Binary Gap Exercise in Codility

Find a specific binary sequence.

Suppose a positive integer `N` is given. Determine the binary representation of `N`, and find the longest subsequence of the form `10*1` in this representation, where `0*` stands for any number of zeros in the sequence. Examples: `11`, `101`, `1001`, `10001` etc. Return the number of zeros in the longest sequence you found. If you didn't find such a sequence, return zero.

You can read the original task description on Codility.

Solution:

Whenever you deal with a riddle, bear in mind, it doesn't matter what techniques you use as long as your solution is correct. Don't try to impress your interviewers with fancy techniques, don't even think about announcing that you are going to use "functional programming" or "recursion" or anything else. Just get the job done.

Do explain your thought process! If you are on the right track, your interviewers will appreciate relating to how you think. If you are on the wrong track, your interviewers will often help you out, because they relate to you, and they want you to succeed.

You can read more interviewing tips in The Developer's Edge.

Before coding, always plan your solution, and explain how you want to solve your task. Your interviewers may correct you, and in the best case, say that you can start coding. In our case, the plan looks as follows:

1. Convert `N` into a binary string
2. Set a sequence counter to zero. Set a maximum sequence counter to zero.
3. Iterate over each digit of the binary representation
    - If you find a zero, increase the sequence counter by one.

    - If you find a one, compare the sequence counter to the maximum sequence counter, and save the higher value in the maximum sequence counter. Then set the sequence counter to zero to read the upcoming sequence lengths.
4. Once you finish, return the maximum sequence counter value.

Obtaining the binary representation:

You may or may not know that integers have a `toString` method, and the first argument of `toString` is the base in which the number should be interpreted. Base `2` is binary, so all you need to do to convert an integer into its binary representation is:

```
const n1 = 256, n2 = 257;
var print = "";

print=n1.toString(2)
console.log(print)
//"100000000"

print=n2.toString( 2 )
console.log(print)
//"100000001"
```

Chances are, you don't know this trick. No problem. In most tech interviews, you can use google. If you formulate the right search expression such as "javascript binary representation of a number", most of the time, you get a nice and short StackOverflow page explaining the solution. Be careful with copy-pasting tens of lines of code. Look for a deep understanding of the problem, and implement a compact solution.

Never google for the exact solution of the task, because your interviewers may not know how to handle such an attempt.

In the unlikely case you are not allowed to use Google, nothing is lost. You can still solve the same problem in vanilla JavaScript. How do we convert a decimal number to binary on paper?

Suppose your number is 18.

- `18 / 2 = 9`, and the remainder is `0`.

- `9 / 2 = 4`, and the remainder is `1`.
- `4 / 2 = 2`, and the remainder is `0`.
- `2 / 2 = 1`, and the remainder is `0`.
- `1 / 2 = 0`, and the remainder is `1`.

Read the digits from bottom-up to get the result: `10010`.

Let's write some code to get the same result:

```
const IntToBinary = N => {
    let result = '';
    while ( N > 0 ) {
        result = (N % 2) + result;
        N = Math.trunc( N / 2 );
    }
    return result;
}

console.log(IntToBinary(256)) //print the output
```

The `%` (modulus) operator gives you the remainder of the division. The `trunc` function truncates the results. For instance, `Math.trunc(9.5)` becomes `9`.

If you can't come up with this algorithm on your own, think in another way:

```
// 18 is
(1 * 16) + (0 * 8) + (0 * 4) + (1 * 2) + (0 * 1)
// yielding 10010
```

First, we have to get the largest digit value, which is `16`:

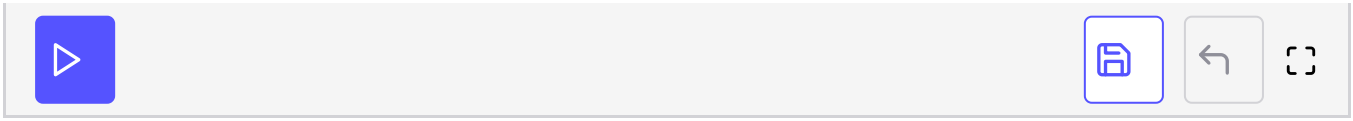
```
// Constraint: N > 0.
const getLargestBinaryDigit = N => {
    let digit = 2;
    while ( N >= digit ) digit *= 2;
    return digit / 2;
}
console.log(getLargestBinaryDigit(20))
```

Then we divide this digit value by `2` until we get `1` to retrieve the digits of the

binary number one by one. Whenever `N` is greater than or equal to the digit

value, our upcoming digit is `1`, and we have to subtract `digit` from `N`. Otherwise, our upcoming digit value is `0`:

```
const IntToBinary = N => {
    let result = '';
    for ( let digit = getLargestBinaryDigit( N ); digit >= 1; digit /= 2 ) {
        if ( N >= digit ) {
            N -= digit;
            result += '1';
        } else {
            result += '0';
        }
    }
    return result;
}
console.log(IntToBinary(10))
```



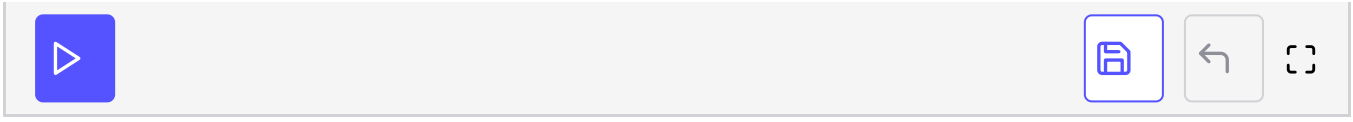
Enough said about the integer to binary conversion. Let's continue with the state space of the solution.

Determining the state space:

```
function solution( N ) {
    let str = N.toString( 2 ),
        zeroCount = 0,
        result = 0;

    // ...

    return result;
}
```



We will use `N.toString( 2 )` here to get the binary representation of `N`.

To identify a sequence of zeros bounded by ones, we have to know if the sequence has a left border. As every single positive binary number starts with `1`, this condition is automatically true.

Side note: if `N` was allowed to be `0`, even then, our function would return the correct result, because the string `'0'` does not have a trailing `1` in the

sequence.

Therefore, the state space is quite simple: we need to know the binary string of the input, the number of zeros currently read in the sequence, and the longest string found so far.

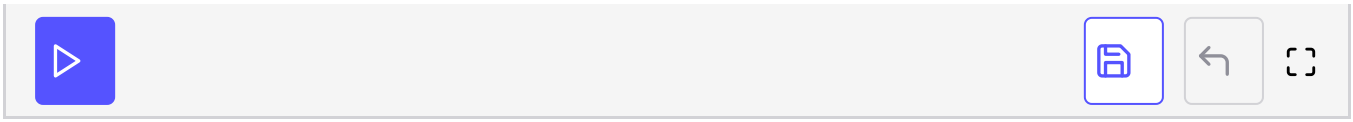
Iteration:

We have to read each digit of the solution one by one. The traditional way in most programming languages is a `for` loop.

```
function solution( N ) {
    let str = N.toString( 2 ),
        zeroCount = 0,
        result = 0;

    for ( let i = 0; i < str.length; ++i ) {
        // ...
    }

    return result;
}
```

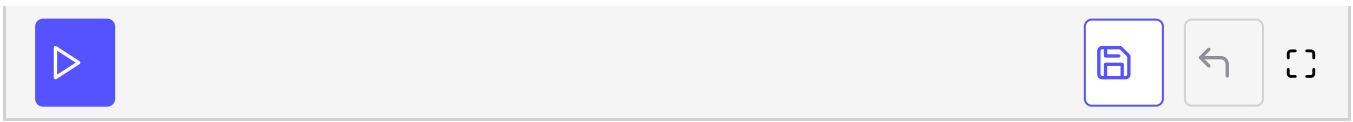


We can also use the `for..of` loop of ES6 that enumerates each character of the string. Strings work as *iterators* and *iterable objects* in ES6. For more information, read my article titled ES6 Iterators and Generators in Practice. You can also find six more exercises belonging to this topic in this blog post.

```
function solution( N ) {
    let str = N.toString( 2 ),
        zeroCount = 0,
        result = 0;

    for ( let digit of str ) {
        // ...
    }

    return result;
}
```



Reading the digits:

Each digit can either be a zero or a one. We will branch off with an if-else statement:

```
function solution( N ) {
    let str = N.toString( 2 ),
        zeroCount = 0,
        result = 0;

    for ( let digit of str ) {
        if ( digit === '0' ) {
            // ...
        } else /* if ( digit === '1' ) */ {
            // ...
        }
    }

    return result;
}
```

Process the digits:

If we read a zero, we have to increment the zero counter by one. If we read a one, we have to determine if we have just read the longest sequence of zeros by taking the maximum of `result` and `zeroCount`, and saving this maximum in `result`. After determining the new `result` value, we have to make sure to reset `zeroCount` to `0`.

```
function solution( N ) {
    let str = N.toString( 2 ),
        zeroCount = 0,
        result = 0;

    for ( let digit of str ) {
        if ( digit === '0' ) {
            zeroCount += 1;
        } else /* if ( digit === '1' ) */ {
            result = Math.max( result, zeroCount );
            zeroCount = 0;
        }
    }

    return result;
}
console.log(solution(456))
```

If you execute this algorithm in Codility, you can see that all your tests pass. I

encourage you to solve other Codility tasks, as Codility is a great platform to practice coding challenges.