ASSIGNMENT – 1
Advanced Programming Lab

Name: Utsav Balar
Roll.no. T24cs003

Problem Statement:

In the realm of hnancial trading, a trader has access to a series of stock prices over N cOnsecutive days. The trader is permitted to perform up to k transac tious, where each transaction cousists of a single buy followed by a single sell. liportantly, a new transaction can only be initiated after the completion of the previous One.
The goal is to determine the maximum possible profit the trader can achieve given the constraints on the number of transactions and the requirement that ench buy must precede its corresponding scll.
Inputs

Stock Trading Problem

- An integer n representing the number of days for which stock prices are available.
- An integer k indicating the maximum number of transactions allowed.
- An array of integers prices:] where pricesli] denotes the stock price on day i(0<i<n).

Objective
Calculate the maxinmum profit that can be attained by strategically executing up to k buy-sell transactions. Each transaction must start with a buy on one day and end with a sell on a subsequent day.
Constraints:

l<n< 105
l<k< 100
0< pricesli] < 104

Solution

main.rs

```
// use rand::{thread_rng, Rng};

/// # Stock Trading Problem
///
/// Trader has series of stock prices over 'n' consecutive days.
/// Trader is permitted to perform k transactions.
///
/// A transaction is a single buy followed by a single sell.
///
```

```rust
/// A new transaction can only be initiated after the completion of
previous one.
///
/// GOAL: Determine the max possible profit after the trader can
achieve given
/// the contraints on the number of transactions and the
requirement that each
/// buy must precede its corresponding sell.
///
/// INPUTS:
/// - Int 'n' number of days
/// - Int 'k' max number of transactions
/// - Array of prices where prices[i] is the stock price on day i
(0 ⩽ i < n)
///
/// OBJECTIVE:
/// Calculate max profit that can be attained by strategically
executing up
/// to 'k' buy-sell transactions.
///
/// Each transaction must start with a buy on the day and end with
a sell on a
/// subsequent days.
///
/// CONTRAINTS:
/// 1. 1 ⩽ n ⩽ 10^5
/// 2. 1 ⩽ k ⩽ 100
/// 3. 0 ⩽ prices[i] ⩽ 10^4

fn maximum_profit(k: usize, prices: Vec<isize>) → isize {
    let n = prices.len();

    // If k is 0 no transaction can be performed
    // If n is 0 no profit can be made
    if n == 0 || k == 0 {
        return 0;
    }

    // If k is larger than n/2, we can perform transactions every
day.
    if k ⩾ n / 2 {
        let mut max_profit = 0;
        (1..n).for_each(|i| max_profit += (prices[i] - prices[i -
1]).max(0));
        return max_profit;
    }
```

```
    // DP table to store the maximum profit on day i with at most j
transactions
    let mut dp = vec![vec![0; k + 1]; n];

    for t in 1..=k {
        let mut max_price_diff: isize = -prices[0];
        for d in 1..n {
            // println!(
            //     "On day {d} with transaction {t} price on day
{}, price on previous day {}",
            //     prices[d],
            //     prices[d - 1]
            // );
            dp[d][t] = dp[d - 1][t].max(prices[d] +
max_price_diff);
            // println!(
            //     "dp[{d}][{t}] is the max of {} and {}",
            //     dp[d - 1][t],
            //     prices[d] + max_price_diff
            // );
            max_price_diff = max_price_diff.max(dp[d - 1][t - 1] as
isize - prices[d]);
            // println!("max_price_diff is set to {}",
max_price_diff);
        }
    }

    dp[n - 1][k]
}

fn main() {
    // let mut rng = thread_rng();
    // let k: usize = rng.gen_range(0..100);

    // let prices_capacity = rng.gen_range(0..10_000);
    // let mut prices = Vec::with_capacity(prices_capacity);
    // (0..prices_capacity).for_each(|_|
prices.push(rng.gen_range(0..10_000)));

    let k = 3;
    let prices: Vec<isize> = Vec::from([30, 40, 10, 45, 20, 15]);
    // let prices = vec![11, 85, 67, 33, 45, 12, 8, 77, 11];
    // let prices = vec![2, 4, 1];

    println!("k: {k}");
```

```
    println!("prices: {prices:?}");
    println!("max profit: {}", maximum_profit(k, prices));
}
```

Output: