Problem 1518: Water Bottles

Problem Information

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

Problem Description

There are

numBottles

water bottles that are initially full of water. You can exchange

numExchange

empty water bottles from the market with one full water bottle.

The operation of drinking a full water bottle turns it into an empty bottle.

Given the two integers

numBottles

and

numExchange

, return

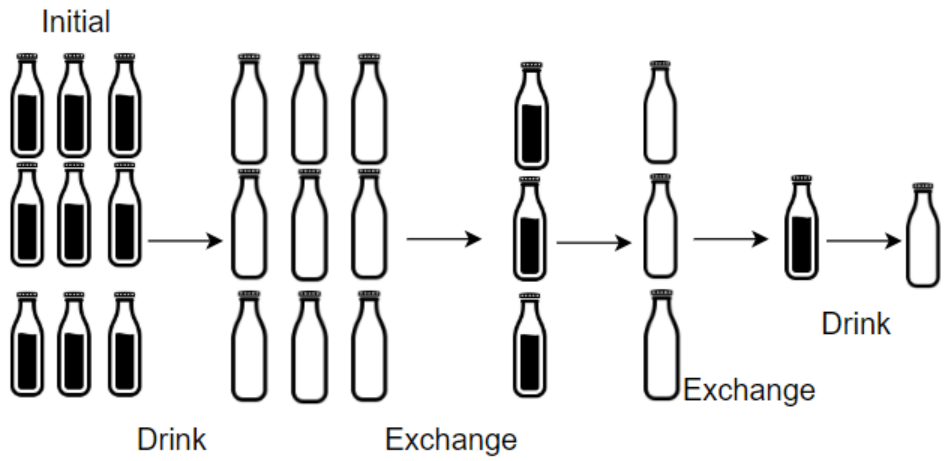
the

maximum

number of water bottles you can drink

.

Example 1:



Input:

numBottles = 9, numExchange = 3

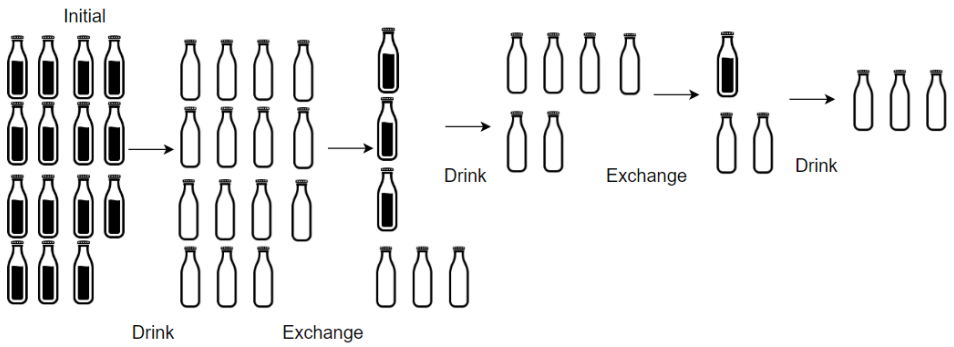
Output:

13

Explanation:

You can exchange 3 empty bottles to get 1 full water bottle. Number of water bottles you can drink: 9 + 3 + 1 = 13.

Example 2:



Input:

numBottles = 15, numExchange = 4

Output:

19

Explanation:

You can exchange 4 empty bottles to get 1 full water bottle. Number of water bottles you can drink: 15 + 3 + 1 = 19.

Constraints:

1 <= numBottles <= 100

2 <= numExchange <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (num-water-bottles numBottles numExchange)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Water Bottles
* Difficulty: Easy
* 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 numWaterBottles(int numBottles, int numExchange) {

}
};
```

Java Solution:

```
/**
* Problem: Water Bottles
* Difficulty: Easy
* 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 numWaterBottles(int numBottles, int numExchange) {

}
}
```

Python3 Solution:

```
"""
Problem: Water Bottles
Difficulty: Easy
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 numWaterBottles(self, numBottles: int, numExchange: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Water Bottles
* Difficulty: Easy
* 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 numWaterBottles = function(numBottles, numExchange) {

};
```

TypeScript Solution:

```
/**
* Problem: Water Bottles
* Difficulty: Easy
* 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 numWaterBottles(numBottles: number, numExchange: number): number {

};
```

C# Solution:

```
/*
* Problem: Water Bottles
* Difficulty: Easy
```

```
* 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 NumWaterBottles(int numBottles, int numExchange) {

}
}
```

C Solution:

```
/*
* Problem: Water Bottles
* Difficulty: Easy
* 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 numWaterBottles(int numBottles, int numExchange) {

}
```

Go Solution:

```
// Problem: Water Bottles
// Difficulty: Easy
// 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 numWaterBottles(numBottles int, numExchange int) int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Water Bottles
// Difficulty: Easy
// 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 num_water_bottles(num_bottles: i32, num_exchange: i32) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-water-bottles numBottles numExchange)
(-> exact-integer? exact-integer? exact-integer?)
)
```