Algoritmos e Estruturas de Dados I - Trabalho 1
Experimentos em Complexidade

Daniele Ramos de Souza

12 de agosto de 2015

Resumo

Este trabalho é a primeira das avaliações da disciplina de Algoritmos e Estruturas de Dados I e tem como objetivo analisar o desempenho de oito algoritmos propostos pelo professor João Batista, usando técnicas experimentais de maneira comparativa.

Introdução

Analisar a eficiência de um algoritmo é uma tarefa difícil, pois ela depende de uma série de fatores. Analisar a eficiência de um algoritmo em comparação com outro algoritmo, por sua vez, é uma tarefa relativamente mais simples. E é o que vamos fazer: analisar o desempenho de oito algoritmos de maneira comparativa, usando uma técnica experimental que possui como vantagem a simplicidade.

Todavia, esta técnica possui duas desvantagens. A primeira delas é que ela só pode ser aplicada depois que o programa estiver pronto. Logo, se o resultado for de um algoritmo ruim ou menos eficiente do que outro algoritmo, terá sido uma perda de tempo e, em uma situação empresarial, de dinheiro. A segunda é que, pelo fato de utilizarmos dados de execução, a escolha dos dados de entrada pode influenciar a obtenção de informações realistas.

Mas estamos falando da primeira das avaliações da disciplina de Algoritmos e Estruturas de Dados I, então tudo se trata de aprendizado. Vamos analisar o desempenho dos oito algoritmos propostos pelo professor João Batista, tentando descobrir todos os detalhes possíveis sobre eles, a partir de uma "receita de bolo"que explicaremos no próximo tópico.

A "receita do bolo"

A primeira coisa que iremos fazer é verificar o comportamento do algoritmo à medida em que os dados de entrada variam. Para isso, vamos programá-lo para

ser executado com diferentes valores de n (escolhemos o intervalo entre 1 e 25 como padrão) e vamos calcular o número de operações, ou seja, identificar a operação mais significativa e utilzar uma variável para contar quantas vezes ela se repete em função de n.

Tendo estas informações, plotaremos gráficos com o Gnuplot a fim de visualizar o comportamento do algoritmo e identificar sua função característica. Existem três casos principais:

1. Função linear: esta é a mais fácil de ser visualizada. Se o gráfico tomar a forma de uma reta, certamente estamos falando de uma função linear. Resta saber a inclinação da reta (r), que pode ser encontrada com a fórmula abaixo:

$$r \approx \frac{y - y_o}{x - x_o}$$

,

2. Função exponencial: se não estivermos falando de uma função linear, teremos que aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$. Se o gráfico tomar a forma de uma reta, certamente estamos falando de uma função exponencial. Resta saber o expoente. Para isso, precisamos encontrar a inclinação da reta (r) utilizando dois valores obtidos com o algoritmo e assim saberemos a base (b), por meio das fórmulas a seguir, respectivamente:

$$r \approx \frac{log(f(x)) - log(f(x_o))}{x - x_o}$$

$$b = 10^r$$

3. Função polinomial: se não estivermos falando de uma função linear e nem de uma função exponencial ao aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$, teremos que aumentar a escala em ambos os eixos. Se o gráfico tomar a forma de uma reta, estamos falando de uma função polinomial. Resta saber a grandeza. Para isso, também utilizaremos dois valores obtidos com o algoritmo para descobrir a inclinação da reta (r), conforme a seguinte fórmula:

$$r \approx \frac{log(f(x)) - log(f(x_o))}{log(x) - log(x_o)}$$

O resultado nos dirá o grau da função.

É importante destacar que estes são resultados experimentais, sujeitos a erros, mas muito próximos. Com eles, podemos estimar a velocidade do algoritmo e o seu tipo de crescimento.

Nos próximos tópicos, analiseramos os oitos algoritmos propostos pelo professor João Batista com base na "receita de bolo"que acabamos de explicar.

Algoritmo 1

```
int f1(n):
    local i, r=0;
    para i = 1 a n
        r = r + i;
    retorna r;
```

Este algoritmo executa um simples somatório dos n primeiros números naturais. Sua operação mais significativa é o incremento da variável acumuladora r, que ocorre dentro do laço que é executado n vezes. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

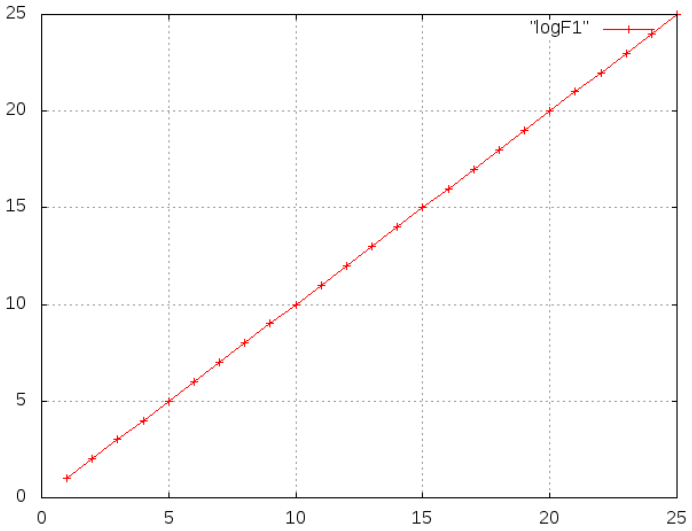

Figura 1: Relação entre n e o número de operações

Neste caso, é fácil identificar que a função é linear, levando-se em consideração que o gráfico toma a forma de uma reta. O número de operações é sempre igualmente proporcional a n, portanto o coeficiente angular é 1.

Algoritmo 2

```
int f2(n):
    local i, j, r=0;
    para i = 1 a n-1
        para j = i+1 a n
            r = r + 2;
    retorna r;
```

Este algoritmo calcula $n(n-1)$ através de sucessivas somas de 2. Isto é possível devido à propriedade matemática de paridade, na qual operações de multiplicação entre um número par e um número ímpar sempre resultam em números pares. Sua operação mais significativa, bem como no algoritmo 1, é o incremento da variável acumuladora r. Mas, desta vez, o algoritmo é composto por dois laços. A operação mais significativa ocorre dentro do laço mais interno. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

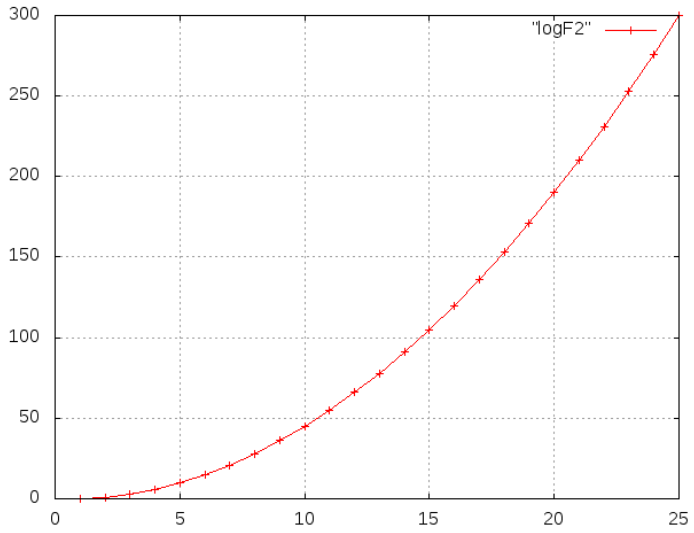

Figura 2: Relação entre n e o número de operações

Neste caso, não é possível identificar com precisão o comportamento da função em questão se somente observarmos o gráfico gerado. Portanto, torna-se necessário aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$ para verificar se estamos falando de uma função exponencial, conforme o gráfico a seguir:

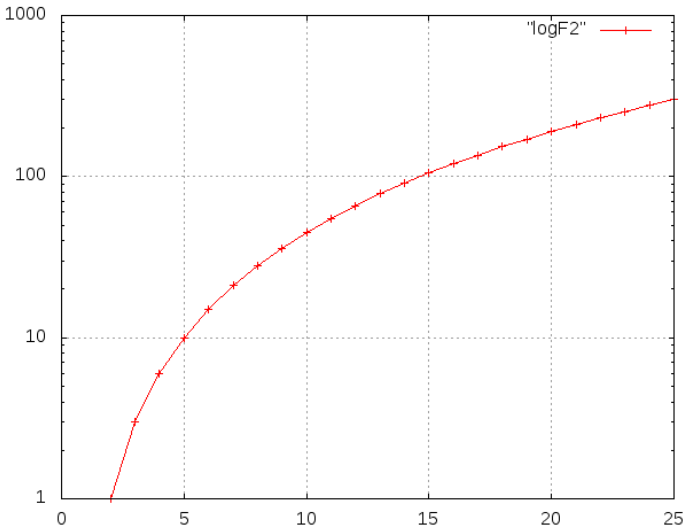

Figura 3: Relação entre n e o número de operações com escala logarítmica no eixo y

Como o gráfico resultante não toma a forma de uma reta, sabemos que a função não é exponencial. A próxima alternativa é verificar se a função é polinomial, aumentando a escala em ambos os eixos utilizando o logaritmo da função $f(n)$, conforme o seguinte gráfico:

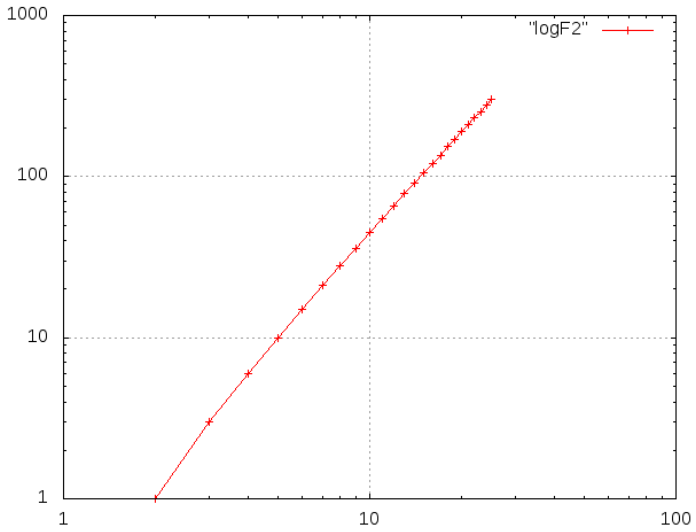

Figura 4: Relação entre n e o número de operações com escala logarítmica em ambos os eixos

O gráfico toma a forma de uma reta e, portanto, a função é polinomial. Agora, podemos descobrir a sua grandeza através do cálculo da inclinação da reta (r), conforme a "receita de bolo":

$$r \approx \frac{log(300) - log(1)}{log(25) - log(2)} = 2,2582...$$

Para obtermos um resultado mais preciso, vamos fazer o cálculo mais uma vez com valores diferentes:

$$r \approx \frac{log(190) - log(45)}{log(20) - log(10)} = 2,07...$$

É possível perceber que o resultado se aproxima cada vez mais de 2. Isso indica que a função característica do algoritmo se comporta de maneira semelhante a uma função polinomial de segundo grau $f(n) = n^{2,07}$.

Algoritmo 3

```
int f3(n):
    local i, j, k r=0;
    para i = 1 a n
        para j = i a 2i
            para k = i a j
                r = r + 1;
    retorna r;
```

Este algoritmo gera a sequência 3, 9, 19, 34, 55, 83, 119, 164,... que pode ser referenciada na Enciclopédia Online de Sequências de Inteiros como A062748. Sua operação mais significativa é o incremento da variável acumuladora r, que ocorre dentro do laço mais interno. A principal diferença deste algoritmo para o algoritmo 2 é que este possui três laços. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

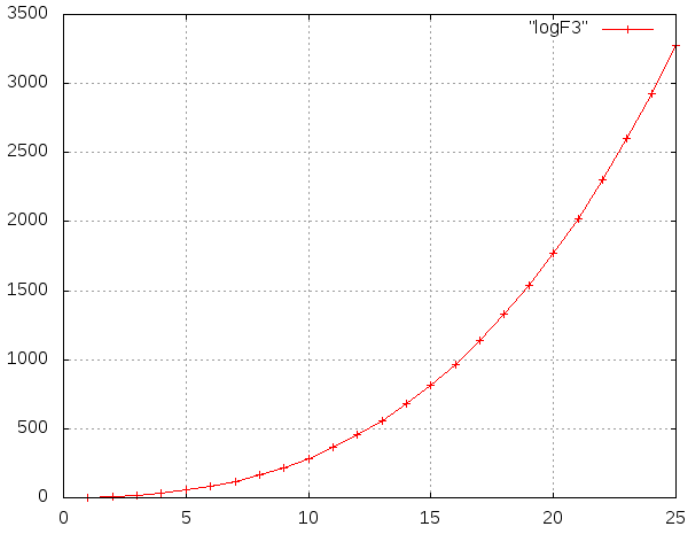

Figura 5: Relação entre n e o número de operações

Como no caso do algoritmo 2, não é possível identificar com precisão o comportamento da função em questão se somente observarmos o gráfico gerado. Portanto, torna-se necessário aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$ para verificar se estamos falando de uma função exponencial, conforme o gráfico a seguir:

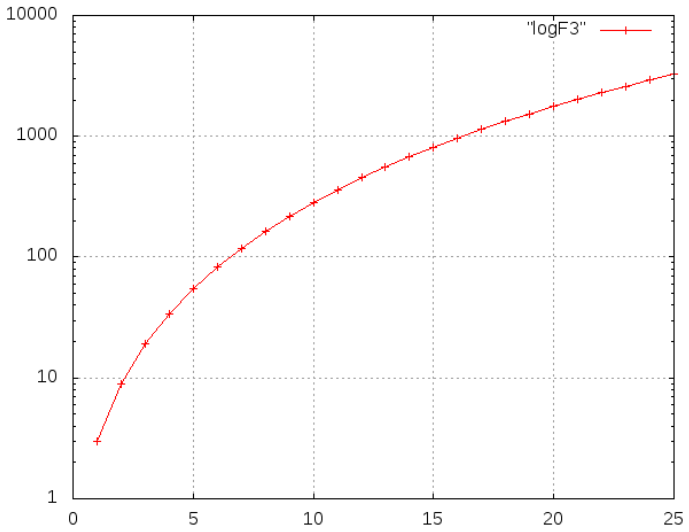

Figura 6: Relação entre n e o número de operações com escala logarítmica no eixo y

Também como no caso do algoritmo 2, o gráfico resultante não toma a forma de uma reta, sabemos que a função não é exponencial. A próxima alternativa é verificar se a função é polinomial, aumentando a escala em ambos os eixos utilizando o logaritmo da função $f(n)$, conforme o seguinte gráfico:

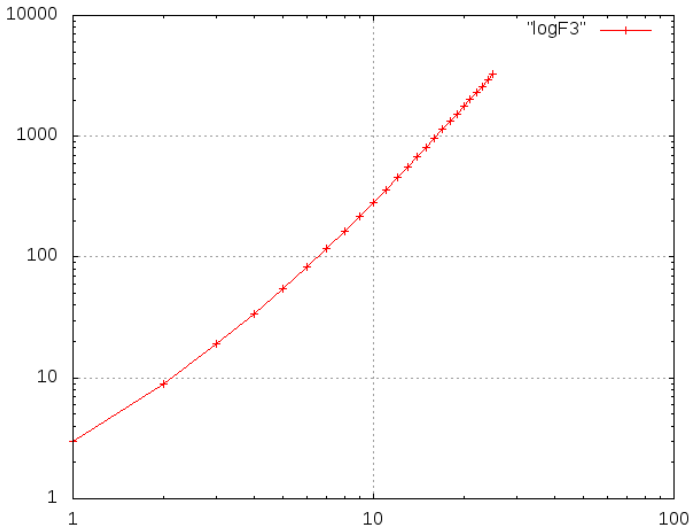

Figura 7: Relação entre n e o número de operações com escala logarítmica em ambos os eixos

O gráfico toma a forma de uma reta e, portanto, a função é polinomial. Agora, podemos descobrir a sua grandeza através do cálculo da inclinação da reta (r):

$$r \approx \frac{log(3275) - log(3)}{log(25) - log(1)} = 2,1732...$$

Para obtermos um resultado mais preciso, vamos fazer o cálculo mais uma vez com valores diferentes:

$$r \approx \frac{log(1770) - (log285)}{log(20) - log(10)} = 2,6347...$$

É possível perceber que o resultado se aproxima cada vez mais de 3. Isso indica que a função característica do algoritmo se comporta de maneira semelhante a uma função polinomial de terceiro grau $f(n) = n^{2,63}$.

Algoritmo 4

```
int f4(n):
    local i, j, k, r=0;
    para i = 1 a n
        para j = i a i+3
            para k = i a j
                r = r + 1;
    retorna r;
```

Este algoritmo calcula $10n$ através de sucessivas somas de 1 dentro de uma estrutura de três laços, semelhante à do algoritmo 3. Sua operação mais significativa é o incremento da variável acumuladora r. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

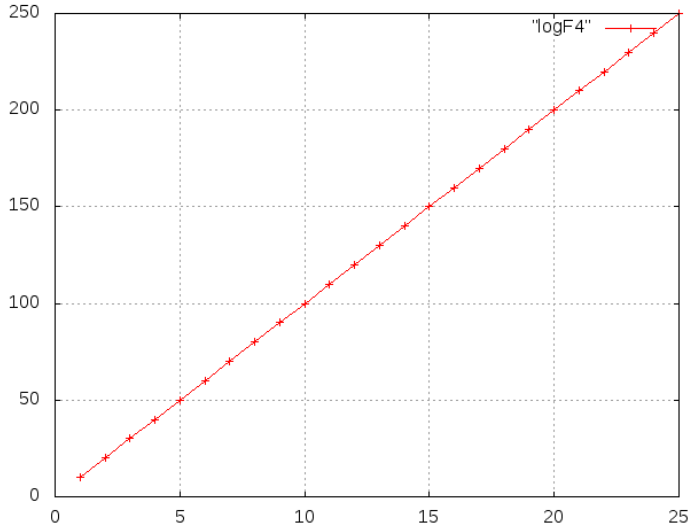

Figura 8: Relação entre n e o número de operações

O gráfico toma a forma de uma reta, portanto é uma função linear. Vamos descobrir a inclinação da reta (r) aplicando a fórmula do coeficiente angular:

$$r \approx \frac{250 - 10}{25 - 1} = 10$$

É possível afirmar que a função característica deste algoritmo é uma função polinomial de primeiro grau $f(n) = 10n$.

Algoritmo 5

```
int f5(n):
    se n = 0 então retorna 1;
    retorna f(n-1) + f(n-1);
```

Este algoritmo consiste em uma função recursiva que calcula 2^n. Justamente por ser recursiva, sua operação mais significativa é a sua própria chamada. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

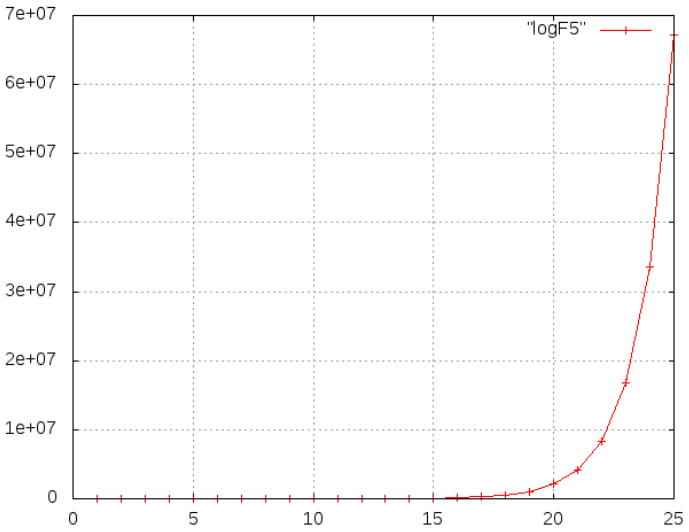

Figura 9: Relação entre n e o número de operações

Não é possível identificar com precisão o comportamento da função em questão se somente observarmos o gráfico gerado. Portanto, torna-se necessário aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$ para verificar se estamos falando de uma função exponencial, conforme o gráfico a seguir:

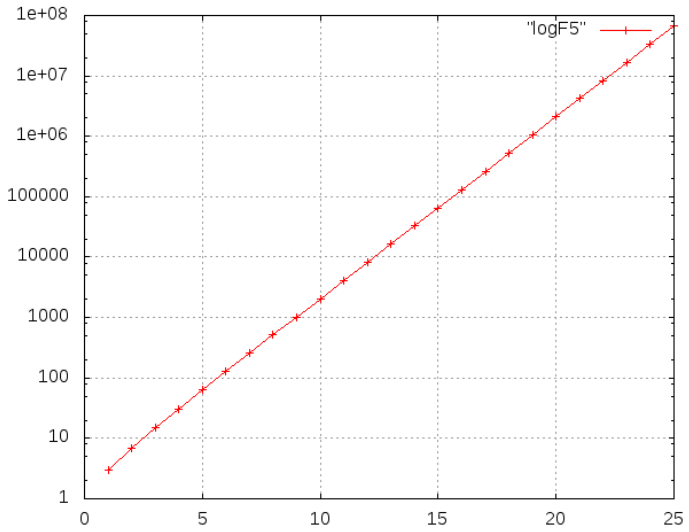

Figura 10: Relação entre n e o número de operações com escala logarítmica no eixo y

O gráfico toma a forma de uma reta e, portanto, a função é exponencial. Agora, precisamos descobrir o expoente. Para isso, precisamos calcular a inclinação da reta (r) e a base (b), por meio das fórmulas a seguir, respectivamente:

$$r \approx \frac{(log(67108863) - log(3)}{25 - 1} = 0,2939...$$

$$b = 10^{0,29} = 1,9674...$$

Ou seja, a função $f(n)$ cresce exponencialmente mesmo com base aproximada de 1,96.

Algoritmo 6

```
int f6(n):
    se n = 0 então retorna 1;
    retorna 2 * f(n-1);
```

Este algoritmo tem a mesma funcionalidade do algoritmo 5, ou seja, consiste em uma função recursiva que calcula 2^n. Contudo, é mais eficiente, pois executa um número muito menor de operações, ou seja, chamadas recursivas, conforme veremos a seguir. Sua operação mais significativa é a sua própria chamada. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

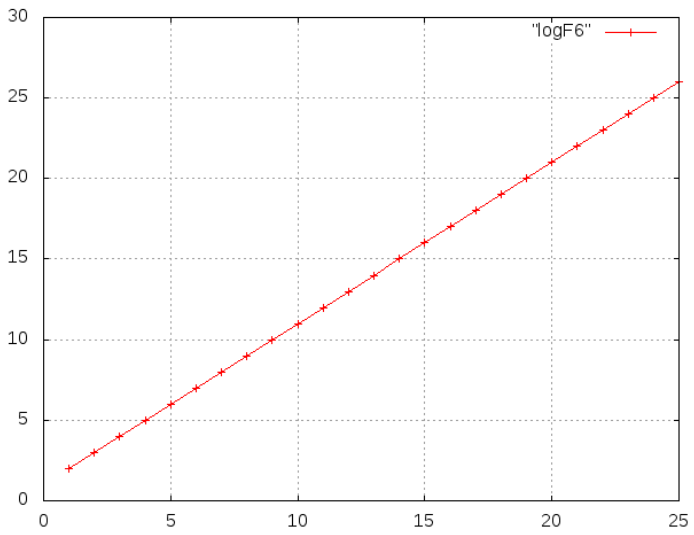

Figura 11: Relação entre n e o número de operações

O gráfico toma a forma de uma reta, portanto é uma função linear. Vamos descobrir a inclinação da reta (r) aplicando a fórmula do coeficiente angular:

$$r \approx \frac{26-2}{25-1} = 1$$

É possível afirmar que a função característica deste algoritmo é uma função polinomial de primeiro grau $f(n) = n + 1$.

Algoritmo 7

```
int f7(n):
    local i, b, r=0;
    para i = 1 a n
        b = i
        enquanto b > 0:
            r = r + 1;
            b = b / 2;
    retorna r;
```

Este algoritmo gera a sequência 0, 1, 3, 5, 8, 11, 14, 17,... que pode ser referenciada na Enciclopédia Online de Sequências de Inteiros como A001855, a qual representa o número máximo de comparações para ordenar n elementos por inserção binária. Sua operação mais significativa é o incremento da variável acumuladora r, que ocorre dentro do laço mais interno. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

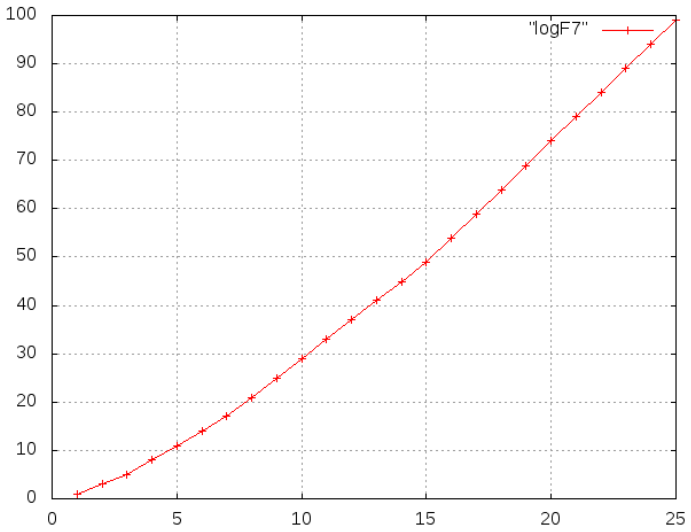

Figura 12: Relação entre n e o número de operações

Não é possível identificar com precisão o comportamento da função em questão se somente observarmos o gráfico gerado. Portanto, torna-se necessário aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$ para verificar se estamos falando de uma função exponencial, conforme o gráfico a seguir:

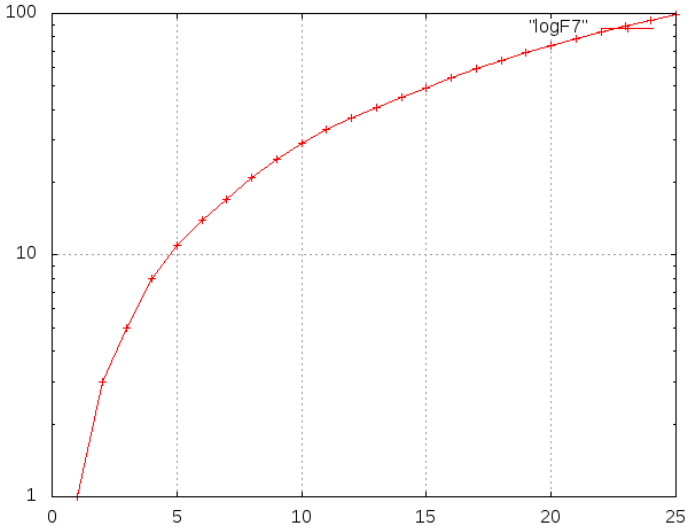

Figura 13: Relação entre n e o número de operações com escala logarítmica no eixo y

Como o gráfico resultante não toma a forma de uma reta, sabemos que a função não é exponencial. A próxima alternativa é verificar se a função é polinomial, aumentando a escala em ambos os eixos utilizando o logaritmo da função $f(n)$, conforme o seguinte gráfico:

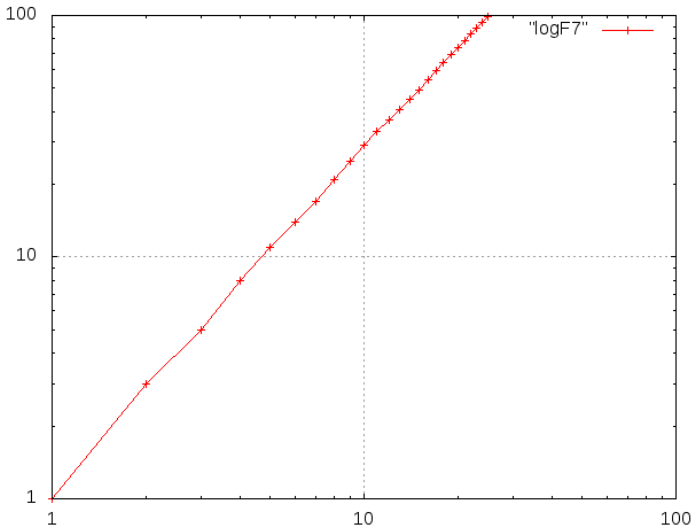

Figura 14: Relação entre n e o número de operações com escala logarítmica em ambos os eixos

O gráfico toma a forma de uma reta e, portanto, a função é polinomial. Agora, podemos descobrir a sua grandeza através do cálculo da inclinação da reta (r), conforme a "receita de bolo":

$$r \approx \frac{log(99) - log(1)}{log(25) - log(1)} = 1,4275...$$

Para obtermos um resultado mais preciso, vamos fazer o cálculo mais uma vez com valores diferentes:

$$r \approx \frac{log(74) - log(29)}{log(20) - log(10)} = 1,3514...$$

É possível perceber que o resultado se aproxima cada vez mais de 1. Isso indica que a função característica do algoritmo se comporta de maneira semelhante a uma função polinomial de primeiro grau $f(n) = n^{1.3}$.

Algoritmo 8

```
void f8(string s, Coleção C):
    se C está vazio:
        imprime s
        retorna
    Item a = C.retiraum();
    f8(s, C);
    f8(s + " " + string(a), C);
```

Este algoritmo é bastante interessante. Ele imprime todas as possíveis combinações entre os n primeiros números inteiros sem repetição. Como ele é recursivo, sua operação mais significativa é a sua própria chamada. A relação entre n e o número de operações consumidas pelo algoritmo pode ser visualizada no gráfico abaixo:

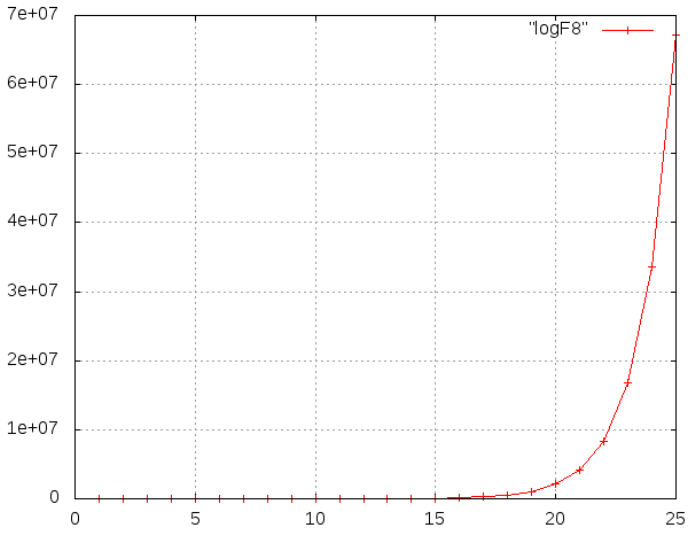

Figura 15: Relação entre n e o número de operações

Não é possível identificar com precisão o comportamento da função em questão se somente observarmos o gráfico gerado. Portanto, torna-se necessário aumentar a escala no eixo y utilizando o logaritmo da função $f(n)$ para verificar se estamos falando de uma função exponencial, conforme o gráfico a seguir:

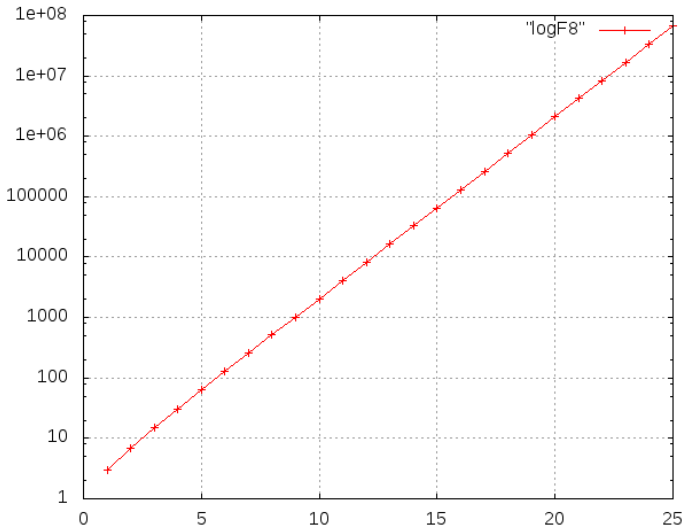

Figura 16: Relação entre n e o número de operações com escala logarítmica no eixo y

O gráfico toma a forma de uma reta e, portanto, a função é exponencial. Agora, precisamos descobrir o expoente. Para isso, precisamos calcular a inclinação da reta (r) e a base (b), por meio das fórmulas a seguir, respectivamente:

$$r \approx \frac{(log(67108863) - log(3)}{25 - 1 = 0,2939...}$$

$$b = 10^{0,29} = 1,9674...$$

Ou seja, a função $f(n)$ cresce exponencialmente mesmo com base aproximada de 1,96. Exatamente igual à função do algoritmo 5.

Considerações

Agora que temos em mãos as funções características de cada algoritmo podemos não somente compará-los, mas também estimar quanto tempo sua execução levaria para qualquer n e assim determinar a sua eficiência e viabilidade para um determinado caso.

Uma questão interessante a ser levantada é se o número de laços de um algoritmo dita o grau de uma função polinomial. Nos algoritmos 2 e 3 este padrão é seguido: o algoritmo 2 possui dois laços e sua função característica é uma polinomial de segundo grau e o algoritmo 3 possui três laços e sua função característica é uma polinomial de terceiro grau. Mas isso só ocorre se a variação

do laço envolver a variável n em questão.

Outro apontamento é que a simples modificação em um algoritmo pode alterar completamente a sua complexidade. Um exemplo disso são os algoritmos 5 e 6, os quais são recursivos. Eles fazem essencialmente a mesma coisa, mas o 6 é muito mais eficiente, pois evita o empilhamento de mais uma chamada igual, substituindo-a por uma simples multiplicação.

Podemos observar, também, que os algoritmos 5 e 8 possuem a mesma função característica. Ambos são recursivos e ambos empilham duas chamadas a cada execução do método (salvo o caso de saída). Isto comprova que a operação mais importante em métodos recursivos é, de fato, sua própria chamada.