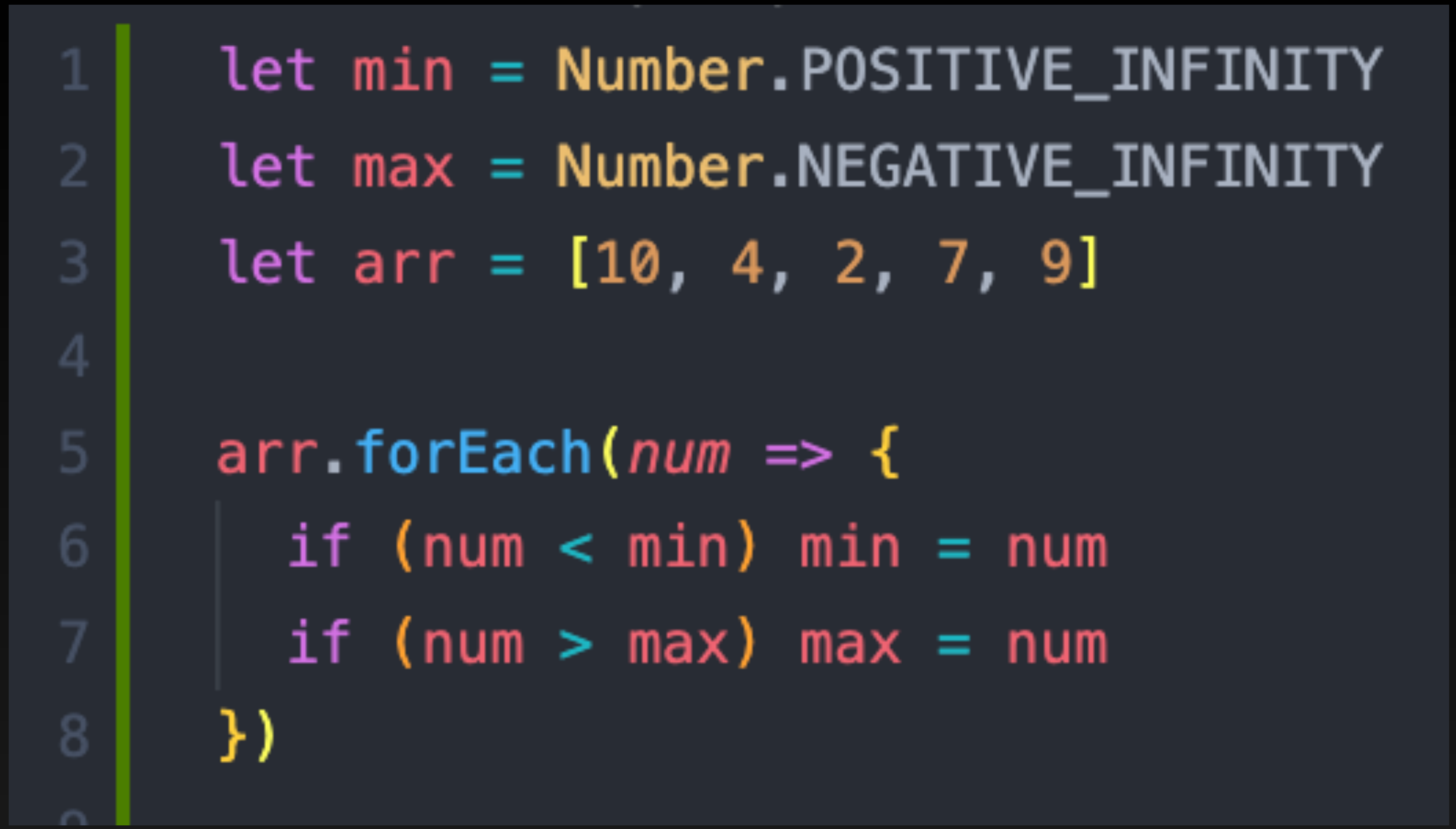
Dropping Constants

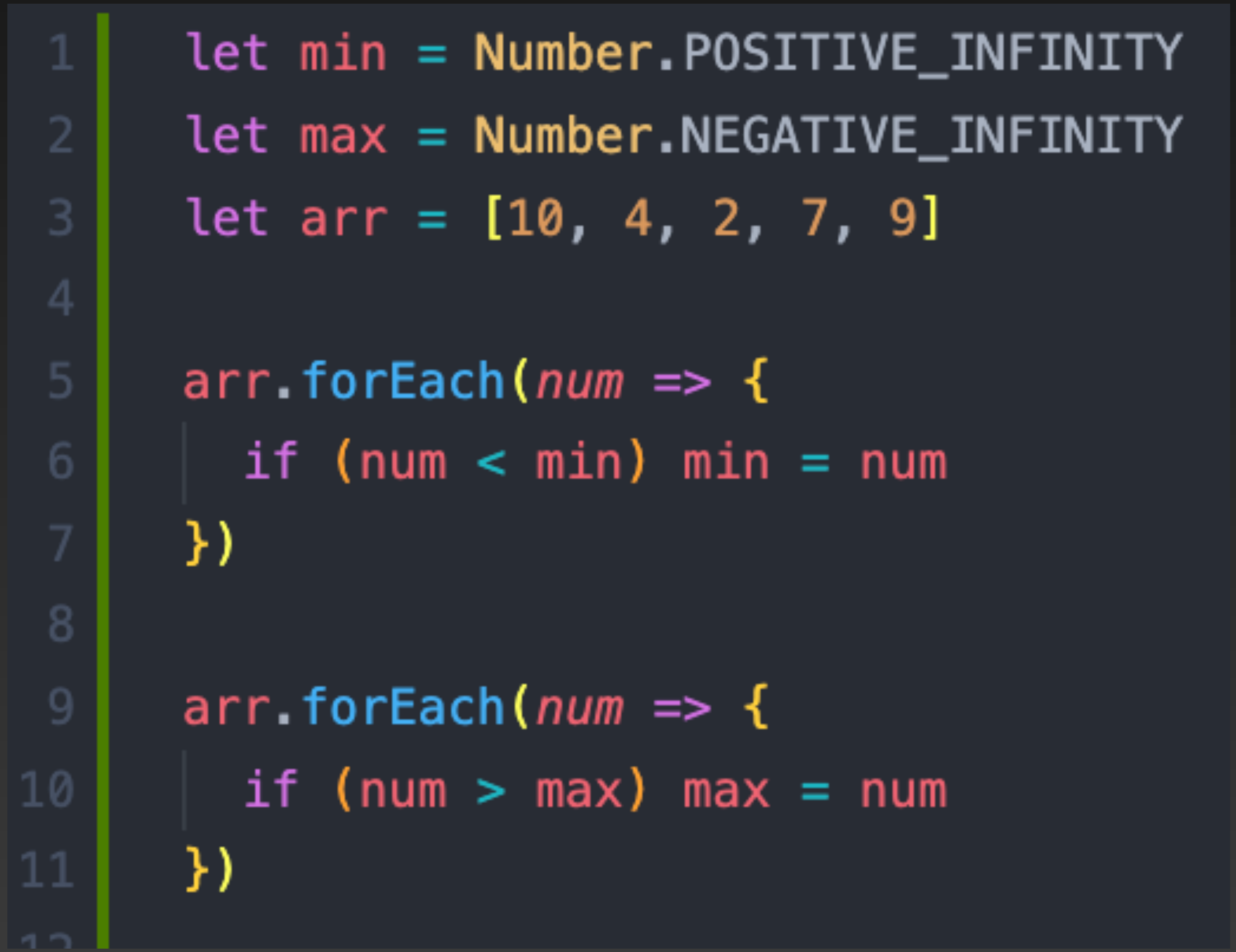
The big idea:

When iterating over the same set of data twice in a single algorithm it may be tempting to label the algorithm as O(2N), but this would be incorrect.

Take these two examples, which one of them is slower?



```javascript
let min = Number.POSITIVE_INFINITY
let max = Number.NEGATIVE_INFINITY
let arr = [10, 4, 2, 7, 9]

arr.forEach(num => {
  if (num < min) min = num
  if (num > max) max = num
})
```

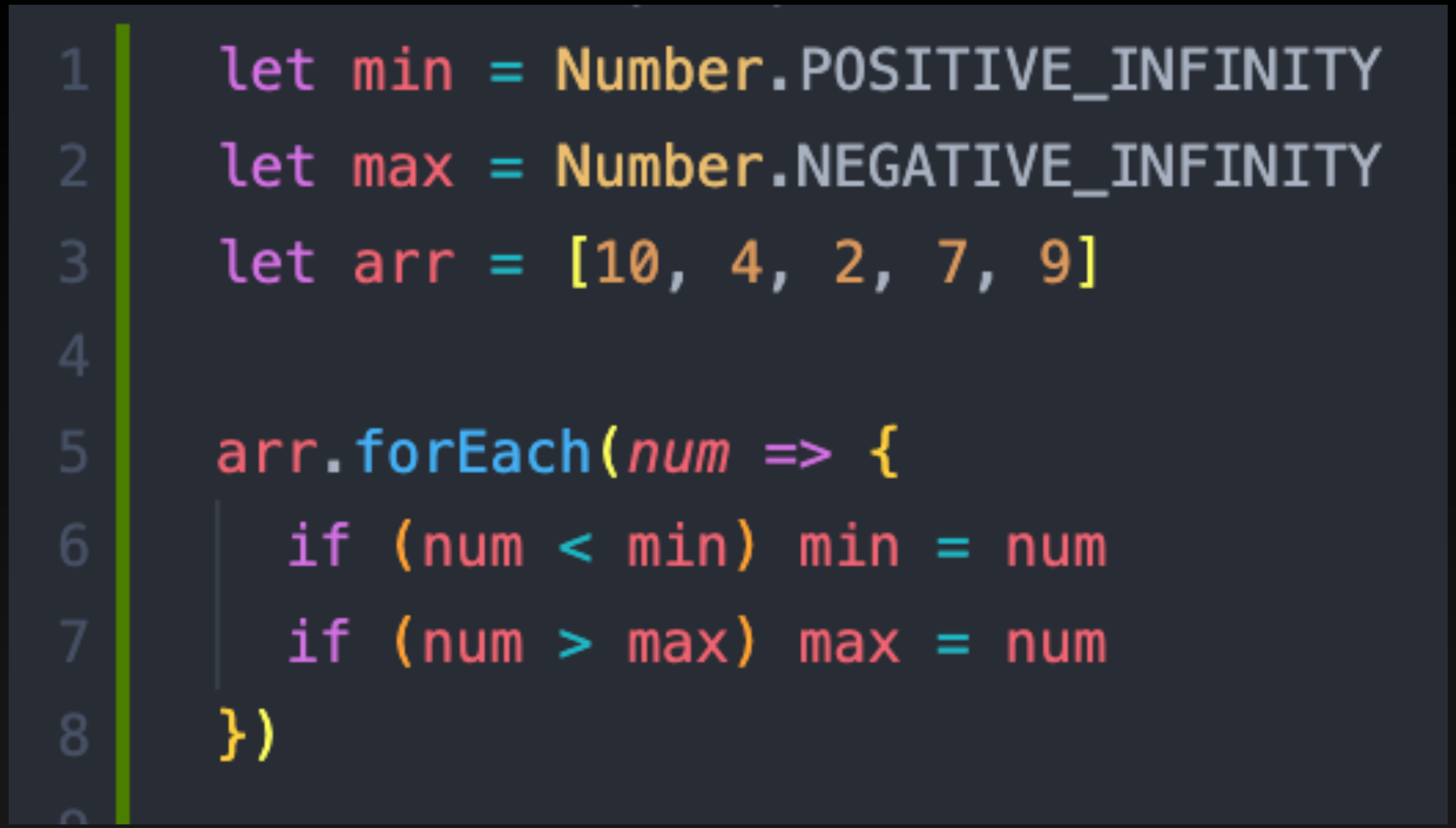


```javascript
let min = Number.POSITIVE_INFINITY
let max = Number.NEGATIVE_INFINITY
let arr = [10, 4, 2, 7, 9]

arr.forEach(num => {
  if (num < min) min = num
})

arr.forEach(num => {
  if (num > max) max = num
})
```

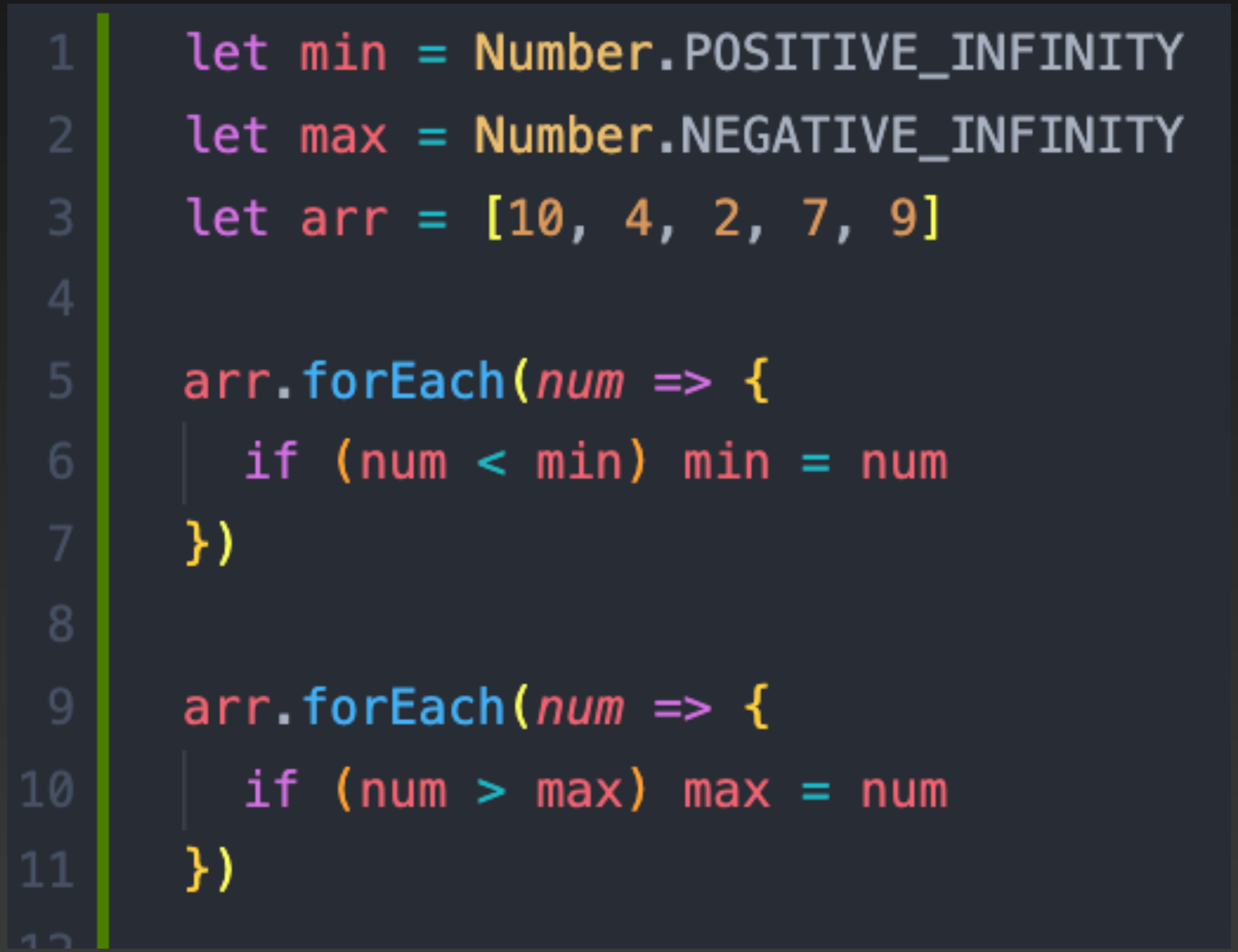
Figuring this out for every algorithm we would write would ultimately be unproductive.

Remember that the ultimate goal of Big O is to determine the major impacts on the runtime of an algorithm as the input scales.

In reality, O(N) algorithms aren't the same as one another, but **they scale in the same way as their inputs grow or shrink**.



```
let min = Number.POSITIVE_INFINITY
let max = Number.NEGATIVE_INFINITY
let arr = [10, 4, 2, 7, 9]

arr.forEach(num => {
  if (num < min) min = num
  if (num > max) max = num
})
```



```
let min = Number.POSITIVE_INFINITY
let max = Number.NEGATIVE_INFINITY
let arr = [10, 4, 2, 7, 9]

arr.forEach(num => {
  if (num < min) min = num
})

arr.forEach(num => {
  if (num > max) max = num
})
```