Pontifícia Universidade Católica do Rio Grande do Sul
Algoritmos e Estruturas de Dados I
Engenharia de Software

Mateus Campos Caçabuena

Relatório do Trabalho 1
Experimentos em Complexidade

Porto Alegre
2023

SUMÁRIO

Sumário

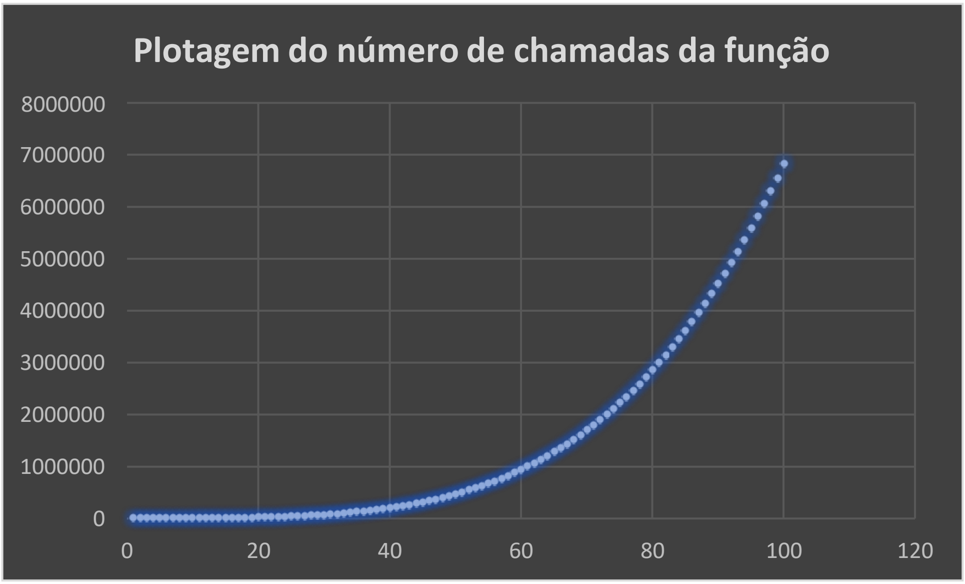
1. Introdução 3
2. 1º Algoritmo..4
3. 2º Algoritmo..6
4. 3º Algoritmo..8
5. 4º Algoritmo..10
6. 5º Algoritmo..12
7. Conclusão..14

Introdução

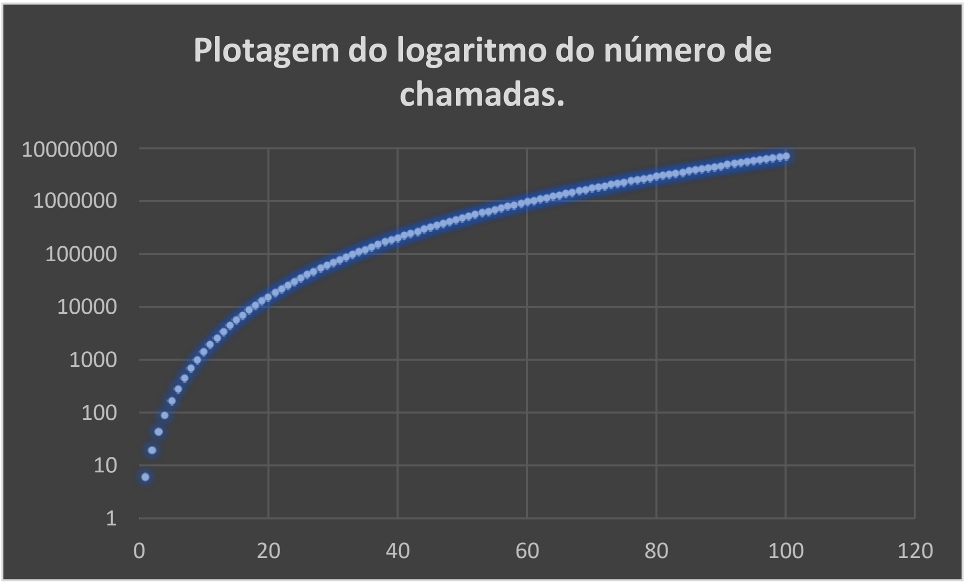
O presente relatório de experimentos em complexidade do curso de Engenharia de Software, possui como objetivo determinar a função dos algoritmos propostos pelo enunciado do exercício. Para tanto foi utilizado os gráficos e comparações deles no Excel, como forma de realizar a coleta de dados.

1º Algoritmo

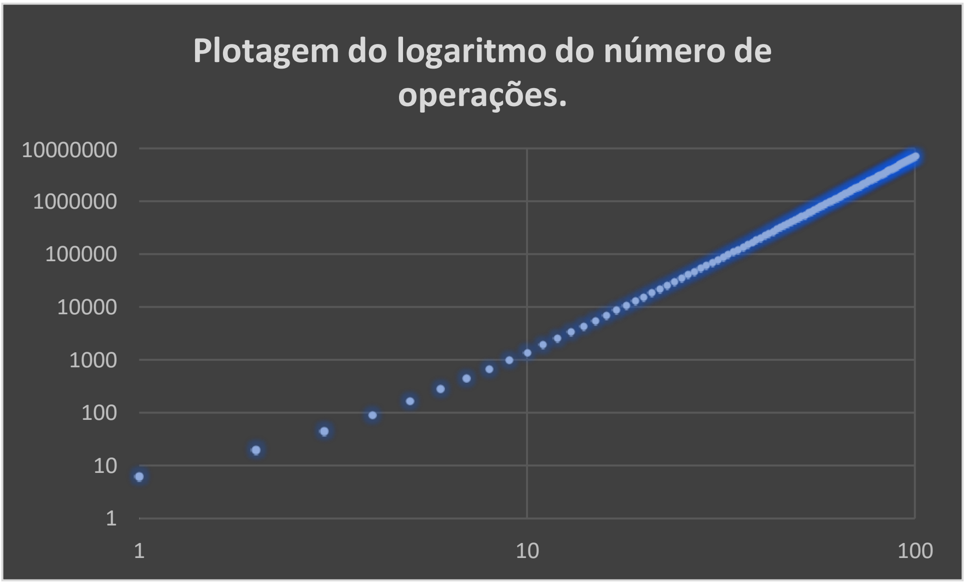
```
// Código implementado para medir operações:
public class exercicio1

{
    public static void main (String[]  args)
        {for (int i=1; i<=100; i++) {
            System.out.println(i + "\t" + f(i));
        }
    }
    public static int f( int n ) {
        int i, j, k, res = 0, cont_op =
        0;for(i=1; i<=n+1; i+=1)
            for(j=1; j<=i*i; j+=i+1)
                for(k=i/2; k<=n+j; k+=2)
                {
                    res = res + n-1;
                    cont_op++;
                }
        return cont_op;
    }
}
```



Agora, será feita a extração do logaritmo destes números para saber se a função é exponencial ou polinomial:



Como o gráfico apresenta uma curva logarítmica, então a próxima possibilidade é de uma funçãopolinomial. A partir disto, é necessário "espremermos" também o eixo *x*, a fim de saber se trata de um polinômio:



Percebe-se que o gráfico apresentou uma quase reta, portanto, trata-se de uma função polinomial. A conta para descobrir o *b* e consequentemente o expoente de uma função polinomial é:

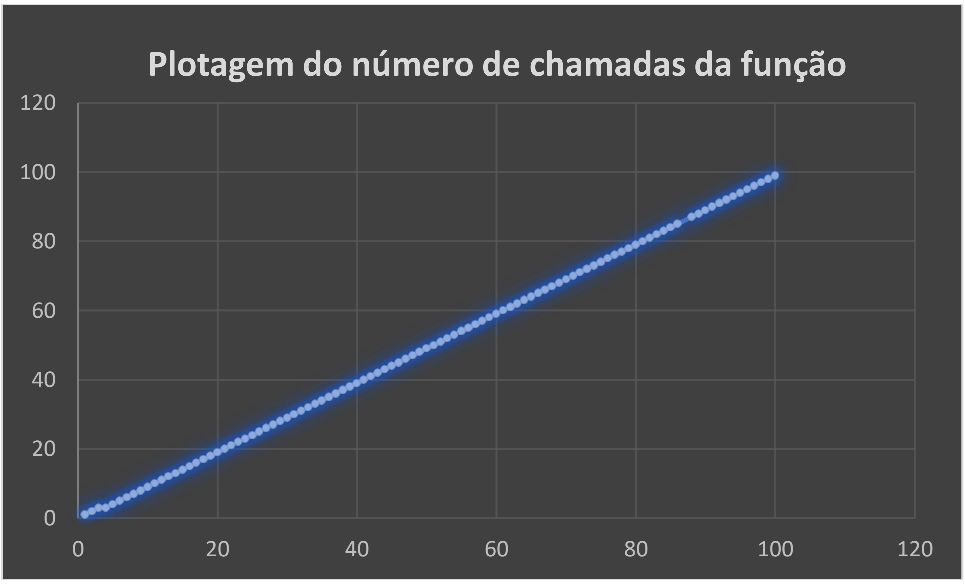
$f(1) = 6$
$f(100) = 6809401$

$b \approx (log(6809401) - log(6)) / (log(100) - log(1)) = 3.0274788299$

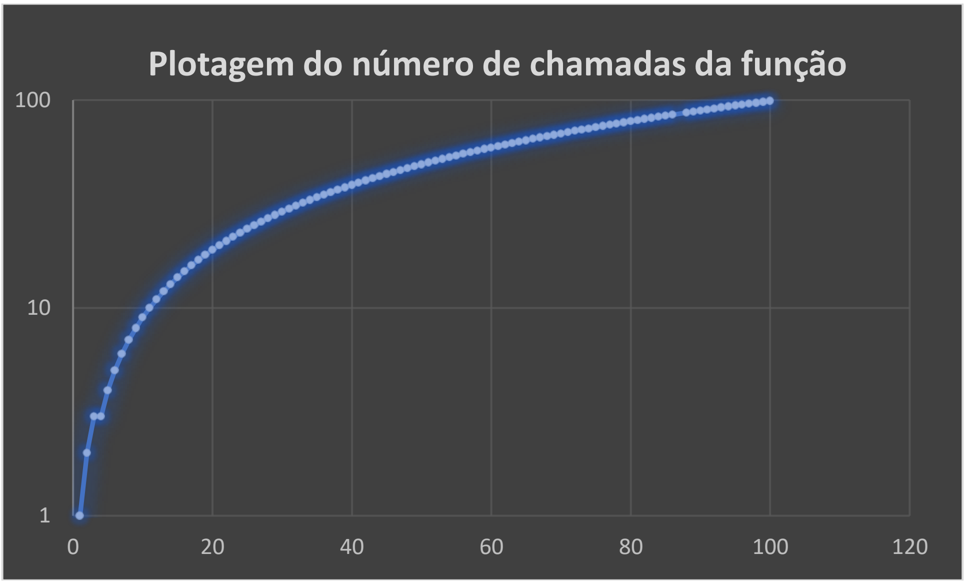
Ou seja, isto sugere que a função f(n) cresce como uma função de segundo grau:
$\boldsymbol{f(n) \approx n^{3,02}}$

2º Algoritmo

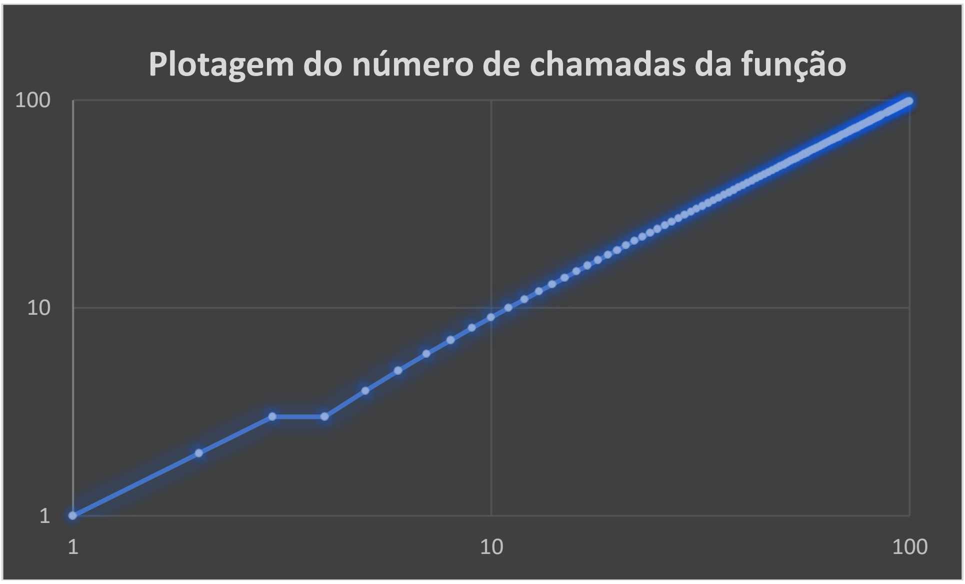
```
// Código implementado para medir operações:

  public static void main(String[] args) throws Exception {
      for (int i = 1; i <= 100; i++) {
          System.out.println(i + "\t" + f(i));
      }

  }

  public static int f(int n) {
      int i, j, k, res = 0;
      int cont_op = 0;
      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;
                  cont_op++;
              }
      return cont_op;
  }
}
```



Agora, será feita a extração do logaritmo destes números para saber se a função é exponencial ou polinomial:



Como o gráfico apresenta uma curva logarítmica, então a próxima possibilidade é de uma função polinomial. A partir disto, é necessário "espremermos" também o eixo *x*, a fim de saber se trata de um polinômio:



Percebe-se que o gráfico apresentou uma quase reta, portanto, trata-se de uma função polinomial. A conta para descobrir o *b* e consequentemente o expoente de uma função polinomial é:

$f(1) = 1$

$f(100) = 99$

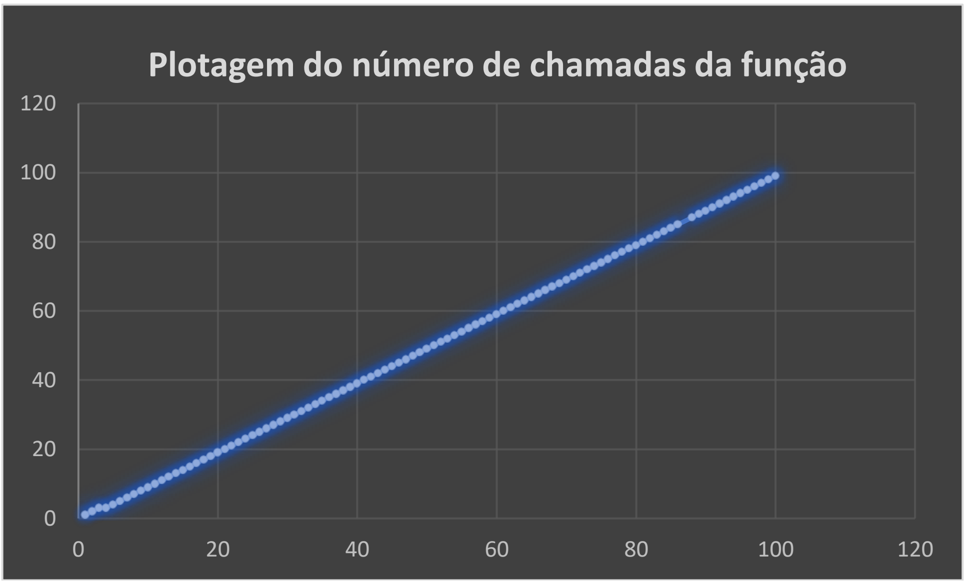
$b \approx (log(99) - log(1)) / (log(100) - log(1)) = 0.9978175973$

$\boldsymbol{f(n) \approx n^1}$

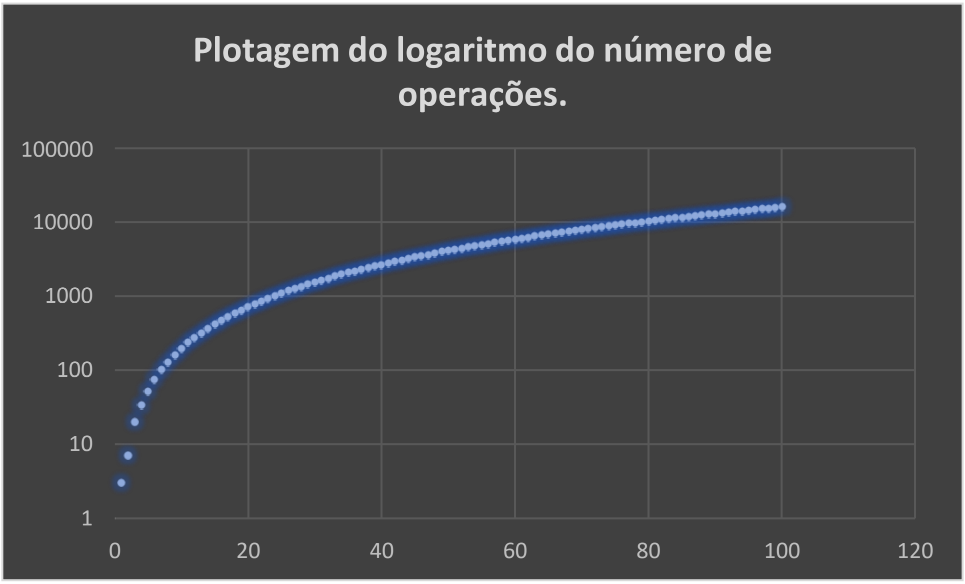
3º Algoritmo

// Código implementado para medir operações:

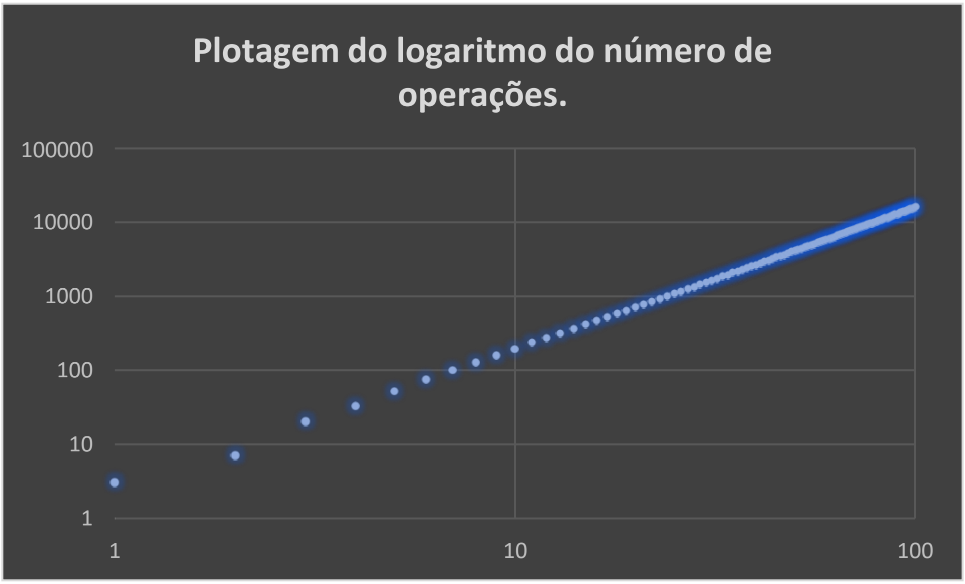
```
public class exercicio3{
    public static void main
    (){
        for (int i = 1; i <= 100; i++){
            System.out.println(i + "\t" + f(i));
        }
    }
    public static int f( int n ) {
        int i, j, k, res = 0, cont_op =
        0;for( i = 1; i <= n*n; i += 2 )
            for( j = i/2; j <= 2*i; j += i/2+1 )
                for( k = j+1; k <= n+j; k += k/2+1 ) {
                    res = res + (j-i);
                    cont_op++;
                }
        return cont_op;
    }
}
```



Agora, será feita a extração do logaritmo destes números para saber se a função é exponencial ou polinomial:



Como o gráfico apresenta uma curva logarítmica, então a próxima possibilidade é de uma funçãopolinomial. A partir disto, é necessário "espremermos" também o eixo *x*, a fim de saber se trata de um polinômio:



Percebe-se que o gráfico apresentou uma quase reta, portanto, trata-se de uma função polinomial. A conta para descobrir o *b* e consequentemente o expoente de uma função polinomial é:

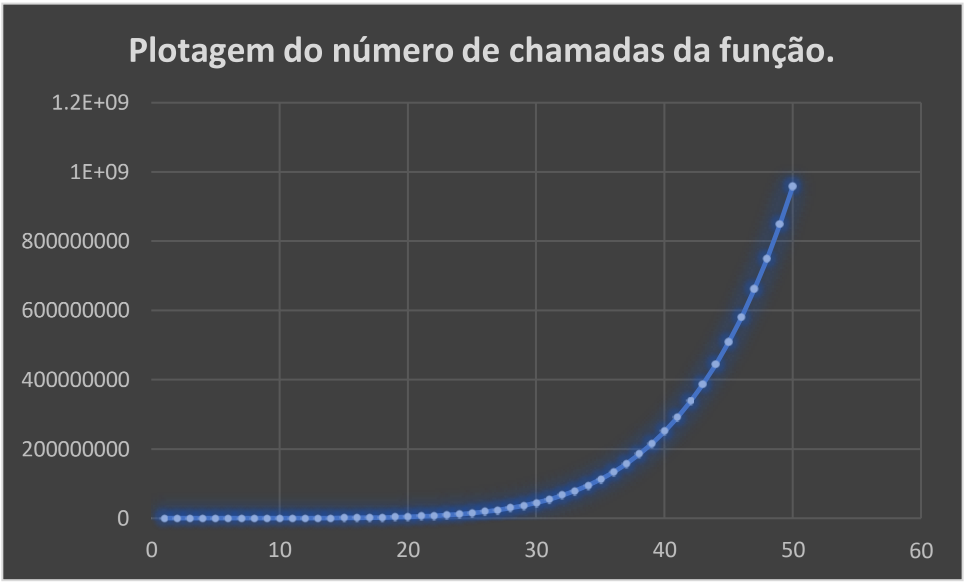
$f(1) = 3$
$f(100) = 15662$

$b \approx (log(15662) - log(3)) / (log(100) - log(1)) = 1.85886298246$

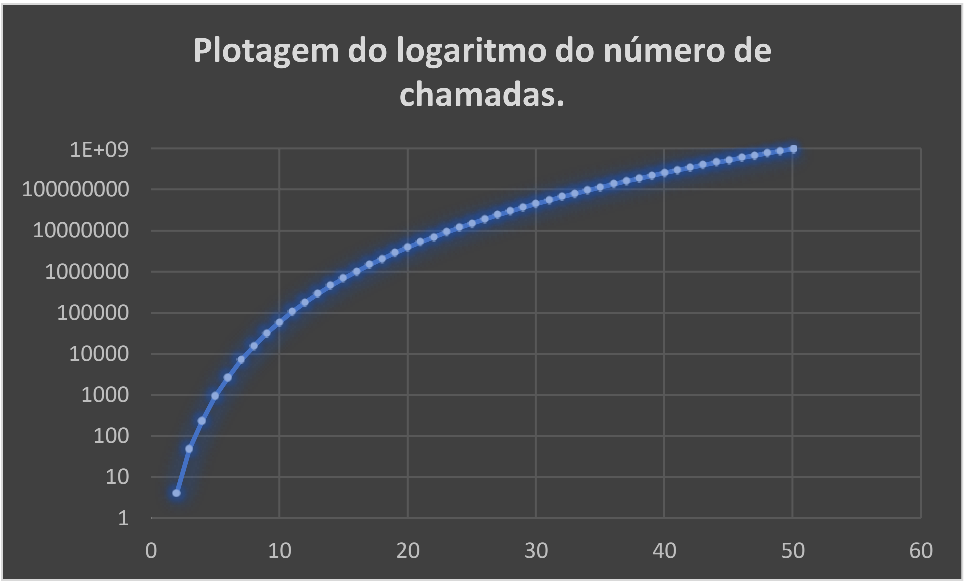
Ou seja, isto sugere que a função f(n) cresce como uma função de segundo grau:
$\boldsymbol{f(n) \approx n^{1,86}}$

4º Algoritmo

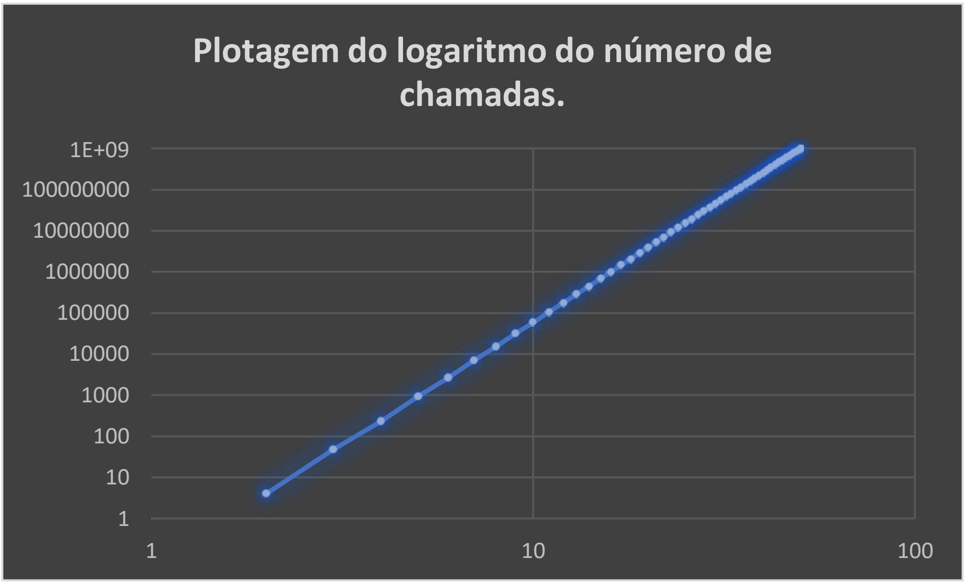
```
// Código implementado para medir operações:
public class exercicio4{
    public static void main
    (){
        for (int i = 1; i <= 50; i++){
            System.out.println(i + "\t" + f(i));
        }
    }
    public static int f( int n ) {
        int i, j, k, res = 0, cont_op =
        0;for( 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;
                    cont_op++;
                }
        return cont_op;
    }
}
```



Agora, será feita a extração do logaritmo destes números para saber se a função é exponencial ou polinomial:



Como o gráfico apresenta uma curva logarítmica, então a próxima possibilidade é de uma funçãopolinomial. A partir disto, é necessário "espremermos" também o eixo *x*, a fim de saber se trata de um polinômio:



Percebe-se que o gráfico apresentou uma quase reta, portanto, trata-se de uma função polinomial. A conta para descobrir o b e consequentemente o expoente de uma função polinomial é:

$f(1) = 0$
$f(50) = 6809401$

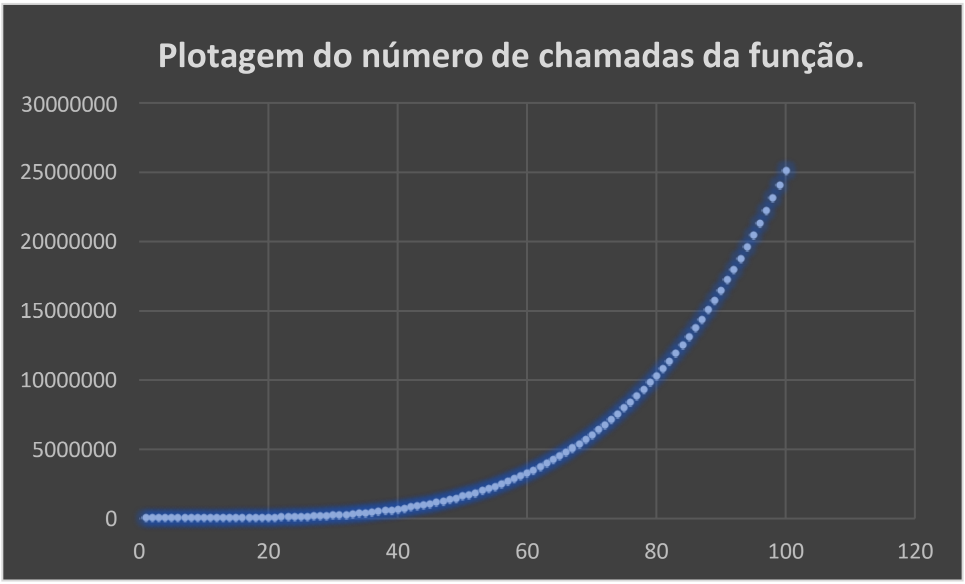
$b \approx log(958180300) / (log(50) - log(1)) = 5.28640718475$

Ou seja, isto sugere que a função f(n) cresce como uma função de segundo grau:
$\boldsymbol{f(n) \approx n^{5,29}}$

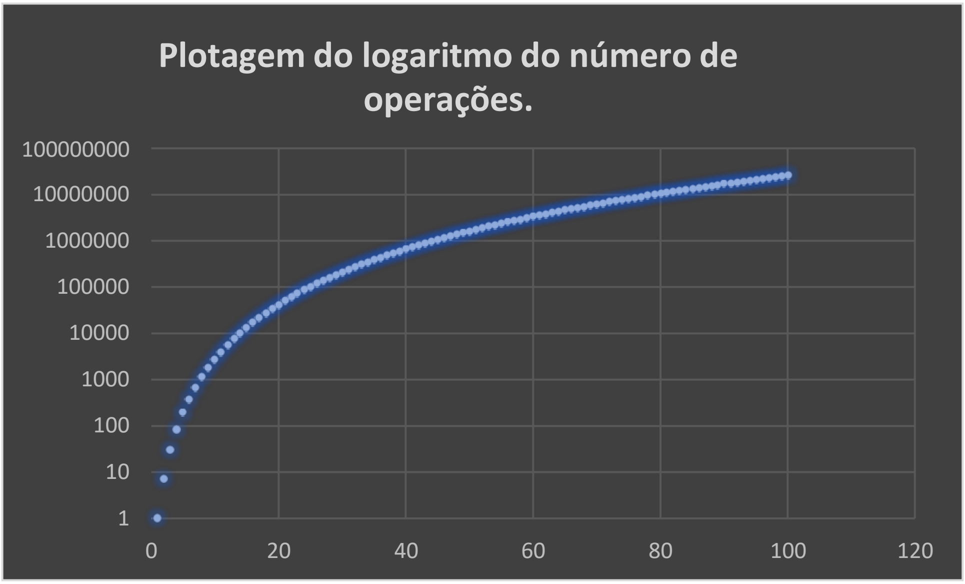
5º Algoritmo

// Código implementado para medir operações:

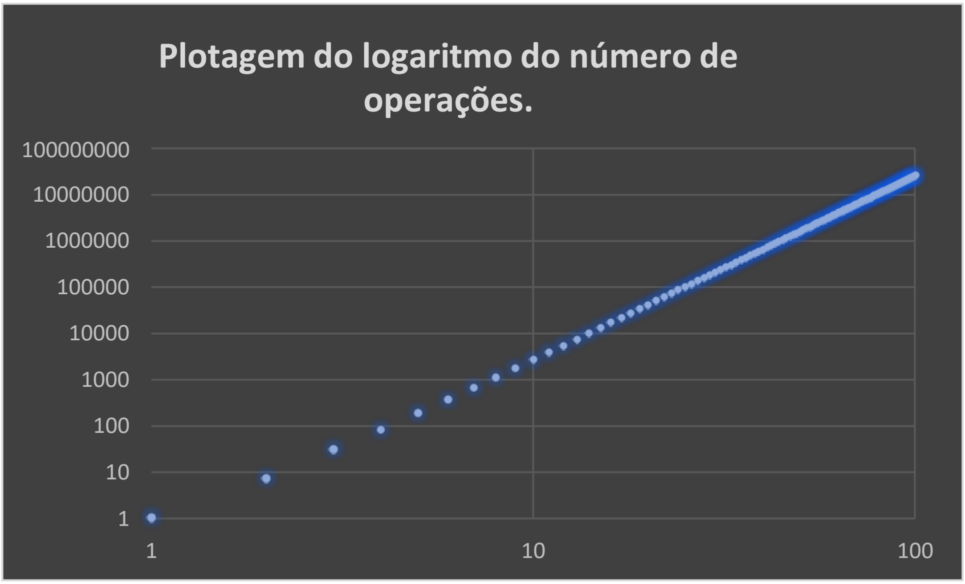
```
public class exercicio5{
    public static void main
    (){
        for (int i = 1; i <= 100; i++){
            System.out.println(i + "\t" + f(i));
        }
    }
    public static int f( int n ) {
        int i, j, k, res = 0, cont_op =
        0;for( i = 1; i <= n*n; i += 1 )
            for( j = 1; j <= i; j += 2 )
                for( k = n+1; k <= 2*i; k += i*j ) {
                    res = res + k+1;
                    cont_op++;

                }
        return cont_op;
    }
}
```



Agora, será feita a extração do logaritmo destes números para saber se a função é exponencial ou polinomial:



Como o gráfico apresenta uma curva logarítmica, então a próxima possibilidade é de uma funçãopolinomial. A partir disto, é necessário "espremermos" também o eixo *x*, a fim de saber se trata de um polinômio:



Percebe-se que o gráfico apresentou uma quase reta, portanto, trata-se de uma função polinomial. A conta para descobrir o *b* e consequentemente o expoente de uma função polinomial é:

$f(1) = 1$
$f(100) = 25014250$

$b \approx (log(25014250) - log(1)) / (log(100) - log(1)) = 3.699093743$

Ou seja, isto sugere que a função f(n) cresce como uma função de segundo grau:

$\boldsymbol{f(n) \approx n^{3,70}}$

Conclusão

Após estudar e analisar todos os 5 algoritmos, percebe-se como uma simples alteração no código pode mudar drasticamente o custo, como é mostrado nos gráficos. O algoritmo que possui o menor custo de operações é o 2º, pois possui o menor número no expoente comparado às outras funções polinomiais. Enquanto isso, o algoritmo que possui o maior custo de operações é o 6º, pois cresce exponencialmente e, como foi visto no gráfico do algoritmo, a partir da 25ª contagem ele começa a consumir operações de forma mais rápida que qualquer outro algoritmo anterior. Voltando a citar o fato de pequenas alterações gerarem grandes diferenças, deve ser destacado a diferença gritante que há entre a quantia de operações do 2º e 3º algoritmo. O 3º algoritmo não chega nem perto de ter a função que mais consome operações deste relatório, nem mesmo citando apenas as funções polinomiais. Mas possui a menor diferença no numeral do expoente do algoritmo que menos consome, já mostrando uma diferença enorme. Em 100 execuções, um consumiu 99 operações contra 15662 operações do outro e, detalhe: o expoente do 3º algoritmo é “apenas” aproximadamente 0,86 maior que o expoente do 2º.