Space Complexity

Learn about the concept of space complexity using examples in JavaScript.

Asymptotic notation is often used to describe the runtime of a program or algorithm, but it can also be used to describe the space, or memory, that a program or algorithm will need. Think about a simple function that takes in two numbers and returns their sum:

```
function addNumbers(a, b) {
  return a + b;
}
```

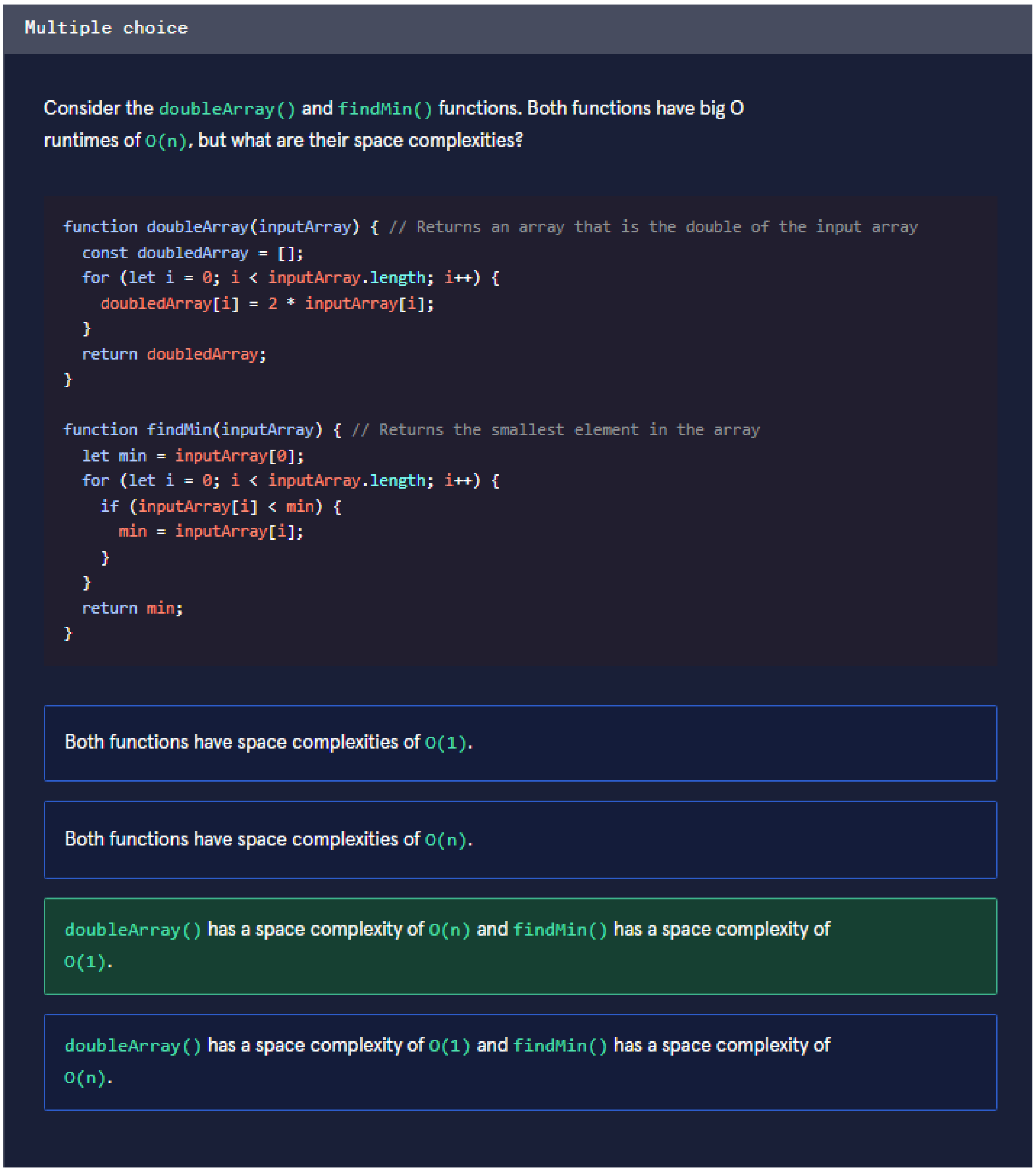
This function has a space complexity of O(1), because the amount of space it needs will not change based on the input. While this function also has a constant runtime of O(1), most functions do not have matching space and time complexities:

```
function simpleLoop(inputArray) {
  for (let i = 0; i < inputArray.length; i++) {
    console.log(i);
  }
}
```

As we know, a simple for loop that goes through every element in an array of size n has a linear runtime of O(n). However, this function takes O(1) space since no new variables are being created and therefore no more space must be allocated.

A recursive function that is passed the same array or object in each call doesn’t add to the space complexity if the array or object is passed by reference (which it is in JavaScript).

Like with time complexity, space complexity denotes space growth in relation to the input size. It’s also important to note that space complexity usually refers to any additional space that will be needed, and doesn’t count the space of the input. So a function could have 10 arrays passed into it, but if all it does inside is print 'Hello World!', then it still takes O(1) space.


Multiple choice

Consider the `doubleArray()` and `findMin()` functions. Both functions have big O runtimes of `O(n)`, but what are their space complexities?

```
function doubleArray(inputArray) { // Returns an array that is the double of the input array
  const doubledArray = [];
  for (let i = 0; i < inputArray.length; i++) {
    doubledArray[i] = 2 * inputArray[i];
  }
  return doubledArray;
}

function findMin(inputArray) { // Returns the smallest element in the array
  let min = inputArray[0];
  for (let i = 0; i < inputArray.length; i++) {
    if (inputArray[i] < min) {
      min = inputArray[i];
    }
  }
  return min;
}
```

Both functions have space complexities of `O(1)`.

Both functions have space complexities of `O(n)`.

[✓ `doubleArray()` has a space complexity of `O(n)` and `findMin()` has a space complexity of `O(1)`.]

`doubleArray()` has a space complexity of `O(1)` and `findMin()` has a space complexity of `O(n)`.

Yes! doubleArray() creates a new array that matches the size of the input array, so the space needed for this function will change as the size of the input array changes. findMin() only creates one new variable regardless of the input, so its size is constant.

Show less

Space complexity is important to consider alongside time complexity when comparing data structures and algorithms. While two functions may have very similar runtimes, one could use less space. Consider the doubleArray() function from above. It has a runtime of O(n), and takes O(n) space. Could we optimize it to have a better space complexity?

```
function doubleInPlace(inputArray) {
 for (let i = 0; i < inputArray.length; i++) {
  inputArray[i] *= 2;
 }
 return inputArray;
}
```

doubleInPlace() does the same thing as doubleArray() and in the same amount of time, but only takes O(1) space, simply because it doesn’t create a new array. As you move forward, remember that just because a program has the best runtime possible, doesn’t mean it can’t still be optimized.