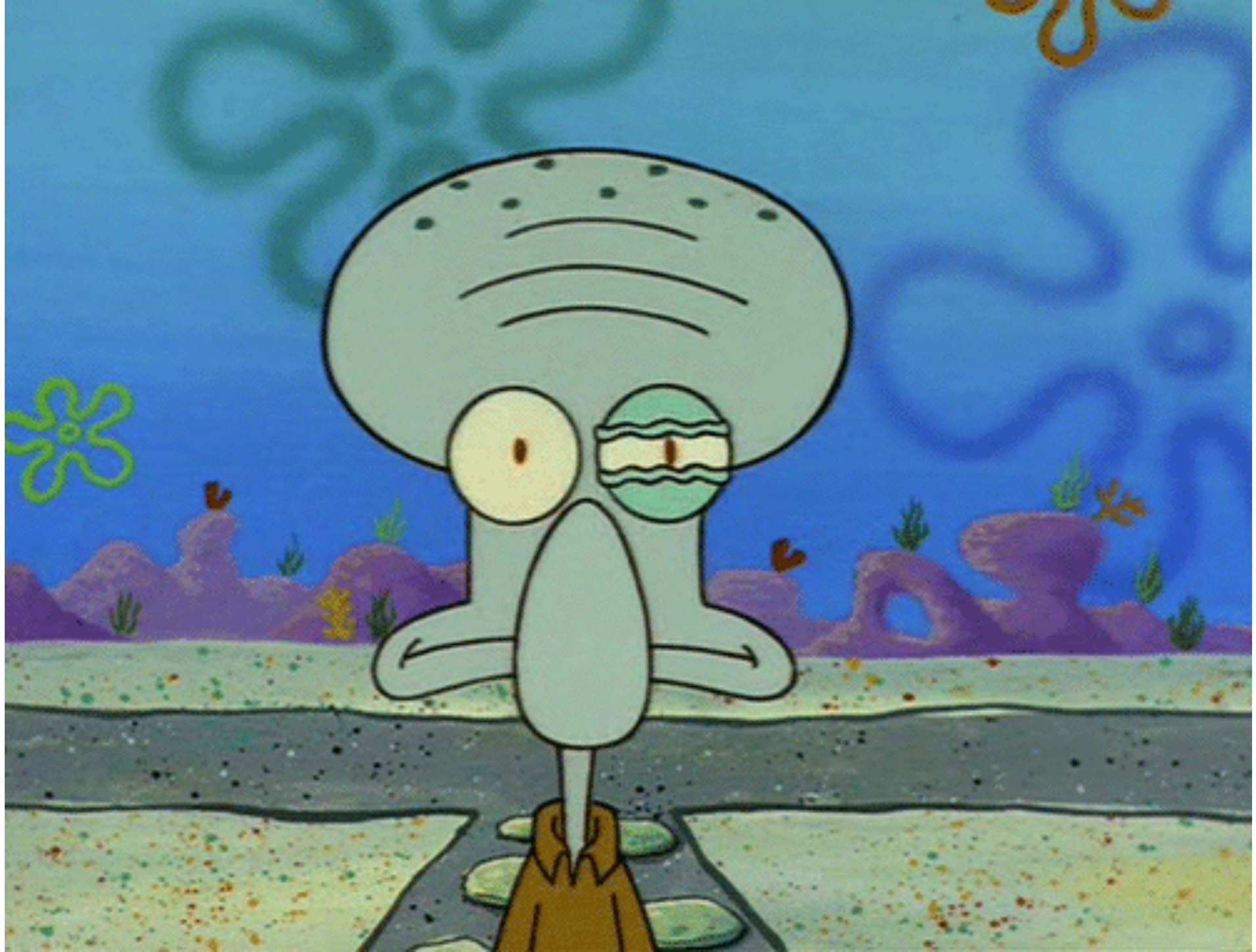
Big O para Devs JavaScript: Porque Seu Código Bonito Pode Ser Lento

Gustavo Caetano

Big O



- ▶16 anos de experiência
- ▶Mestre em Ciência da Computação
- ▶Google Developer Group’s
- ▶Professor
- ▶Especialista Android no Banco Inter
- ▶Canal Gustavo Caetano 3MM de views
- ▶Papinho Tech - O melhor podcast tech do Brasil*

* Fonte: Mamãe

Até o final da palestra eu vou te mostrar como ganhar pontos com o seu chefe e ser um programador melhor.

(Ou menos pior)

Então calma.

Big O é tipo a régua que mede quão rápido seu código piora conforme o problema cresce.

“Não precisa ser PhD em matemática, é só saber quando seu algoritmo é um patinete e quando é uma Ferrari.”

Carlos Drummond de Andrade

```
1 const first = arr[0]; // acesso direto
```

$$O(1)$$

Constante

```
1 const array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
2
3 array.forEach(num => console.log(num));
```

$$O(n)$$

$O(n)$

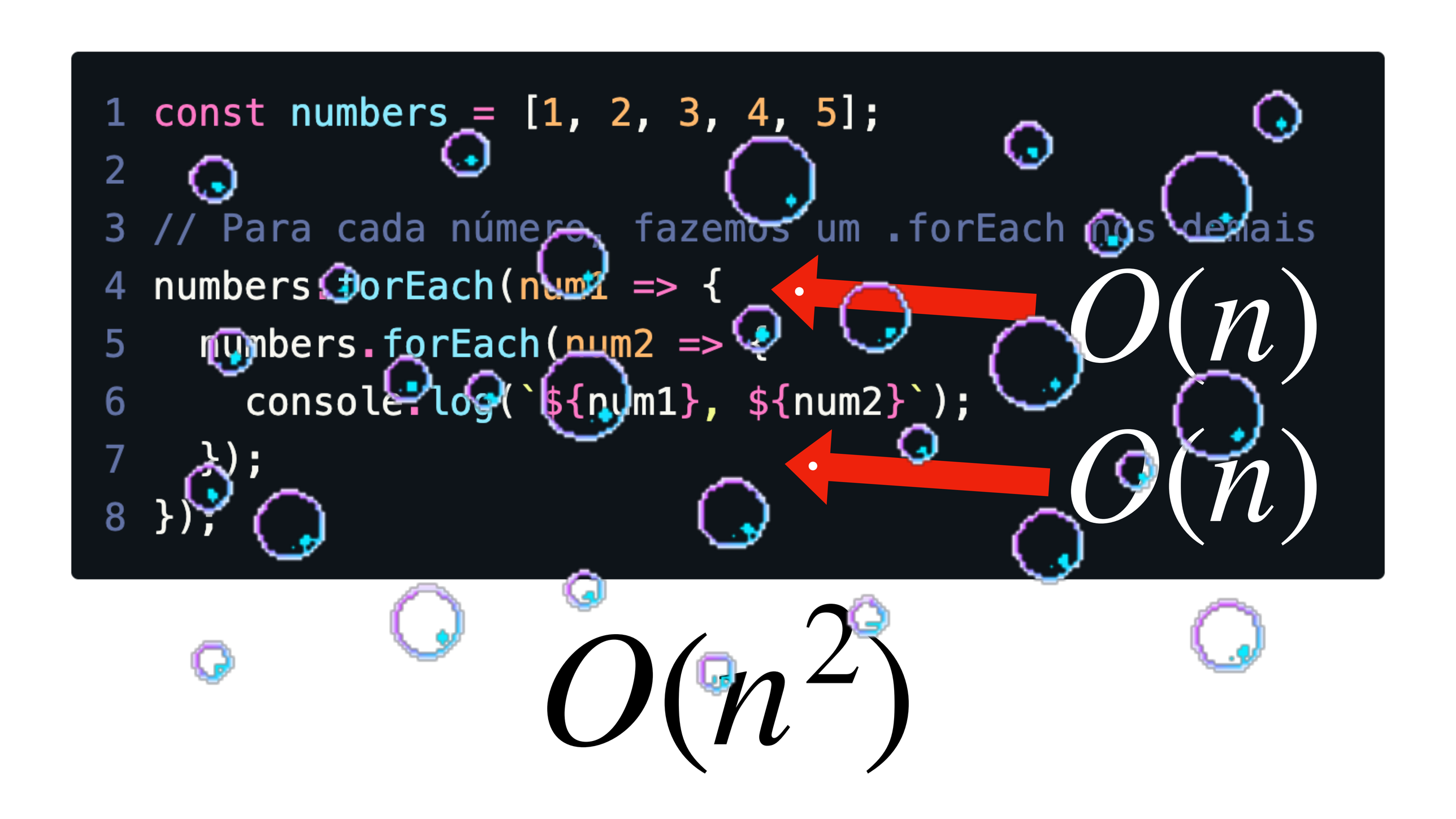
$n = 10 \rightarrow O(10)$

$n = 100 \rightarrow O(100)$

$n = 1000 \rightarrow O(1000)$

Crescimento linear

```
1 const numbers = [1, 2, 3, 4, 5];
2
3 // Para cada número, fazemos um .forEach nos demais
4 numbers.forEach(num1 => {
5   numbers.forEach(num2 => {
6     console.log(`${num1}, ${num2}`);
7   });
8 });
```

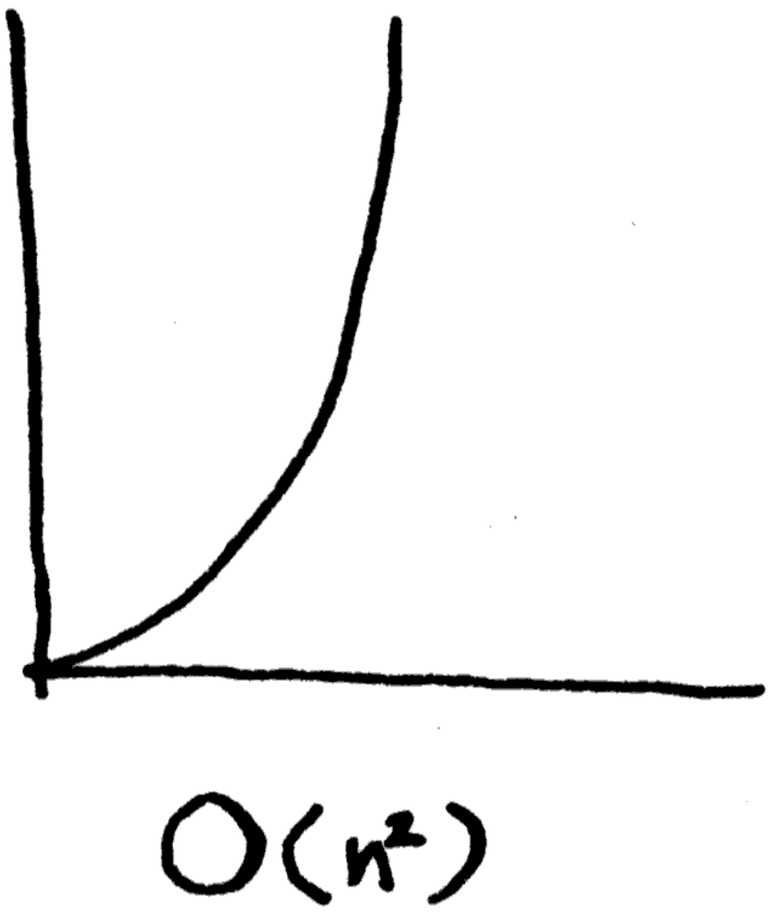


$O(n^2)$

$n = 10 \rightarrow O(100)$

$n = 100 \rightarrow O(1000)$

$n = 1000 \rightarrow O(10000)$



Crescimento quadrático

```javascript
// Busca binária
function busca(arr, target) {
  let left = 0, right = arr.length - 1;

  while (left <= right) {
    const mid = Math.floor((left + right) / 2);
    if (arr[mid] === target) return mid;
    if (arr[mid] < target) left = mid + 1;
    else right = mid - 1;
  }
};
```

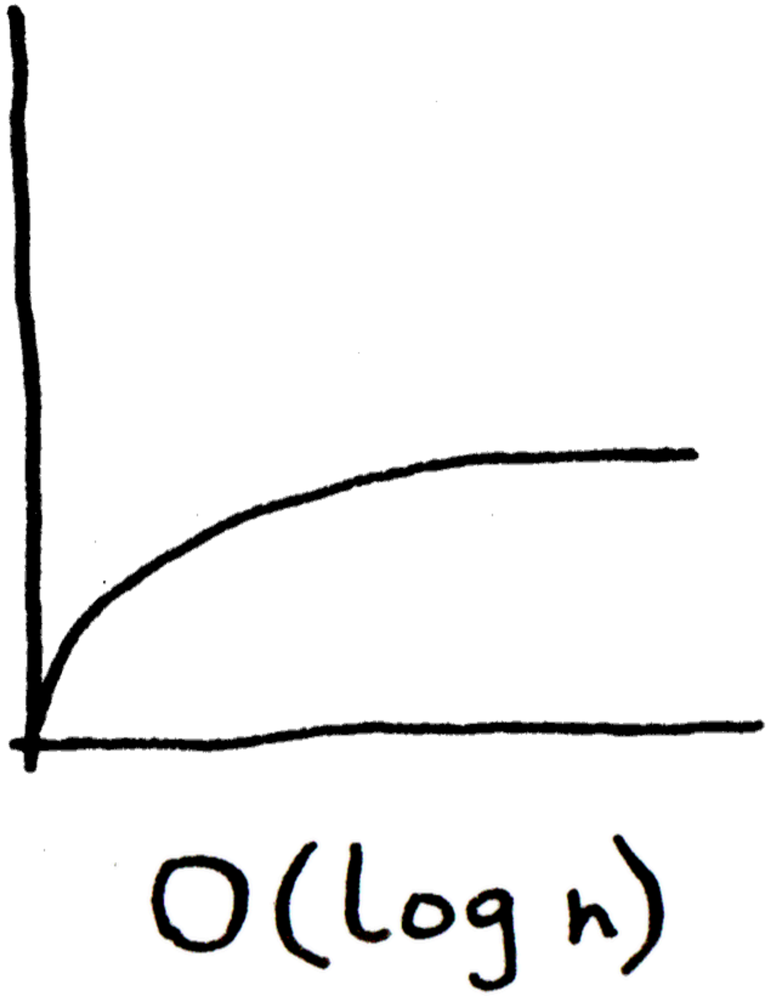
$$O(\log n)$$

$O(log\ n)$

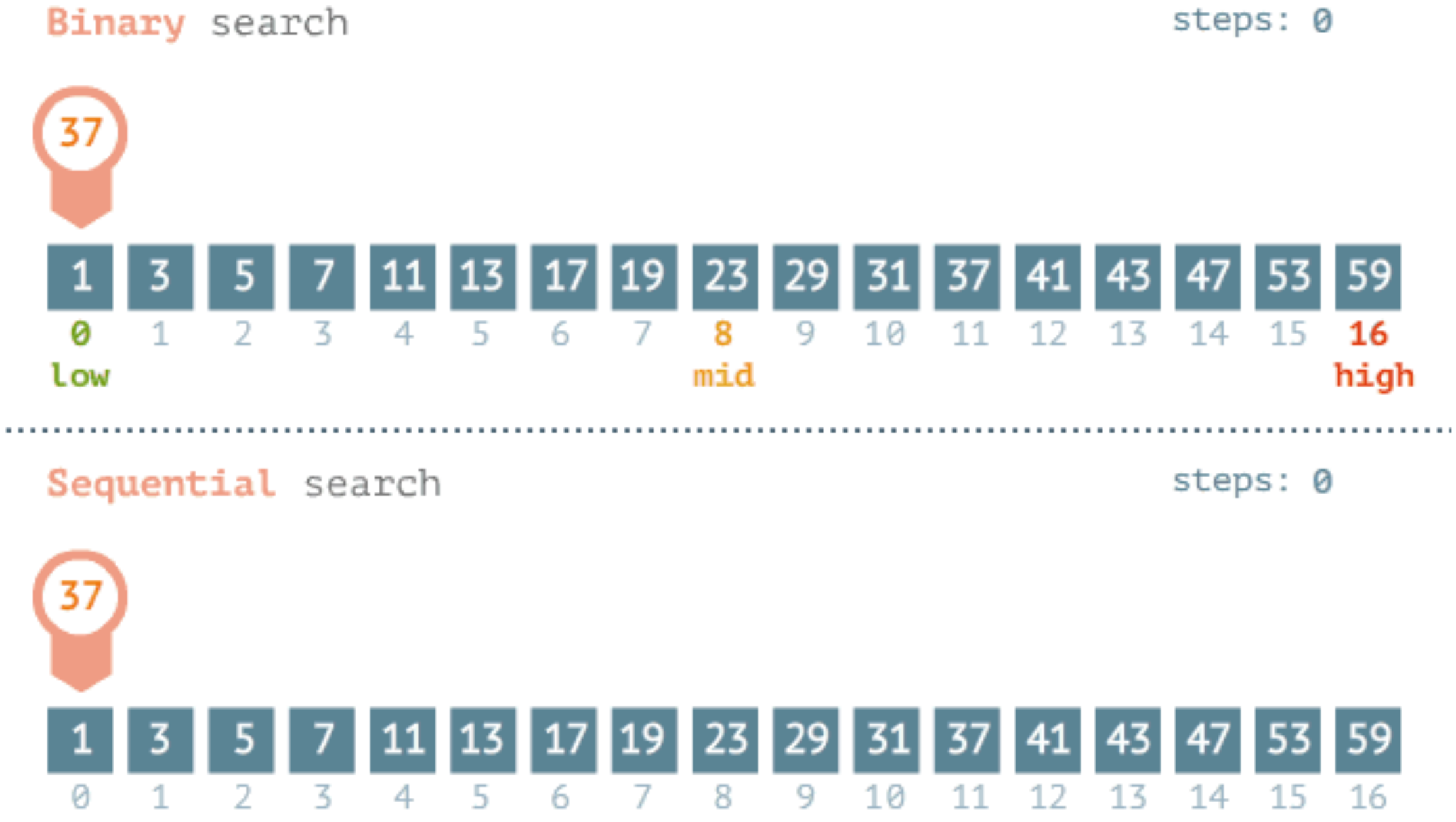
$n = 10 \rightarrow O(3)$

$n = 100 \rightarrow O(6)$

$n = 1000 \rightarrow O(9)$



Crescimento logarítmico


Binary search
steps: 0
37
1 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
low
mid
high
Sequential search
steps: 0
37
1 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
1 const array = [9, 7, 6, 3, 8, 4, 2, 1, 0];
2
3 array.sort();
```

$$O(n \log n)$$

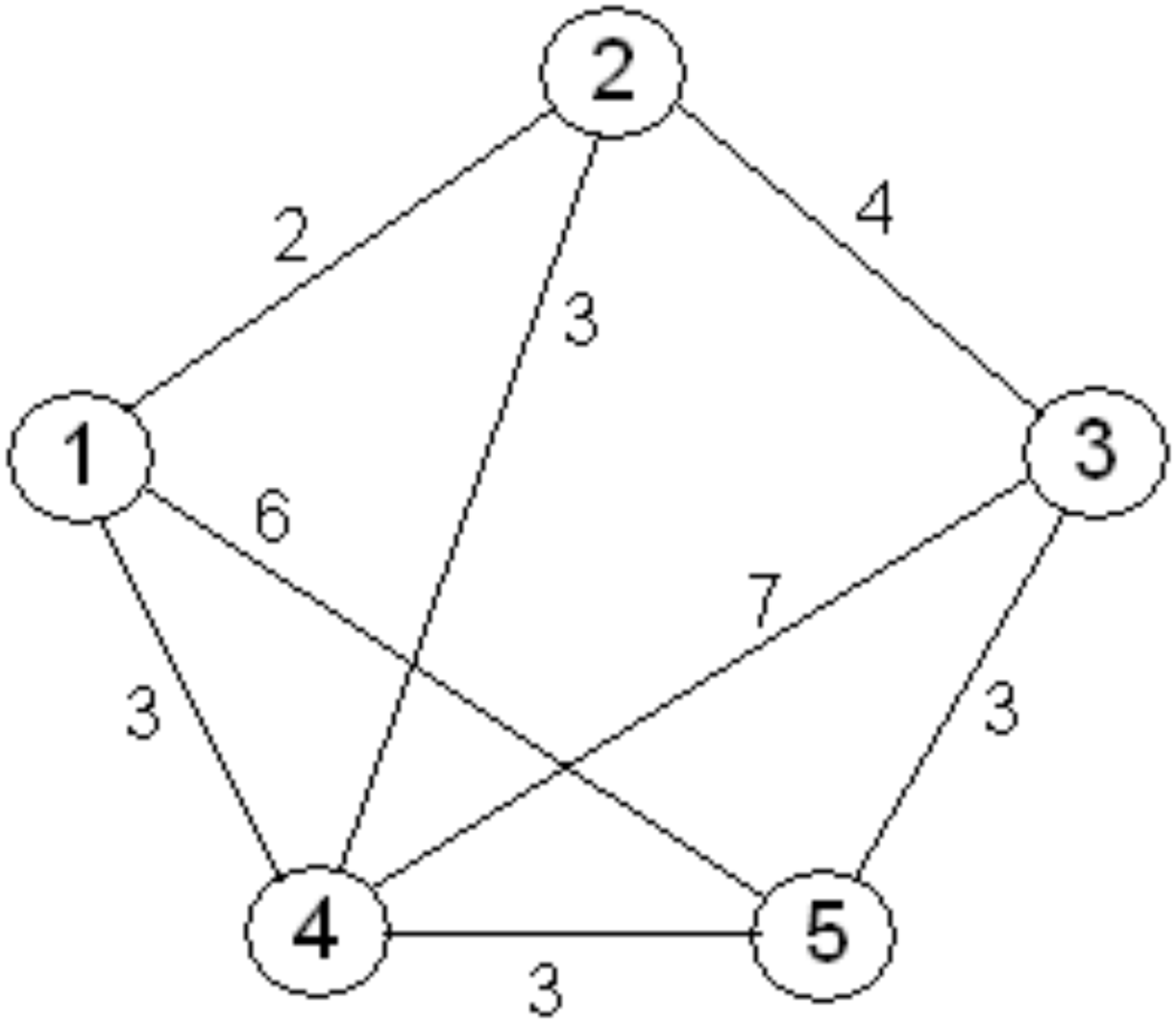
Quick Sort

$O(n \log n)$

$n = 10 \rightarrow O(30)$

$n = 100 \rightarrow O(60)$

$n = 1000 \rightarrow O(90)$



$O(n!)$

$n = 5 \rightarrow 120$

$n = 6 \rightarrow 720$

$n = 7 \rightarrow 5040$

$n = 20 \rightarrow 2432902008176640000$

Algoritmo de Dijkstra

Breaking the Sorting Barrier for Directed Single-Source Shortest Paths

Ran Duan * Jiayi Mao * Xiao Mao † Xinkai Shu ‡ Longhui Yin *

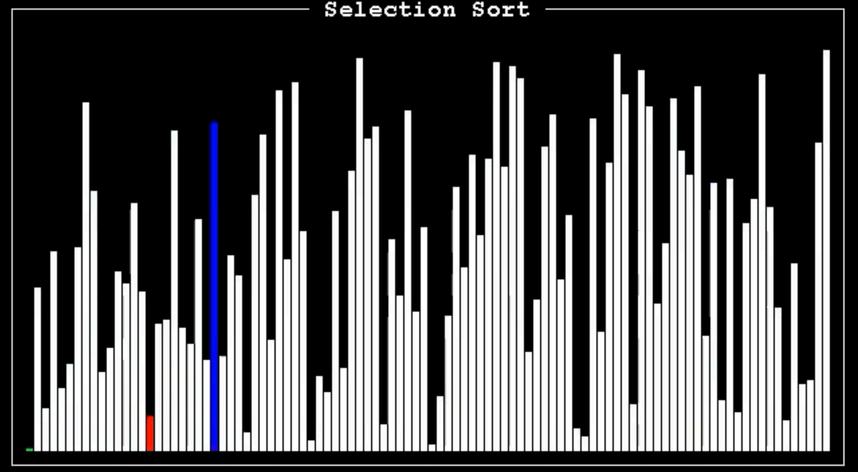
July 31, 2025

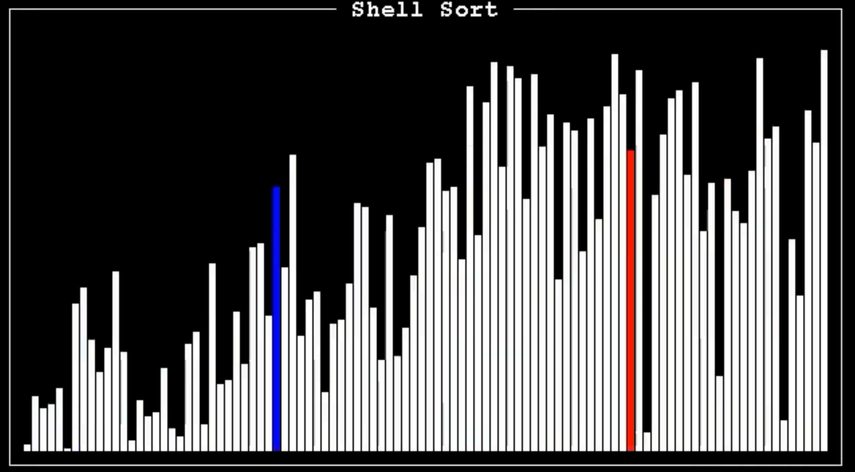
Abstract

We give a deterministic $O(m \log^{2/3} n)$-time algorithm for single-source shortest paths (SSSP) on directed graphs with real non-negative edge weights in the comparison-addition model. This is the first result to break the $O(m + n \log n)$ time bound of Dijkstra's algorithm on sparse graphs, showing that Dijkstra's algorithm is not optimal for SSSP.

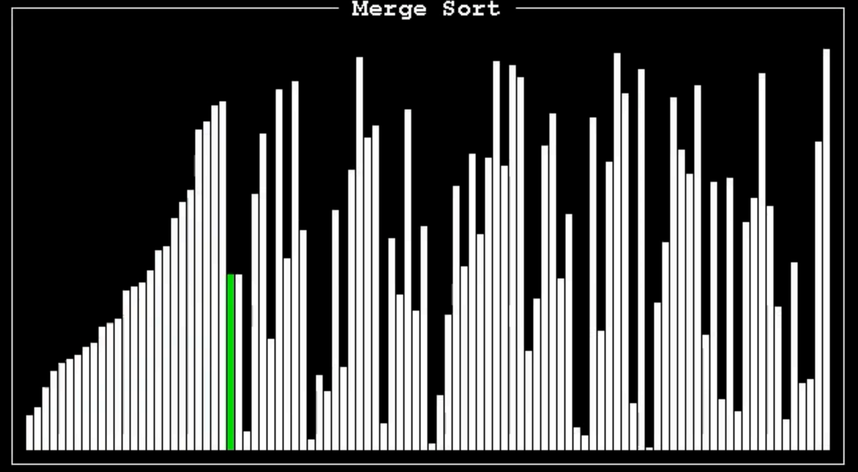
1 Introduction

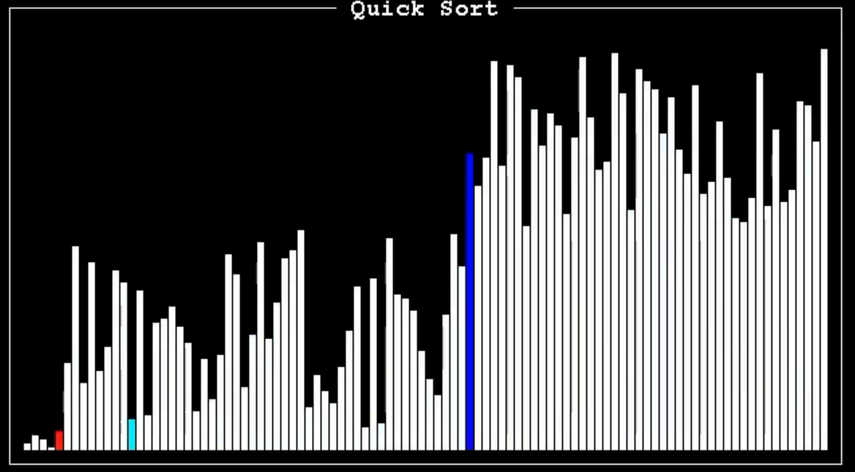
In an m-edge n-vertex directed graph $G = (V, E)$ with a non-negative weight function $w : E \to \mathbb{R}_{\geq 0}$, single-source shortest path (SSSP) considers the lengths of the shortest paths from a source vertex s to all $v \in V$. Designing faster algorithms for SSSP is one of the most fundamental problems in graph theory, with exciting improvements since the 50s.

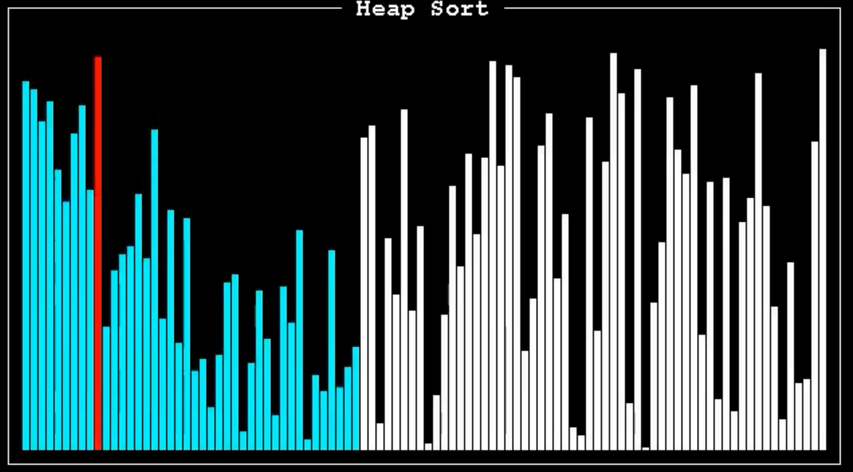

Selection Sort

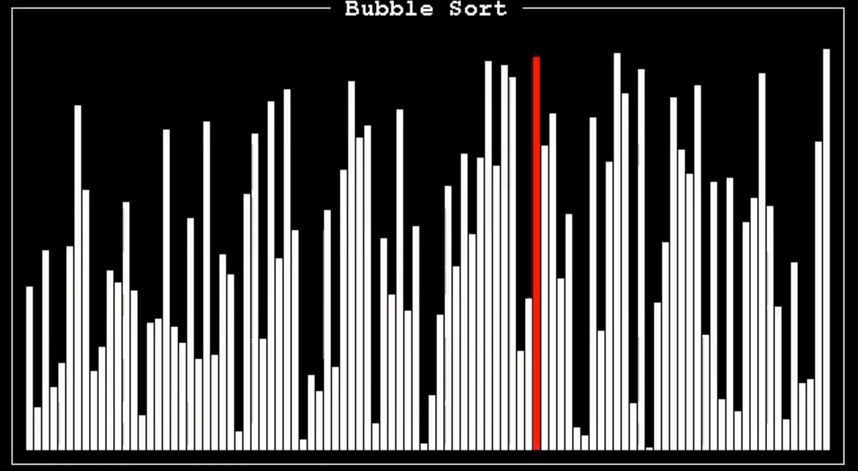

Shell Sort

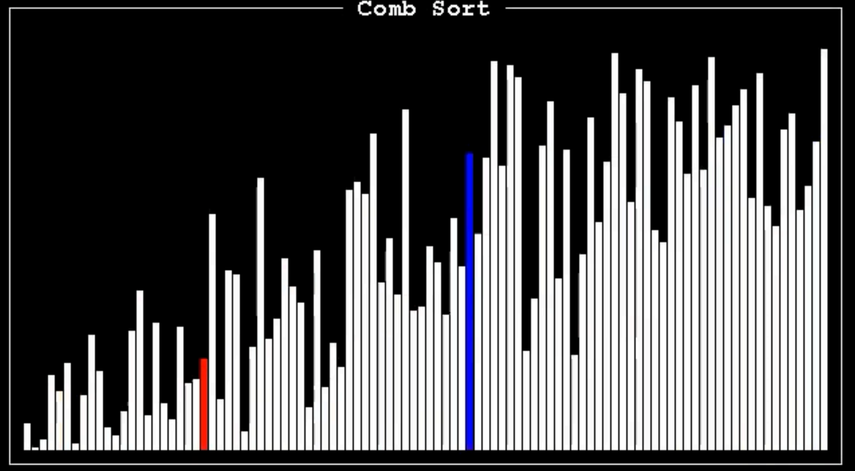

Insertion Sort


Merge Sort

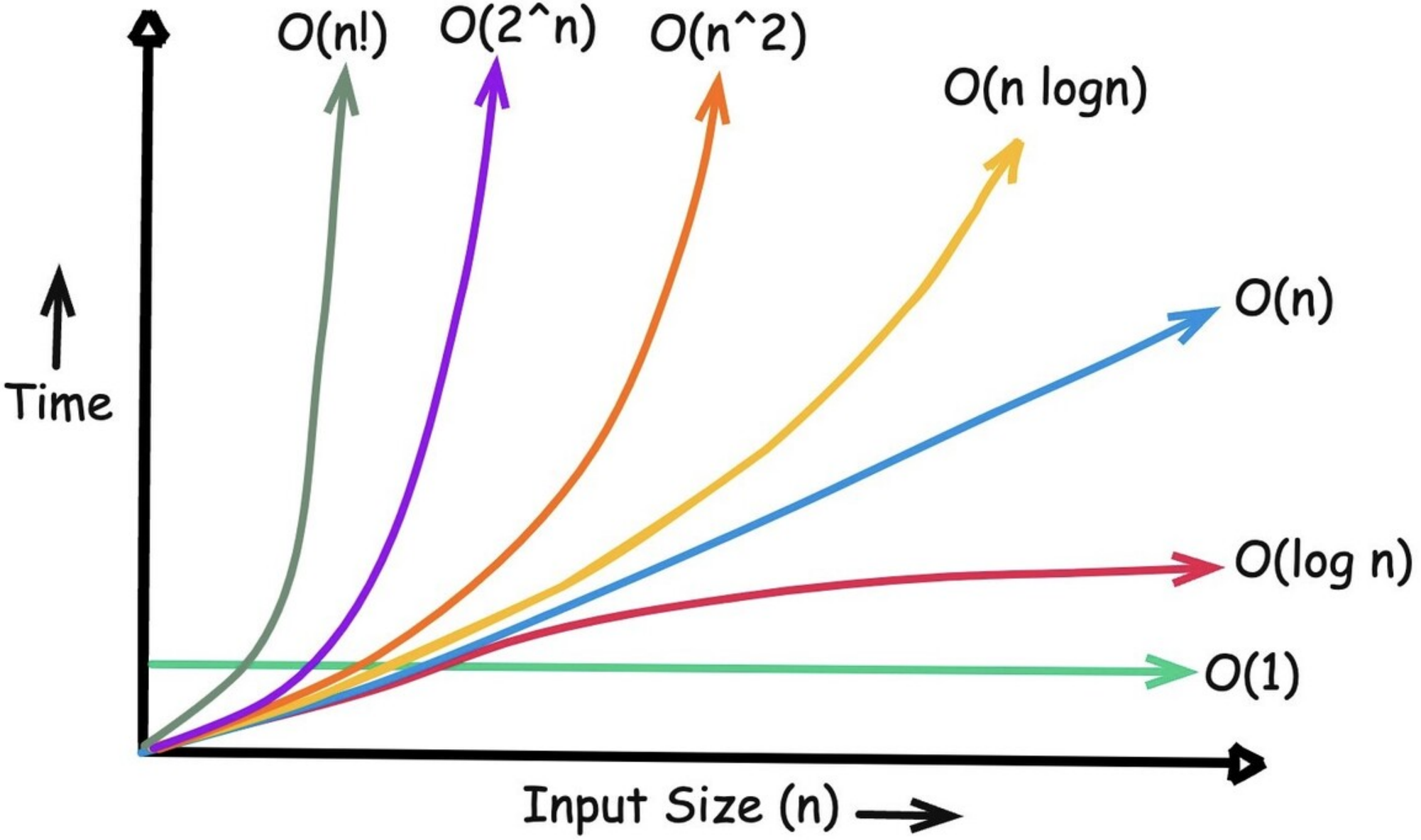

Quick Sort

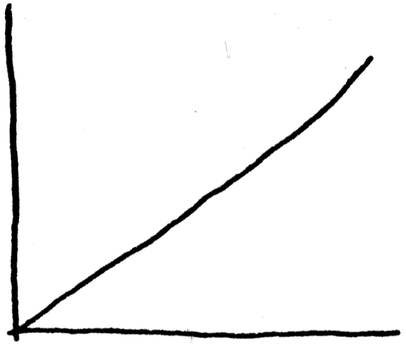
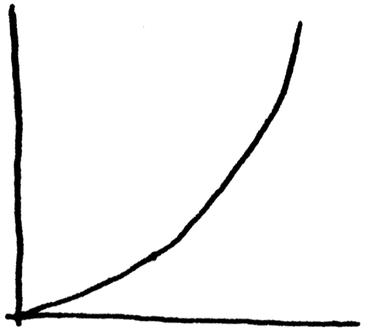
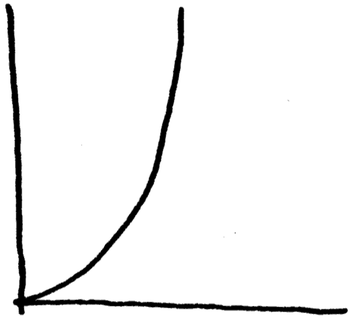

Heap Sort


Bubble Sort


Comb Sort


Cocktail Sort


O(n!)
O(2^n)
O(n^2)
O(n logn)
O(n)
O(log n)
O(1)
Time
Input Size (n)

EXAMPLE ALGORITHM:	BINARY SEARCH	SIMPLE SEARCH	QUICKSORT	SELECTION SORT	THE TRAVELING SALESMAN
					
ARRAY SIZE	O(log n)	O(n)	O(n log n)	O(n²)	O(n!)
10	0.3sec	1sec	3.3sec	10sec	4.2 days
100	0.6sec	10 sec	664 sec	16.6min	2.9×10^{149} years
1000	1sec	100sec	996 sec	27.7 hours	1.27×10^{2559} years

1.27×10^{2559} years

“Isso aí é útil pra mim na onde?”

Craque Neto

1. Código que escala de verdade
2. Economia de Infraestrutura
3. Entrevistas técnicas
4. Melhor tomada de decisão
5. Discutir arquitetura
6. Código eficiente

“Saber Big O não vai te fazer escrever menos bugs… mas vai garantir que seu bug rode rápido.”

Carlos Drummond de Andrade

Ah, mas eu sou front…

1. O usuário não quer saber se é Back ou front
2. Nem sempre o back presta
3. Transformações de dados são responsabilidade do front
4. Cenário offline
5. Custo de energia (mobile)

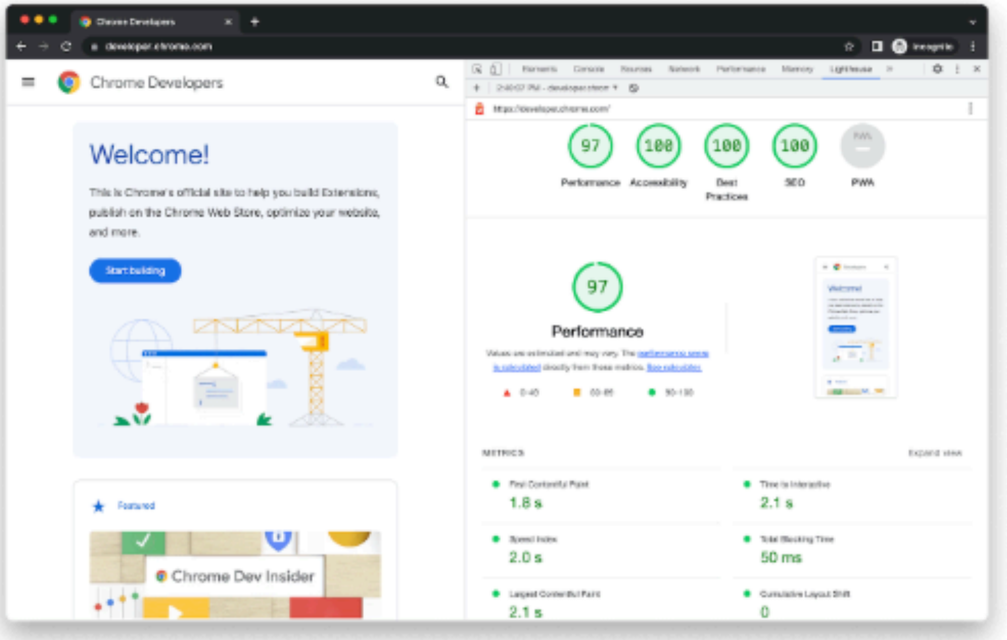
Lighthouse

Lighthouse has audits for performance, accessibility, progressive web apps, SEO, and more. You can run Lighthouse against any web page, public or requiring authentication.

You can run Lighthouse as part of PageSpeed Insights, in Chrome DevTools, from the command line, or as a Node module. You give Lighthouse a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicators on how to improve the page.

Each audit has a reference document explaining why the audit is important, as well as how to fix it.

Test a site Overview



Como melhorar sua pontuação de TTI

Tempo de Interação

Uma melhoria que pode ter um efeito particularmente grande no TTI é adiar ou remover trabalhos JavaScript desnecessários. Procure oportunidades para otimizar seu JavaScript. Em particular, considere reduzir payloads do JavaScript com a divisão de código e aplicar o padrão PRPL. A otimização do JavaScript de terceiros também gera melhorias significativas para alguns sites.

Sabia que não existe só Array em JavaScript?

$$O(n)$$

```
1 // ARRAY
2 const array = [1, 2, 3, 4, 5];
3 console.log(array.includes(5)); // true, mas percorreu a lista
4
5 // SET
6 const set = new Set([1, 2, 3, 4, 5]);
7 console.log(set.has(5)); // true, instantâneo
```

$$O(1)$$

```
// Código fezes
const users = [
  { id: 1, name: "Ana", active: true },
  { id: 2, name: "Carlos", active: false },
  { id: 3, name: "Mariana", active: true },
  { id: 4, name: "Pedro", active: true }
];

const result = users
  .map(user => ({ ...user, name: user.name.toUpperCase() }))   // 1ª passada
  .filter(user => user.active)                                 // 2ª passada
  .map(user => user.name);                                     // 3ª passada

console.log(result); // ["ANA", "MARIANA", "PEDRO"]
```

```
1 // Código otimizado
2 const result = users.reduce((acc, user) => {
3   if (user.active) acc.push(user.name.toUpperCase());
4   return acc;
5 }, []);
6
7 console.log(result); // ["ANA", "MARIANA", "PEDRO"]
```

API

$O(n)$

```
1 // O dev poeta puxando 1 milhão de registros na fé
2 async function getActiveUsers() {
3   const res = await fetch("https://api.xablau.com/users");
4   const users = await res.json();
5   return users.filter(u => u.active);
6 }
```

API

```
async function getActiveUsers() {
  const res = await fetch("https://api.xablau.com/users?active=true&limit=100");
  return res.json();
}
```

RESUMO PARA QUEM NÃO PRESTOU ATENÇÃO:

- Big O não é um bicho de sete cabeças
- Bubble Sort é um lixo
- O seu .map.map.map ainda vai te demitir
- Não tenha uma vida amorosa O(n2), onde cada tentativa dobra o sofrimento

Gostaria que vc desse a sua opinião sobre a minha palestra. Por favor não me decepcione como das outras vezes.

Gus, percebi que seus códigos são horríveis do ponto de vista de Big O. E de outros pontos de vista também. Se eu fosse expressar seus códigos em notação Big O, seria:

$$O(meu\ deus)$$

Como diria minha ex…

terminamos

@oguscaetano