Dear reader, welcome to the article on the problem named **‘**[**Buy & Sell Stock - Infinite Transactions Allowed**](https://www.pepcoding.com/resources/online-java-foundation/dynamic-programming-and-greedy/buy-and-sell-stocks-ita-official/ojquestion)**’.** It is the 2nd problem in the series Buy & Sell Stock.

***Problem Statement:***

* You are given an array of n elements. Element at index i represents a stock value on ith day.
* You need to print the maximum profit you can make if you are allowed ***infinite transactions***.

Note - There can be ***no overlapping transactions***. One transaction needs to be closed (a buy followed by a sell) before opening another transaction (another buy). We cannot first buy 2 stocks and then sell them, i.e. states like ‘BBSS’ are not allowed.

Nobody will laugh at you, if you didn't know this! It is very trivial, but still, I want to let you know that you cannot sell a stock before buying it! You cannot have a state “SB”.

***Example***:

Consider the array of days as : *[7,1,5,3,6,4]* *.* The maximum profit can be achieved by (days are 0 based indexed):

- Buying at day 1 at price 1

- Selling at day 2 at price 5

- Buying at day 3 at price 3

- Selling at day 4 at price 6

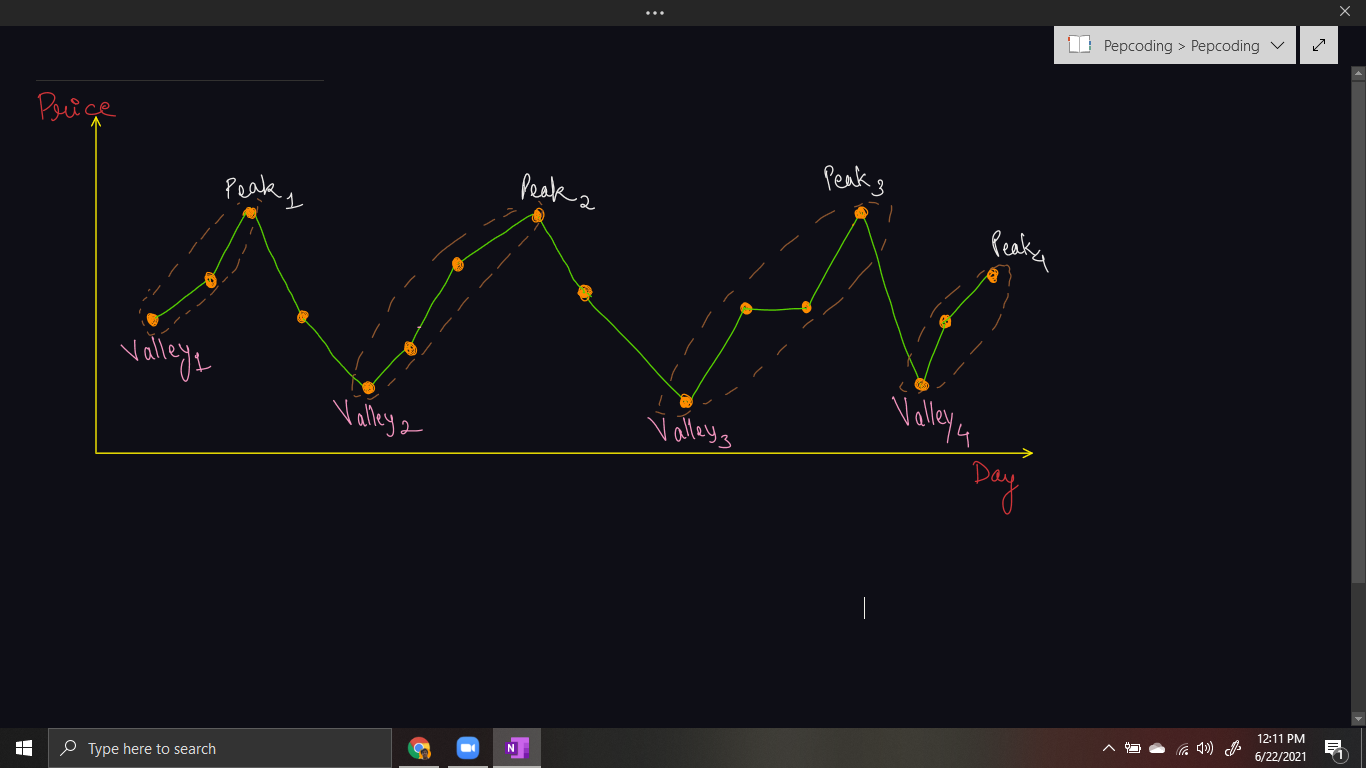
The total profit is ((5 - 1) + (6 - 3)) = 7.

***Solution: O(N) Time & O(1) Space***

Since, we can make any number of transactions, hence we need to find all the buying - selling point pairs, such that their cumulative profit is maximum possible.

***WHAT:***

Let us try to plot the array on a price vs day graph, and analyze how we can extract maximum profit:



The only points of interest are the ***peaks*** and the ***valleys.***

* The **valleys are the potential buying points**.
* The **peaks are the potential selling points**.

Hence, we can pair all the corresponding valleys with peaks and the difference between their values will be the profit. We will add all such pairs differences to our result.

***WHY:***

**Q)** Why cannot we take dips (peak to valley points) into consideration?

**R)** If you will not take peak to valley point, then we are buying for a higher price and selling for a lower price, hence making a loss. We cannot bear such ***losses,*** when our career is concerned.

Also, we cannot make a pair of any point in the dip with the peak, because the profit of peak minus this dip point will be less than the profit of peak minus valley point.

The ***key point*** is we need to consider every peak immediately following a valley to maximize the profit. In case we ***skip one of the peaks*** (trying to obtain more profit), we will end up losing the profit over one of the transactions leading to an overall ***lesser profit***.

***HOW:***

How can we make pairs of peaks and valleys?

* We will start with the first element as both the potential peak and the potential valley.
* While going ***upwards***, i.e. up the hill, we will update the peak element as the current element. (Note, that we will not update the valley element, but only the peak element).
* Whenever we find a dip, i.e. we have travelled ***downwards*** or down the hill, it means the previous point was the peak, hence we will ***add the difference*** of peak (previous element) and valley to our answer.
* After adding the answer, we will make the current element (dip) as the ***potential peak and the potential valley***.
* We will continue with our algorithm until the last element of the array.
* But, there is one ***corner case***. When the last element is not a dip, i.e. it is a peak element, then we have not added the corresponding profit to our result.
* Hence, after the loop terminates, we will add the difference peak - valley, to take into account the last pair of peak and valley.

***Implementation (Java)***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

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

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

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

for (int i = 0; i < arr.length; i++) {

arr[i] = Integer.parseInt(br.readLine());

}

int bon = 0;

int son = 0;

int op = 0;

for(int i = 1; i < arr.length; i++){

if(arr[i] < arr[i - 1]){

op += arr[son] - arr[bon];

bon = son = i;

} else {

son++;

}

}

op += arr[son] - arr[bon];

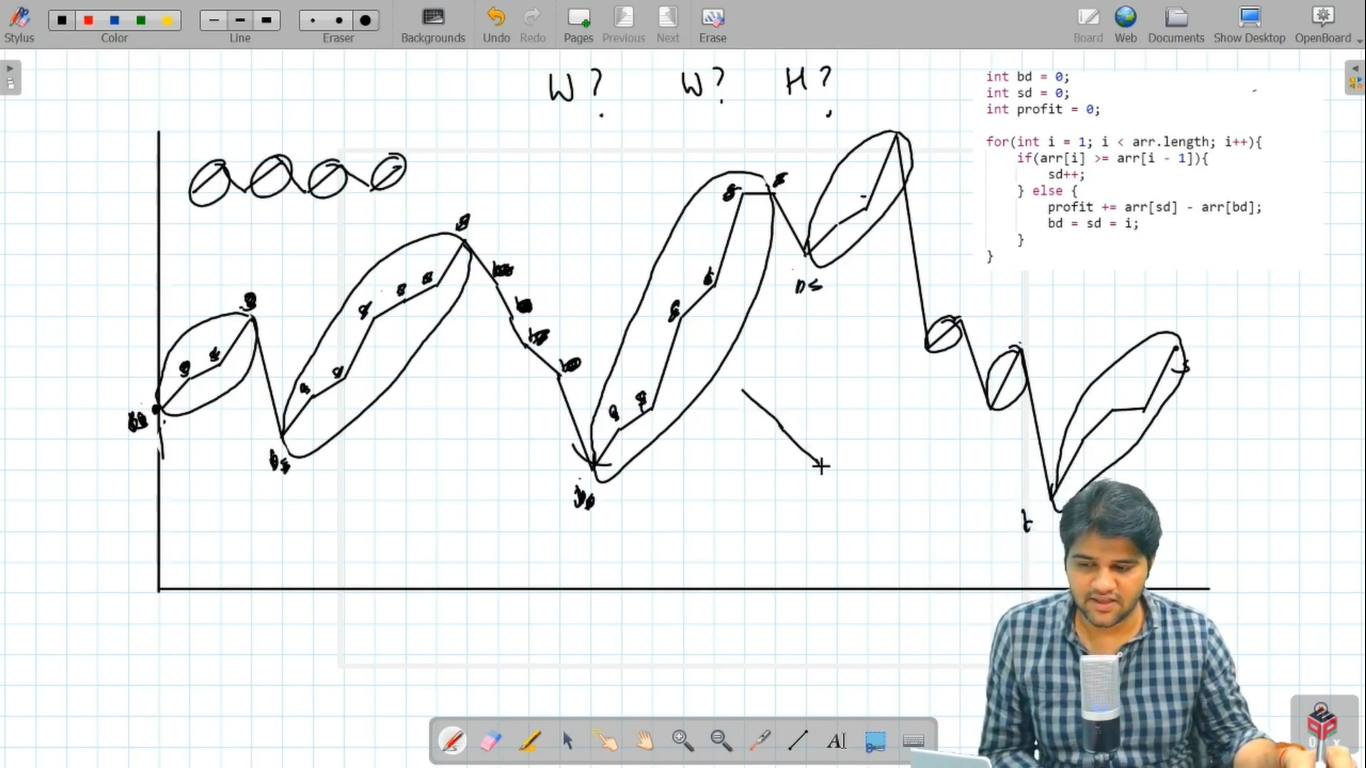
System.out.println(op);

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=HWJ9kIPpzXs) from timestamp *[10:00 , 12:25]*.

***Dry Run*** of the code using an example is taken and elaborated in the same video from *[12:25, 14:45]*.



* What is the ***time complexity*** of the above code?

Since, we are traversing through the array of n integers (representing days), and doing constant work inside the loop, hence time complexity will be O(N).

* What is the ***space complexity*** of the above code?

We have used few integer variables, hence constant ***O(1)*** auxiliary space is required.

***Follow Up:***

**Q)** If someone asks you to print the state in which you get the maximum profit, i.e. the sequence of the days in which stocks were bought and sold, then will you be able to adapt according to the problem requirements?

**R)** Well, I know you can handle such variations. Whenever you add the difference of peak - valley, add this pair to a resultant string as well. Finally, the string will contain all the days on which stocks are bought or sold.

***Asked in Companies***: *Accolite, Amazon, Directi, Flipkart, Goldman Sachs, Hike, MakeMyTrip, Microsoft, Morgan Stanley, Ola Cabs, Oracle, Paytm, Pubmatic, Quikr, Samsung, SAP Labs, Sapient, Swiggy, Walmart*

Hope that you liked the article on *‘Buy And Sell Stock - Infinite Transactions Allowed’*. Next problem will be the 3rd problem in this series [*‘Buy And Sell Stock - With Transaction Fee - Infinite Transactions’*](https://www.pepcoding.com/resources/online-java-foundation/dynamic-programming-and-greedy/buy-sell-stocks-transaction-fee-ita-official/ojquestion).

Subscribe to Pepcoding’s youtube channel for more such amazing content on Data Structures & Algorithms and follow the resources available for all students in the resources section of Pepcoding’s website.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by:

Archit Aggarwal