



1º TRABALHO DE APA

Prof. Raul Fonseca Neto

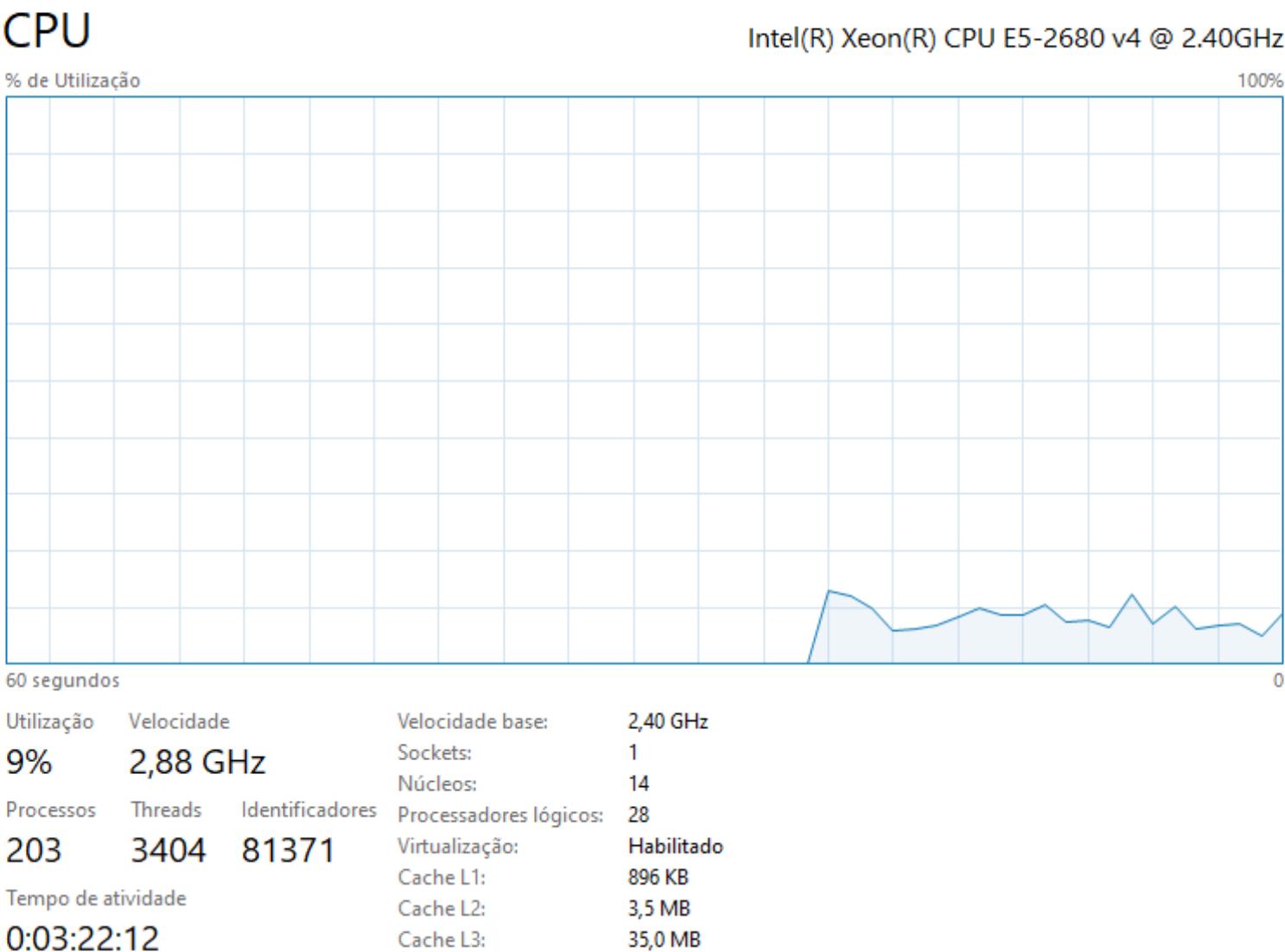
**Caio Assis de Lima
Thiago Ramalho Vieira**

INTRODUÇÃO

Linguagem Utilizada
JAVASCRIPT

INTRODUÇÃO

Processador Utilizado
XEON E5-2680V4



INTRODUÇÃO

Entrada para Avaliação de Polinômio
grau int 1000

Entrada para Sequência de Fibonacci
int 40

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

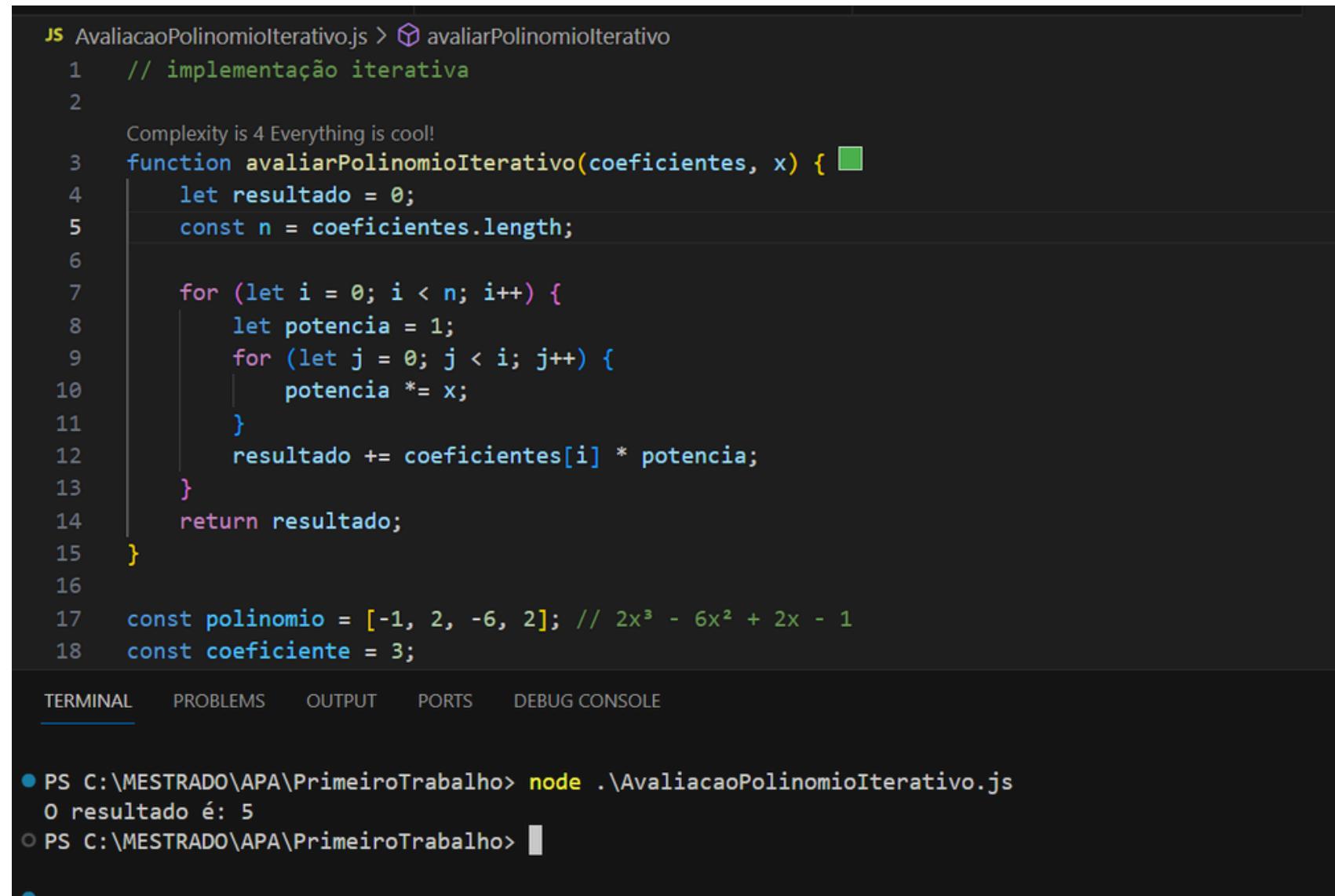
ITERATIVO

COMPLEXIDADE

$\Theta(N^2)$ QUADRÁTICA

FUNÇÃO DE COMPLEXIDADE

$$T(N) = \sum_{i=0}^{N-1} i = \frac{N(N-1)}{2}$$



```

JS AvaliacaoPolinomioIterativo.js > avaliarPolinomioIterativo
1 // implementação iterativa
2
3 Complexity is 4 Everything is cool!
4 function avaliarPolinomioIterativo(coeficientes, x) {
5     let resultado = 0;
6     const n = coeficientes.length;
7
8     for (let i = 0; i < n; i++) {
9         let potencia = 1;
10        for (let j = 0; j < i; j++) {
11            potencia *= x;
12        }
13        resultado += coeficientes[i] * potencia;
14    }
15    return resultado;
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
 0 resultado é: 5
- PS C:\MESTRADO\APA\PrimeiroTrabalho>

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO ITERATIVO

The screenshot shows a code editor with a dark theme. On the left, there's a sidebar with file names: Iterativo.js, IterativoLLM.js, Recursivo.js, RecursivoLLM.js, LLM.js, o.js, and oLLM.js. The main area displays the content of Iterativo.js:

```
1 const { performance } = require('perf_hooks');
2
3 // implementação iterativa
4
5 Complexity is 4 Everything is cool
6 > function avaliarPolinomioIterativo(coeficientes, x) {
7 }
8
9 Complexity is 4 Everything is cool
10 > function gerarPolinomio(n) {
11 }
12
13 const grau = 1000;
14 const polinomio = gerarPolinomio(grau);
15 const coeficiente = 1.0001;
16
17 const inicioExecucao = performance.now();
18 const resultado = avaliarPolinomioIterativo(polinomio, coeficiente);
19 const fimExecucao = performance.now();
20
21 console.log("O resultado é: " + resultado);
22 console.log(`Tempo de execução: ${fimExecucao - inicioExecucao} ms`);
```

Below the code editor is a terminal window with the following output:

```
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE
PS C:\ESTRADO\PAV\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativo.js
O resultado é: -552.582572673945
Tempo de execução: 4.37 ms
o ps: C:\ESTRADO\PAV\PrimeiroTrabalho>
```

The terminal also shows the current working directory as C:\ESTRADO\PAV\PrimeiroTrabalho.

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

RECURSIVA

COMPLEXIDADE

$O(N)$

FUNÇÃO DE COMPLEXIDADE

$T(N) = T(N-1) + O(1)$

The screenshot shows a code editor window with a dark theme. At the top, it says "JS AvaliacaoPolinomioRecursivo.js > avaliacaoPolinomioRecursiva". The code is as follows:

```
1 // implementação recursiva
2
3 Complexity is 4 Everything is cool!
4 function avaliacaoPolinomioRecursiva(coeficientes,x,i = 0) {
5     const n = coeficientes.length;
6     if (i === n - 1) {
7         return coeficientes[i];
8     }
9     return coeficientes[i] + x * avaliacaoPolinomioRecursiva(coeficientes, x, i + 1);
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
```

Below the code, there is a terminal window showing the execution of the script:

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

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO RECURSIVO

The screenshot shows a code editor with several tabs open, including 'avaliacaoPolinomioRecuritivo.js' which is the active file. The code implements a recursive algorithm to evaluate a polynomial. It defines a function `avaliacaoPolinomioRecurtivo` that takes coefficients and a variable `x`, and a helper function `gerarPolinomio` that generates a polynomial of a given degree. The terminal below shows the execution of the script and its output.

```
// implementação recursiva
function avaliacaoPolinomioRecurtivo(coeficientes, x, i = 0) {
    const n = coeficientes.length;
    if (i === n - 1) {
        return coeficientes[i];
    }
    return coeficientes[i] + x * avaliacaoPolinomioRecurtivo(coeficientes, x, i + 1);
}

function gerarPolinomio(n) {
}

const grau = 1000;
const polinomio = gerarPolinomio(grau);
const coeficiente = 1.0001;

const inicioExecucao = performance.now();
const resultado = avaliacaoPolinomioRecurtivo(polinomio, coeficiente);
const fimExecucao = performance.now();
console.log("O resultado é: " + resultado);
console.log(`Tempo de execução: ${((fimExecucao - inicioExecucao).toFixed(2))} ms`);
```

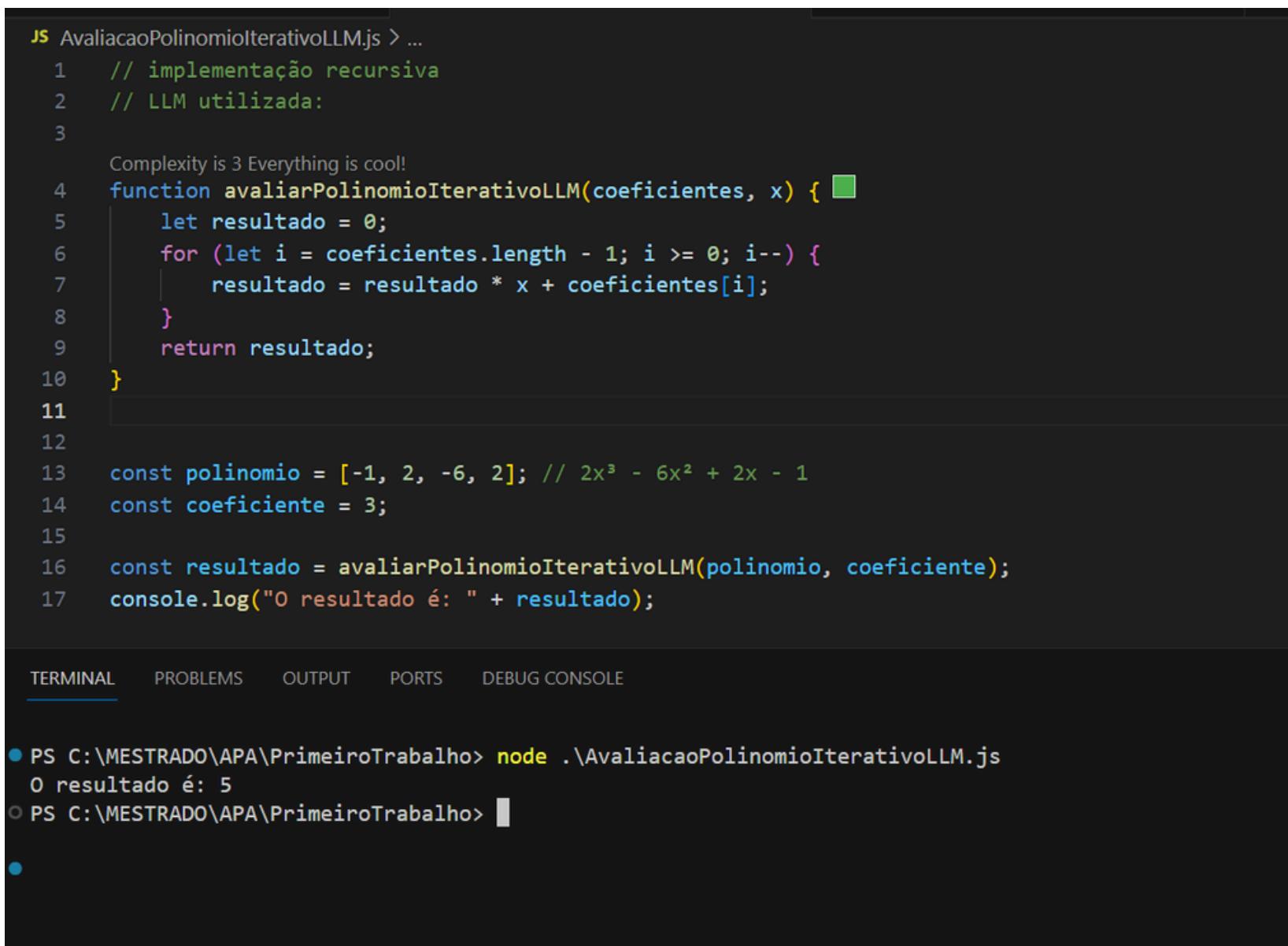
TERMINAL

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecurtivo.js
O resultado é: -552.421357267394
Tempo de execução: 0.26 ms
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

ITERATIVA (LLM)

COMPLEXIDADE
 $O(N)$
FUNÇÃO DE COMPLEXIDADE
 $T(N) = CN$



The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with icons for File, Edit, View, Insert, Select, Find, and Help. The main area displays a JavaScript file named `AvaliacaoPolinomioIterativoLLM.js`. The code implements a recursive function to evaluate a polynomial. The terminal below shows the execution of the script and its output.

```
JS AvaliacaoPolinomioIterativoLLM.js > ...
1 // implementação recursiva
2 // LLM utilizada:
3
4 Complexity is 3 Everything is cool!
5 function avaliarPolinomioIterativoLLM(coeficientes, x) {
6     let resultado = 0;
7     for (let i = coeficientes.length - 1; i >= 0; i--) {
8         resultado = resultado * x + coeficientes[i];
9     }
10    return resultado;
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>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO ITERATIVO (LLM)

The screenshot shows a code editor with a dark theme. The file is named `AvaliacaoPolinomioIterativoLLM.js`. The code implements an iterative algorithm to evaluate a polynomial. It uses `perf_hooks` for performance measurement.

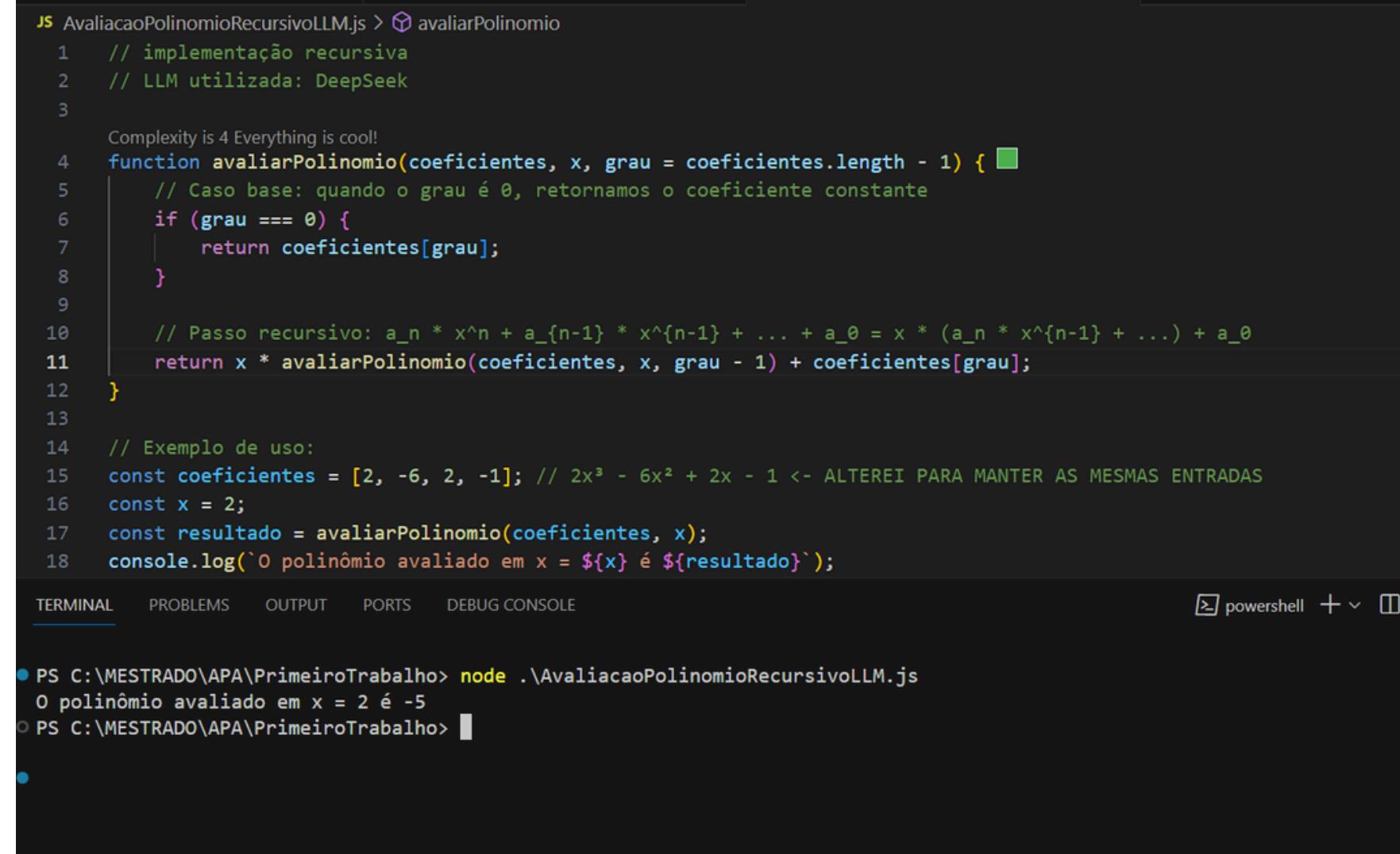
```
1 const { performance } = require('perf_hooks');
2
3 // implementação recursiva
4 // LLM utilizada: Claude
5
6 Complexity is 3 Everything is cool
7 > function avaliarPolinomioIterativoLLM(coefficientes, x) {
8 }
9
10 Complexity is 4 Everything is cool
11 > function gerarPolinomio(n) {
12 }
13
14 const grau = 1000;
15 const polinomio = gerarPolinomio(grau);
16 const coeficiente = 1.0001;
17
18 const inicioExecucao = performance.now();
19 const resultado = avaliarPolinomioIterativoLLM(polinomio, coeficiente);
20 const fimExecucao = performance.now();
21 console.log("O resultado é: " + resultado);
22 console.log(`Tempo de execução: ${((fimExecucao - inicioExecucao).toFixed(2))} ms`);
```

Below the code editor is a terminal window showing the execution of the script. The output shows the result and the execution time.

```
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE
PS C:\RESTRADOUAPA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativoLLM.js
O resultado é: -552.581357267394
Tempo de execução: 0.01 ms
PS C:\RESTRADOUAPA\PrimeiroTrabalho>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

RECURSIVA(LLM)



```

JS AvaliacaoPolinomioRecursovLLM.js > avaliarPolinomio
1 // implementação recursiva
2 // LLM utilizada: DeepSeek
3
4 Complexity is 4 Everything is cool!
5 function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
6     // Caso base: quando o grau é 0, retornamos o coeficiente constante
7     if (grau === 0) {
8         return coeficientes[grau];
9     }
10
11    // Passo recursivo: a_n * x^n + a_{n-1} * x^{n-1} + ... + a_0 = x * (a_n * x^{n-1} + ...) + a_0
12    return x * avaliarPolinomio(coeficientes, x, grau - 1) + coeficientes[grau];
13
14 // Exemplo de uso:
15 const coeficientes = [2, -6, 2, -1]; // 2x^3 - 6x^2 + 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

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

COMPLEXIDADE

 $O(N)$

FUNÇÃO DE COMPLEXIDADE

$$T(N) = T(N-1) + O(1)$$

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO RECURSIVA (LLM)

```
src.js
avaliam.js
arquivo.js
arshavall...
Ajs

1 // implementação recursiva
2 // LLM utilizada: DeepSeek
3
4 Complexity is 4 Everything is cool
5 > function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
6   if (grau === 0) {
7     return coeficientes[grau];
8   }
9   const resultado = avaliarPolinomio(coeficientes, x, grau - 1);
10  const termo = coeficientes[grau] * Math.pow(x, grau);
11  return resultado + termo;
12 }
13
14 Complexity is 4 Everything is cool
15 > function gerarPolinomio(n) {
16   const coeficientes = [];
17   for (let i = 0; i < n; i++) {
18     const valor = Math.random() * 100;
19     coeficientes.push(valor);
20   }
21   return coeficientes;
22 }
23
24 // Exemplo de uso:
25 const grau = 1000;
26 const coeficientes = gerarPolinomio(grau);
27 const x = 1.0001;
28
29 const inicioExecucao = performance.now();
30 const resultado = avaliarPolinomio(coeficientes, x);
31 const fimExecucao = performance.now();
32 console.log(`O polinômio avaliado em x = ${x} é ${resultado}`);
33 console.log(`Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

PS: C:\ESTRADO\AVP\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivoLLM.js

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

ITERATIVA

COMPLEXIDADE

$O(N)$

FUNÇÃO DE COMPLEXIDADE

$T(N) = O(N)$

The screenshot shows a code editor window with a dark theme. The file is named `FibonacciIterativo.js`. The code implements an iterative algorithm to calculate the n -th Fibonacci number. It uses a for loop to iterate from 2 to n , updating two variables (`anterior` and `atual`) to store the previous and current Fibonacci numbers respectively. The complexity is noted as $O(N)$.

```
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>

ALGORITMO DA SEQUÊNCIA DEFIBONACCI ITERATIVO

The screenshot shows a code editor with a dark theme. On the left, there's a file tree with files like `fibonacci.js`, `fibonacciLLM.js`, `fibonacciLLM...`, and `fibonacciLLM...`. The main editor area contains the following code:

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

// implementação iterativa
Complexity is O(n). Everything is cost
function fibonacciIterativo(n) {
    if (n <= 1) {
        return n;
    }
    let anterior = 0;
    let atual = 1;
    for (let i = 2; i <= n; i++) {
        const proximo = anterior + atual;
        anterior = atual;
        atual = proximo;
    }
    return atual;
}

const numero = 40; // Número 40

const inicioExecucao = performance.now();
const resultado = fibonacciIterativo(numero);
const fimExecucao = performance.now();
console.log(`O resultado é: ${resultado}`);
console.log(`Tempo de execução: ${((fimExecucao - inicioExecucao).toFixed(2))} ms`);
```

Below the code editor is a terminal window showing the execution of the script:

```
PS C:\RESTRACC\PA\PA\PrimeiroTrabalho> node .\FibonacciIterativo.js
O resultado é: 102334355
Tempo de execução: 0.05 ms
PS C:\RESTRACC\PA\PA\PrimeiroTrabalho>
```

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

RECURSIVA

COMPLEXIDADE

$O(2^n)$ EXPOENCIAL

FUNÇÃO DE COMPLEXIDADE

$$T(N) = T(N-1) + T(N-2) + O(1)$$

The screenshot shows a terminal window with the following content:

```
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>
-

ALGORITMO DA SEQUÊNCIA DEFIBONACCI RECURSIVO

The screenshot shows a code editor with a dark theme. On the left, there's a sidebar with file navigation. The main area contains the following code:

```
1 // Current 1 performance ) = require('perf_hooks');
2
3 // implementação recursiva
4 // Complexity is O(2^n) time to do something...
5 function fibonacciRecursiva(numero) {
6     if(numero <= 0){
7         return 0;
8     }
9     if(numero == 1){
10        return 1;
11    }
12    return fibonacciRecursiva(numero - 1) + fibonacciRecursiva(numero - 2);
13}
14
15 const numero = 40; // Número 40
16
17 const inicioExecucao = performance.now();
18 const resultado = fibonacciRecursiva(numero);
19 const fimExecucao = performance.now();
20
21 console.log("O resultado é: " + resultado);
22 console.log(`Tempo de execução: ${((fimExecucao - inicioExecucao).toFixed(2))} ms`);
```

Below the code, the terminal window shows the command: `ps C:\ESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivo.js`. The output of the program is visible as a long sequence of digits.

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

ITERATIVA (LLM)

```
JS FibonacciliterativoLLM.js > ...
1 // implementação iterativa
2 // LLM utilizada:
3
4 Complexity is 5 Everything is cool!
5 function fibonacciIterativoLLM(n) {
6     if (n <= 1) return n;
7     let a = 0, b = 1;
8     for (let i = 2; i <= n; i++) {
9         [a, b] = [b, a + b];
10    }
11 }
12
13
14 const numero = 40; // Número 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
0 resultado é: 102334155
- PS C:\MESTRADO\APA\PrimeiroTrabalho>

COMPLEXIDADE

$O(N)$

FUNÇÃO DE COMPLEXIDADE

$T(N)=O(N)$

ALGORITMO DA SEQUÊNCIA DEFIBONACCI ITERATIVO (LLM)

The screenshot shows a computer interface with a dark-themed code editor and a terminal window below it.

Code Editor Content:

```
4 // LLM utilizada: Claude
5
6 Complexity is 5 Everything is cool
7 function fibonacciterativollm(n) {
8     if (n <= 1) return n;
9     let a = 0, b = 1;
10    for (let i = 2; i <= n; i++) {
11        [a, b] = [b, a + b];
12    }
13    return b;
14
15
16 const numero = 40; // Número 40
17
18 const inicioExecucao = performance.now();
19 const resultado = fibonacciterativollm(numero);
20 const fimExecucao = performance.now();
21
22 console.log("O resultado é: " + resultado);
23 console.log("Tempo de execução: ${((fimExecucao - inicioExecucao).toFixed(2))} ms");
```

Terminal Output:

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\Fibonacciterativollm.js
O resultado é: 102334155
Tempo de execução: 0.00 ms
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

ALGORITMO DE SEQUÊNCIA DE FIBONACCI

RECURSIVA(LLM)

```
JS FibonacciRecursivoLLM.js > [o] posicao
1 // implementação iterativa
2 // LLM utilizada: DeepSeek
3
4 Complexity is 6 It's time to do something...
5 function fibonacci(n) {
6     // Casos base: fib(0) = 0 e fib(1) = 1
7     if (n === 0) return 0;
8     if (n === 1) return 1;
9
10    // Passo recursivo: fib(n) = fib(n-1) + fib(n-2)
11    return fibonacci(n - 1) + fibonacci(n - 2);
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

$O(2^n)$ EXPOENCIAL

FUNÇÃO DE COMPLEXIDADE

$T(N)=O(2^n)$

ALGORITMO DA SEQUÊNCIA DEFIBONACCI RECURSIVO (LLM)

The image shows a computer monitor displaying a code editor and a terminal window. The code editor is showing a file named `FibonacciRecursivoLLM.js`. The code implements a recursive algorithm for generating Fibonacci numbers. It includes base cases for $n=0$ and $n=1$, and a recursive step where it calculates the sum of the previous two numbers. It also includes a usage example where it calculates the 40th Fibonacci number and measures the execution time.

```
// Implementação iterativa
// LLM utilizada: DeepSeek

Complexity is 6 it's time to do something...
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 = 40; // Número 40

const inicioExecucao = performance.now();
console.log(`O ${posicao}º número de Fibonacci é ${fibonacci(posicao)}`);
const fimExecucao = performance.now();
console.log(`Tempo de execução: ${((fimExecucao - inicioExecucao).toFixed(2))} ms`);
```

The terminal window below the code editor shows the output of running the script. It prints the 40th Fibonacci number, which is 102334155, and the execution time, which is 1500.44 ms.

```
TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivoLLM.js
O 40º número de Fibonacci é 102334155
Tempo de execução: 1500.44 ms
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

PROMPTS

Gere um código recursivo para avaliação de polinomios em javascript

Copy Download

Código Recursivo para Avaliação de Polinômios em JavaScript

Aqui está uma implementação recursiva do método de Horner para avaliação de polinômios:

```
javascript
/**
 * 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:  $a_n * x^n + a_{n-1} * x^{n-1} + \dots + a_0 = x * (a_n * x^{n-1} + \dots) + a_0$ 
    return x * avaliarPolinomio(coeficientes, x, grau - 1) + coeficientes[grau];
}

// Exemplo de uso:
const coeficientes = [2, -4, 1, 3]; // Representa  $2x^3 - 4x^2 + x + 3$ 
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

Copy

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: $\cong 4.37$ MS

COMPLEXIDADE: $O(N^2)$ \leftarrow QUADRÁTICA

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 0.26$ MS

COMPLEXIDADE: $O(N)$ \leftarrow LINEAR

CONCLUSÃO

AVALIAÇÃO DE POLINÔMIO

LLM → DEEPSEEK E CLAUDE

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 0.08 \text{ MS}$

COMPLEXIDADE: $O(N) \leftarrow \text{LINEAR}$

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 0.23\text{MS}$

COMPLEXIDADE: $O(N) \leftarrow \text{LINEAR}$

CONCLUSÃO

SEQUÊNCIA DE FIBONACCI CÓDIGO HUMANO

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 0.05$ MS

COMPLEXIDADE: $O(N)$ \leftarrow LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 1560$ MS

COMPLEXIDADE: $O(2^n)$ \leftarrow EXPONENCIAL

CONCLUSÃO

SEQUÊNCIA DE FIBONACCI

LLM → DEEPSEEK E CLAUDE

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 0.09$ MS

COMPLEXIDADE: $O(N)$ ← LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 1580$ MS

COMPLEXIDADE: $O(2^n)$ ← EXPONENCIAL