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

Brute Force :-

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

(idx, k, ctr, profit)

ctr → 1, 0

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
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         return help(prices, 0, 0, k);
14     }
15     int help(vector<int> &prices, int idx, int ctr, int k){
16         int n=prices.size();
17         if(idx==n || k==0)return 0;
18         if(ctr==0){
19             int a=0;
20             if(idx!=n-1)a=help(prices, idx+1, 1, k)-prices[idx];
21             int b=0;
22             b=help(prices, idx+1, 0, k);
23             return max(a, b);
24         }
25         int a=0;
26         a=help(prices, idx+1, 0, k-1)+prices[idx];
27             int b=0;
28             if(idx!=n-1)b=help(prices, idx+1, 1, k);
29             return max(a, b);
30     }
31 };
32
```

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) Start Timer

```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×k)

S.C. :- O(n×k)