Maximizing Profit from Stocks

Your algorithms have become so good at predicting the market that you now know what the share price of Silly Purple Toothpicks Inc. (SPT) will be for a number of minutes going forward. Each minute, your high frequency trading platform allows you to either buy one share of SPT, sell any number of shares of SPT that you own, or not make any transaction at all. Find the maximum profit you can obtain with an optimal trading strategy.

Purchases are negative and sales are positive cash flows in the following equations. For example, if predicted prices over the next *n = 6* minutes are *prices = [3, 4, 5, 3, 5, 2]*, one way to the best outcome is to purchase a share in each of the first *2* minutes for cash flows *-3 + -4 = -7*, then sell them at the third minute for *2 \* 5 = 10*. Purchase a share in the *4th* minute for *3* and sell it in the *5th* minute for *5*. Total profits are *-3 - 4 + 10 - 3 + 5 = 5*. Another way to the same outcome is to purchase a share in each of the *1st*, *2nd* and *4th* minutes for -*3 - 4 - 3 = -10*, do nothing at minute *2* then sell all three shares at *5* (total *3 \* 5 = 15*) on the *5th* minute, again for a total profit of *-10 + 15 = 5*. There is no reason to purchase in the last minute as there is no time to sell.

****Function Description****

Complete the *maximumProfit* function in the editor below. The function must return a long integer that denotes the maximum possible profit.

maximumProfit has the following parameter:

*price*: an array of *n* integers that denote the stock prices at minutes *1* through *n*.

****Constraints****

* *1 ≤ t ≤ 10*
* *1 ≤ n ≤ 5x105*
* *1* *≤ price[i] ≤* *105*

Input Format For Custom Testing

The first line contains the number of test cases *t*.

Each of the next *t* pairs of lines is as follows:

    The first line of each test case contains a number *n*, the number of predicted prices for SPT stock.

    The next line contains *n* space-separated integers that denote the predicted price of SPT shares for the next *n* minutes.

Sample Case 0

**Sample Input For Custom Testing**

3

3

5 3 2

3

1 2 100

4

1 3 1 2

****Sample Output 0****

0

197

3

****Explanation 0****

For the first case, no profit can be had because the share price never increases, so do nothing.

For the second case, buy one share in each of the first two minutes, then sell both shares in the third minute.

For the third case, buy one share in the first minute, sell one in the second minute, buy one share in the third minute, and sell one share in fourth minute to get a total profit of 3.

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'maximumProfit' function below.

\*

\* The function is expected to return a LONG\_INTEGER.

\* The function accepts INTEGER\_ARRAY price as parameter.

\*/

public static long maximumProfit(List<Integer> price) {

// Write your code here

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int t = Integer.parseInt(bufferedReader.readLine().trim());

IntStream.range(0, t).forEach(tItr -> {

try {

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> price = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

long profit = Result.maximumProfit(price);

bufferedWriter.write(String.valueOf(profit));

bufferedWriter.newLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

});

bufferedReader.close();

bufferedWriter.close();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*