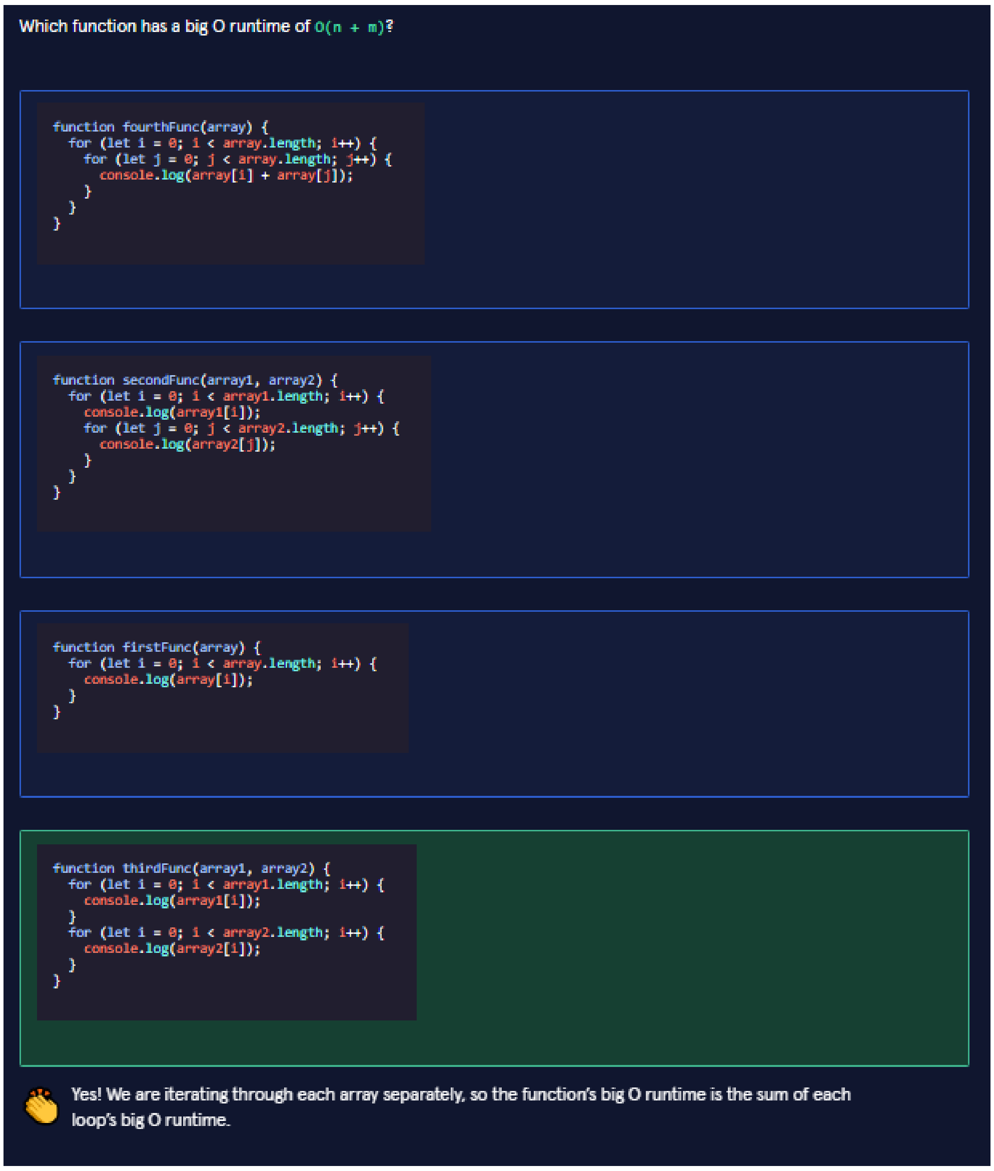
QUIZ



Which function has a big O runtime of `O(n + m)`?

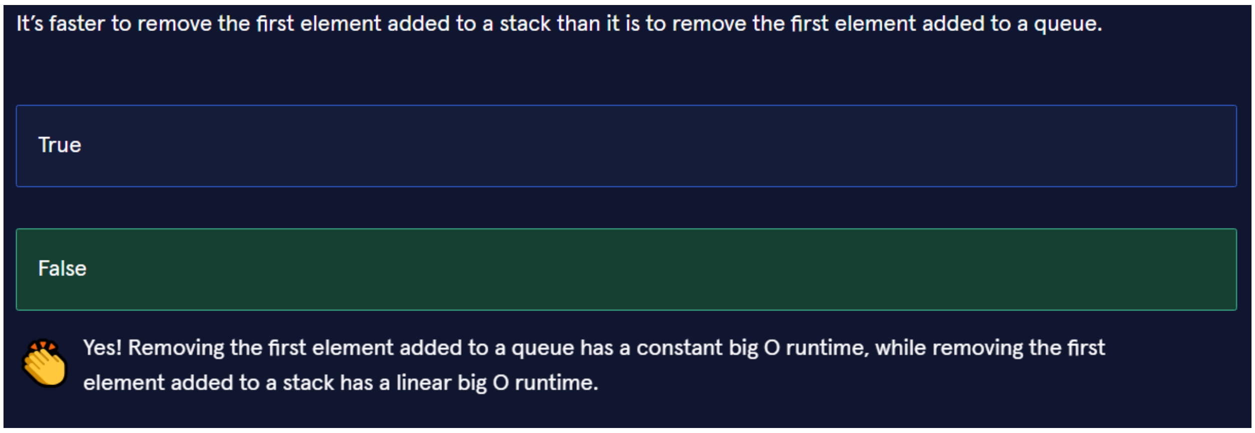
```
function fourthFunc(array) {
  for (let i = 0; i < array.length; i++) {
    for (let j = 0; j < array.length; j++) {
      console.log(array[i] + array[j]);
    }
  }
}
```

```
function secondFunc(array1, array2) {
  for (let i = 0; i < array1.length; i++) {
    console.log(array1[i]);
    for (let j = 0; j < array2.length; j++) {
      console.log(array2[j]);
    }
  }
}
```

```
function firstFunc(array) {
  for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
  }
}
```

[✓
```
function thirdFunc(array1, array2) {
  for (let i = 0; i < array1.length; i++) {
    console.log(array1[i]);
  }
  for (let i = 0; i < array2.length; i++) {
    console.log(array2[i]);
  }
}
```
]

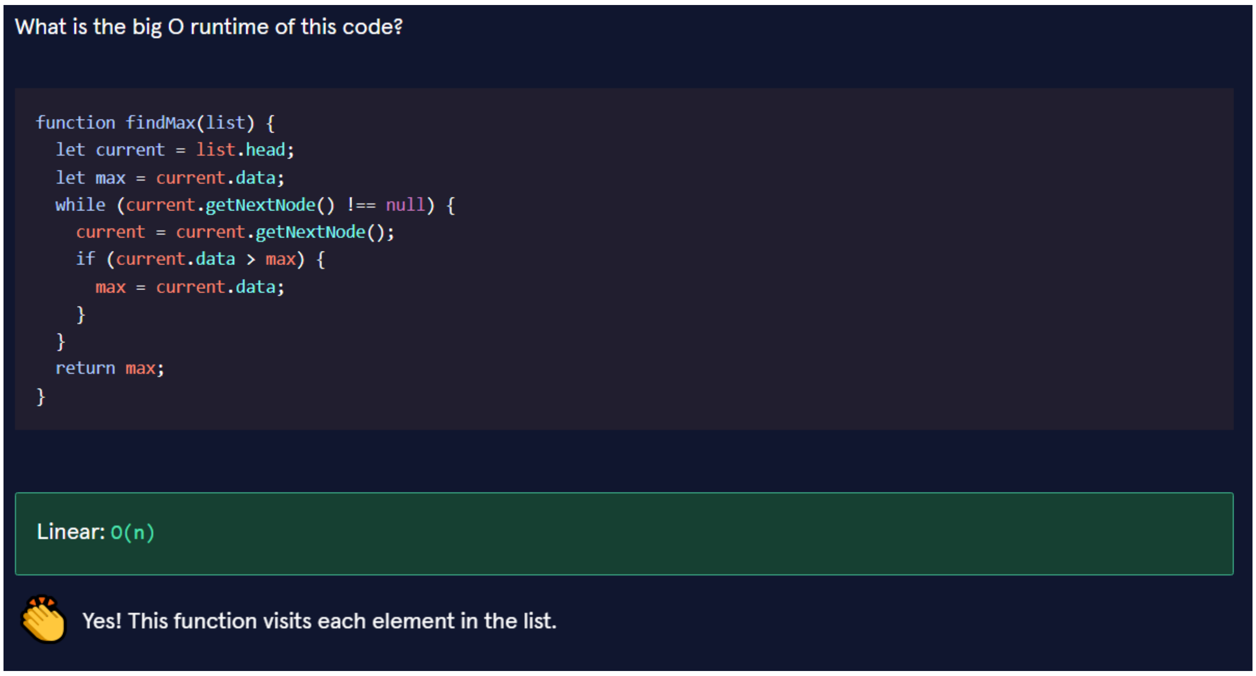
👏 Yes! We are iterating through each array separately, so the function's big O runtime is the sum of each loop's big O runtime.



It's faster to remove the first element added to a stack than it is to remove the first element added to a queue.

True [✓ False]

👏 Yes! Removing the first element added to a queue has a constant big O runtime, while removing the first element added to a stack has a linear big O runtime.

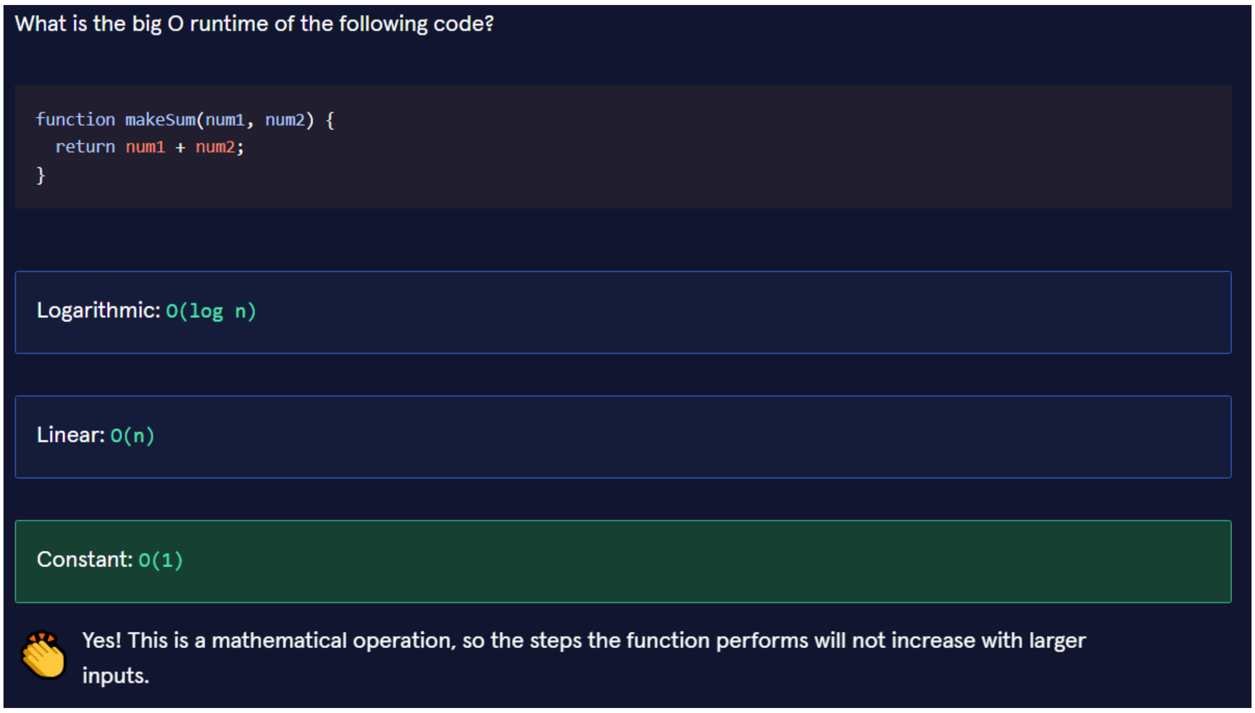


What is the big O runtime of this code?

```
function findMax(list) {
  let current = list.head;
  let max = current.data;
  while (current.getNextNode() !== null) {
    current = current.getNextNode();
    if (current.data > max) {
      max = current.data;
    }
  }
  return max;
}
```

[✓ Linear: `O(n)`]

👏 Yes! This function visits each element in the list.



What is the big O runtime of the following code?

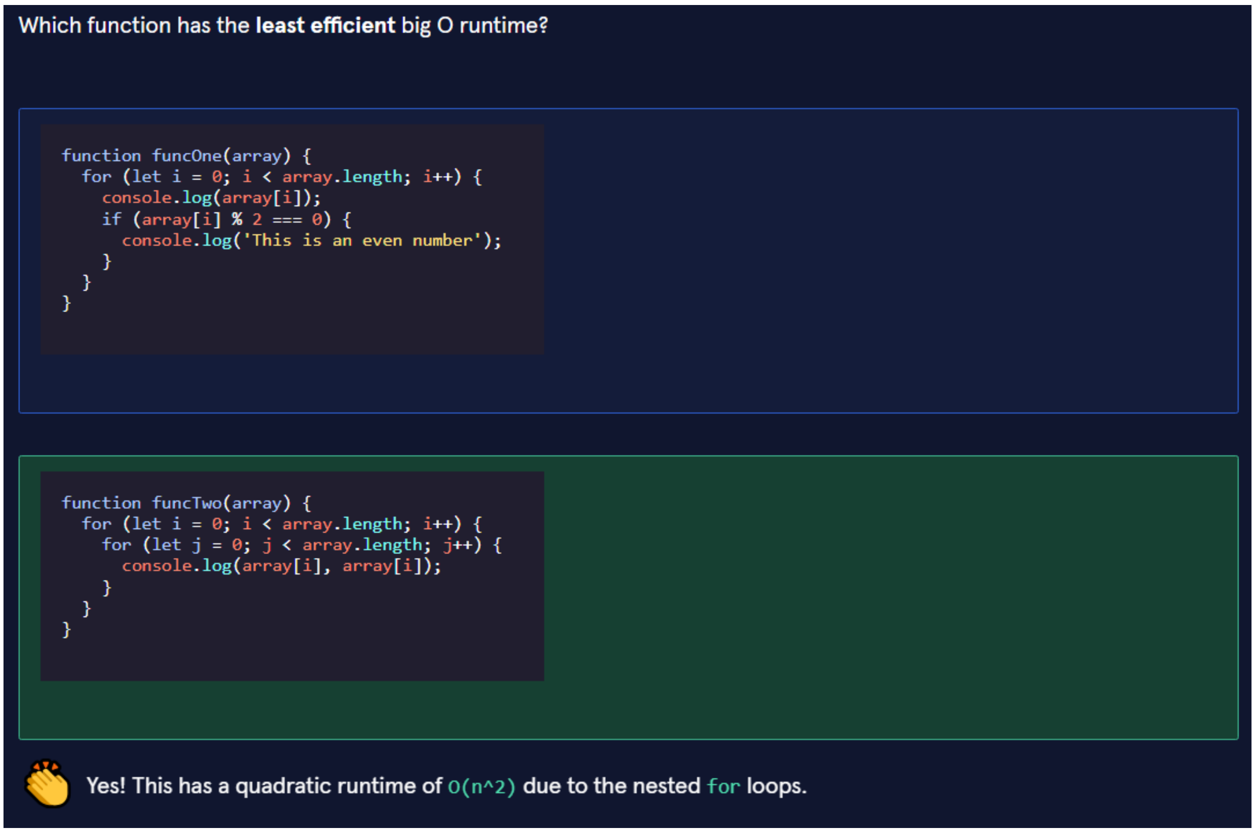
```
function makeSum(num1, num2) {
  return num1 + num2;
}
```

Logarithmic: `O(log n)`

Linear: `O(n)`

[✓ Constant: `O(1)`]

👏 Yes! This is a mathematical operation, so the steps the function performs will not increase with larger inputs.

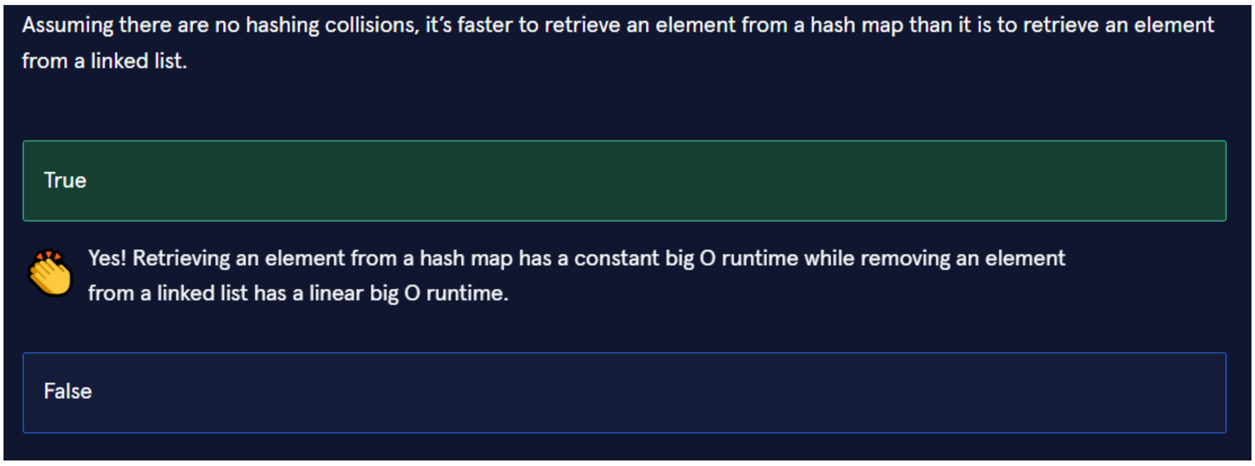

Which function has the **least efficient** big O runtime?

```javascript
function funcOne(array) {
  for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
    if (array[i] % 2 === 0) {
      console.log('This is an even number');
    }
  }
}
```

[✓]

```javascript
function funcTwo(array) {
  for (let i = 0; i < array.length; i++) {
    for (let j = 0; j < array.length; j++) {
      console.log(array[i], array[i]);
    }
  }
}
```

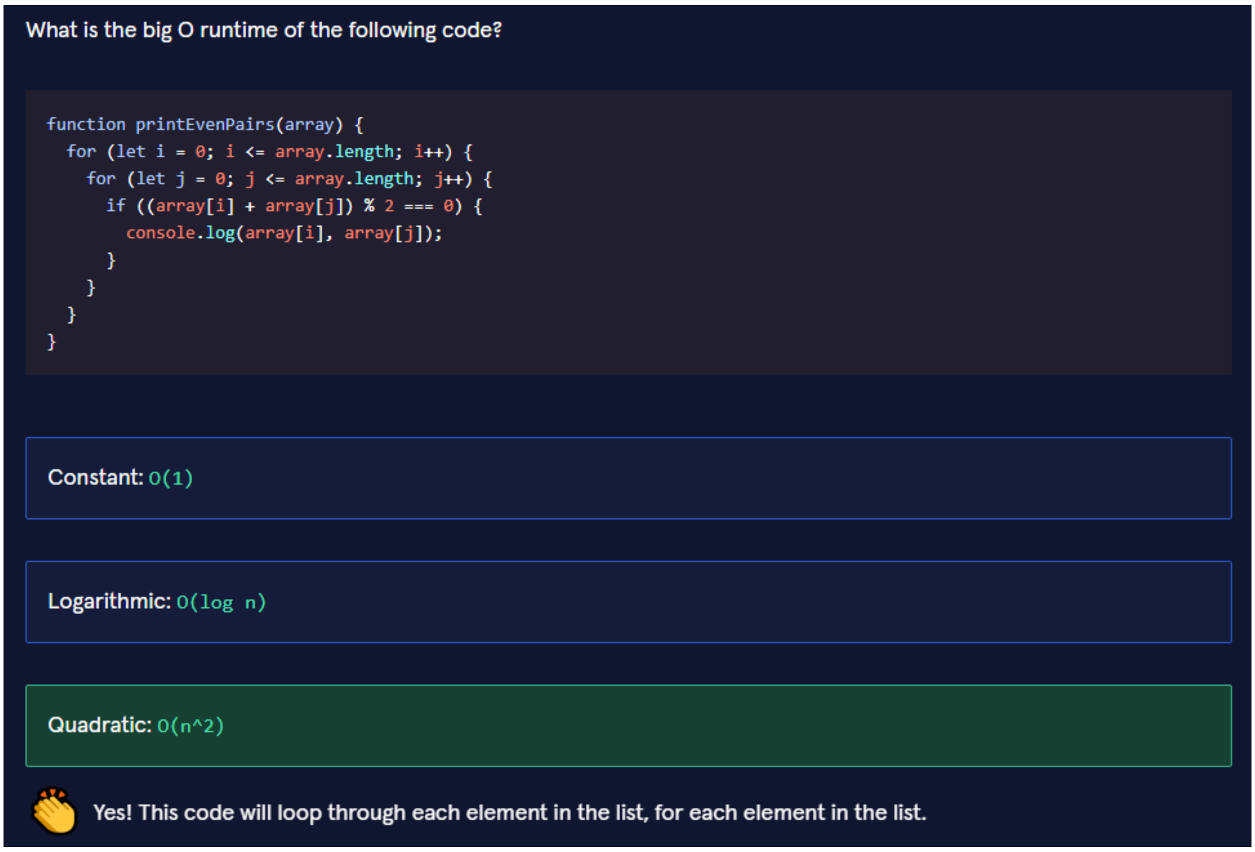
👏 Yes! This has a quadratic runtime of `O(n^2)` due to the nested `for` loops.


Assuming there are no hashing collisions, it’s faster to retrieve an element from a hash map than it is to retrieve an element from a linked list.

[✓ True]

👏 Yes! Retrieving an element from a hash map has a constant big O runtime while removing an element from a linked list has a linear big O runtime.

False



What is the big O runtime of the following code?

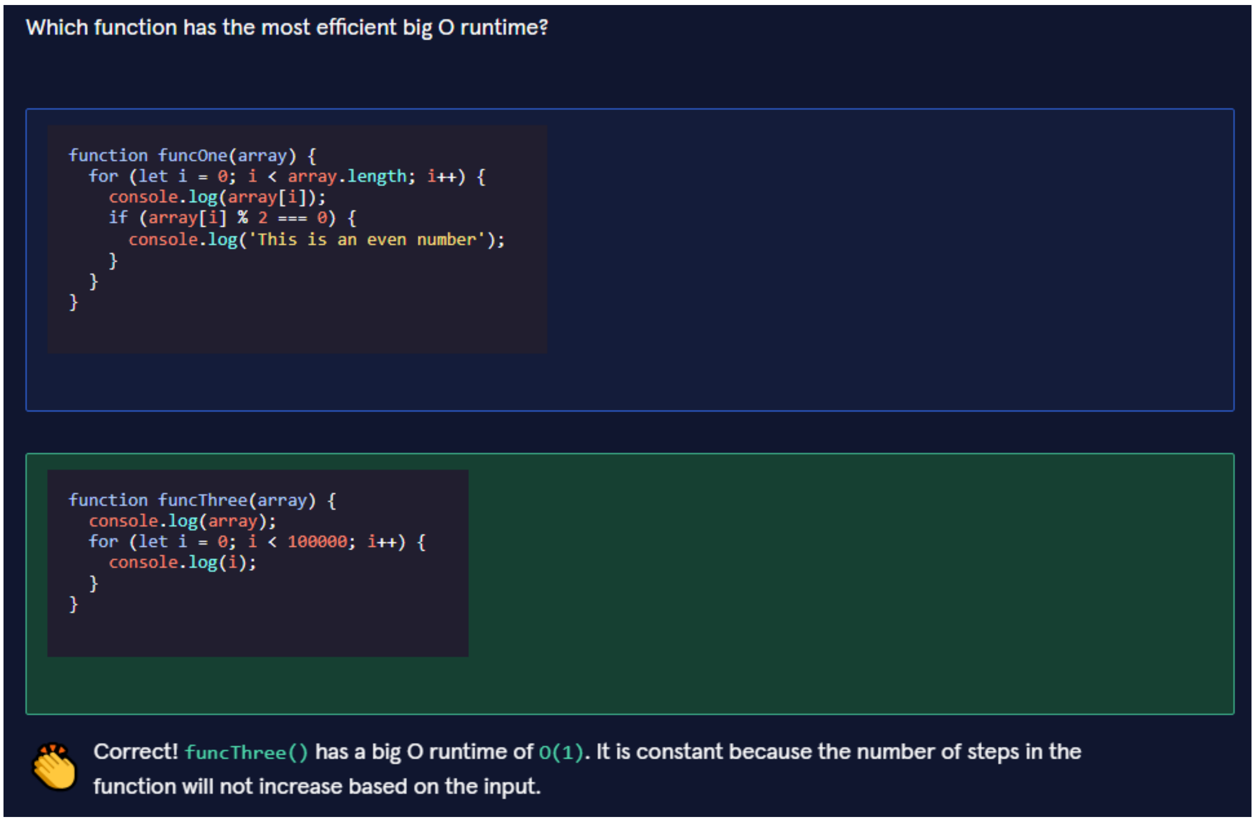
```
function printEvenPairs(array) {
  for (let i = 0; i <= array.length; i++) {
    for (let j = 0; j <= array.length; j++) {
      if ((array[i] + array[j]) % 2 === 0) {
        console.log(array[i], array[j]);
      }
    }
  }
}
```

Constant: `O(1)`

Logarithmic: `O(log n)`

[✓ Quadratic: `O(n^2)`]

Yes! This code will loop through each element in the list, for each element in the list.



Which function has the most efficient big O runtime?

```
function funcOne(array) {
  for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
    if (array[i] % 2 === 0) {
      console.log('This is an even number');
    }
  }
}
```

[✓
```
function funcThree(array) {
  console.log(array);
  for (let i = 0; i < 100000; i++) {
    console.log(i);
  }
}
```
]

Correct! `funcThree()` has a big O runtime of `O(1)`. It is constant because the number of steps in the function will not increase based on the input.