Problem 3100: Water Bottles II

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given two integers

numBottles

and

numExchange

.

numBottles

represents the number of full water bottles that you initially have. In one operation, you can perform one of the following operations:

Drink any number of full water bottles turning them into empty bottles.

Exchange

numExchange

empty bottles with one full water bottle. Then, increase

numExchange

by one.

Note that you cannot exchange multiple batches of empty bottles for the same value of

numExchange

. For example, if

numBottles == 3

and

numExchange == 1

, you cannot exchange

3

empty water bottles for

3

full bottles.

Return

the

maximum

number of water bottles you can drink

.

Example 1:

	Full Bottles	Empty Bottles	numExchange	Bottles Drunk
Initially	13	0	6	0
Drink 13 bottles	0	13	6	13
Exchange	1	7	7	13
Exchange	2	0	8	13
Drink 2 bottles	0	2	8	15

Input:

numBottles = 13, numExchange = 6

Output:

15

Explanation:

The table above shows the number of full water bottles, empty water bottles, the value of numExchange, and the number of bottles drunk.

Example 2:

	Full Bottles	Empty Bottles	numExchange	Bottles Drunk
Initially	10	0	3	0
Drink 10 bottles	0	10	3	10
Exchange	1	7	4	10
Exchange	2	3	5	10
Drink 2 bottles	0	5	5	12
Exchange	1	0	6	12
Drink 1 bottle	0	1	6	13

Input:

numBottles = 10, numExchange = 3

Output:

13

Explanation:

The table above shows the number of full water bottles, empty water bottles, the value of numExchange, and the number of bottles drunk.

Constraints:

1 <= numBottles <= 100

1 <= numExchange <= 100

Code Snippets

C++:

```
class Solution {
public:
int maxBottlesDrunk(int numBottles, int numExchange) {

}
};
```

Java:

```
class Solution {
public int maxBottlesDrunk(int numBottles, int numExchange) {

}
}
```

Python3:

```
class Solution:
def maxBottlesDrunk(self, numBottles: int, numExchange: int) -> int:
```

Python:

```
class Solution(object):
def maxBottlesDrunk(self, numBottles, numExchange):
"""
:type numBottles: int
:type numExchange: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} numBottles
* @param {number} numExchange
* @return {number}
*/
var maxBottlesDrunk = function(numBottles, numExchange) {

};
```

TypeScript:

```
function maxBottlesDrunk(numBottles: number, numExchange: number): number {

};
```

C#:

```
public class Solution {
public int MaxBottlesDrunk(int numBottles, int numExchange) {

}
}
```

C:

```
int maxBottlesDrunk(int numBottles, int numExchange) {

}
```

Go:

```
func maxBottlesDrunk(numBottles int, numExchange int) int {

}
```

Kotlin:

```
class Solution {
fun maxBottlesDrunk(numBottles: Int, numExchange: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxBottlesDrunk(_ numBottles: Int, _ numExchange: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_bottles_drunk(num_bottles: i32, num_exchange: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num_bottles
# @param {Integer} num_exchange
# @return {Integer}
def max_bottles_drunk(num_bottles, num_exchange)

end
```

PHP:

```
class Solution {

/**
* @param Integer $numBottles
* @param Integer $numExchange
* @return Integer
*/
function maxBottlesDrunk($numBottles, $numExchange) {

}
}
```

Dart:

```
class Solution {
int maxBottlesDrunk(int numBottles, int numExchange) {

}
}
```

Scala:

```
object Solution {
def maxBottlesDrunk(numBottles: Int, numExchange: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec max_bottles_drunk(num_bottles :: integer, num_exchange :: integer) ::
integer
def max_bottles_drunk(num_bottles, num_exchange) do

end
end
```

Erlang:

```
-spec max_bottles_drunk(NumBottles :: integer(), NumExchange :: integer()) ->
integer().
max_bottles_drunk(NumBottles, NumExchange) ->
.
```

Racket:

```
(define/contract (max-bottles-drunk numBottles numExchange)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Water Bottles II
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
```

```
public:
int maxBottlesDrunk(int numBottles, int numExchange) {

}
};
```

Java Solution:

```
/**
* Problem: Water Bottles II
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maxBottlesDrunk(int numBottles, int numExchange) {

}
}
```

Python3 Solution:

```
"""
Problem: Water Bottles II
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxBottlesDrunk(self, numBottles: int, numExchange: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxBottlesDrunk(self, numBottles, numExchange):
"""
:type numBottles: int
:type numExchange: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Water Bottles II
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} numBottles
* @param {number} numExchange
* @return {number}
*/
var maxBottlesDrunk = function(numBottles, numExchange) {

};
```

TypeScript Solution:

```
/**
* Problem: Water Bottles II
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
function maxBottlesDrunk(numBottles: number, numExchange: number): number {

};
```

C# Solution:

```
/*
* Problem: Water Bottles II
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaxBottlesDrunk(int numBottles, int numExchange) {

}
}
```

C Solution:

```
/*
* Problem: Water Bottles II
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int maxBottlesDrunk(int numBottles, int numExchange) {

}
```

Go Solution:

```
// Problem: Water Bottles II
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func maxBottlesDrunk(numBottles int, numExchange int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxBottlesDrunk(numBottles: Int, numExchange: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxBottlesDrunk(_ numBottles: Int, _ numExchange: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Water Bottles II
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_bottles_drunk(num_bottles: i32, num_exchange: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} num_bottles
# @param {Integer} num_exchange
# @return {Integer}
def max_bottles_drunk(num_bottles, num_exchange)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $numBottles
* @param Integer $numExchange
* @return Integer
*/
function maxBottlesDrunk($numBottles, $numExchange) {

}
}
```

Dart Solution:

```
class Solution {
int maxBottlesDrunk(int numBottles, int numExchange) {

}
}
```

Scala Solution:

```
object Solution {
def maxBottlesDrunk(numBottles: Int, numExchange: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_bottles_drunk(num_bottles :: integer, num_exchange :: integer) ::
integer
def max_bottles_drunk(num_bottles, num_exchange) do

end
end
```

Erlang Solution:

```
-spec max_bottles_drunk(NumBottles :: integer(), NumExchange :: integer()) ->
integer().
max_bottles_drunk(NumBottles, NumExchange) ->
.
```

Racket Solution:

```
(define/contract (max-bottles-drunk numBottles numExchange)
(-> exact-integer? exact-integer? exact-integer?)
)
```