



Pontifícia Universidade Católica do Rio Grande do Sul

Ciência de Dados e Inteligência Artificial

Morgana Dias Rodrigues

Relatório de Algoritmos e Estruturas de dados I sobre desenvolvimento
e análise algorítmica

Porto Alegre, RS

2021

Sumário

.....	1
<u>1 APRESENTAÇÃO</u>	<u>3</u>
<u>2 INTRODUÇÃO</u>	<u>3</u>
2.1 ESPECIFICAÇÃO DO PROBLEMA	3
2.1.1 ESPECIFICAÇÃO DE FERRAMENTAS E LINGUAGEM USADA	3
<u>3 APRESENTAÇÃO DA SOLUÇÃO.....</u>	<u>4</u>
3.1. CLASSES.....	4
3.1.1. APRESENTAÇÃO DA CLASSE MAIN	4
3.1.1.1 VARIÁVEIS	4
<u>4 ANÁLISANDO A COMPLEXIDADE DO ALGORITMO.....</u>	<u>5</u>
<u>5 APRESENTAÇÃO DOS GRÁFICOS COM O CRESCIMENTO DA SOLUÇÃO ELABORADA.....</u>	<u>7</u>
<u>6 CARACTERÍSTICA DA IMPLEMENTAÇÃO</u>	<u>9</u>
<u>7 CONCLUSÃO</u>	<u>10</u>

1 APRESENTAÇÃO

Este relatório tem por objetivo detalhar o desenvolvimento e análise do algoritmo construído a partir do enunciado do primeiro trabalho da disciplina de algoritmo e estruturas de dados I focando nos conceitos iniciais apresentados na disciplina sendo eles a complexidade algorítmica, apresentação gráfica da complexidade e avaliação da característica da solução implementada.

2 INTRODUÇÃO

2.1 Especificação do problema

Dada uma sequência de números V , tal que V vai de 1 até V , contabilize a quantidade de números primos existentes nesta sequência. Usando a linguagem de programação C, C++, Java ou Python, elabore a solução. Como resultado, elabore um relatório onde (a) é feita a análise da complexidade algorítmica da sua solução, (b) é apresentando um gráfico com o crescimento da solução elaborada e (c) é avaliada a característica desta implementação.

2.1.1 Especificação de ferramentas e linguagem usada


A ferramenta utilizada para a construção do código foi o editor de código Visual Studio Code e a linguagem utilizada foi Java versão 16.0.2.

A ferramenta utilizada para a construção e análise de gráficos foi utilizado o aplicativo Microsoft Excel e para anotações do código foi usado o aplicativo Notepad++.

3 APRESENTAÇÃO DA SOLUÇÃO

3.1.Classes

3.1.1. Apresentação da classe Main



```

1  /**Dada uma sequência de números V, tal que V vai de 1 até V, contabilize a quantidade de
2  números primos existentes nesta sequência*/
3  public class Main {
4      public static void main(String[] args) {
5          int divisores;
6          int V = 200;
7          int contabilizador = 0;
8          int totalOperacoes = 0;
9
10         for (int contador = 1; contador <= V; contador++) {
11             divisores = 0;
12             for (int aux = 1; aux <= V; aux++) {
13                 if (contador%aux == 0) {
14                     divisores++;
15                 }
16                 totalOperacoes++;
17             }
18             if (divisores == 2) {
19                 contabilizador += 1;
20             }
21             System.out.println("=====");
22             System.out.println(contador + ";" + totalOperacoes);
23             System.out.println("=====");
24         }
25         System.out.println("TOTAL DE PRIMOS: " + contabilizador);
26     }
27 }




```

A classe Main tem como finalidade encontrar o total de números primos existentes em uma sequência de números naturais maiores que zero através da iteração de dois laços for, onde o primeiro inicia em um contador de valor 1, passa para o segundo laço que também inicia sua variável aux em valor 1 e vai até o último número da sequência em cada looping para ambos números serem divididos um pelo outro com o intuito de acharmos o total de divisores que o número do contador tem dividido pelos números do aux, da seguinte forma:

Quando temos nosso contador = 5, o laço interno irá iniciar em aux = 1 até n-1 dividindo o contador. No primeiro looping teremos 5/1, depois 5/2, 5/3 até 5/n-1.

Toda vez que a divisão destas variáveis possuírem resto 0 (zero), contabilizamos um divisor. Após isso, veremos os números que possuem apenas dois divisores, como característica primordial dos números primos e também contabilizaremos para ao final do programa mostramos o total de números encontrados.

3.1.1.1 Variáveis

-  **divisores:** variável do tipo inteiro que tem por finalidade contabilizar o total de divisores que determinado número possui.
-  **V:** variável do tipo inteiro que indica qual o tamanho da sequência de número a ser percorrida para encontrar o total de números primos presentes nela.
-  **contabilizador:** variável do tipo inteiro que tem por finalidade contabilizar o total de números primos presentes na sequência.

- ✚ **totalOperacoes:** variável auxiliar do tipo inteiro que tem por finalidade contabilizar o total de vezes que os laços *for* são executados no programa para então podermos analisar a complexidade do nosso algoritmo, não sendo uma variável que se faz necessária para a solução do problema em encontrar primos.
- ✚ **contador:** variável do tipo inteiro que inicia em 1 para indicar o início e os valores que vão do contador até V para ser usado na divisão das estruturas condicionais.
- ✚ **aux:** variável do tipo inteiro que inicia em 1 para indicar o início e os valores que vão do contador até V para ser usado na divisão das estruturas condicionais.

4 ANÁLISANDO A COMPLEXIDADE DO ALGORITMO

Quanto à contagem de operações primitivas temos a seguinte tabela com o total de cada operação presente:

Atribuição de valores a variáveis	8
Operações aritméticas	4
Comparação de dois números	6

O algoritmo é composto por dois laços de repetição, sendo que o mais interno possui duas estruturas condicionais, atrelando para ele o número de vezes em que as suas operações primitivas internas são realizadas, como podemos observar na figura a seguir:

```

10      for (int contador = 1; contador <= V; contador++) {
11          divisores = 0;
12          for (int aux = 1; aux <= V; aux++) {
13              if (contador%aux == 0) {
14                  divisores+=1;
15              }
16              totalOperacoes+=1;
17          }
18          if (divisores == 2) {
19              contabilizador += 1;
20          }

```

A primeira condição serve para contabilizar o total de divisores que cada número de 1 até V que ambos os laços possuem e a segunda condição serve para filtrar os números primos da sequência, já que ela contabiliza apenas os números que possuem apenas dois divisores, como característica principal dos números primos.

O primeiro laço é executado n vezes (de acordo com o número dado a V) e o segundo laço executa n^2 (de acordo com o número dado a V) pois está atrelado ao primeiro laço e possuem as mesmas condições e percorrem a mesma sequência de números.

Com isso podemos analisar todas as operações inclusas no algoritmo e montarmos uma função para ela. Para tanto, por conta das estruturas condicionais, montaremos uma função para dois casos, sendo eles o pior caso e o melhor caso.

A função para o pior caso ocorre quando todas as operações primitivas, incluindo as operações internas das condicionais são executadas para todos os valores de n:

$$f(n) = 7 + n*(5 + 5n)$$

```

public class Main {
    public static void main(String[] args) {
        int divisores;
        int V = 50; // Atribuição de valores a variáveis          1
        int contabilizador = 0; // Atribuição de valores a variáveis 1
        //int totalOperacoes = 0; // Atribuição de valores a variáveis
        // antes do for:
        // contador = 1 --> Atribuição de valores a variáveis      1
        // contador <= V --> Comparação de dois números           1

        for (int contador = 1; contador <= V; contador++) { // número de execução do laço: n vezes      4 + n
            // depois do for:
            // contador++ --> Operações aritméticas                4 + n*(1)
            // contador <= V --> Comparação de dois números        4 + n*(2)

            divisores = 0; // Atribuição de valores a variáveis    4 + n*(3)

            // antes do for:
            // aux = 1 --> Atribuição de valores a variáveis        4 + n*(4)
            // aux <= V --> Comparação de dois números              4 + n*(5)
            for (int aux = 1; aux <= V; aux++) { // número de execução do laço: n*n vezes              n*n
                // depois do for:
                // aux++ --> Operações aritméticas                  4 + n*(5 + 1n)
                // aux <= V --> Comparação de dois números          4 + n*(5 + 2n)
                if (contador%aux == 0) { // Comparação de dois números  4 + n*(5 + 3n)
                    divisores++; // Atribuição de valores a variáveis & Operações aritméticas  4 + n*(5 + 5n)
                }
                //totalOperacoes++; // Atribuição de valores a variáveis & Operações aritméticas
            }
            if (divisores == 2) { // Comparação de dois números        4 + n*(5 + 5n) + 1
                contabilizador += 1; // Atribuição de valores a variáveis & Operações aritméticas  4 + n*(5 + 5n) + 2
            }
            //
            //
        }
        System.out.println("=====");
        System.out.println(contabilizador + "; " + totalOperacoes);
        System.out.println("=====");
    }
}

```

Já a função para o melhor caso ocorre quando as operações primitivas são executadas para metade dos valores de n, considerando que no problema estamos focando em números primos e em diversas sequências podemos considerar como aproximação que no máximo metade dela poderá ser números de primos, reduzindo a execução das operações primitivas das estruturas condicionais:

$$f(n) = 4,5 + n*(5 + 5*n/2)$$

```

public class Main {
    public static void main(String[] args) {
        int divisores;
        int V = 50; // Atribuição de valores a variáveis          1
        int contabilizador = 0; // Atribuição de valores a variáveis 1
        // antes do for:
        // contador = 1 --> Atribuição de valores a variáveis      1
        // contador <= V --> Comparação de dois números           1

        for (int contador = 1; contador <= V; contador++) { // número de execução do laço: n vezes      4 + n
            // depois do for:
            // contador++ --> Operações aritméticas                4 + n*(1)
            // contador <= V --> Comparação de dois números        4 + n*(2)

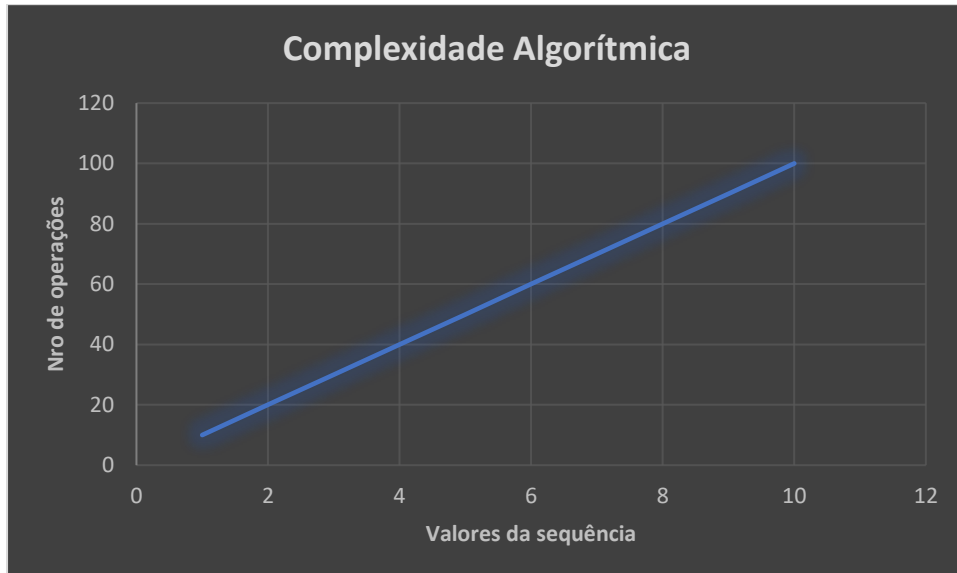
            divisores = 0; // Atribuição de valores a variáveis    4 + n*(3)

            // antes do for:
            // aux = 1 --> Atribuição de valores a variáveis        4 + n*(4)
            // aux <= V --> Comparação de dois números              4 + n*(5)
            for (int aux = 1; aux <= V; aux++) { // número de execução do laço: n*n vezes              n*n
                // depois do for:
                // aux++ --> Operações aritméticas                  4 + n*(5 + 1n)
                // aux <= V --> Comparação de dois números          4 + n*(5 + 2n)
                if (contador%aux == 0) { // Comparação de dois números  4 + n*(5 + 3n)
                    divisores++; // Atribuição de valores a variáveis & Operações aritméticas  4 + n*(5 + 5*n/2)
                }
            }
            if (divisores == 2) { // Comparação de dois números        4 + n*(5 + 5*n/2) + 1/2
                contabilizador += 1; // Atribuição de valores a variáveis & Operações aritméticas  4 + n*(5 + 5*n/2) + 1/2 + 1/2
            }
            //
            //
        }
        System.out.println("=====");
        System.out.println(contabilizador + "; " + totalOperacoes);
        System.out.println("=====");
    }
}

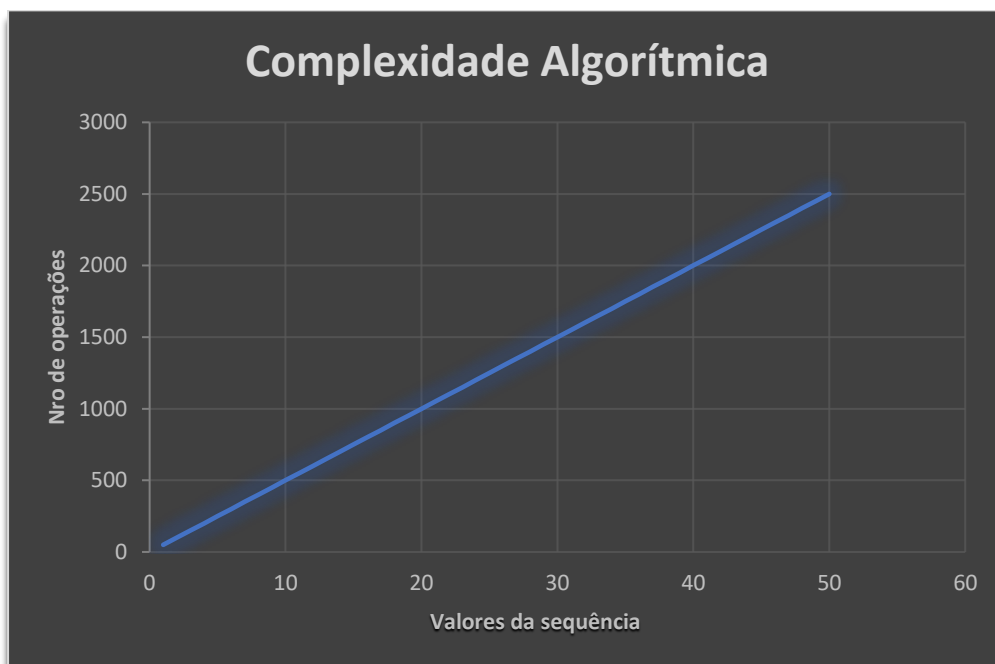
```

5 APRESENTAÇÃO DOS GRÁFICOS COM O CRESCIMENTO DA SOLUÇÃO ELABORADA

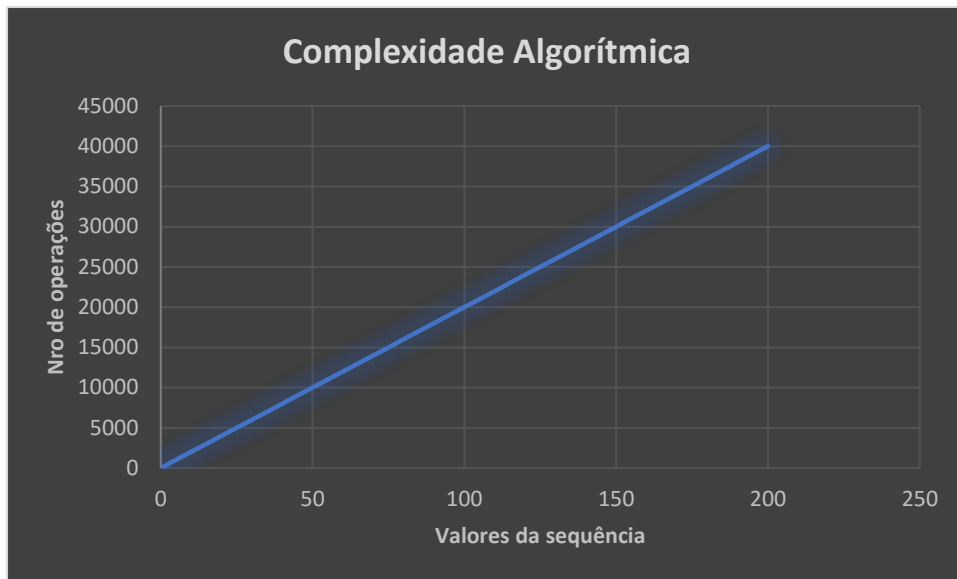
✚ Para $V = 10$



✚ Para $V = 50$



Para $V = 200$



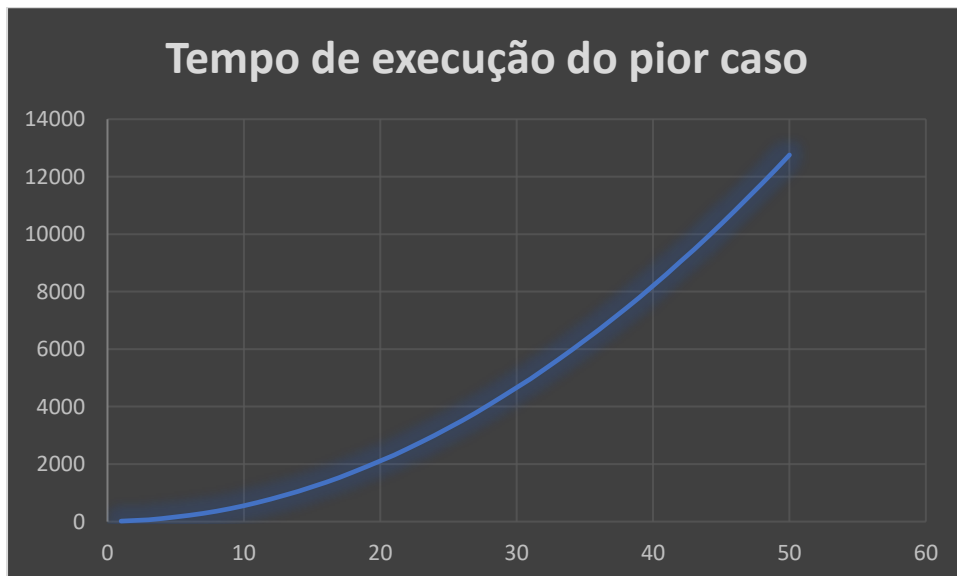
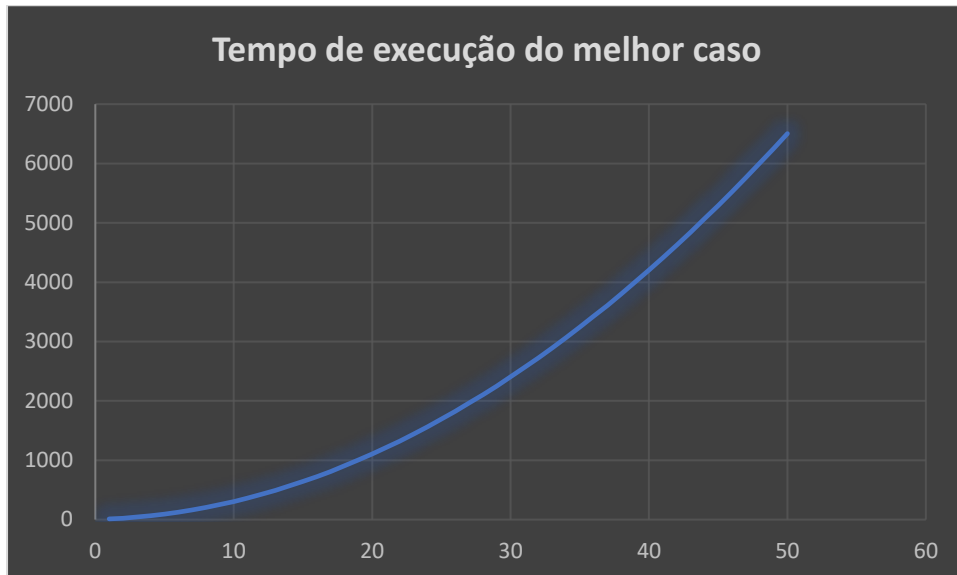
Interpretação

Podemos observar a partir dos gráficos que temos uma construção algorítmica linear, ou seja, conforme o tamanho do nosso V cresce, crescem também o número de operações feitas pelo algoritmo.

6 CARACTERÍSTICA DA IMPLEMENTAÇÃO

Conforme vimos pela função do algoritmo nos dois casos apresentados, temos uma característica de tempo de execução dada por $O(n^2)$ que é o nosso termo de mais alto grau na função:

$f(n) = 7 + n \cdot (5 + 5n)$	$O(n^2)$
$f(n) = 4,5 + n \cdot (5 + 5 \cdot n/2)$	$O(n^2)$



7 CONCLUSÃO

Como primeira análise de um algoritmo autoral a conclusão final nos diz a importância de analisar as operações primitivas e repetições de laços para diversos tamanhos de problemas na solução criada pois apenas desta maneira, testando e analisando, podemos avaliar o desempenho da solução bem como o seu melhoramento para redução de complexidade, operações e tempo de execução, sempre visando tirar o melhor do nosso algoritmo em termos de implementação.