Skip A Level

Alex has to complete a multi-level game. Each level has an entry fee that needs to be paid before starting the level. After each level, Alex receives a point. Alex has to play levels in the given order and can skip at most one level.

Given the initial amount in Alex's wallet *k*, the number of levels in the game, *n* and the cost of each level, *costs*. Find the maximum points Alex can collect.

Note: It is not compulsory to complete all the levels

Example

k = 14

n = 5

costs = [2, 4,1, 8, 6]

Completing 5 levels without skipping any level, entry fees = 2 + 4 + 1 + 8 + 6 = 21 > *k*

Completing 5 levels while skipping the 4_{th} level, entry fees = 2 + 4 + 1 + 6 = 13 ≤ *k,* points collected = 4, as levels 1, 2, 3 and 5 were completed.

It can be proven that you cannot collect more than 4 points. Hence the answer is 4.

Function Description

Complete the function *maximumPoints* in the editor below.

maximumPoints has the following parameter(s):

int k: the initial number of coins in Alex's wallet

int costs[n]: the costs of each level

Returns

int: the maximum number of points Alex can collect after skipping at most one level

Constraints

- $1 \leq k \leq 10^9$
- $1 \leq n \leq 10^5$
- $1 \leq costs[i] \leq 10^9$

Input Format For Custom Testing

The first line contains an integer, *k*, the initial number of coins in Alex's wallet.

The second line contains an integer, *n*, the size of the array *costs*.

Each line *i* of the *n* subsequent lines (where $1 \leq i \leq n$) contains an integer that describes costs*[i]*.

Sample Case 0

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    --------
10    →  k = 10
5     →  n = 5
5     →  costs = [5, 2, 3, 1, 4]
2
3
1
4
10
5
5
2
3
1
4
```

Sample Output

```
4
```

Explanation

Completing 5 levels without skipping any level, entry fees = 5 + 2 + 3 + 1 + 4 = 15 > *k*

Completing 5 levels and skipping the 4th level, entry fees = 5 + 2 + 3 + 4 = 14 > *k*

Completing 5 levels and skipping the 4th level, entry fees = 2 + 3 + 1 + 4 = 10 ≤ *k*, points collected = 4, as levels 2, 3, 4 and 5 were completed.

It can be proven that you cannot collect more than 4 points. Hence the answer is 4.

Sample Case 1

Sample Input For Custom Testing

```
STDIN    FUNCTION

-----    --------

15    →  k = 15

6     →  n = 6

3     →  costs = [3, 2, 6, 4, 6, 1]

2

6

4

6

1
```

Sample Output

```
4
```

Explanation

Completing 6 levels without skipping any level, entry fees = 3 + 2 + 6 + 4 + 6 + 1 = 22 > *k*

Completing 5 levels and skipping the 3rd level, entry fees = 3 + 2 + 4 + 6 = 15 ≤ *k*, points collected = 4, as levels 1, 2, 4 and 5 were completed.

It can be proven that you cannot collect more than 4 points. Hence the answer is 4.

```
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 'maximumPoints' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER k
     *  2. INTEGER_ARRAY costs
     */

    public static int maximumPoints(int k, List<Integer> costs) {
    // 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 k = Integer.parseInt(bufferedReader.readLine().trim());

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

        List<Integer> costs = IntStream.range(0, costsCount).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.maximumPoints(k, costs);

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

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```

****

****