

BIG O RULEBOOK

Friday, May 1, 2020 4:05 PM


Rule 1: "We always care about what is the worst case scenario."

// BIG_O RULE 1: we always care about what is the worst case scenario.

//for example here:

```
const nemo = ['nemo'];
const everyone = ['dory', 'bruce', 'marlin', 'gill', 'bloat', 'nigel', 'squirt', 'darla', 'hank', 'nemo'];
const large = new Array(100000).fill('nemo');
function findNemo(array) {
  for(let i = 0; i < array.length; i++) {
    console.log('running');

    if(array[i] === 'nemo') {
      console.log('FOUND NEMO!');
      break;
    }
  }
}
findNemo(everyone);
// Runtime: will be O(n) as in worst case 'nemo' will be at last position.
```



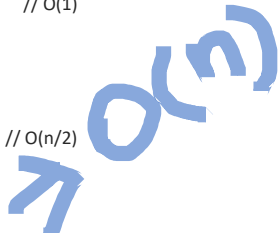
Rule 2: "When n is sufficiently large, we drop the constant term."

// BIG_O RULE 2: when 'n' is sufficiently large as often it is, we drop the constants. if $n/2 \Rightarrow n$. $2n \Rightarrow n$.

// for example here:

```
function printFirstItemThenFirstHalf(items) {
  console.log(items[0]); // O(1)
  var middleIndex = Math.floor(items.length/2);
  var index = 0;

  while(index < middleIndex) { // O(n/2)
    console.log(items[index]);
    index++;
  }
  for(var i = 0; i < 100; i++) { // O(100)
    console.log('hi');
  }
}
printFirstItemThenFirstHalf();
// Runtime: O(1 + n/2 + 100) === O(n), here n is sufficiently large, so drop constants.
```



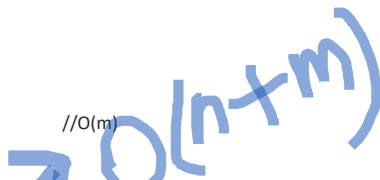
Rule 3: "Just becoz we see 2 for loops doesn't mean that their constant should be dropped or they are looping over the same items. And if they are looping over different items, we can not drop their constant, we need to mention it."

//for example:

```
function compressBoxesTwice(boxes, boxes2) {

  boxes.forEach(function(boxes) { // O(n)
    console.log(boxes);
  });

  boxes2.forEach(function(boxes) { // O(m)
    console.log(boxes);
  });
}
```



```
});
}
compressBoxesTwice(boxes, boxes2);
```

```
// Runtime:  $O(n + m)$ 
```

```
//  $O(n^2)$  or Program that log all pairs of array
```

```
const boxes = [1,2,3,4,5];
```

```
function logAllPairsOfArray(array) {
```

```
  for(let i = 0; i < array.length; i++) {      //  $O(n)$  *
    for(let j = 0; j < array.length; j++) {      //  $O(n)$ 
      console.log(array[i], array[j]);
    }
  }
}
```

$O(n^2)$

```
logAllPairsOfArray(boxes);
```

```
// Runtime:  $O(n * n) === O(n^2)$ 
```

Rule 4: "ignore all non-dominant terms."

```
function printAllNumbersThenAllPairSums(numbers) {
  console.log('these are numbers:');
```

```
  numbers.forEach(function(number) {          // *comment:  $O(n)$ 
    console.log(number);
  });
```

```
  console.log('and these are their sums: ');
```

```
  numbers.forEach(function(firstNumber) {      // *comment:  $O(n*n)$ 
    numbers.forEach(function(secondNumber) {
      console.log(firstNumber + secondNumber);
    });
  });
}
```

```
printAllNumbersThenAllPairSums([1,2,3,4,5];
```

We see that runtime is: $O(n + n^2)$, $n \ll n^2$. So, $O(n + n^2) === O(n^2)$.