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.

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)

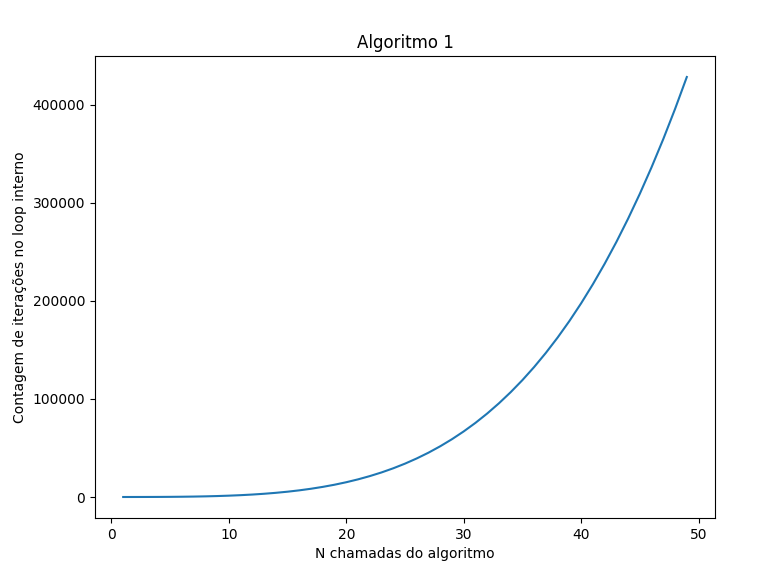
{

res = res + n - 1;

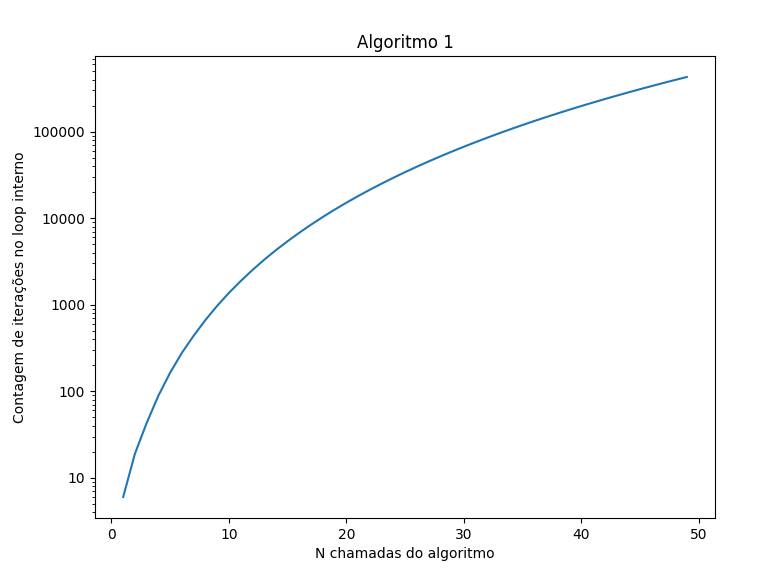
op\_count++;

}

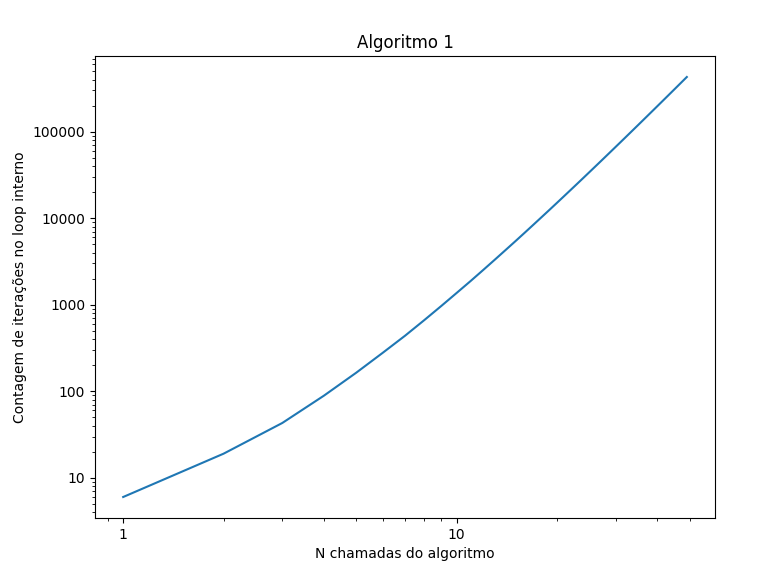
}

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:



Aplicando os valores temos a seguinte fórmula:



Tendo o resultado *n ≅ 3.6154*

Logo temos que a função que demonstra o crescimento do algoritmo 1

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)

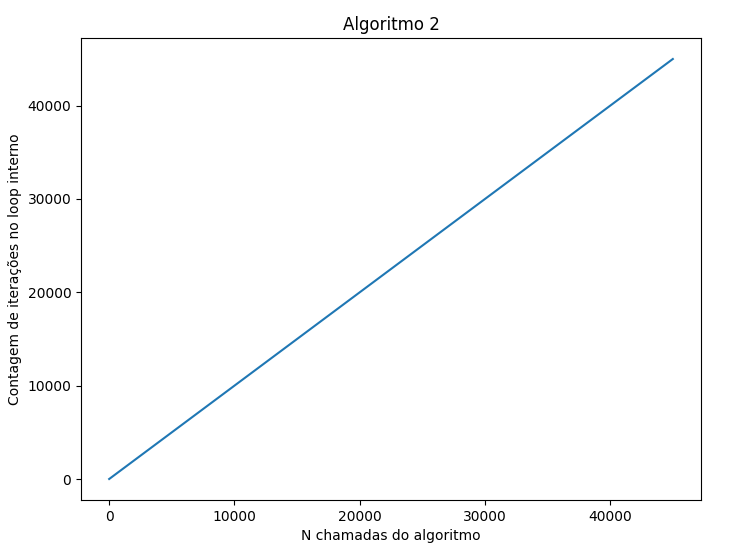
{

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.

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)

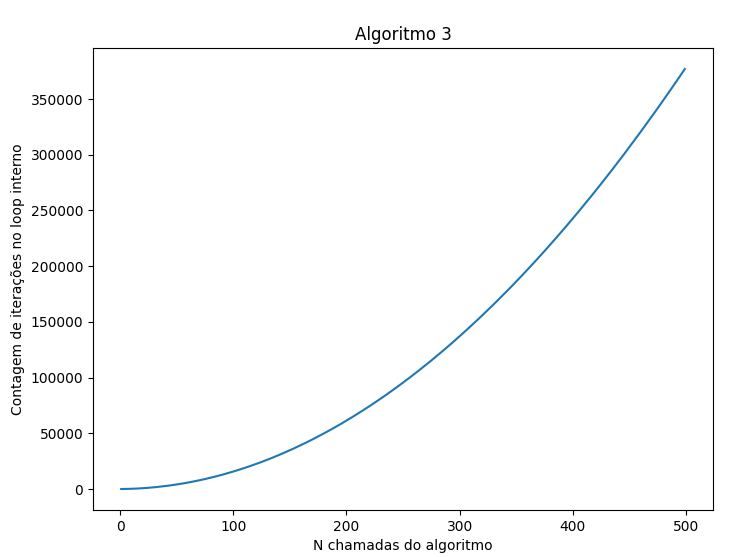
{

res = res + abs(j - i);

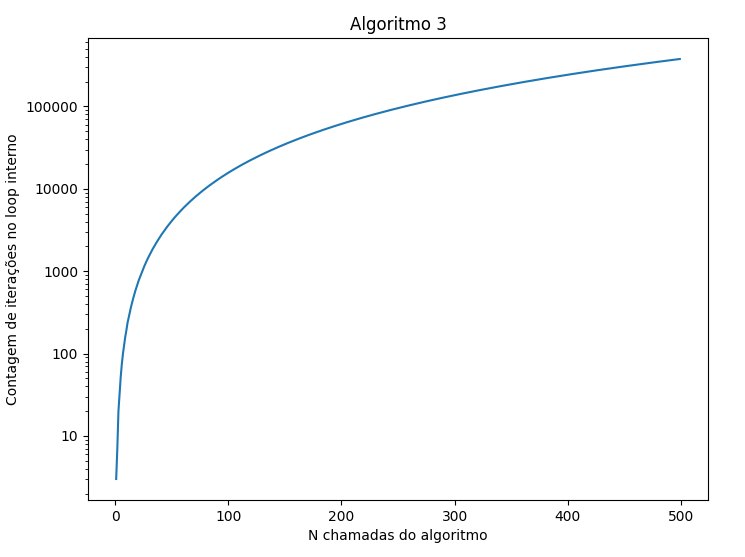
op\_count++;

}

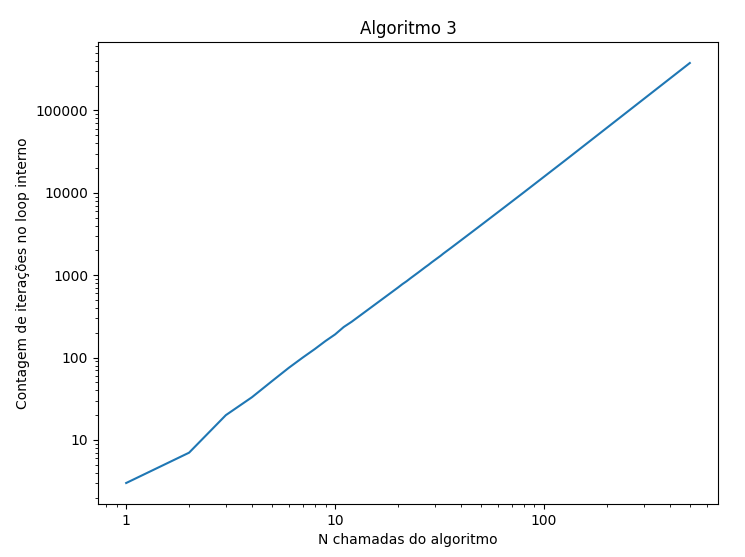
}



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.



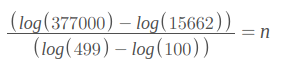
Após aplicarmos a escala logarítmica no eixo y, ainda não chegamos a reta, aplicaremos a escala no eixo x.



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



Aplicando a fórmula em 2 pontos:



Temos o resultado *n ≅ 1.97893…*

Logo a função que representa o crescimento é

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)

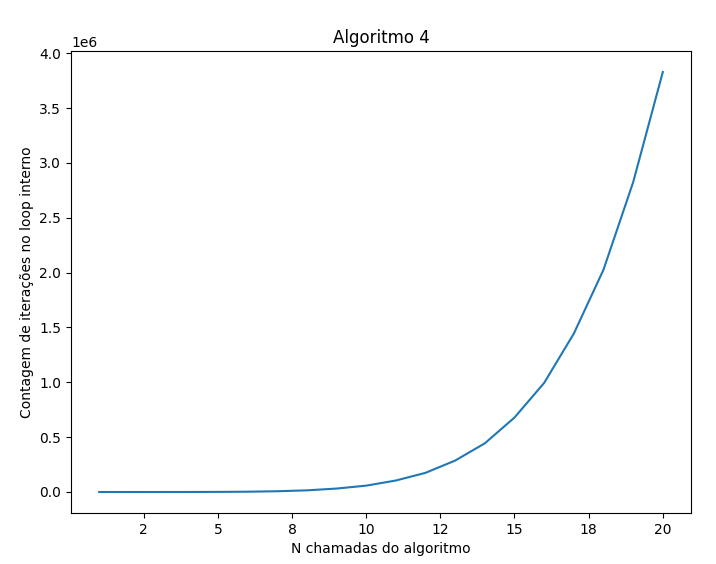
{

res = res + 1;

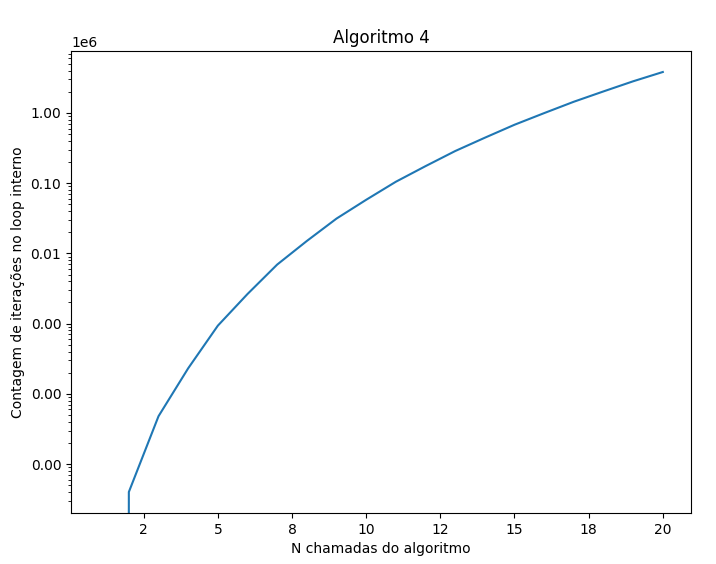
op\_count++;

}

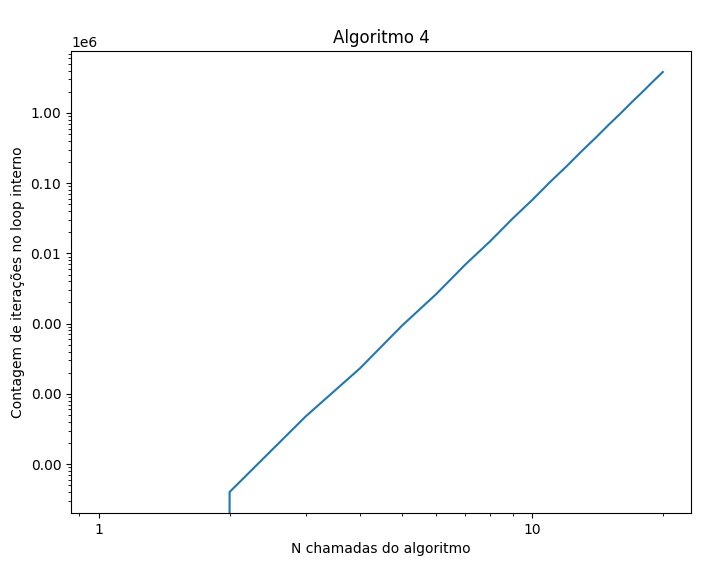
}



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



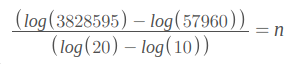
Após aplicarmos a escala logarítmica no eixo y, ainda não chegamos a reta, aplicaremos a escala no eixo x.



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



Aplicando a fórmula em 2 pontos:



Temos o resultado *n ≅ 6.04561…*

Logo a função que representa o crescimento é

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)

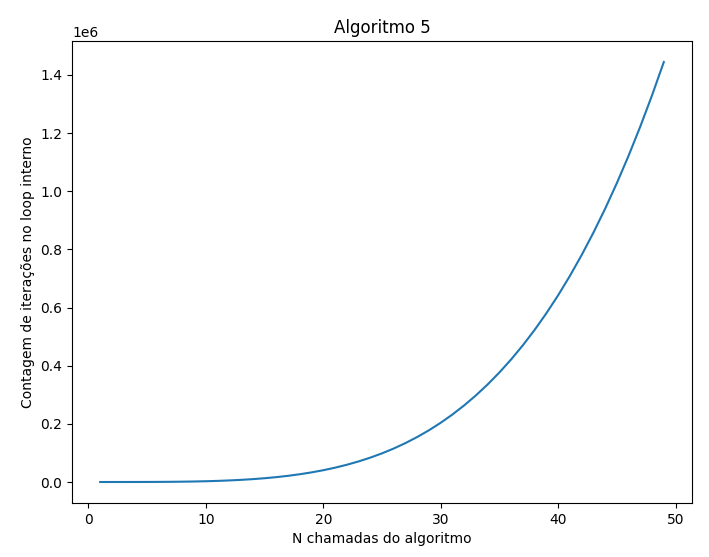
{

res = res + k + 1;

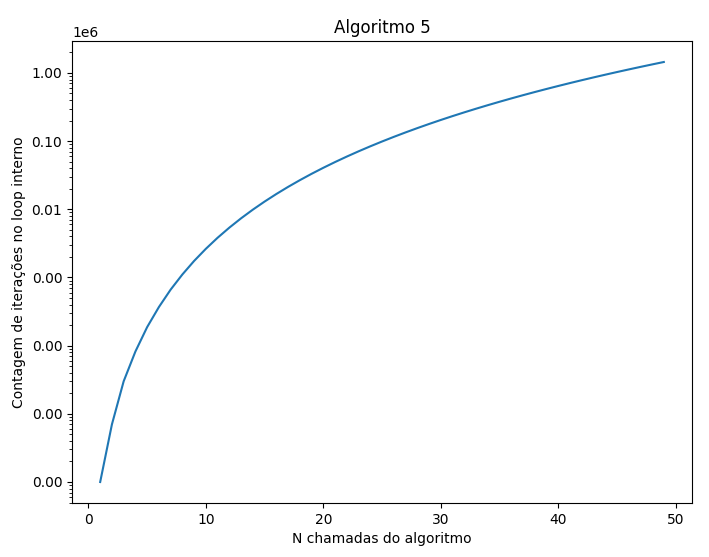
op\_count++;

}

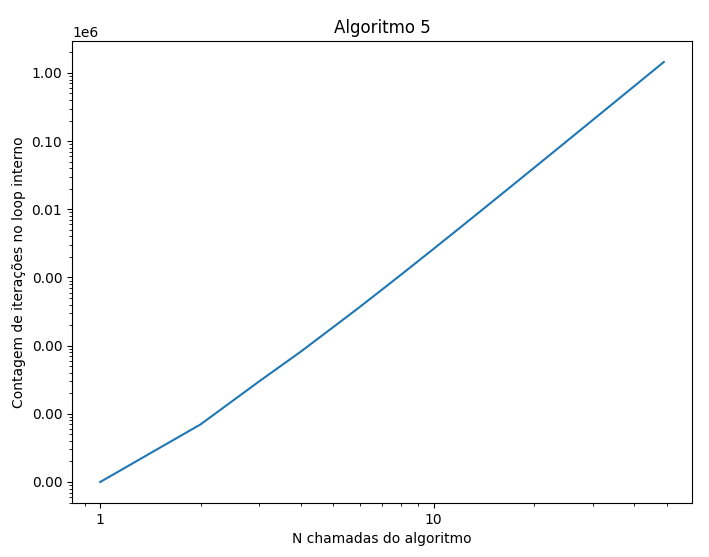
}



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:



Aplicando a fórmula em 2 pontos:



Temos o resultado n ≅ 3.96934…

Logo a função que representa o crescimento é