Dear reader, welcome to the article on the problem named **‘**[**Buy And Sell Stocks With Transaction Fee - Infinite Transactions Allowed**](https://www.pepcoding.com/resources/online-java-foundation/dynamic-programming-and-greedy/buy-sell-stocks-transaction-fee-ita-official/ojquestion)**’.** It is the 3rd 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 are also given a number ‘*fee*’ representing the ***transaction fee*** for every transaction.
* You need to print the maximum profit you can make if you are allowed ***infinite transactions***, but have to pay a "fee" for every complete transaction.

Note - There can be ***no overlapping transaction***. 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 : *[1,3,2,8,4,9].* The maximum profit can be achieved by:

- Buying at day 0 = 1

- Selling at day 3 = 8

- Buying at day 4 = 4

- Selling at day 5 = 9

The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.

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

We are discussing a very classic ***dynamic programming*** based approach. We will maintain ***2 states for each day***.

* For any day, the ***first state*** will be the maximum profit we can make if there is one stock left to be sold. This state does not necessarily require you to buy the stock at the current day. It just requires that there is one stock extra, i.e. the number of stocks bought should be one more than the number of stocks sold so far.
* The ***second state*** will be the maximum profit we can make if we have no stock available to sell. This state does not necessarily require you to sell the stock at the current day. It just requires that the number of stocks bought should be equal to the number of stocks sold so far.
* Let us name the first state as ***bsp*** (bought state profit) and the second state as ***ssp*** (sold state profit). We will create two arrays, one for each state, of size equal to the number of days.

Can you derive a relation for the states at day i, using the values of bsp state and ssp state at day i-1?

***BSP State:*** If it is necessary to have one stock extra in our hand, then we have to two options:

1. We can use the ssp state of previous day and buy the stock at current day, i.e. take the maximum profit upto previous day and buy the stock at current day.

**Q)** Why we cannot take the maximum profit upto previous day from bsp state?

**R)** This is because if we consider the previous day, which already has one stock extra, then buying on the current day will make two stocks extra. But we are not allowed such transactions, as we must sell stock before buying it again.

1. We do not buy on the current day, but instead use the previous bsp state, i.e. we can take the maximum profit with one stock left upto previous day only. This will make us left with one stock extra only, as we are not buying or selling today.

We will consider that option whichever gives us more profit (or less loss).

***SSP State***: If it is necessary to have to equal number of stocks bought and sold, then we have two options:

1. We take the bsp state of the previous day and sell that extra stock which was left in the previous day’s state today. Hence, now, we will be left with no extra stocks in our hand, i.e. the number of stocks bought and sold will become extra.

Please remember that selling the stock at the current day means we are completing a transaction (“*BS*”). Hence we need to reduce the ***transaction fee***also from the profit so far.

**Q)** Why can't we take the ssp state of the previous day and sell a stock today?

**R)** Since ssp state means we do not have any extra stock in our hand, we cannot sell a stock without even buying it first. A state like “SB” is not possible.

1. We take the ssp state of the previous day as it is, and do not buy or sell stock today. Hence, ssp state of previous day, which gives the maximum profit with no stocks left upto previous day will become the maximum profit with no stock left upto the current day.

We will consider that option whichever gives us more profit (or less loss).

The only task left is to analyze the ***corner case***.

* It is the ***first day***, when we will not have any previous day.
* In this case, for the bsp state, we must have to buy the stock at the current day as option 2 is not available.
* For the ssp state, we do not have any option to sell stock, hence the maximum profit will be 0 (no stocks bought, no stocks sold).

*Note*: Please note that there can be negative value in bsp, or ssp state, which represents we are at a ***loss***. (Profit of -x is equivalent to loss of x).

At last, we will return the value of ***ssp state of the last day*** as our ***result***, since it will contain the maximum profit with no extra stock left in our hand (no incomplete transaction) and it has taken into account all the n days.

Please note that the bsp state of the last day cannot be considered as our result, because it has one incomplete transaction, because there is one stock extra in our hand, which is still left to be sold.

***Pseudo Code/ Algorithm***

* Create two arrays of size = n, one named *bsp* and other named *ssp*.
* Initialize bsp[0] as - arr[0] (bought the stock at day 0, hence in loss of arr[0], or in profit of -arr[0]).
* Initialize ssp[0] as 0 (no stocks bought or sold).
* Now loop from index/day 1 to n-1, and Update the bsp & ssp states as given:
  + **bsp[i] = max ( ssp[i - 1] - arr[i], bsp[i - 1] )**
  + **ssp[i] = max ( bsp[i - 1] + arr[i] - fee, ssp[i - 1] );**
* Return the value of ssp[n-1] as the maximum profit with infinite transactions allowed.

*Some Modifications in the logic to reduce Space Complexity*

Uptil now, we have used two arrays of size n each. But, you can easily see that there is no need to store the states of all the previous days for the calculations. We just need the state of the ***immediate previous day*** ((i - 1)th previous day for ith current day) and all the days before (i-1), i.e. days (i-2), (i-3), and so on, need not be stored.

Hence, we can replace our logic with 4 integer variables, two for previous day bsp and ssp state, namely ***bstp*** and ***sstp*** , and other two variables for the current day bsp and ssp state, namely ***nbstp*** and ***nsstp***.

Well, there will be a little difference in the implementation part, but, I expect from you that you can easily make these amendments accordingly.

***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 fee = Integer.parseInt(br.readLine());

int bstp = -arr[0];

int sstp = 0;

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

int nsstp = 0;

int nbstp = 0;

if(sstp - arr[i] > bstp){

nbstp = sstp - arr[i];

} else {

nbstp = bstp;

}

if(bstp + arr[i] - fee > sstp){

nsstp = bstp + arr[i] - fee;

} else {

nsstp = sstp;

}

bstp = nbstp;

sstp = nsstp;

}

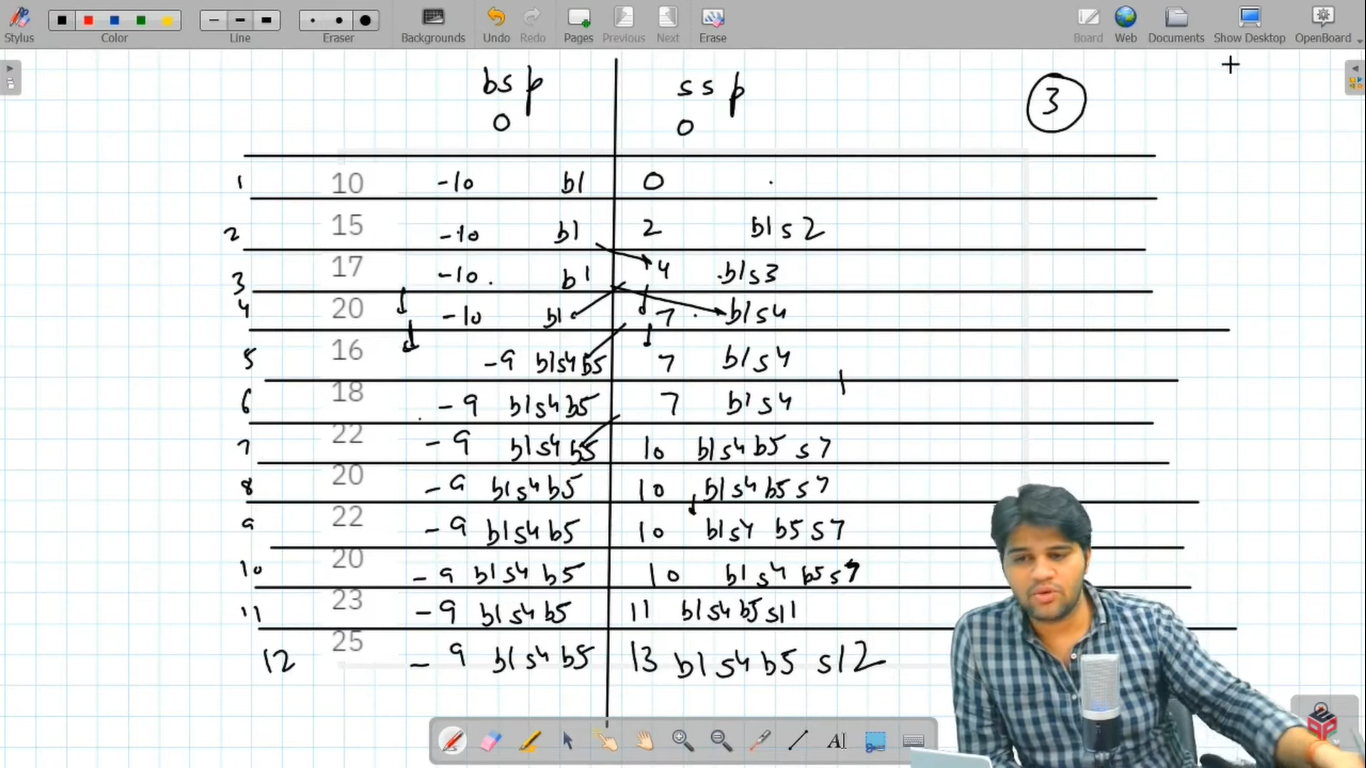
System.out.println(sstp);

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=pTQB9wbIpfU) from timestamp *[22:30 , 27:35]*.

***Dry Run*** of the code using an example is taken and elaborated in the same video from *[27:35,30:55]*.



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

Since, we are traversing through the array of n integers (representing days), and filling 2 states for each day, the time complexity will be O(2 \* N) = ***O(N)***.

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

Since we have replaced our logic of two arrays with just 4 integer variables, the space complexity will reduce from O(N) to ***O(1)***.

***Follow Up:***

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 ?

Along with the maximum profit, we can also maintain a string (or an array) of the current bsp and ssp state, the string will represent the sequence of stocks buy and sell upto the current day. Finally, we will print the resultant sequence which is stored in the ssp state of the last day.

***Asked in Companies***: *Facebook*

Hope that you liked the article on *‘Buy And Sell Stock - Transaction Fee - Infinite Transactions Allowed’*. Next problem will be the 4th problem in this series ‘[*Buy And Sell Stock - Cooldown - Infinite Transactions Allowed*](https://www.pepcoding.com/resources/online-java-foundation/dynamic-programming-and-greedy/buy-and-sell-stocks-cooldown-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