1º TRABALHO DE APA

Prof. Raul Fonseca Neto

Caio Assis de Lima
Thiago Ramalho Vieira

INTRODUÇÃO

Linguagem Utilizada

JAVASCRIPT

ANÁLISE E PROJETO DE ALGORITMOS
Prof. Raul Fonseca Neto

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

ITERATIVO

JS AvaliacaoPolinomioIterativo.js > avaliarPolinomioIterativo

```
1   // implementação iterativa
2
    Complexity is 4 Everything is cool!
3   function avaliarPolinomioIterativo(coeficientes, x) {
4       let resultado = 0;
5       const n = coeficientes.length;
6
7       for (let i = 0; i < n; i++) {
8           let potencia = 1;
9           for (let j = 0; j < i; j++) {
10              potencia *= x;
11          }
12          resultado += coeficientes[i] * potencia;
13      }
14      return resultado;
15  }
16
17  const polinomio = [-1, 2, -6, 2]; // 2x³ - 6x² + 2x - 1
18  const coeficiente = 3;
```

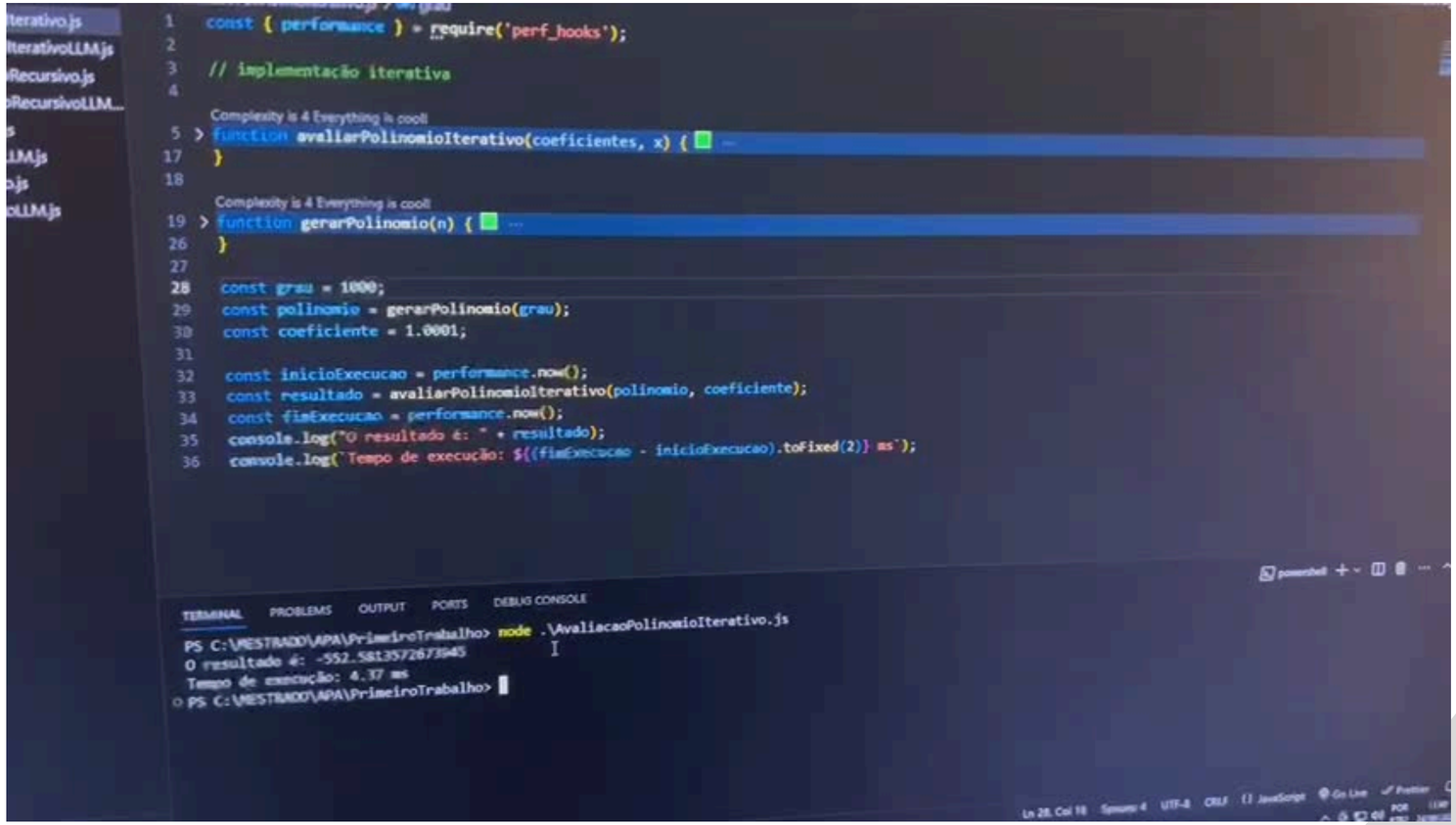
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativo.js
O resultado é: 5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

$O(N^2)$ **QUADRÁTICA**

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO ITERATIVO



ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

RECURSIVA

JS AvaliacaoPolinomioRecursivo.js > avaliacaoPolinomioRecursiva

```
1   // implementação recursiva
2
    Complexity is 4 Everything is cool!
3   function avaliacaoPolinomioRecursiva(coeficientes,x,i = 0) {
4       const n = coeficientes.length;
5       if (i === n - 1) {
6           return coeficientes[i];
7       }
8       return coeficientes[i] + x * avaliacaoPolinomioRecursiva(coeficientes, x, i + 1);
9   }
10
11  const polinomio = [-1, 2, -6, 2]; // 2x³ - 6x² + 2x - 1
12  const coeficiente = 3;
13
14  const resultado = avaliacaoPolinomioRecursiva(polinomio,coeficiente);
15  console.log("O resultado é: " + resultado);
16
17
```

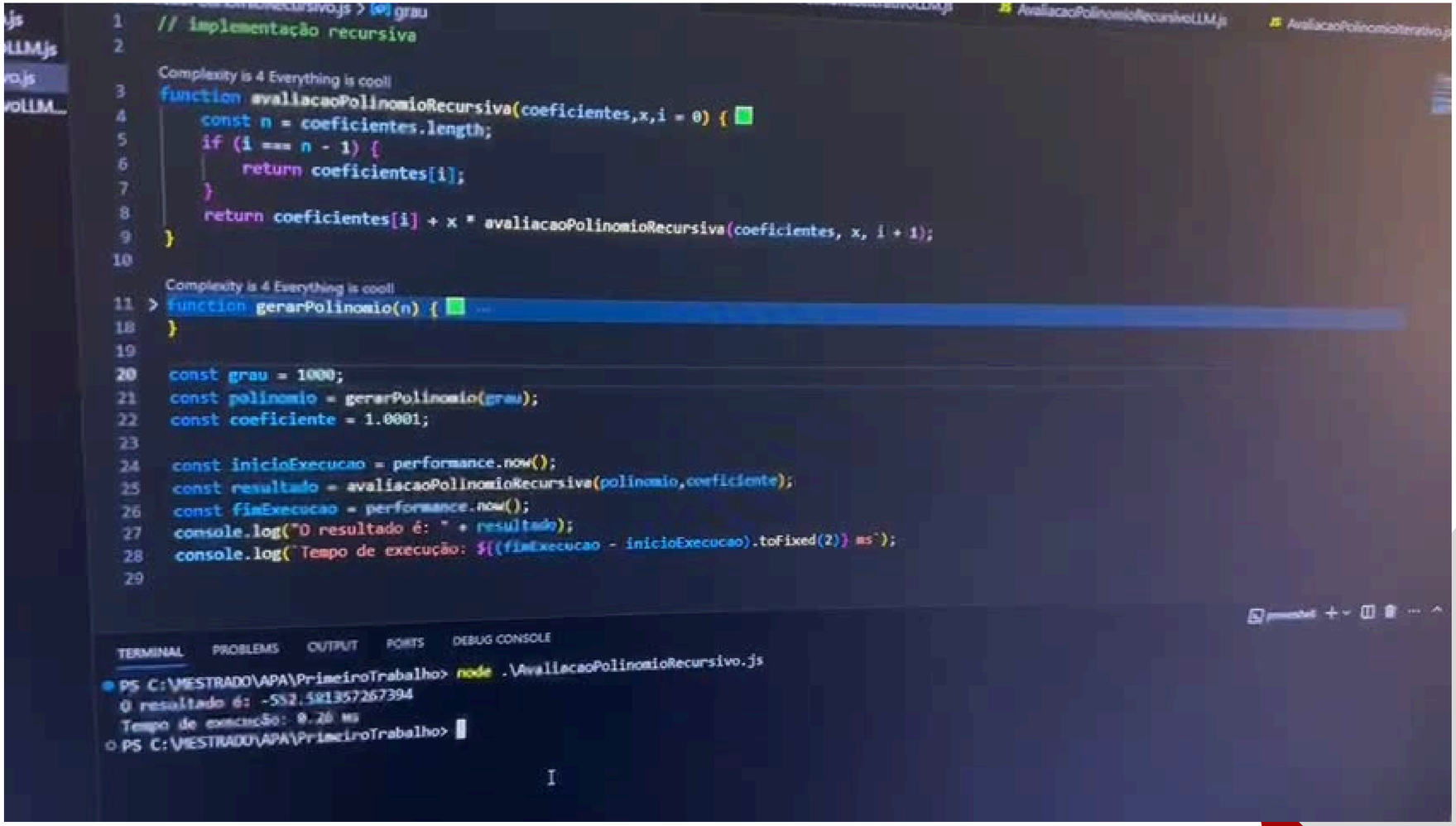
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivo.js
O resultado é: 5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

O(N)

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO RECURSIVO



ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

ITERATIVA (LLM)

JS AvaliacaoPolinomioIterativoLLM.js > ...

```
1   // implementação recursiva
2   // LLM utilizada:
3
    Complexity is 3 Everything is cool!
4   function avaliarPolinomioIterativoLLM(coeficientes, x) {
5       let resultado = 0;
6       for (let i = coeficientes.length - 1; i >= 0; i--) {
7           resultado = resultado * x + coeficientes[i];
8       }
9       return resultado;
10  }
11
12
13  const polinomio = [-1, 2, -6, 2]; // 2x³ - 6x² + 2x - 1
14  const coeficiente = 3;
15
16  const resultado = avaliarPolinomioIterativoLLM(polinomio, coeficiente);
17  console.log("O resultado é: " + resultado);
```

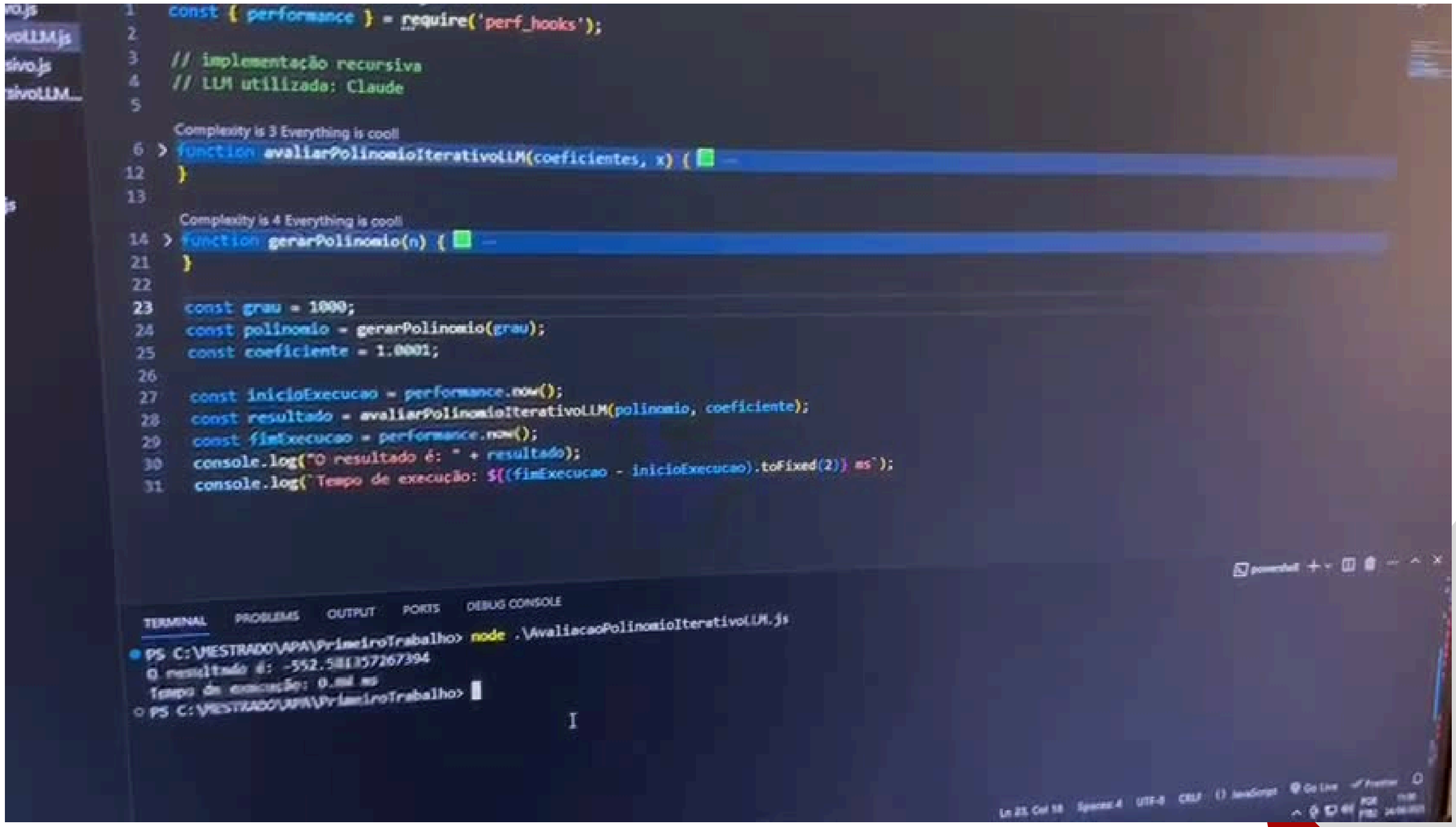
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativoLLM.js
O resultado é: 5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

O(N)

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO ITERATIVO (LLM)



ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

RECURSIVA(LLM)

JS AvaliacaoPolinomioRecursivoLLM.js > avaliarPolinomio

```
1   // implementação recursiva
2   // LLM utilizada: DeepSeek
3
    Complexity is 4 Everything is cool!
4   function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
5       // Caso base: quando o grau é 0, retornamos o coeficiente constante
6       if (grau === 0) {
7           return coeficientes[grau];
8       }
9
10      // Passo recursivo: a_n * x^n + a_{n-1} * x^{n-1} + ... + a_0 = x * (a_n * x^{n-1} + ...) + a_0
11      return x * avaliarPolinomio(coeficientes, x, grau - 1) + coeficientes[grau];
12  }
13
14  // Exemplo de uso:
15  const coeficientes = [2, -6, 2, -1]; // 2x³ - 6x² + 2x - 1 <- ALTEREI PARA MANTER AS MESMAS ENTRADAS
16  const x = 2;
17  const resultado = avaliarPolinomio(coeficientes, x);
18  console.log(`O polinômio avaliado em x = ${x} é ${resultado}`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE powershell

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivoLLM.js
O polinômio avaliado em x = 2 é -5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

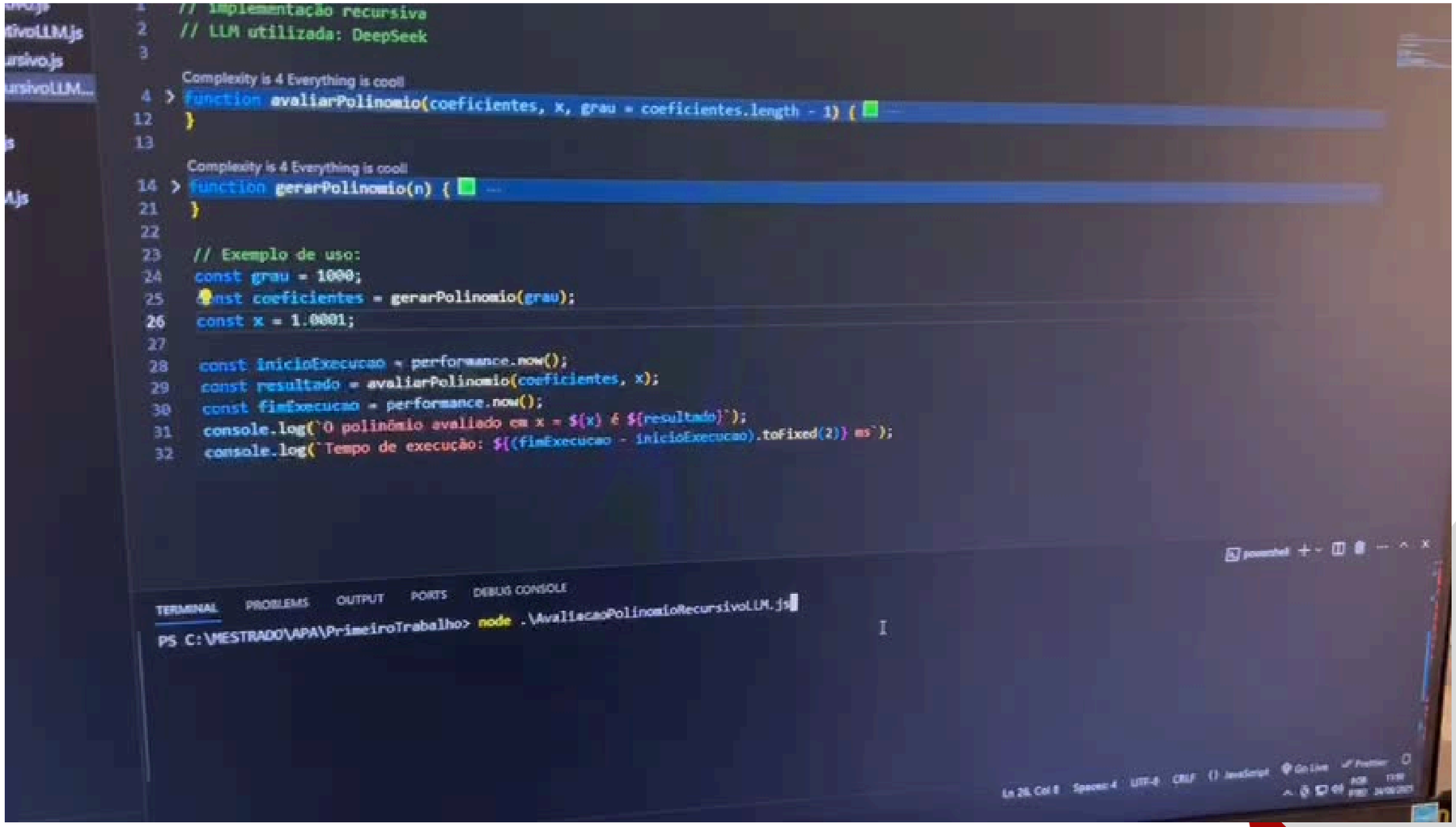
COMPLEXIDADE

O(N)

ANÁLISE E PROJETO DE ALGORITMOS

Prof. Raul Fonseca Neto

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO RECURSIVA (LLM)



ALGORITMO DA SEQUÊNCIA DE FIBONACCI

ITERATIVA

```
JS FibonacciIterativo.js > ...
1   // implementação iterativa
    Complexity is 5 Everything is cool!
2   function fibonacciIterativo(n) {
3       if (n <= 1) {
4           return n;
5       }
6       let anterior = 0;
7       let atual = 1;
8       for (let i = 2; i <= n; i++) {
9           const proximo = anterior + atual;
10          anterior = atual;
11          atual = proximo;
12      }
13      return atual;
14  }
15
16  const numero = 40; // Numero 40
17
18  const resultado = fibonacciIterativo(numero);

TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciIterativo.js
O resultado é: 102334155
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

O(N)

ALGORITMO DA SEQUÊNCIA DEFIBONACCI ITERATIVO

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

RECURSIVA

```
JS FibonacciRecursivo.js > ...
1   // implementação recursiva
    Complexity is 6 It's time to do something...
2   function fibonacciRecursiva(numero) {
3       if(numero <= 0){
4           return 0;
5       }
6       if(numero == 1){
7           return 1;
8       }
9       return fibonacciRecursiva(numero - 1) + fibonacciRecursiva(numero - 2);
10  }
11
12  const numero = 40; // Numero 40
13
14  const resultado = fibonacciRecursiva(numero);
15  console.log("O resultado é: " + resultado);
```

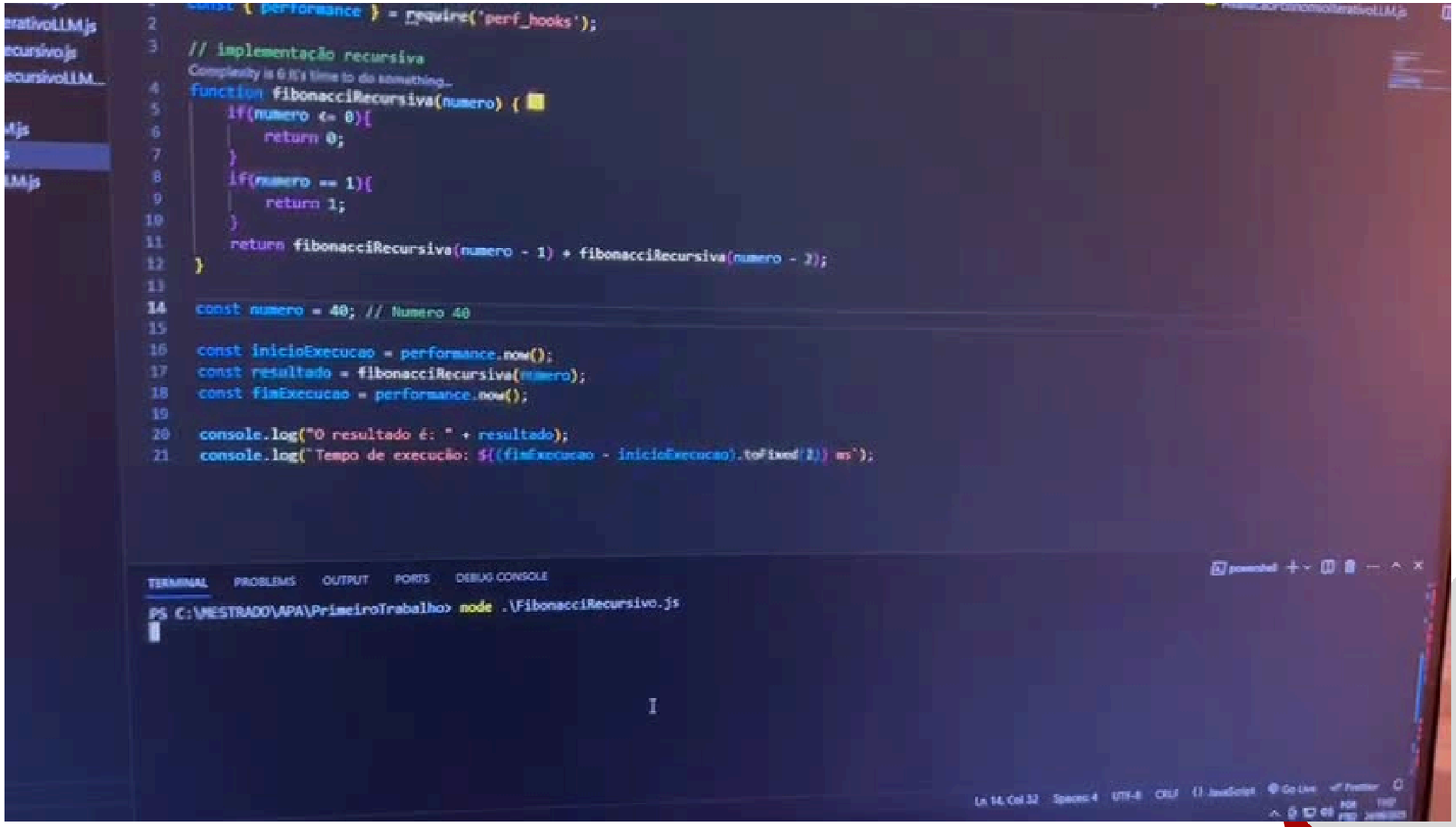
```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivo.js
O resultado é: 102334155
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

$O(2^n)$ **EXPONENCIAL**

ALGORITMO DA SEQUÊNCIA DEFIBONACCI RECURSIVO



ALGORITMO DA SEQUÊNCIA DE FIBONACCI

ITERATIVA (LLM)

JS FibonacciIterativoLLM.js > ...

```
1   // implementação iterativa
2   // LLM utilizada:
3
    Complexity is 5 Everything is cool!
4   function fibonacciIterativoLLM(n) {
5       if (n <= 1) return n;
6       let a = 0, b = 1;
7       for (let i = 2; i <= n; i++) {
8           [a, b] = [b, a + b];
9       }
10      return b;
11  }
12
13
14  const numero = 40; // Numero 40
15
16  const resultado = fibonacciIterativoLLM(numero);
17  console.log("O resultado é: " + resultado);
```

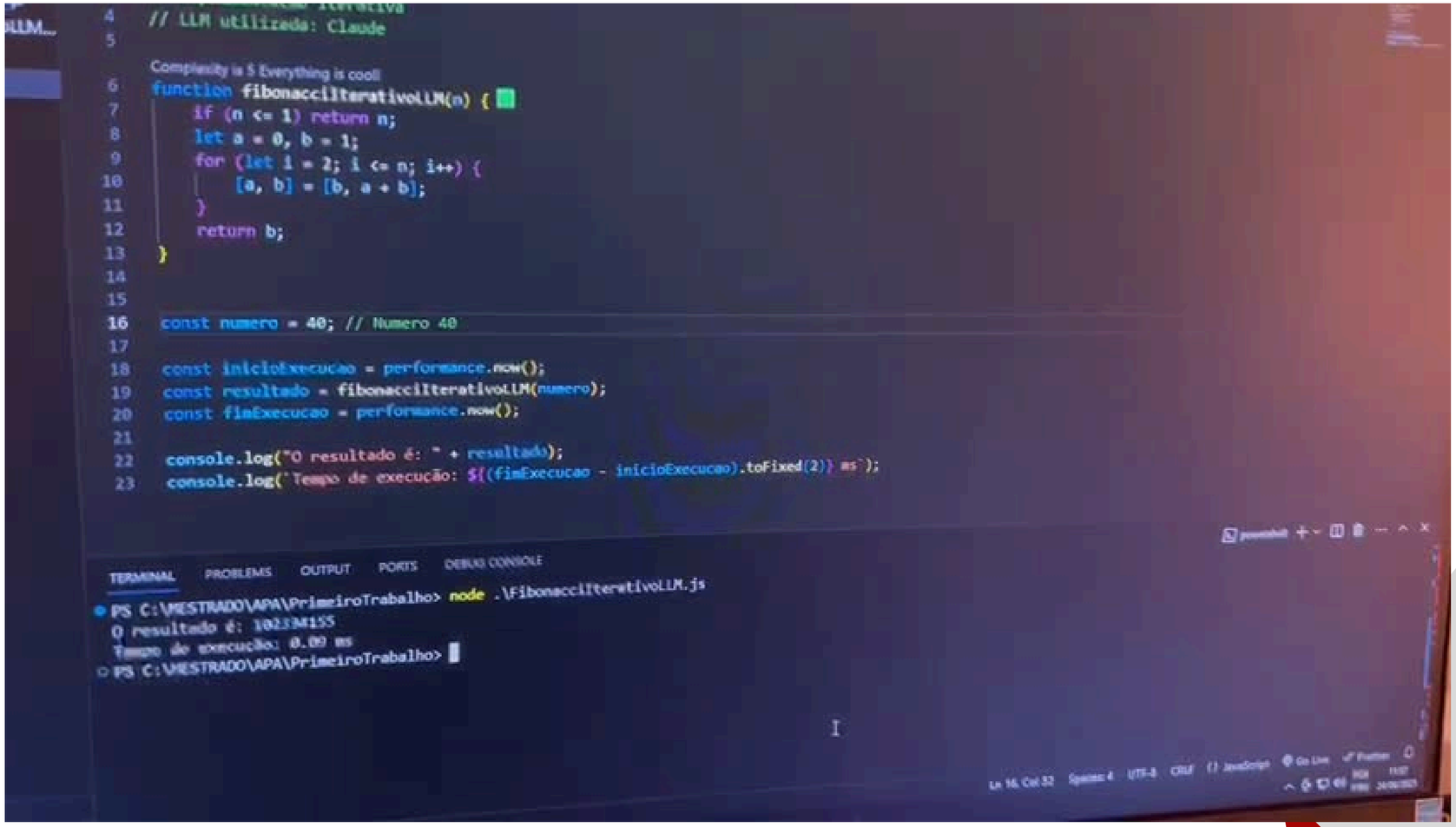
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciIterativoLLM.js
O resultado é: 102334155
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

O(N)

ALGORITMO DA SEQUÊNCIA DEFIBONACCI ITERATIVO (LLM)



ALGORITMO DE SEQUÊNCIA DE FIBONACCI

RECURSIVA(LLM)

JS FibonacciRecursivoLLM.js > posicao

```
1   // implementação iterativa
2   // LLM utilizada: DeepSeek
3
    Complexity is 6 It's time to do something...
4   function fibonacci(n) {
5       // Casos base: fib(0) = 0 e fib(1) = 1
6       if (n === 0) return 0;
7       if (n === 1) return 1;
8
9       // Passo recursivo: fib(n) = fib(n-1) + fib(n-2)
10      return fibonacci(n - 1) + fibonacci(n - 2);
11  }
12
13  // Exemplo de uso:
14  const posicao = 40;
15  console.log(`O ${posicao}º número de Fibonacci é ${fibonacci(posicao)}`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

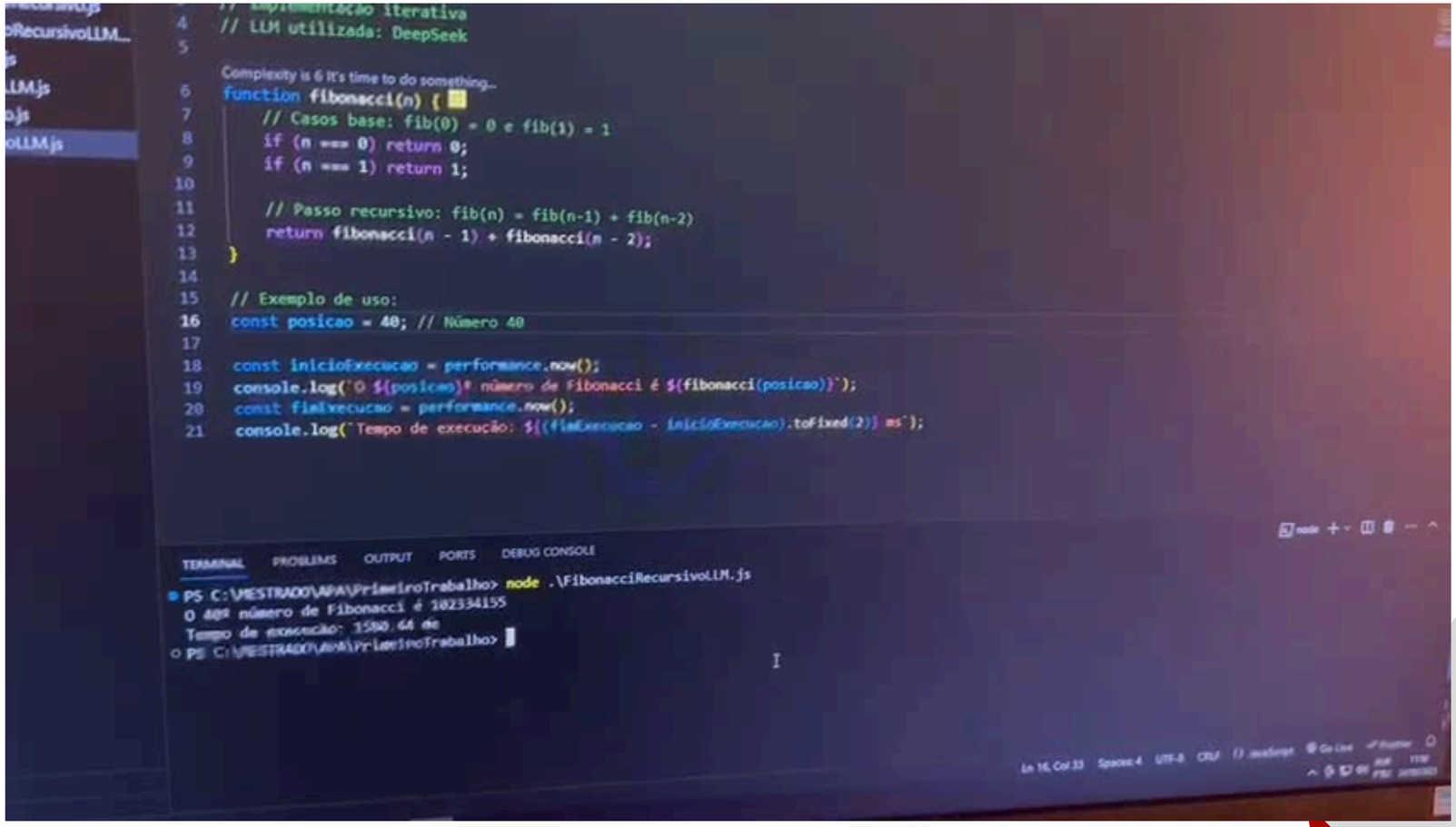
```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivo.js
O resultado é: 102334155
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

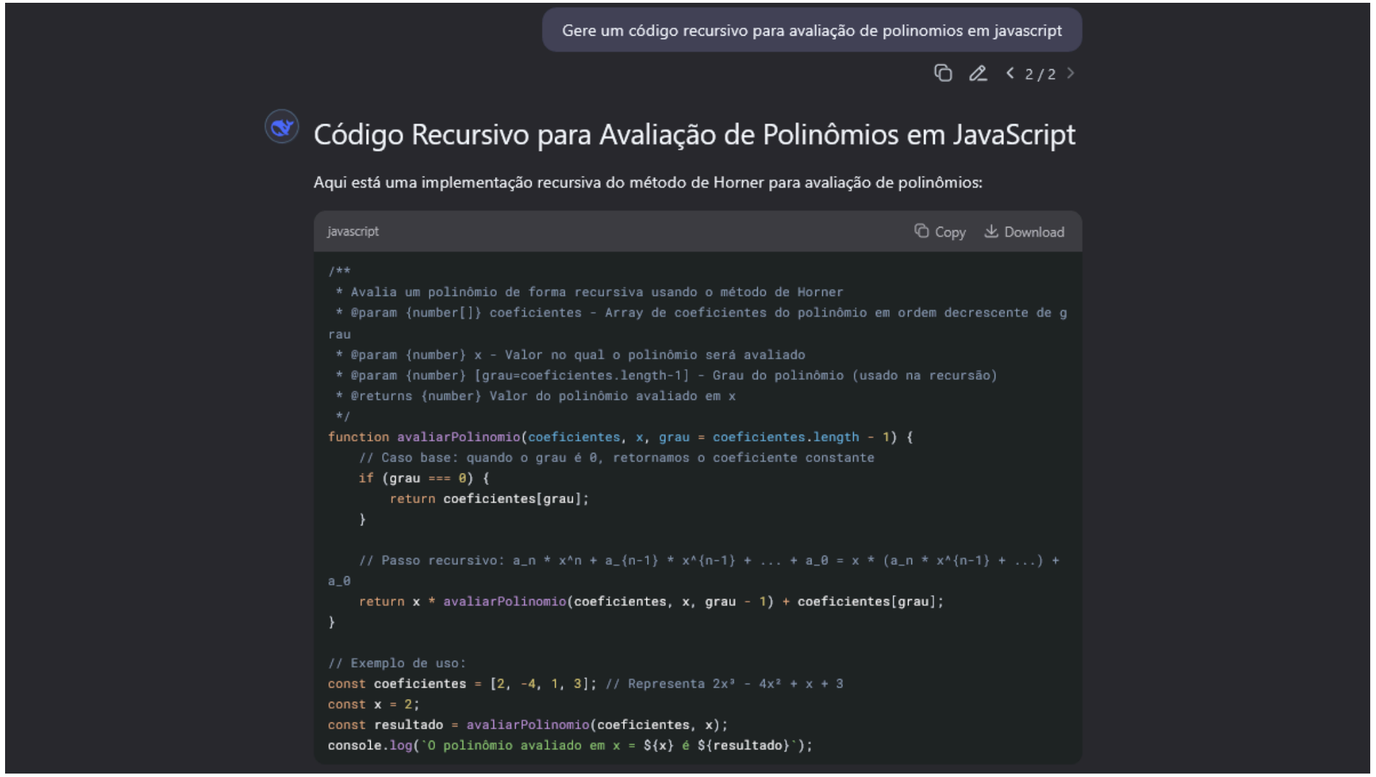
$O(2^n)$ **EXPONENCIAL**

ANÁLISE E PROJETO DE ALGORITMOS
Prof. Raul Fonseca Neto

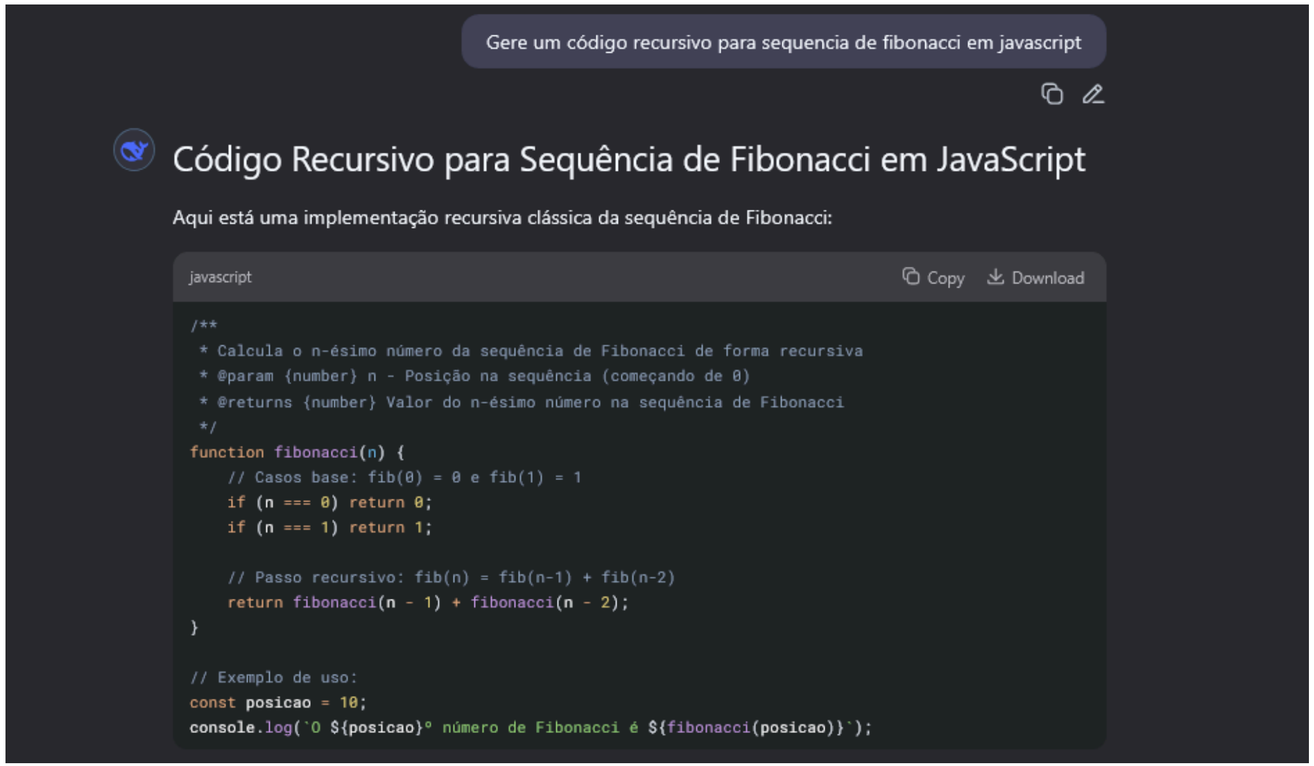
ALGORITMO DA SEQUÊNCIA DEFIBONACCI RECURSIVO (LLM)



PROMPTS


Gere um código recursivo para avaliação de polinomios em javascript
2 / 2
Código Recursivo para Avaliação de Polinômios em JavaScript
Aqui está uma implementação recursiva do método de Horner para avaliação de polinômios:
javascript
Copy
Download
/**
* Avalia um polinômio de forma recursiva usando o método de Horner
* @param {number[]} coeficientes - Array de coeficientes do polinômio em ordem decrescente de grau
* @param {number} x - Valor no qual o polinômio será avaliado
* @param {number} [grau=coeficientes.length-1] - Grau do polinômio (usado na recursão)
* @returns {number} Valor do polinômio avaliado em x
*/
function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
// Caso base: quando o grau é 0, retornamos o coeficiente constante
if (grau === 0) {
return coeficientes[grau];
}
// Passo recursivo: a_n * x^n + a_{n-1} * x^{n-1} + ... + a_0 = x * (a_n * x^{n-1} + ...) + a_0
return x * avaliarPolinomio(coeficientes, x, grau - 1) + coeficientes[grau];
}
// Exemplo de uso:
const coeficientes = [2, -4, 1, 3]; // Representa 2x³ - 4x² + x + 3
const x = 2;
const resultado = avaliarPolinomio(coeficientes, x);
console.log(`O polinômio avaliado em x = ${x} é ${resultado}`);

PROMPTS


Gere um código recursivo para sequencia de fibonacci em javascript
Código Recursivo para Sequência de Fibonacci em JavaScript
Aqui está uma implementação recursiva clássica da sequência de Fibonacci:
javascript
Copy
Download
/**
* Calcula o n-ésimo número da sequência de Fibonacci de forma recursiva
* @param {number} n - Posição na sequência (começando de 0)
* @returns {number} Valor do n-ésimo número na sequência de Fibonacci
*/
function fibonacci(n) {
// Casos base: fib(0) = 0 e fib(1) = 1
if (n === 0) return 0;
if (n === 1) return 1;
// Passo recursivo: fib(n) = fib(n-1) + fib(n-2)
return fibonacci(n - 1) + fibonacci(n - 2);
}
// Exemplo de uso:
const posicao = 10;
console.log(`O ${posicao}º número de Fibonacci é ${fibonacci(posicao)}`);

CONCLUSÃO

AVALIAÇÃO DE POLINÔMIO

CÓDIGO HUMANO

ITERATIVO

TEMPO DE EXECUÇÃO: ≅ 4.37 MS

COMPLEXIDADE: $O(N^2)$ ← QUADRÁTICA

RECURSIVO

TEMPO DE EXECUÇÃO: ≅ 0.26 MS

COMPLEXIDADE: $O(N)$ ← LINEAR

CONCLUSÃO

AVALIAÇÃO DE POLINÔMIO

LLM → DEEPSEEK E CLAUDE

ITERATIVO

TEMPO DE EXECUÇÃO: ≅ 0.08 MS

COMPLEXIDADE: O(N) ← LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: ≅ 0.23MS

COMPLEXIDADE: O(N) ← LINEAR

CONCLUSÃO

SEQUÊNCIA DE FIBONACCI

CÓDIGO HUMANO

ITERATIVO

TEMPO DE EXECUÇÃO: ≅ 0.05 MS

COMPLEXIDADE: O(N) ← LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: ≅ 1560 MS

COMPLEXIDADE: $O(2^n)$ ← EXPONENCIAL

CONCLUSÃO

SEQUÊNCIA DE FIBONACCI

LLM → DEEPSEEK E CLAUDE

ITERATIVO

TEMPO DE EXECUÇÃO: ≅ 0.09 MS

COMPLEXIDADE: O(N) ← LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: ≅ 1580 MS

COMPLEXIDADE: $O(2^n)$ ← EXPONENCIAL