Selecting Stocks

An investor has saved some money and wants to invest in the stock market. There are a number of stocks to choose from, and they want to buy at most *1* share in any company. The total invested cannot exceed the funds available. A friend who is a stock market expert has predicted the values of each stock after 1 year. Determine the maximum profit that can be earned at the end of the year assuming the predictions come true.

****Example****

*saving = 250*

*currentValue = [175, 133, 109, 210, 97]*

*futureValue = [200, 125, 128, 228, 133]*

To maximize profits, the investor should buy stocks at indices *2* and *4* for an investment of *109 + 97 = 206*. At the end of the year the stocks are sold for *128 + 133 = 261*, so total profit is *261 - 206 = 55.*

**Function Description**

Complete the function *selectStock* in the editor below. The function should return an integer that denotes the maximum profit after one year.

selectStock has the following parameter(s):

int *saving:*  amount available for investment

int *currentValue[n]:*  the current stock values

int *futureValue[n]:*  the values of the stocks after one year

**Constraints**

* 0 < *n* ≤ 100
* 0 < *saving* ≤ 30000
* 0 ≤ *currrentValue[i], futureValue[i]*  ≤ 300

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *currrentValue*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *currrentValue[i]*.

The next line contains an integer, *n*, the number of elements in *futureValue*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *futureValue[i]*.

Sample Case 0

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

30    →  saving = 30

4     →  currentValue[] size n = 4

1     →  currentValue = [1, 2, 4, 6]

2

4

6

4     →  futureValue[] size n = 4

5     →  futureValue = ]5, 3, 5, 6]

3

5

6

**Sample Output**

6

**Explanation**

The investor can buy all *4* stocks and gain a profit of *(5-1)+(3-2)+(5-4)+(6-6) = 4+2+1+0 = 6*.

Sample Case 1

**Sample Input For Custom Testing**

STDIN    Function

-----    --------

500   →  saving = 500

5     →  currentValue[] size n = 5

150   →  currentValue = [150, 199, 200, 168, 153]

199

200

168

153

5     →  futureValue[] size n = 5

140   →  futureValue = [140, 175, 199, 121, 111]

175

199

121

111

**Sample Output**

0

**Explanation**

All the stocks lose value during the year, so no investment is made. There is no way to make a profit.

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 'selectStock' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER saving

\* 2. INTEGER\_ARRAY currentValue

\* 3. INTEGER\_ARRAY futureValue

\*/

public static int selectStock(int saving, List<Integer> currentValue, List<Integer> futureValue) {

// 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 saving = Integer.parseInt(bufferedReader.readLine().trim());

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

List<Integer> currentValue = IntStream.range(0, currentValueCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

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

List<Integer> futureValue = IntStream.range(0, futureValueCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int result = Result.selectStock(saving, currentValue, futureValue);

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

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

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