1. Parking Dilemma

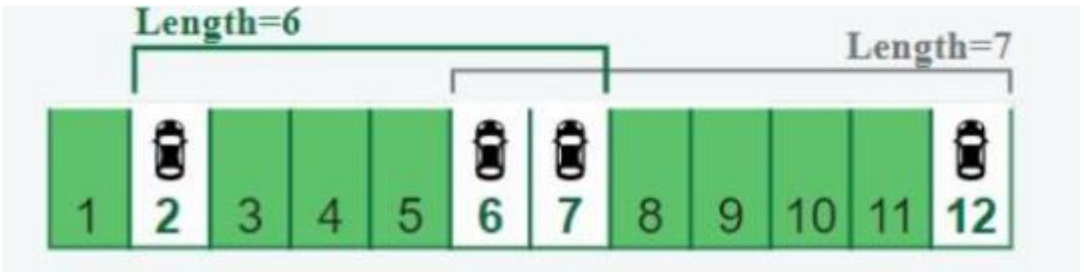
There are many cars parked in a parking lot. The parking lot is a straight line with a parking spot for every meter. There are n cars currently parked and a roofer wants to cover them with a roof. The requirement is that at least k cars currently in the lot are covered by the roof. Determine the minimum length of the roof to cover k cars.

Example:

n = 4

cars = [6, 2, 12, 7]

k = 3



Two roofs that cover three cars are possible: one covering spots 2 through 7with a length of 6, and another covering slots 6 through 12 with a length of 7. The shortest roof that meets the requirement is of length 6.

Function Description

Complete the function carParkingRoof in the editor below.

carParkingRoof has the following parameter(s):

int cars[n]: the parking spots where cars are parked

int k: the number of cars that have to be covered by the roof

Returns

int: the minimum length of a roof that covers k cars where they are parked currently

Constraints

- 1 <= n<=10^5
- 1<=k<=n
- 1<=cars[i] <= 10^14

• All spots taken by cars are unique

Input Format Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

In the first line, there is a single integer, n, the size of cars[].

Then, n line follows. In the ith of them, there is a single integer cars[i]

In the last line, there is a single integer k.

Sample Case 0

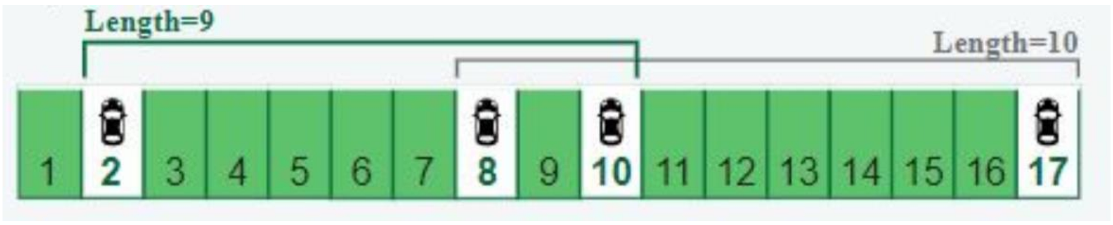
Sample Input

```
STDIN                    Function
--------------           ---------------
4               →         cars[ ]  size n = 4
2               →         cars[ ]  = [2,10,8,17]
10
8
17
3               →         k = 3
```

Sample Output

```
9
```

Explanation



A roof can be built of length 9 covering all parking spots from the 2nd through the 10th, so covering 3 cars in spots 2, 8, and 10. There is no shorter roof that can cover 3 cars.

Sample Case 1

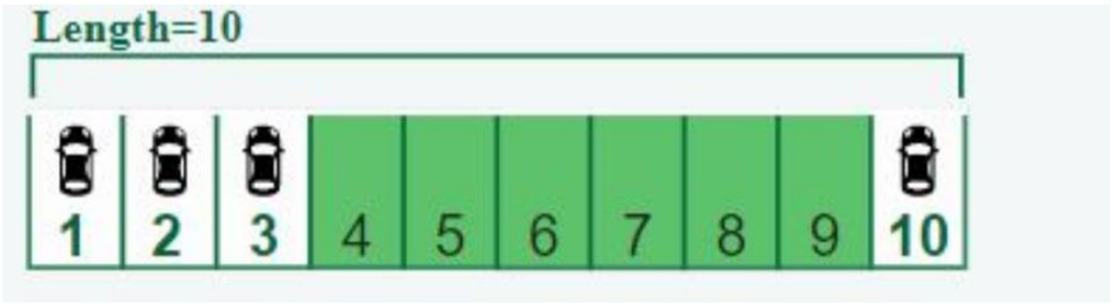
Sample Input

STDIN		Function
--------------		---------------
4	→	cars[ ] size n = 4
1	→	cars[ ] = [1,2,3,10]
2		
3		
10		
4	→	k = 4

Sample Output

10

Explanation



All of the cars must be covered. The shortest roof that can cover them has a length of 10 and starts at slot 1 and ends at slot 10.

Edit Code

```
const fs = require('fs');
process.stdin.resume();
process.stdin.setEncoding('utf-8');
let inputString = '';
let currentLine = 0;
process.stdin.on('data', function(inputStdin) {
        inputString += inputStdin;
} ) ;
process.stdin.on('end', function() {
         inputString = inputString.split('\n');
        main();
});

function readLine(){
         return inputString[currentLine++];
}

/*
        Complete the 'carParkingRoof' function below.
        The function is expected to return a LONG_INTEGER.
        The function accepts following parameters:
                1.  LONG_INTEGER_ARRAY cars
                2.  INTEGER k
*/

function carParkingRoof(cars,k){
        //write your code here
}

function main(){
        const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
        const carsCount = parseInt(readline().trim(),10);
        let cars = [ ];
        for(let I = 0; i<carsCount; i++){
                const carsItem = parseInt(readLine().trim(),10);
                cars.push(carsItem);
        }
        const k = parseInt(readLine().trim(),10);
        const result = carParkingRoof(cars,k);
        ws.write(result+'\n');
        ws.end();
}
```

2. Competitive Gaming

A group of friends are playing a video game together. During the ALL game, each player earns a number of points. At the end of a round, players who achieve at least a certain rank get to "level up" their characters to gain increased abilities. Given the scores of the 0 players at the end of a round, how many players will be able to level up?

Note: Players with equal scores will have equal ranks, but the player with the next lower score will be ranked based on the position within the list of all players' scores. For example, if there are four players, and three players tie for first place, their ranks are 1, 1, 1, and 4.

No player with a score of 0 can level up, regardless of rank.

Example:

n = 4
k = 3
scores = [100, 50, 50, 25]

These players' ranks are [1, 2, 2, 4]. Because the players need to have a rank of at least 3 to level up, only the first three players qualify. Therefore, the answer is 3.

Function Description

Complete the function numPlayers in the editor below.

numPlayers has the following parameters:
int k: an integer denoting the cutoff rank for leveling up a player's character
int scores[n]: an array of integers denoting the scores of the players
Returns:
int: the number of players who can level up after this round

Constraints

- 1 <= n<=10^5
- k<=n
- 0<=scores[i] <= 100

♦ Input Format For Custom Testing

The first line contains an integer, k, the cutoff rank for leveling up a player's character. The second line contains an integer, 17, the size of scores[ ]. 3 Each line i of the n subsequent lines (where 0<= i <=n) contains an integer, scores[i].

Sample Case 0

Sample Input

```
STDIN                   Function
--------------          ---------------
4            →           k = 4
5            →           scores[ ]   size  n=5
20           →           scores  = [20,40,60,80,100]
40
60
80
100
```

Sample Output

```
4
```

Explanation

In order,the players achieve the ranks[5,4,3,2,1].Since the cutoff, k, is rank >= 4, there are 4 players who will be able to level up.

Sample Case 1

Sample Output

```
STDIN                   Function
--------------          ---------------
4            →           k = 4
5            →           scores[ ]   size  n=5
2            →           scores  = [2,2,3,4,5]
2
3
4
5
```

Sample Output

```
5
```

Explanation

In order,the players achieve the ranks[4,4,3,2,1].Since the cutoff rank is 4,all 5 players will be able to level up.

Edit Code

```
const fs = require('fs');
process.stdin.resume();
process.stdin.setEncoding('utf-8');
let inputString = '';
let currentLine = 0;
process.stdin.on('data', function(inputStdin) {
        inputString += inputStdin;
} ) ;
process.stdin.on('end', function() {
         inputString = inputString.split('\n');
        main();
});

function readLine(){
         return inputString[currentLine++];
}

/*
        Complete the 'numPlayers' function below.
        The function is expected to return an INTEGER.
        The function accepts following parameters:
                1.  INTEGER k
                2.  INTEGER_ARRAY  scores

*/
function numPlayers(k,scores){
        //write your code here
}

function main(){
        const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
        const k= parseInt(readline().trim(),10);
        const scoresCount= parseInt(readline().trim(),10);
        let scores= [ ];
        for(let I = 0; i<scoresCount; i++){
                const scoresItem = parseInt(readLine().trim(),10);
                scores.push(scoresItem);
        }
        const result = numPlayers(k,scores);
        ws.write(result+’\n’);
        ws.end();
}
```

3. Reach the End in Time

A 2-D grid consisting of some blocked (represented as '#') and some unblocked (represented as '.') cells is given. The starting position of a pointer is in the top-left corner of the grid. It is guaranteed that the starting position is in an unblocked cell. It is also guaranteed that the bottom-right cell is unblocked. Each cell of the grid is connected with its right, left, top, and bottom cells (if those cells exist). It takes 1 second for a pointer to move from a cell to its adjacent cell. If the pointer can reach the bottom-right corner of the grid within k seconds, return the string 'Yes. Otherwise, return the string 'No'.

Example

rows = 3

grid = [‘..##’,’#.##’,’#...’]

maxTime = 5

```
 ..##
#.##
#...
```

It will take the pointer 5 seconds to reach the bottom right corner. As long as k >= 5, return ' Yes'.

Function Description

Complete the function reachTheEnd in the editor below.

reachTheEnd has the following parameter(s):

string grid[r]: the rows of the grid

int maxTime: the maximum time to complete the traversal

Returns:

string: the final string; either 'Yes' or 'No'

Constraints

- 1 <= rows<=500
- 1<=maxTime<=10^6

♦ Input Format For Custom Testing

The first line contains an integer, rows, that denotes the number of rows of the 2-D grid. In each of the next rows lines, the ith line contains a string denoting the configuration of the ith row of the grid. The last line contains an integer, maxTime, that represents the maximum time in seconds the pointer has to reach the bottom right cell.

Sample Case 0

```
STDIN                    Function

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

2                  →      size of grid[ ] rows = 2

..                 →      grid = [ '..','..' ]

..

3                  →      maxTime = 3
```

Sample Output

Yes

Explanation

The grid has 2 rows and 2 columns and the time within which the pointer needs to reach the bottom-right cell is 3 seconds. Starting from the top-left cell, the pointer can either move to the top-right unblocked cell or bottom-left unblocked cell then to the bottom-right cell. It takes 2 seconds to reach the bottom-right cell on either path. Thus, the pointer reaches the bottom-right cell within the 3 seconds allowed, and 'Yes' is returned.

Sample Case 1

Sample Input

```
STDIN                    Function

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

2                         grid[ ]  size rows = 2

.#                        grid = [ '.#','#.']

#.

2                          maxTime = 2
```

Sample Output

```
No
```

Explanation

The grid has 2 rows and 2 columns and the time within which the pointer needs to reach the bottom-right cell is 2 seconds. It can neither move to the top-right cell nor to the bottom-left cell and so the pointer cannot reach the bottom-right cell, regardless of the time constraint.

Edit Code

```
const fs = require('fs');

process.stdin.resume();

process.stdin.setEncoding('utf-8');

let inputString = '';

let currentLine = 0;

process.stdin.on('data', function(inputStdin) {

            inputString += inputStdin;

} ) ;

process.stdin.on('end', function() {

             inputString = inputString.split('\n');

            main();

});

function readLine() {

            return inputString[currentLine++];

}

/*

            Complete the 'reachTheEnd' function below.

            The function is expected to return an STRING.

            The function accepts following parameters:

                        1.  STRING_ARRAY grid;
                        2.  INTEGER  maxTime

*/


function reachTheEnd(grid,maxTime){

            //write your code here

}

function main(){

            const ws = fs.createWriteStream(process.env.OUTPUT_PATH);

            const gridCount = parseInt(readLine().trim(), 10);

            let grid = [];

            for (let i = 0; i < gridCount; i++){

             const gridItem = readLine();
             grid.push(gridItem);
            }
            const maxTime = parseInt(readLine().trim(), 10);
            const result = reachTheEnd(grid, maxTime);
            ws.write(result + 1\n');
            ws.end();
}
```