

dp[i][l][1] = max (dp[i+1][l][0] - prices[i], dp[i+1][l][1])

- buy = 0 → we are allowed to sell

dp[i][l][0] = max (prices[i] + dp[i+1][l - 1][1], dp[i+1][l][0])



```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to return max profit from k
// transactions
int maxProfit(vector<int> &prices, int k){
    int n = prices.size();
    if (n == 0 || k == 0)
        return 0;

    // DP table to store the maximum profit:
    // dp[day][l][buy_or_sell]
    vector<vector<vector<int>>> dp(n + 1,
        vector<vector<int>>(k + 1,
                            vector<int>(2, 0)));

    // Iterate from last day to the first (bottom-up)
    for (int i = n - 1; i >= 0; i--){
        for (int l = 1; l <= k; l++){

            // Buy state
            dp[i][l][1] = max(dp[i + 1][l][0]
                    - prices[i], dp[i + 1][l][1]);

            // Sell state
            dp[i][l][0] = max(prices[i]
            + dp[i + 1][l - 1][1], dp[i + 1][l][0]);
        }
    }

    // Result is maximum profit starting from day 0,
    // with k transactions, and in buy state
    return dp[0][k][1];
}
```