

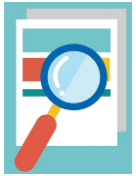


Academlo

Desarrollo Web Full-Stack y Ciencias de la computación

Big O notation

¿Qué es un buen código?



1. Legible



2. Escalable

Big O

¿Qué es un buen código?



```
01101100  
01101111  
01110110  
01100101
```



¿Cómo medir el rendimiento de nuestro código?

```
const { performance } = require('perf_hooks');

const linearSearch = (arr, n) => {

  let t0 = performance.now();

  for(let i = 0; i < arr.length; i++) {
    if ( n === arr[i] ) {
      return i;
    }
  }
  let t1 = performance.now();
  console.log(`la función tomó ${t1-t0} milisegundos en completarse`);
  return -1;
}
```

¿Cuál es la complejidad o **big o notation** de este algoritmo?

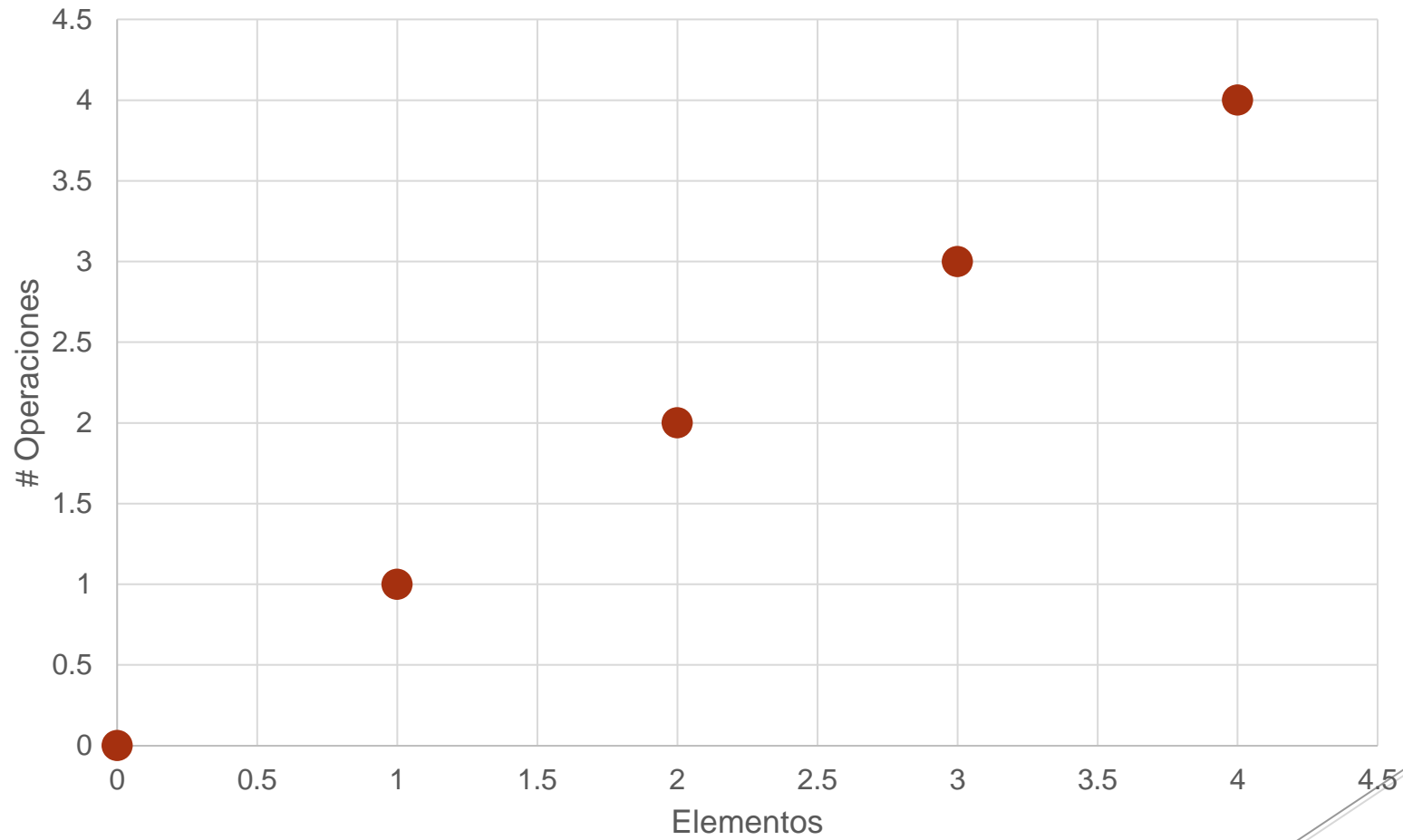
```
const { performance } = require('perf_hooks');

const linearSearch = (arr, n) => {

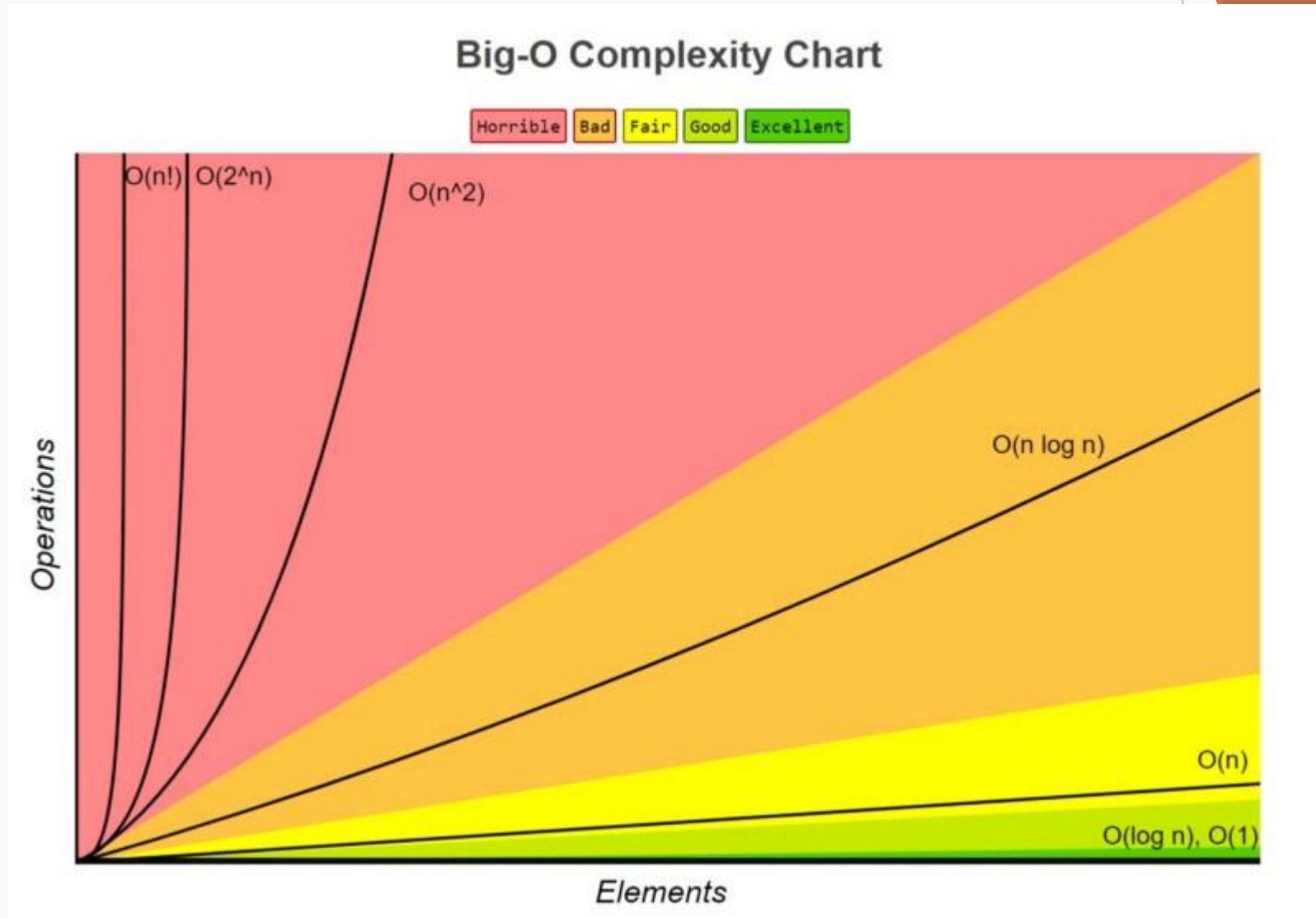
  let t0 = performance.now();

  for(let i = 0; i < arr.length; i++) {
    if ( n === arr[i] ) {
      return i;
    }
  }
  let t1 = performance.now();
  console.log(`la función tomó ${t1-t0} milisegundos en completarse`);
  return -1;
}
```

$O(n)$ o tiempo lineal



Big-O

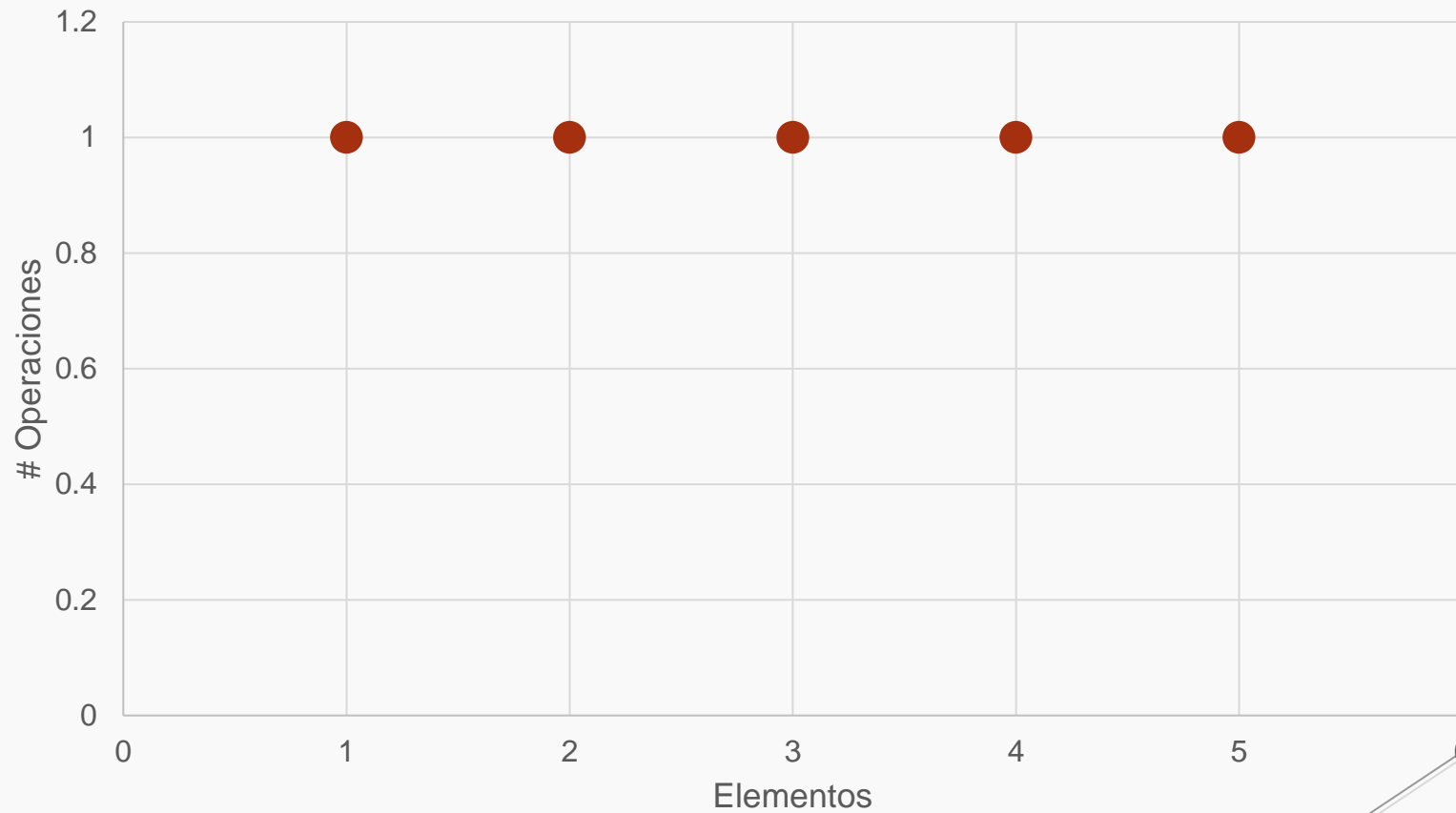


¿Cuál es la complejidad o **big o notation** en esta función?

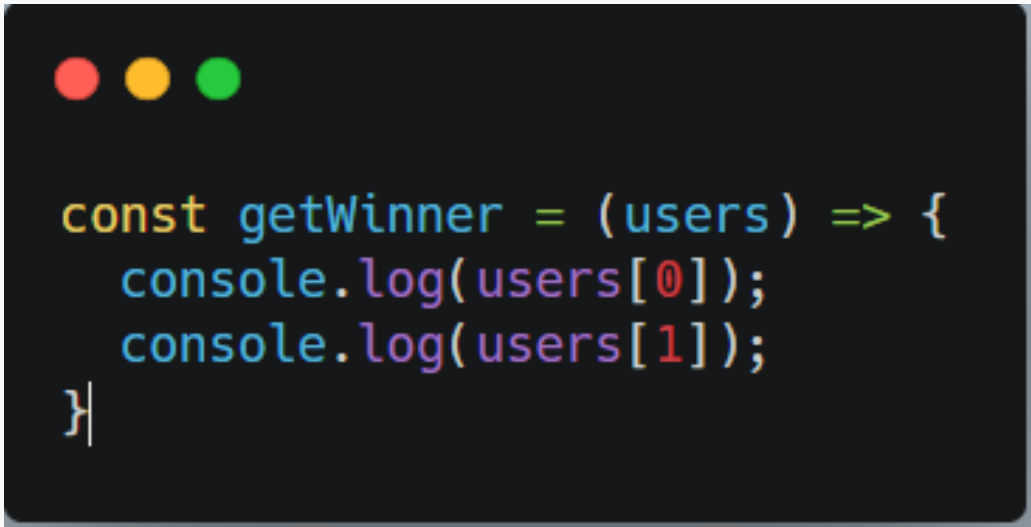


```
const getWinner = (users) => {  
  console.log(users[0]);  
}
```


$O(1)$ o tiempo constante

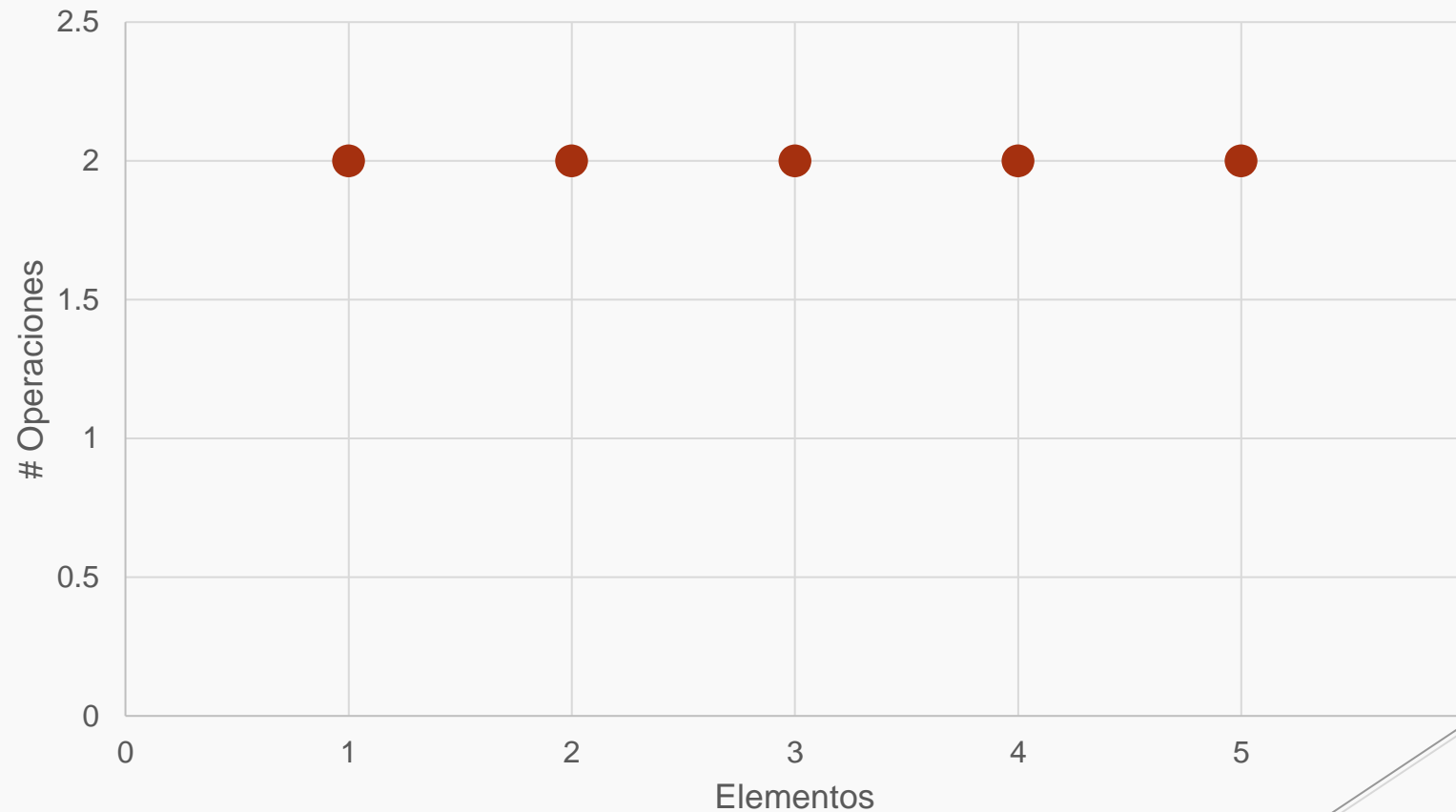


¿Cuál es la complejidad o **big o notation** en esta función?



```
const getWinner = (users) => {  
  console.log(users[0]);  
  console.log(users[1]);  
}
```

¿Cuál es la complejidad o **big o notation** en esta función?



Ejercicio



```
const funChallenge = (input) => {  
  let a = 10;  
  a = 50 + 3;  
  
  for (let i = 0; i < input.length; i++) {  
    anotherFunction();  
    let stranger = true;  
    a++;  
  }  
  return a;  
}
```

Solución

```
const funChallenge = (input) => {  
  let a = 10; // O(1)  
  a = 50 + 3; // O(1)  
  
  for (let i = 0; i < input.length; i++) { // O(n)  
    anotherFunction(); // O(n)  
    let stranger = true; // O(n)  
    a++; // O(n)  
  }  
  return a; // O(1)  
}  
  
funChallenge(); //O(3 + 4n) -> //O(n)  
|
```

Ejercicio

```
function anotherFunChallenge(input) {  
  let a = 5;  
  let b = 10;  
  let c = 50;  
  for (let i = 0; i < input; i++) {  
    let x = i + 1;  
    let y = i + 2;  
    let z = i + 3;  
  
  }  
  for (let j = 0; j < input; j++) {  
    let p = j * 2;  
    let q = j * 2;  
  }  
  let whoAmI = "I don't know";  
}
```

Solución

```
function anotherFunChallenge(input) {  
  let a = 5; //O(1)  
  let b = 10; //O(1)  
  let c = 50; //O(1)  
  for (let i = 0; i < input; i++) {  
    let x = i + 1; //O(n)  
    let y = i + 2; //O(n)  
    let z = i + 3; //O(n)  
  }  
  for (let j = 0; j < input; j++) {  
    let p = j * 2; //O(n)  
    let q = j * 2; //O(n)  
  }  
  let whoAmI = "I don't know"; //O(1)  
}  
  
anotherFunChallenge(); //O(4 + 5n)|
```

Reglas para simplificar Big O notation

- ▶ Regla 1: El peor caso
- ▶ Regla 2: Quitar las constantes
- ▶ Regla 3: Diferentes términos para las entradas (inputs)
- ▶ Regla 4: Soltar los no dominantes

Regla 1: El peor caso



```
1  const linearSearch = (arr, n) => {  
2    for(let i = 0; i < arr.length; i++) {  
3      if ( n === arr[i] ) {  
4        return i;  
5      }  
6    }  
7    return -1;  
8  }  
9  
10 const numeros = [3, 4, 6, 1, 5, 10];  
11  
12 linearSearch(numeros, 6);
```



```
1  const linearSearch = (arr, n) => {  
2    for(let i = 0; i < arr.length; i++) {  
3      if ( n === arr[i] ) {  
4        return i;  
5      }  
6    }  
7    return -1;  
8  }  
9  
10 const numeros = [3, 4, 6, 1, 5, 10];  
11  
12 linearSearch(numeros, 10);
```

Regla 2: Quitar las constantes



```
1  const randomFuncOperation = (items) => {  
2    console.log(items[0]);  
3  
4    let middleIndex = Math.floor(items.length / 2);  
5    let index = 0;  
6  
7    while (index < middleIndex) {  
8      console.log(items[index]);  
9      index++;  
10   }  
11  
12   for (let i = 0; i < 100; i++) {  
13     console.log('hello world');  
14   }  
15 }
```

Regla 2: Quitar las constantes

```
1  const randomFuncOperation = (items) => {  
2    console.log(items[0]);  
3  
4    let middleIndex = Math.floor(items.length / 2);  
5    let index = 0;  
6  
7    while (index < middleIndex) {  
8      console.log(items[index]);  
9      index++;  
10   }  
11  
12   for (let i = 0; i < 100; i++) {  
13     console.log('hello world');  
14   }  
15 }  
16  
17 //O(1 + n/2 + 100) -> O(n)
```

Regla 2: Quitar las constantes



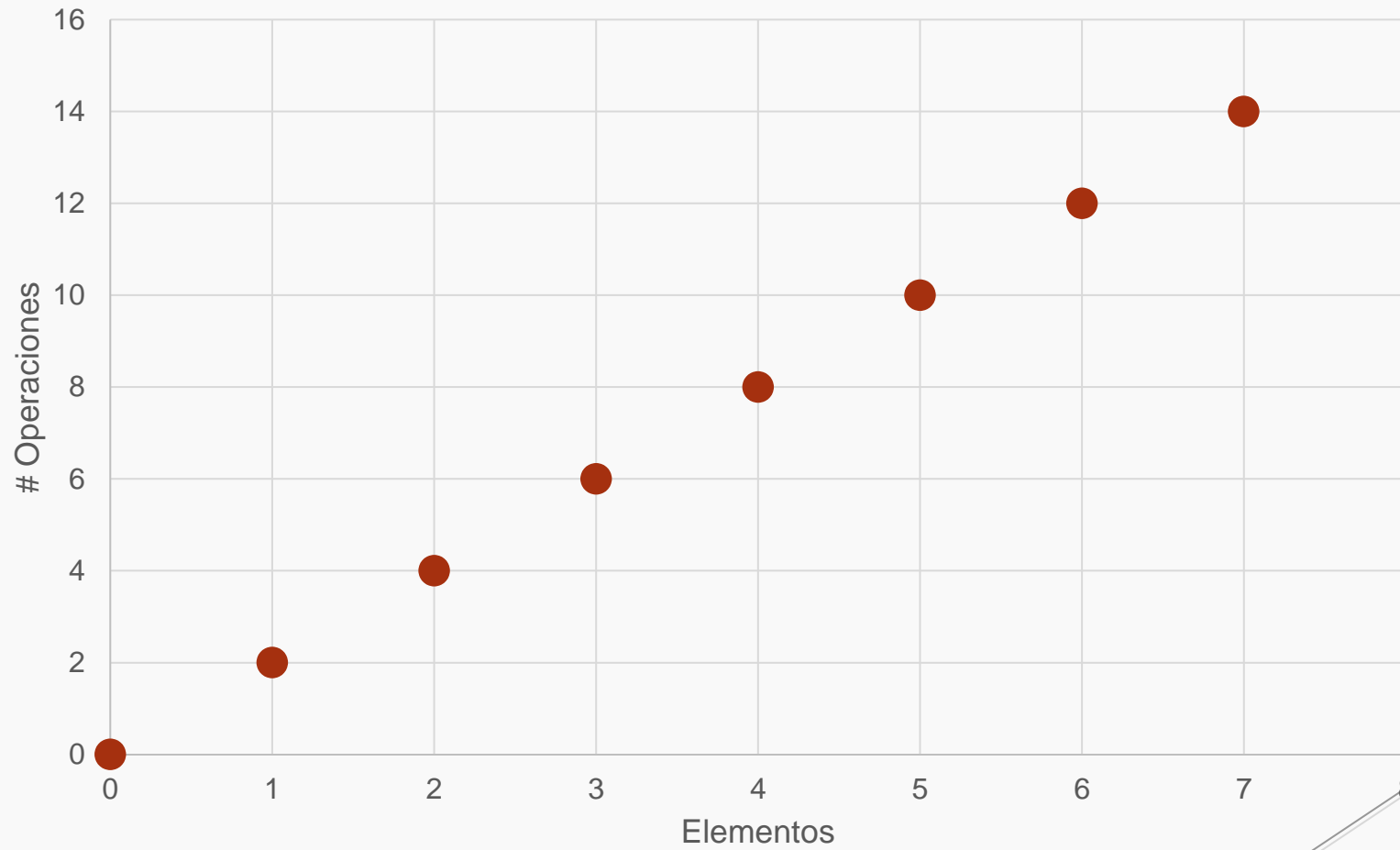
```
1  const randomFuncOperation = (items) => {  
2    items.forEach(item => console.log(`Primer ciclo ${item}`));  
3  
4    items.forEach(item => console.log(`Segundo ciclo ${item}`));  
5  }
```

Regla 2: Quitar las constantes



```
1  const randomFuncOperation = (items) => {  
2    items.forEach(item => console.log(`Primer ciclo ${item}`));  
3  
4    items.forEach(item => console.log(`Segundo ciclo ${item}`));  
5  }  
6  
7  //O(2n) -> O(n)
```

$O(n)$ o tiempo lineal



Regla 3: Diferentes términos para las entradas (inputs)



```
1  const randomFuncOperation = (items, items2) => {  
2    items.forEach(item => console.log(`Primer ciclo ${item}`));  
3  
4    items2.forEach(item => console.log(`Segundo ciclo ${item}`));  
5  }
```

Regla 3: Diferentes términos para las entradas (inputs)



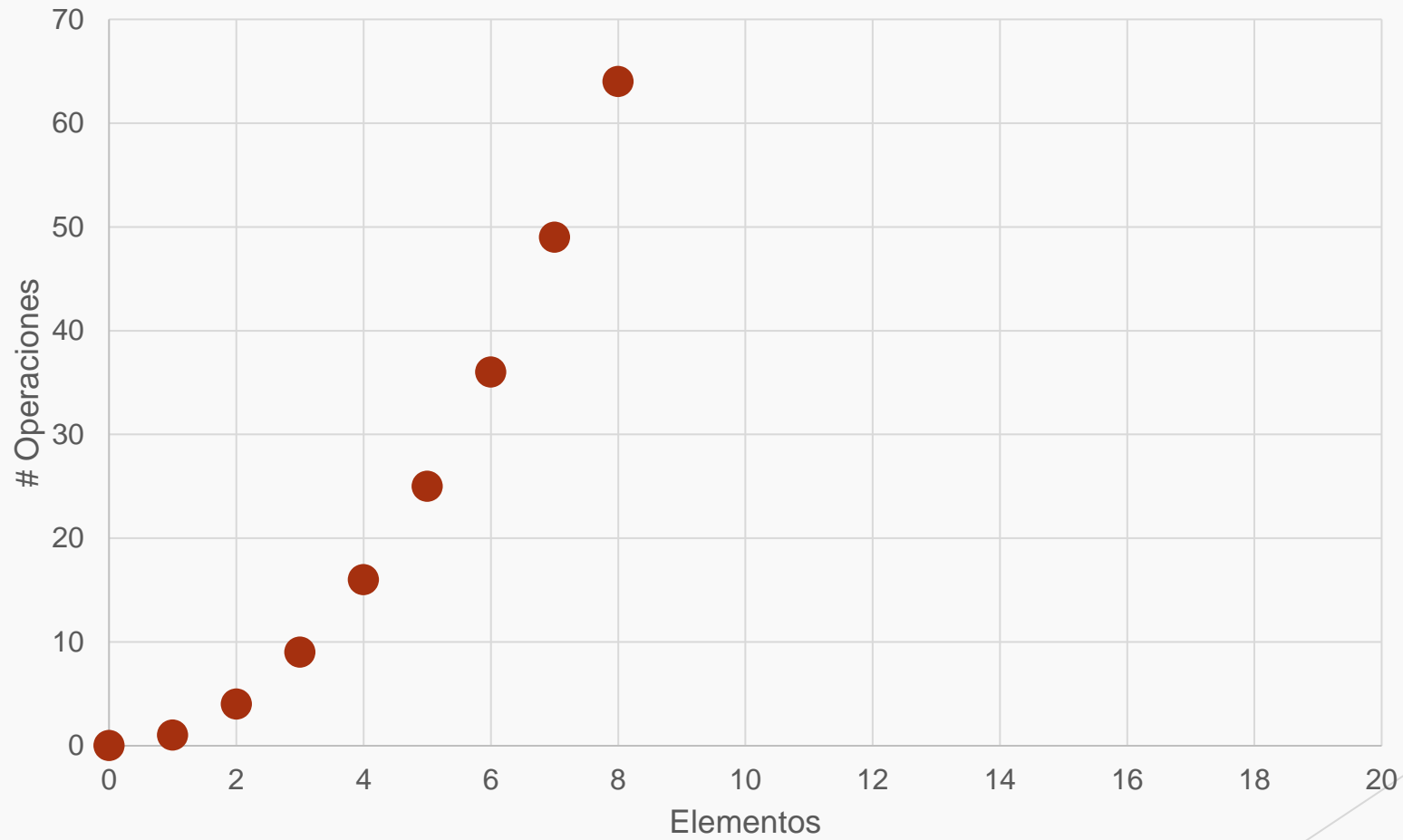
```
1  const randomFuncOperation = (items, items2) => {  
2    items.forEach(item => console.log(`Primer ciclo ${item}`));  
3  
4    items2.forEach(item => console.log(`Segundo ciclo ${item}`));  
5  }  
6  
7  //O(a + b)
```


¿Qué complejidad tiene el siguiente algoritmo?



```
1  const randomFuncOperation = (items) => {  
2    for (let i = 0; i < items.length; i++) {  
3      for (let j = 0; j < items.length; j++) {  
4        console.log(items[i], items[j]);  
5      }  
6    }  
7  }
```

$O(n^2)$ – Tiempo cuadrático



Regla 4: Soltar los no dominantes



```
1  const randomFuncOperation = (items) => {  
2    items.forEach(item => console.log(items));  
3  
4    for (let i = 0; i < items.length; i++) {  
5      for (let j = 0; j < items.length; j++) {  
6        console.log(items[i] + items[j]);  
7      }  
8    }  
9  }
```

Regla 4: Soltar los no dominantes



```
1  const randomFuncOperation = (items) => {  
2    items.forEach(item => console.log(item));  
3  
4    for (let i = 0; i < items.length; i++) {  
5      for (let j = 0; j < items.length; j++) {  
6        console.log(items[i] + items[j]);  
7      }  
8    }  
9  }  
10 randomFuncOperation(numeros); //O(n + n*n) -> O(n + n^2) -> O(n^2)
```

Regla 4: Soltar los no dominantes



```
1 randomFuncOperation(numeros); //O(3n + n^2 + 100 + n/2) -> ¿Cuál es el más importante?
```

Regla 4: Soltar los no dominantes



```
1 randomFuncOperation(numeros); //O(3n + n^2 + 100 + n/2) -> ¿Cuál es el más importante?  
2 // 500  
3 //O(1500 + 250000 + 100 + 250) -> ¿Cuál es el más importante?
```

Resumen (Cheat Sheet)

- ▶ **$O(1)$** Constantes – sin ciclos (no loops)
- ▶ * **$O(\log N)$** Logaritmico – Usualmente en algoritmos de busquedas (Binary Search)
- ▶ **$O(n)$** Lineal - Para ciclos (loops), ciclos while recorriendo n items
- ▶ * **$O(n \log(n))$** Log Lineal – Operaciones para ordenar
- ▶ * **$O(n^2)$** Cuadratico – Por cada elemento en una lista, se necesita comparar con otro elemento. Dos ciclos anidados
- ▶ * **$O(2^n)$** Exponencial – Algoritmos recursivos que resuelven problemas de tamaño N
- ▶ **$O(n!)$** Factorial – Estás agregando un ciclo por cada elemento.