|  |  |
| --- | --- |
| **Best time to buy and sell stocks in C++** | |
| #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  class BestTimeToBuyAndSellStock {  public:      int maxProfit(vector<int>& prices) {          if (prices.empty()) return 0;          int maxP = 0;          int minBP = prices[0];          for (int prc : prices) {              int tp = prc - minBP;              if (tp > maxP) {                  maxP = tp;              }              minBP = min(minBP, prc);          }          return maxP;      }  };  int main() {      BestTimeToBuyAndSellStock solution;      // Test case 1      vector<int> prices1 = {7, 1, 5, 3, 6, 4};      int maxProfit1 = solution.maxProfit(prices1);      cout << "Max profit for prices1: " << maxProfit1 << endl; // Output: 5      return 0;  } | **Dry Run of the BestTimeToBuyAndSellStock Problem:**  **Input:**  prices1 = {7, 1, 5, 3, 6, 4};  We need to determine the maximum profit that can be made by buying and then selling a stock. The goal is to find the lowest price to buy and the highest price to sell after buying, and calculate the profit.  **Step 1: Initialize the Variables**   1. maxP = 0: This variable stores the maximum profit found so far. 2. minBP = prices[0] = 7: This variable stores the minimum price found so far (the price at the beginning of the array).   **Step 2: Iterate Through the Prices**  **Iteration 1 (Price = 7):**   * Calculate potential profit: tp = 7 - 7 = 0. * maxP is updated to max(0, 0) = 0 (no change). * minBP remains 7 since min(7, 7) = 7.   **Iteration 2 (Price = 1):**   * Calculate potential profit: tp = 1 - 7 = -6. * maxP remains 0 (no update). * Update minBP to 1 because min(7, 1) = 1.   **Iteration 3 (Price = 5):**   * Calculate potential profit: tp = 5 - 1 = 4. * maxP is updated to max(0, 4) = 4. * minBP remains 1 (no change).   **Iteration 4 (Price = 3):**   * Calculate potential profit: tp = 3 - 1 = 2. * maxP remains 4 (no update). * minBP remains 1 (no change).   **Iteration 5 (Price = 6):**   * Calculate potential profit: tp = 6 - 1 = 5. * maxP is updated to max(4, 5) = 5. * minBP remains 1 (no change).   **Iteration 6 (Price = 4):**   * Calculate potential profit: tp = 4 - 1 = 3. * maxP remains 5 (no update). * minBP remains 1 (no change).   **Step 3: Return the Maximum Profit**  After iterating through all the prices, the maxP variable holds the maximum profit, which is 5. |
| **Output:-** maxP = 5 (Maximum profit) | |