Introdução

A análise da complexidade de algoritmos é um tema muito pertinente na área da computação, que se dedica ao estudo da eficiência e desempenho de algoritmos. A análise de complexidade é útil para determinar a quantidade de recursos que determinado algoritmo consome ao ser utilizado dentro de um programa de computador.

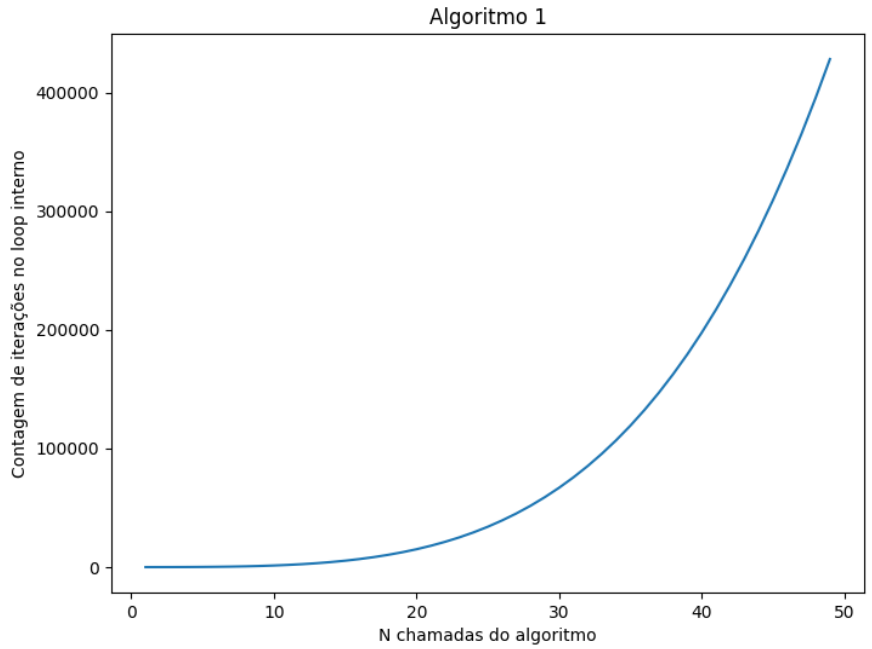
Estar familiarizado com esse conceito é importante pois ajuda o programador a entender como diferentes algoritmos se comportam e seu impacto na eficiência de programas, scripts, e especialmente em casos de grandes bancos de dados, como na área de inteligência artificial e machine learning.

Análise dos algoritmos

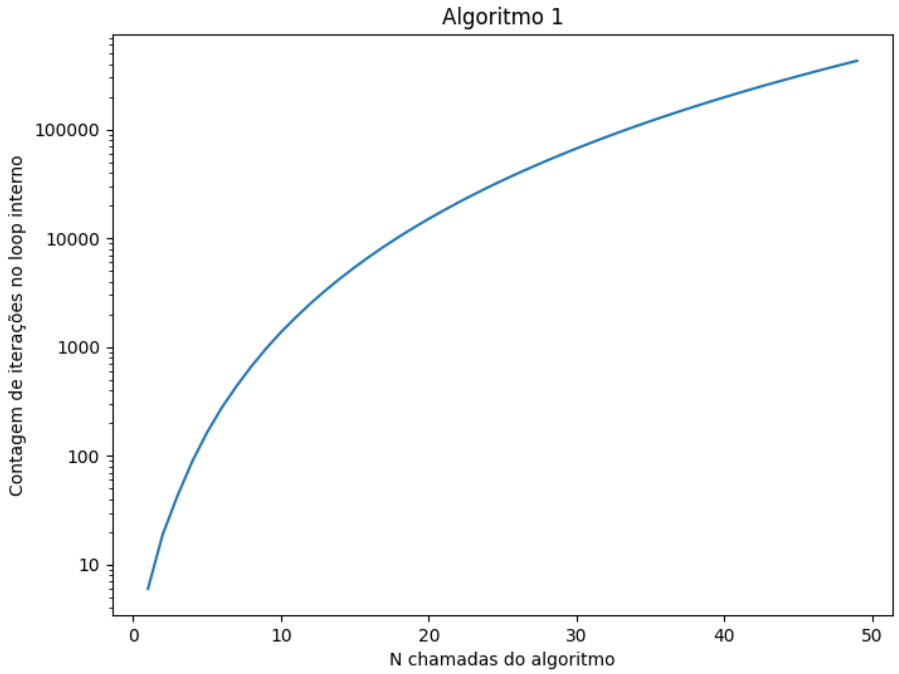
Algoritmo 1

```
for (i = 0; i <= n + 1; i++)
{
    for (j = 1; j <= i * i; j += i + 1)
    {
        for (k = i / 2; k <= n + j; k += 2)
        {
            res = res + n - 1;
            op_count++;
        }
    }
```

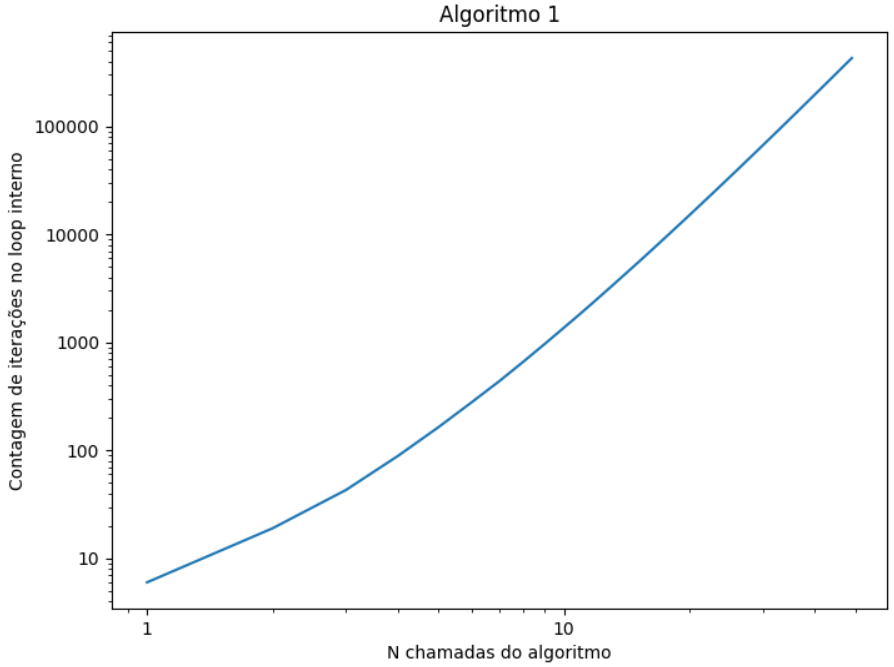
Esse é o gráfico de *n* por *Número de operações*



Pela curva do gráfico é possível perceber que a função que mostra o número de chamadas da função é do tipo polinomial. Para isso aplicaremos uma escala logarítmica no eixo y, para tentarmos encontrar uma reta.



Ainda não encontramos a reta, então aplicaremos a escala logarítmica no eixo x.



Agora sim, chegamos em uma reta, então podemos aplicar a seguinte fórmula, para descobrir a função que mostra o crescimento da função:

$$\frac{(log(y2) - log(y1))}{(log(x2) - log(x1))}$$

Aplicando os valores temos a seguinte fórmula:

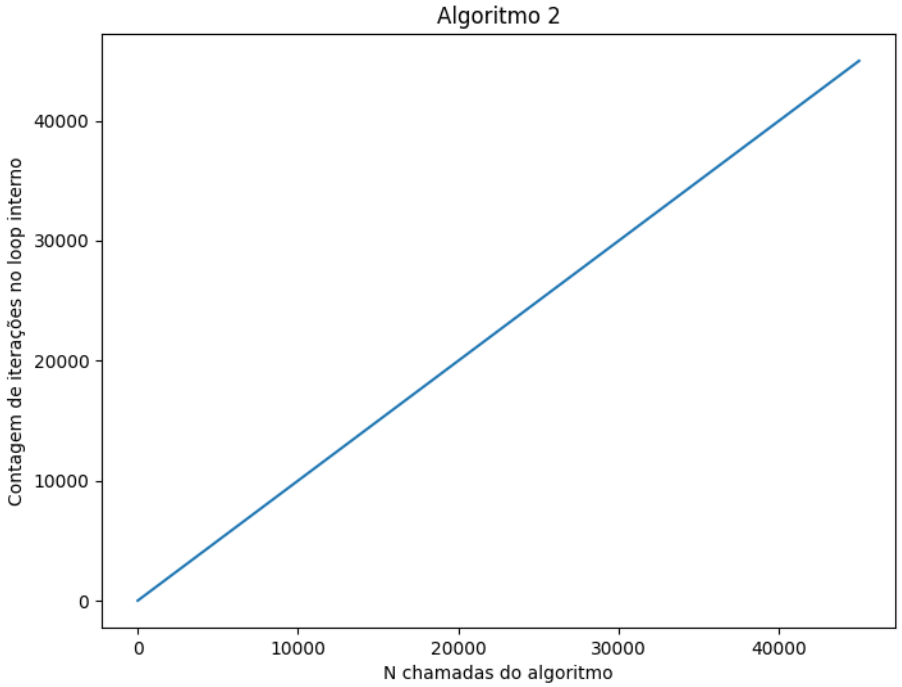
$$\frac{(log(428350) - log(1369))}{(log(49) - log(10))} = b$$

Tendo o resultado *n ≅ 3.6154*

Logo temos que a função que demonstra o crescimento do algoritmo 1 $f(n) \cong n^{3.6154}$...

Algoritmo 2

```
for (i = n; i <= n; i += i / 2 + 1)
{
    for (j = i / 2; j <= i * i; j += i + 1)
    {
        for (k = n; k <= 2 * n; k += i + 1)
        {
            res = res + n;
            op_count++;
        }
    }
```

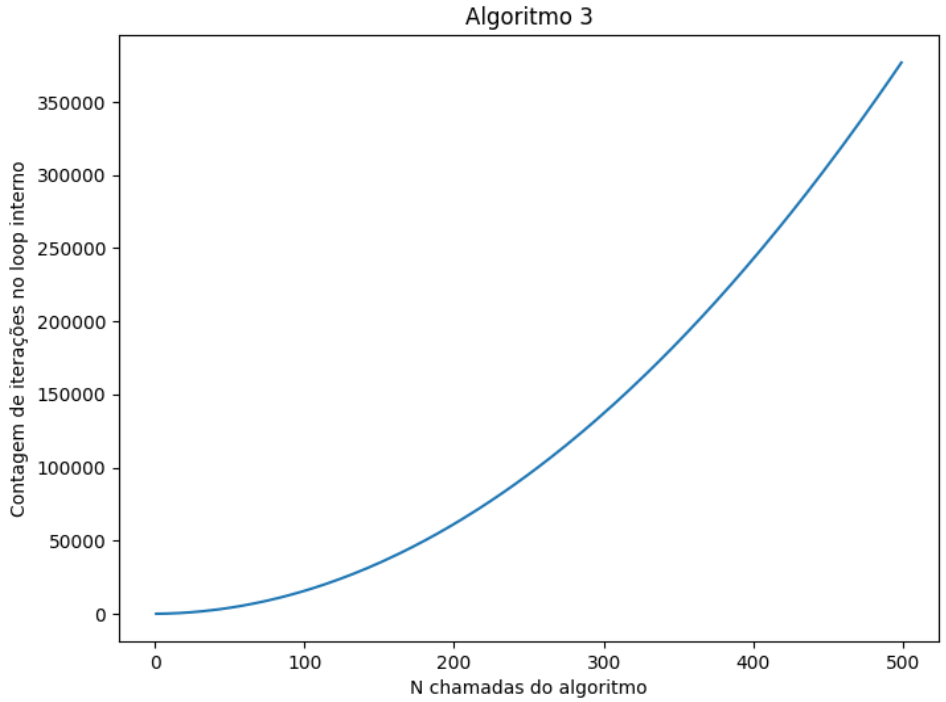


Esse é o gráfico de *n* por *Número de operações.*

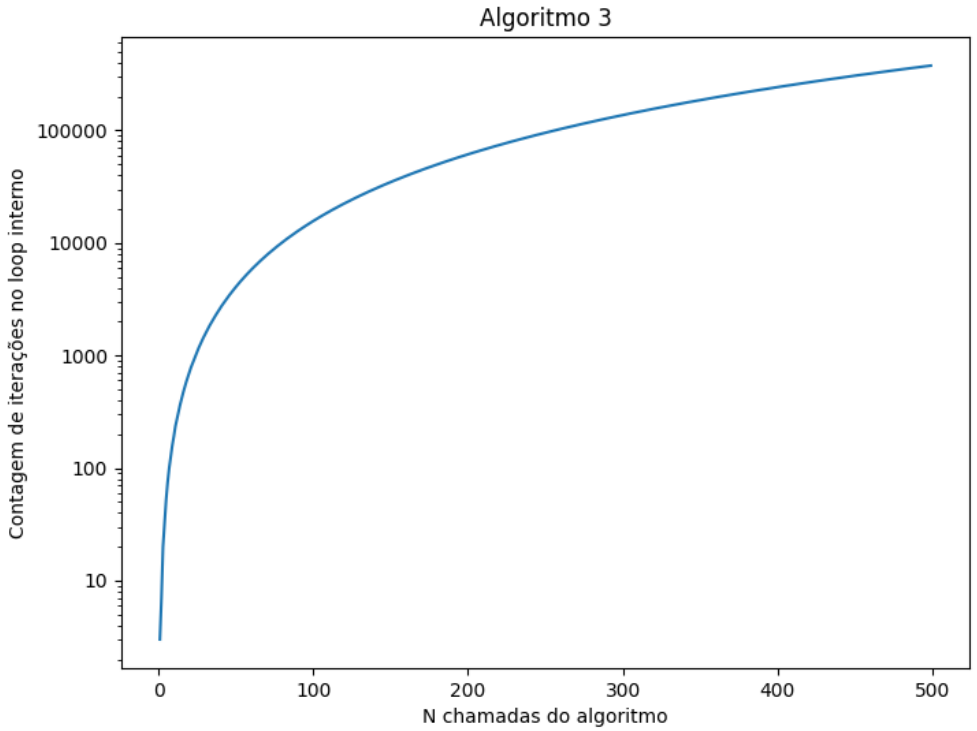
É possível perceber pelo gráfico, que não é necessário fazer nenhum tipo de manipulação para saber o crescimento dessa função.
$f(n) = n$

Algoritmo 3

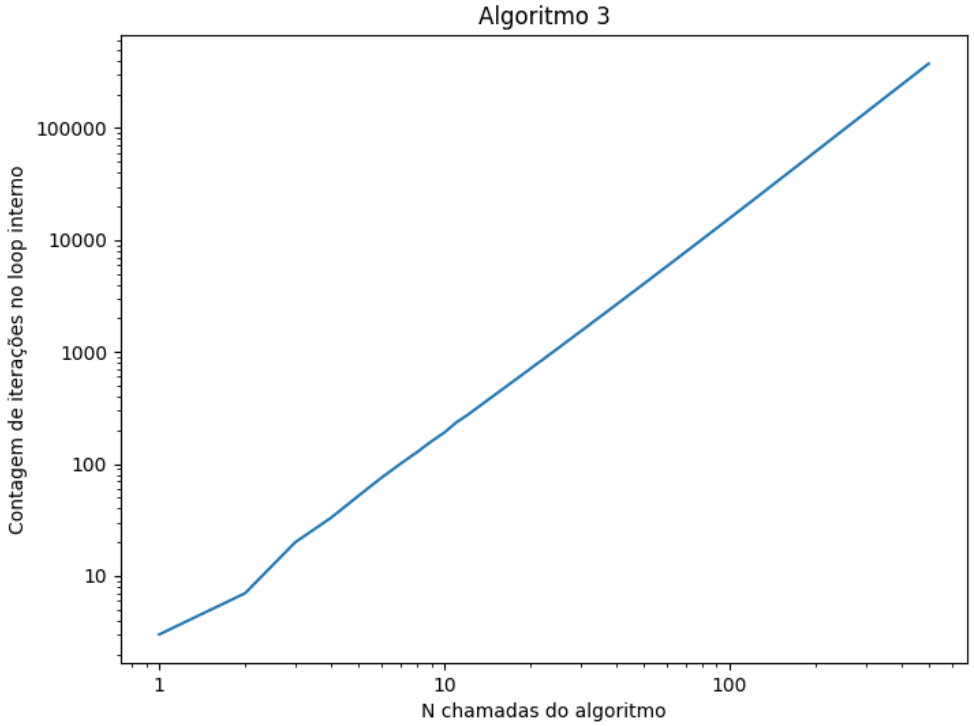
```
for (i = 1; i <= n * n; i += 2)
{
    for (j = i / 2; j <= 2 * i; j += i / 2 + 1)
    {
        for (k = j + 1; k <= n + j; k += k / 2 + 1)
        {
            res = res + abs(j - i);
            op_count++;
        }
    }
```



Ao observarmos o gráfico do algoritmo 3, podemos perceber a similaridade com o algoritmo 1. Aplicaremos o mesmo processo para descobrir a função f(n) que representa o crescimento da quantidade de operações.



Após aplicarmos a escala logarítmica no eixo y, ainda não chegamos a reta, aplicaremos a escala no eixo x.



Agora sim, chegamos em uma reta, então podemos aplicar a mesma fórmula, para descobrir a função que mostra o crescimento da quantidade de operações:

$$\frac{(log(y2) - log(y1))}{(log(x2) - log(x1))}$$

Aplicando a fórmula em 2 pontos:

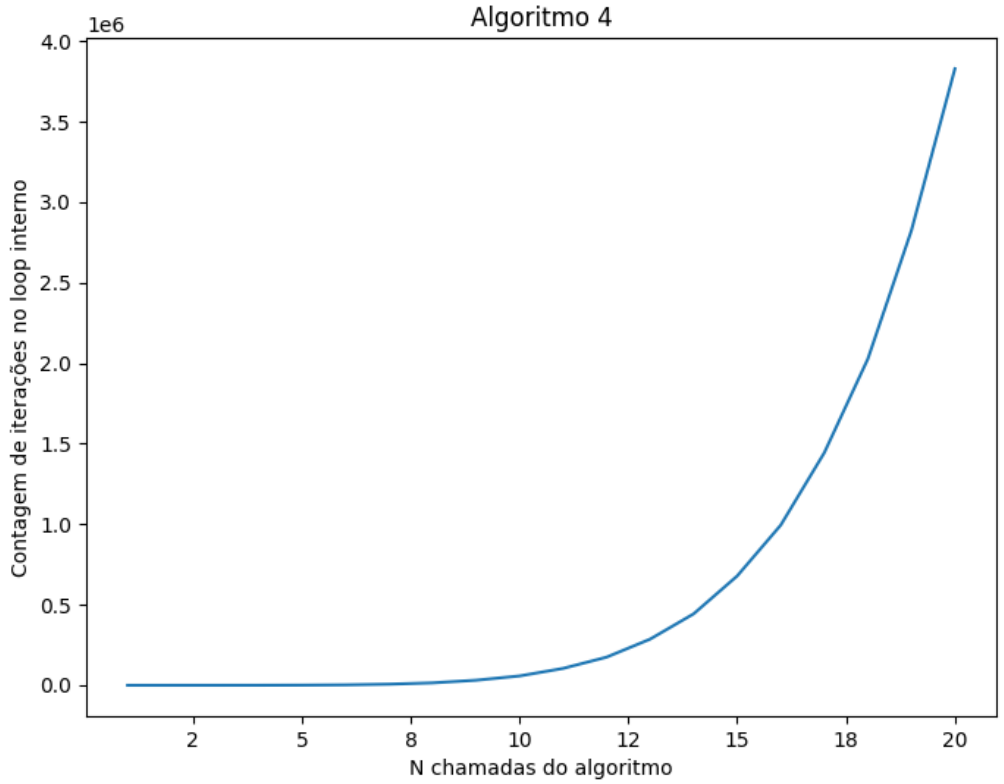
$$\frac{(log(377000) - log(15662))}{(log(499) - log(100))} = n$$

Temos o resultado *n ≅ 1.97893…*

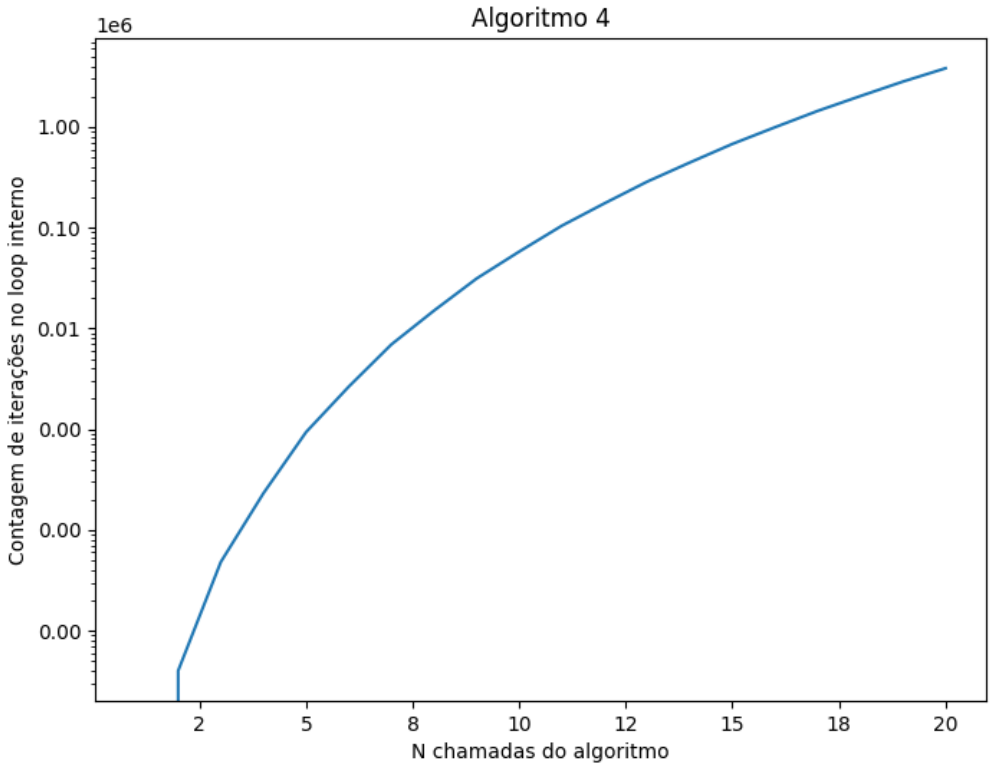
Logo a função que representa o crescimento é $f(n) \cong n^{1.9789}$...

Algoritmo 4

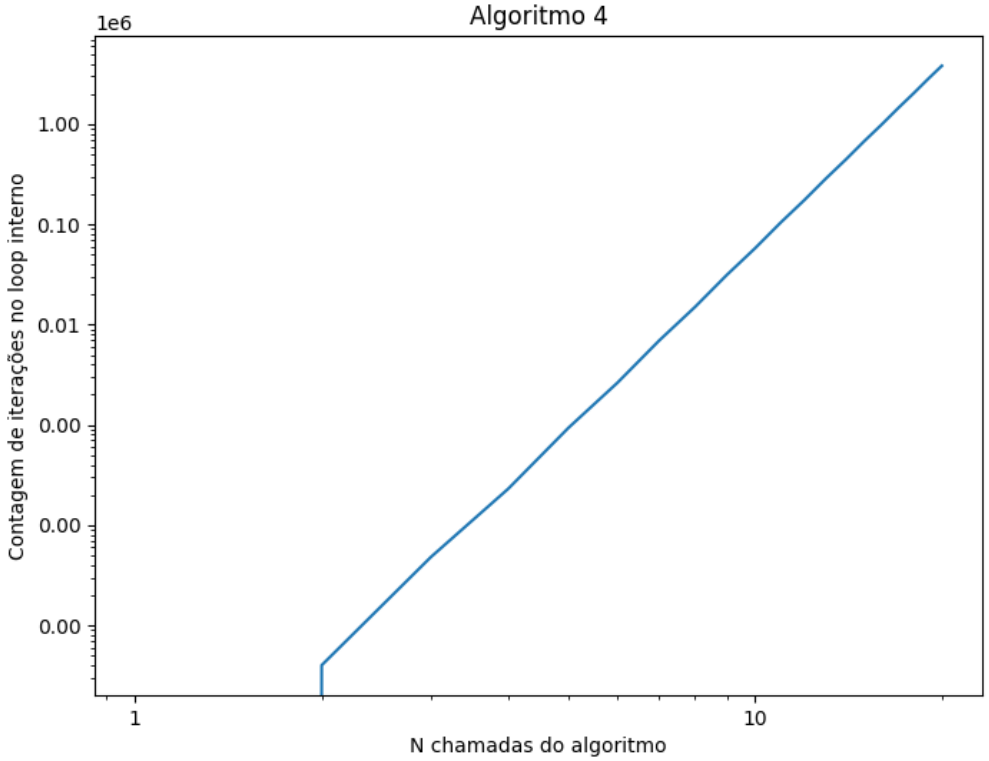
```
for (i = n; i <= n * n; i += 2)
{
    for (j = n + 1; j <= n * n; j += 2)
    {
        for (k = j; k <= 2 * j; k += 2)
        {
            res = res + 1;
            op_count++;
        }
    }
```



Mesma situação dos algoritmos 1 e 3, aplicaremos o mesmo processo.



Após aplicarmos a escala logarítmica no eixo y, ainda não chegamos a reta, aplicaremos a escala no eixo x.



Agora sim, chegamos em uma reta, então podemos aplicar a mesma fórmula, para descobrir a função que mostra o crescimento da quantidade de operações:

$$\frac{(log(y2) - log(y1))}{(log(x2) - log(x1))}$$

Aplicando a fórmula em 2 pontos:

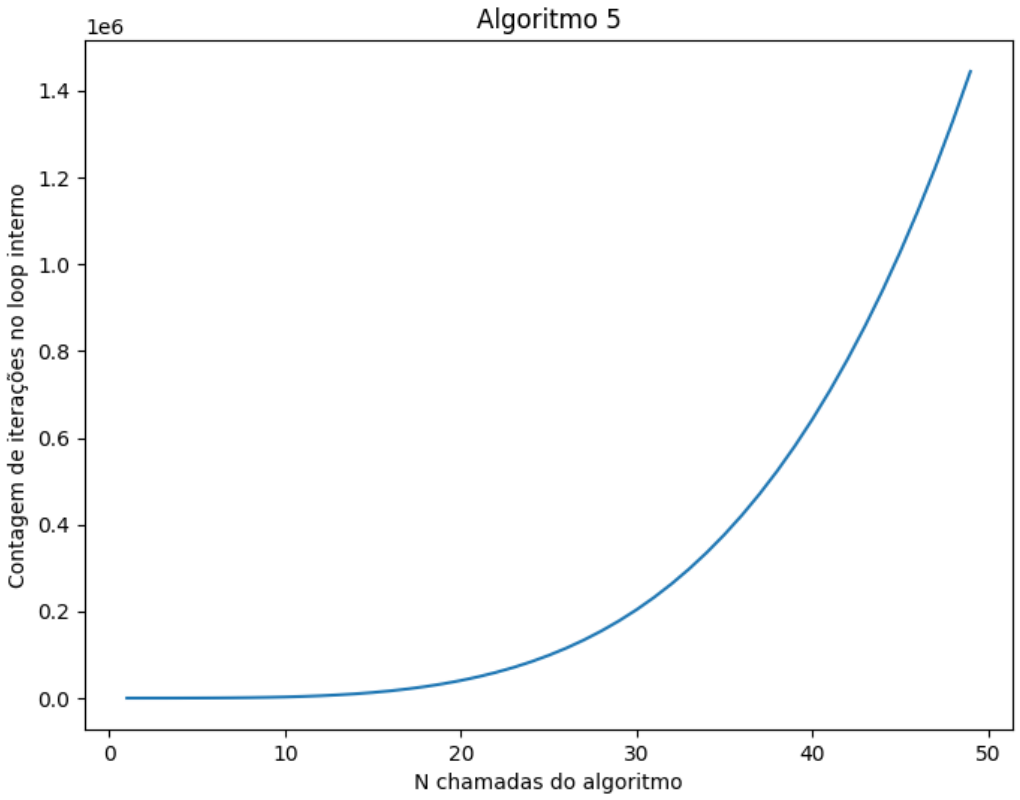
$$\frac{(log(3828595) - log(57960))}{(log(20) - log(10))} = n$$

Temos o resultado *n ≅ 6.04561…*

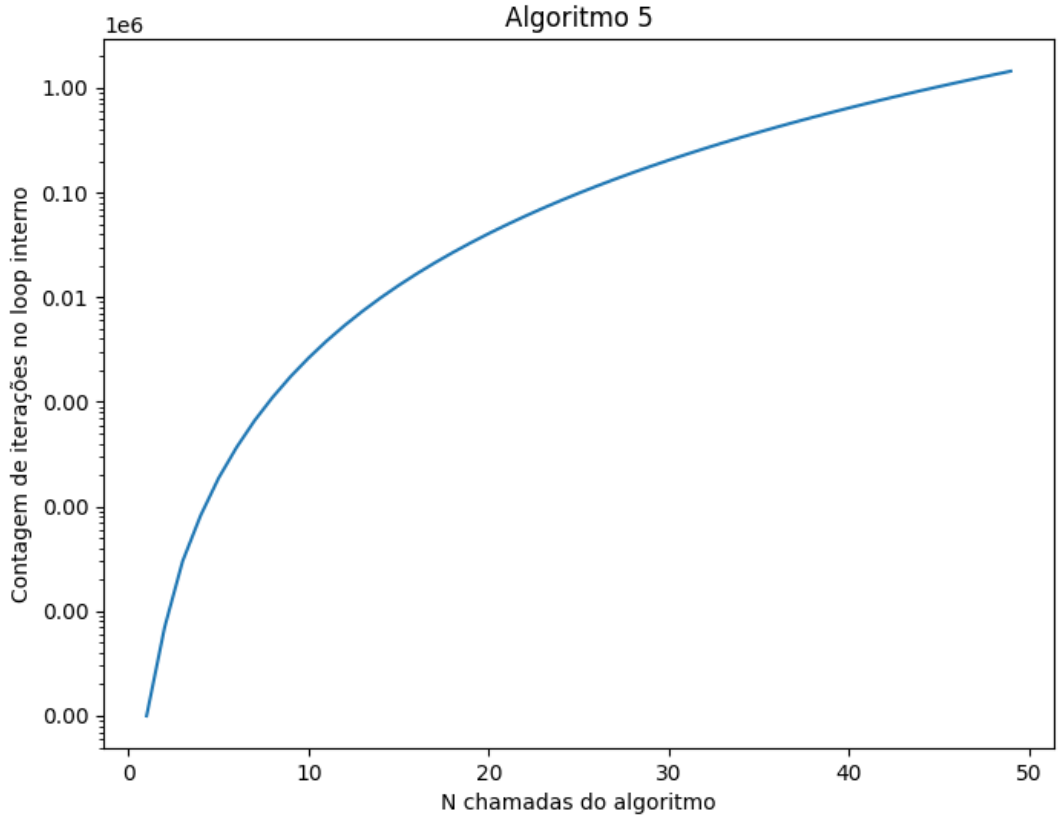
Logo a função que representa o crescimento é $f(n) \cong n^{6.04561}$...

Algoritmo 5

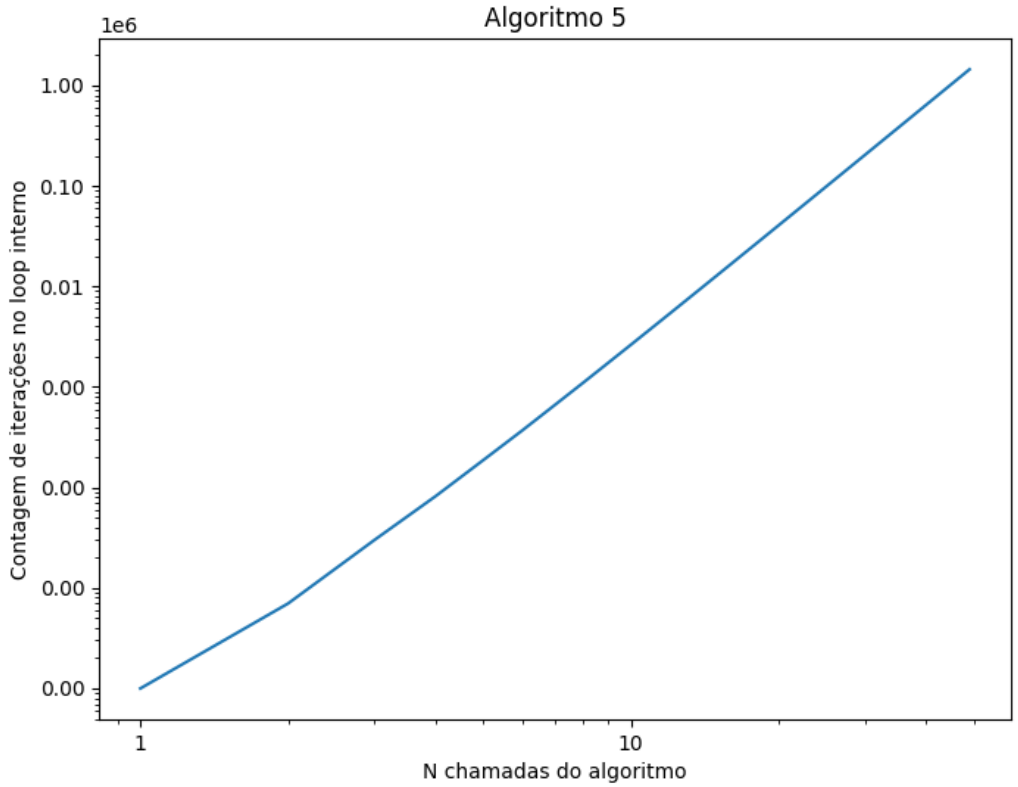
```
for (i = 1; i <= n * n; i += 1)
{
    for (j = 1; j <= i; j += 2)
    {
        for (k = n + 1; k <= 2 * i; k += i * j)
        {
            res = res + k + 1;
            op_count++;
        }
```



Mesma situação dos algoritmos 1,3 e 4, aplicaremos o mesmo processo.



Após aplicarmos a escala logarítmica no eixo y, ainda não chegamos a reta, aplicaremos a escala no eixo x.



Agora sim, chegamos em uma reta, então podemos aplicar a mesma fórmula, para descobrir a função que mostra o crescimento da quantidade de operações:

$$\frac{(log(y2) - log(y1))}{(log(x2) - log(x1))}$$

Aplicando a fórmula em 2 pontos:

$$\frac{(log(1444597) - log(2631))}{(log(49) - log(10))} = n$$

Temos o resultado n ≅ 3.96934…
Logo a função que representa o crescimento é $f(n) \cong n^{3.96934}$...

Ambiente de implementação

A implementação dos algoritmos foi feita na linguagem C++, e a plotagem dos gráficos foi feita em python, utilizando as bibliotecas Pandas e MatPlotLib.

O algoritmo a ser executado é selecionado através da linha de comando, junto com o valor de N a ser passado para os algoritmos da seguinte maneira:

```
t/alest1_t1/codes$ ./bin/main 3 50
```

Sendo o valor “3”, o algoritmo 3, e “50” o número de vezes que o algoritmo 3 será chamado.

Cada algoritmo emite um arquivo .csv, em uma pasta previamente escolhida, com a seguinte estrutura:

```
1;3
2;7
3;20
4;33
5;52
6;75
7;100
8;127
9;159
10;191
```

Dados retirados do algoritmo 3

A primeira coluna é o valor de “*n”* passado ao algoritmo, e a segunda coluna é o número de iterações realizada pelo laço “*for*” mais interno do algoritmo.

Para a formação visual do gráfico, é utilizado o código em python que vai até a pasta previamente escolhida e puxa os dados do .csv para um *dataframe* biblioteca *Pandas.*

```python
import pandas as pd
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt

df1 = pd.read_csv("./data/Algoritmo3.csv", sep=';', header = None)
df1.columns = ['x', 'y'] #indexação dos dados
#formatação da GUI
f = plt.figure()
f.set_figwidth(8)
f.set_figheight(6)
plt.title('Algoritmo 3')
plt.xlabel('N chamadas do algoritmo')
plt.ylabel('Contagem de iterações no loop interno')

#Linhas que selecionam a escala do gráfico
plt.plot(df1['x'], df1['y'])
#plt.semilogy(df1['x'], df1['y'])
#plt.loglog(df1['x'], df1['y'])

# define a função de formatação de rótulo personalizada
def format_x_ticks(x, pos):
    """
    Retorna a notação decimal para os rótulos do eixo x em vez da notação científica.
    """
    return '{:.0f}'.format(x)
# atribui a função de formatação de rótulo personalizada para o eixo x
plt.gca().xaxis.set_major_formatter(ticker.FuncFormatter(format_x_ticks))
plt.gca().yaxis.set_major_formatter(ticker.ScalarFormatter())

plt.show()
```

Após a inserção dos gráficos no *dataframe, plotamos* o gráfico com a Matplotlib, concluindo dessa maneira a análise dos algoritmos.

Toda a implementação utilizada nesse relatório se encontra neste repositório: https://github.com/LucasDamo22/alest1_t1