Complexidade de Algoritmos

Prof. Andrey Masiero

1 de setembro de 2017

Agenda

Introdução

- Identificar se a complexidade do problema e algoritmo proposto, geram uma solução eficiente ou não;

Introdução

- Identificar se a complexidade do problema e algoritmo proposto, geram uma solução eficiente ou não;
- Técnicas de programação auxiliam durante a implementação dele (Estrutura de Dados, Métodos de Divisão de Problema, etc);

Introdução

- Identificar se a complexidade do problema e algoritmo proposto, geram uma solução eficiente ou não;
- Técnicas de programação auxiliam durante a implementação dele (Estrutura de Dados, Métodos de Divisão de Problema, etc);
- A análise é feita a partir de uma entrada n de elementos, como é o comportamento do algoritmo.

Introdução

Como é feita a análise?

- Remover elemento de um vetor com n elementos:
  No pior caso, leva $n - 1$ iterações;

Introdução

Como é feita a análise?

- Remover elemento de um vetor com n elementos:
  No pior caso, leva $n - 1$ iterações;
- Alterar elemento na posição i do vetor:
  Leva 1 iteração;

Introdução

Como é feita a análise?

- Remover elemento de um vetor com n elementos:
  No pior caso, leva $n - 1$ iterações;
- Alterar elemento na posição i do vetor:
  Leva 1 iteração;
- E assim por diante.

Exemplo 01

Determinar quantas vezes o laço executa:

```
cont = 0
i = 1
while(i <= 1000) {
    print i
    i = i + 1
    cont = cont + 1
}

print cont
```

Exemplo 02

E esse outro laço, quantas vezes executa:

```
// n é informado pelo usuário
n = ?
cont = 0
i = 1
while(i <= n) {
    print i
    i = i + 1
    cont = cont + 1
}

print cont
```

Analisando os exemplos 01 e 02

- Número de repetições do exemplo 1 = 1000;

Analisando os exemplos 01 e 02

- Número de repetições do exemplo 1 = 1000;
- Número de repetições do exemplo 2 = n;

Analisando os exemplos 01 e 02

- Número de repetições do exemplo 1 = 1000;
- Número de repetições do exemplo 2 = n;
- Então pode-se dizer que o número de repetições é de acordo com n;

Analisando os exemplos 01 e 02

- Número de repetições do exemplo 1 = 1000;
- Número de repetições do exemplo 2 = n;
- Então pode-se dizer que o número de repetições é de acordo com n;
- Sendo assim, minha função é $f(n) = n$.

Exemplo 03

```
cont = 0
i = 1
while(i <= 1000) {
    print i
    i = i * 2
    cont = cont + 1
}

print cont
```

Exemplo 04

```
// n é informado pelo usuário
n = ?
cont = 0
i = 1
while(i <= n) {
    print i
    i = i * 2
    cont = cont + 1
}

print cont
```

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;
- Sendo assim, o exemplo 3 tem 10 repetições;

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;
- Sendo assim, o exemplo 3 tem 10 repetições;
- Número de repetições do exemplo 4 = ?;

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;
- Sendo assim, o exemplo 3 tem 10 repetições;
- Número de repetições do exemplo 4 $=$?;
- Na matemática, para esse caso, o n é encontrado por uma função $\log_2$;

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;
- Sendo assim, o exemplo 3 tem 10 repetições;
- Número de repetições do exemplo 4 = ?;
- Na matemática, para esse caso, o n é encontrado por uma função $\log_2$;
- Então, o número de repetições do exemplo 4 = $\log_2 n$;

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;
- Sendo assim, o exemplo 3 tem 10 repetições;
- Número de repetições do exemplo 4 = ?;
- Na matemática, para esse caso, o n é encontrado por uma função $\log_2$;
- Então, o número de repetições do exemplo 4 = $\log_2 n$;
- Pode-se dizer que o número de repetições é de acordo com $\log_2 n$;

Analisando os exemplos 03 e 04

- Se observarmos a operação $i \times 2$, é a mesma coisa que 2^i;
- Sendo assim, o exemplo 3 tem 10 repetições;
- Número de repetições do exemplo 4 = ?;
- Na matemática, para esse caso, o n é encontrado por uma função $\log_2$;
- Então, o número de repetições do exemplo 4 = $\log_2 n$;
- Pode-se dizer que o número de repetições é de acordo com $\log_2 n$;
- Minha função resulta em $f(n) = \log_2 n$.

Exemplo 05

```
// n é informado pelo usuário
n = ?
cont = 0
for(i = 1; i <= n; i + 1) {
    for( j = 1; j <= n; j * 2) {
        cont = cont + 1
    }
}
print cont
```

Analisando os exemplos 05

- Analisando o primeiro `for` ele tem o número de repetições $= n$;

Analisando os exemplos 05

- Analisando o primeiro `for` ele tem o número de repetições $= n$;
- O segundo `for` ele tem o número de repetições $= \log_2 n$;

Analisando os exemplos 05

- Analisando o primeiro `for` ele tem o número de repetições $= n$;
- O segundo `for` ele tem o número de repetições $= \log_2 n$;
- Combinando os dois temos $n \times \log_2 n$;

Analisando os exemplos 05

- Analisando o primeiro `for` ele tem o número de repetições $= n$;
- O segundo `for` ele tem o número de repetições $= \log_2 n$;
- Combinando os dois temos $n \times \log_2 n$;
- Minha função fica: $f(n) = n \log_2 n$.

Exemplo 06

```
// n é informado pelo usuário
n = ?
cont = 0
for(i = 1; i <= n; i + 1) {
    for( j = 1; j <= n; j + 1) {
        cont = cont + 1
    }
}
print cont
```

Analisando os exemplos 06

- Analisando o primeiro `for` ele tem o número de repetições $= n$;

Analisando os exemplos 06

- Analisando o primeiro `for` ele tem o número de repetições $= n$;
- O segundo `for` ele tem o número de repetições $= n$;

Analisando os exemplos 06

- Analisando o primeiro `for` ele tem o número de repetições $= n$;
- O segundo `for` ele tem o número de repetições $= n$;
- Combinando os dois temos $n \times n$;

Analisando os exemplos 06

- Analisando o primeiro `for` ele tem o número de repetições $= n$;
- O segundo `for` ele tem o número de repetições $= n$;
- Combinando os dois temos $n \times n$;
- Minha função fica: $f(n) = n^2$.

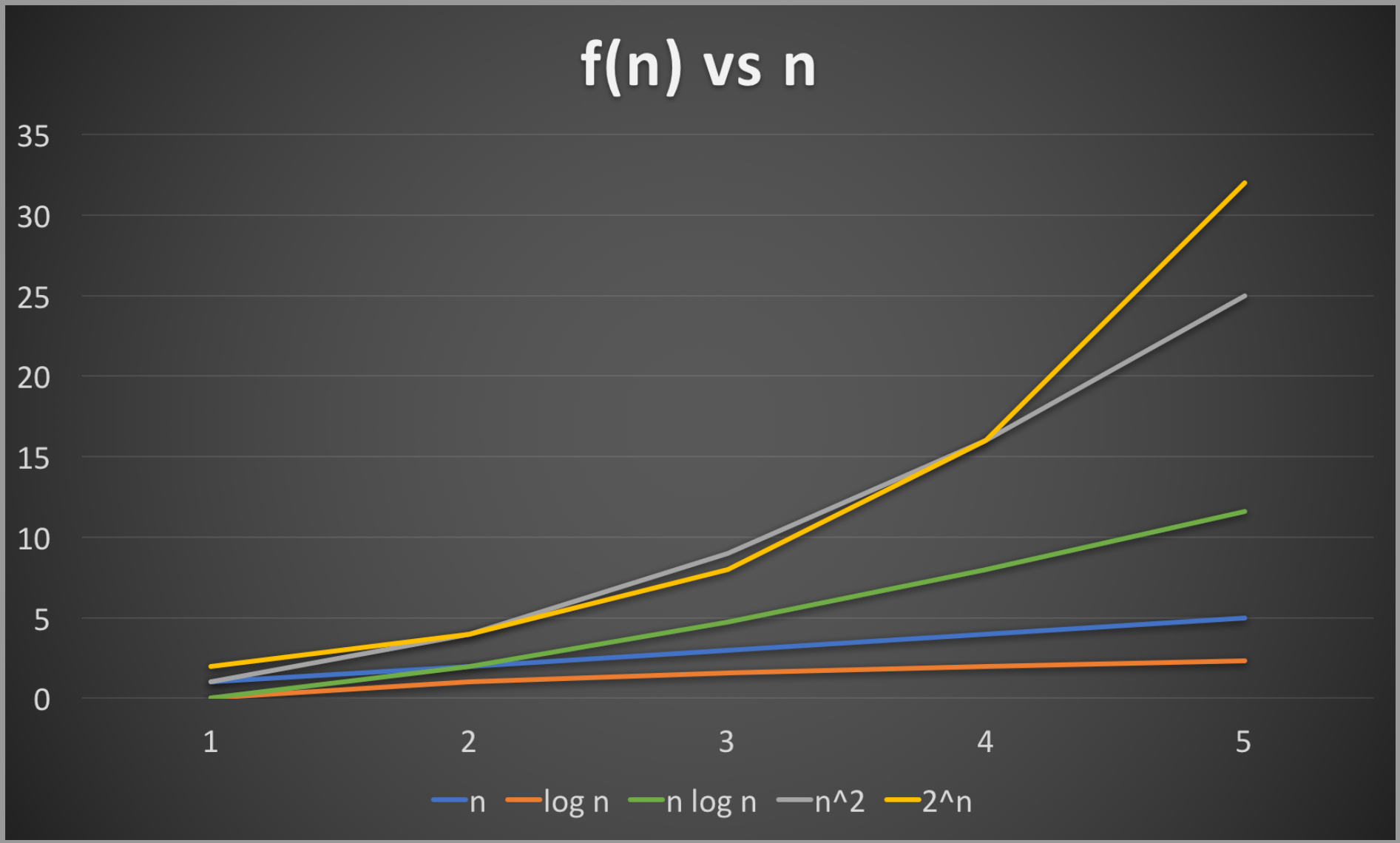
Notação Big O

- Em complexidade de algoritmos utilizamos a notação $O(f(n))$ para dizer qual é o tempo de execução do algoritmo, no pior caso;
- Então se minha função é $f(n) = n^2$;
- Na notação Big O é: $O(n^2)$.

Categoria de Grandezas

- Logarítmica: $\log_2 n$;
- Linear: n;
- Logarítmica Linear: $n \log_2 n$;
- Quadrática: n^2;
- Polinomial: n^k;
- Exponencial: 2^n;
- Fatorial: $n!$.

Categoria de Grandezas



Exercícios

- Analise os códigos a seguir e encontre sua complexidade.
- Diga a complexidade na notação Big O.

Exercício 01

```
// n é informado pelo usuário
i = 1
j = 1

while(i <= n) {
    i = i * 2
}

while(j <= n) {
    j = j + 1
}
```

Exercício 02

```
// n é informado pelo usuário
cont = 0
for(i = 1; i <= n; i + 1) {
    for( j = 1; j <= n; j + 1) {
        cont = cont + 1
    }
}
k = 1
while(k <= n) {
    k = k + 1
}
```

Exercício 03

```
// n é informado pelo usuário
cont = 0
for(i = 1; i <= n; i + 1) {
    for( j = 1; j <= n; j + 1) {
        for( k = 1; k <= n; k + 1) {
            cont = cont + 1
        }
    }
}
```

Referências Bibliográficas

1. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to algorithms second edition." (2001).
2. Tamassia, Roberto, and Michael T. Goodrich. "Estrutura de Dados e Algoritmos em Java." Porto Alegre, Ed. Bookman 4 (2007).
3. Ascencio, Ana Fernanda Gomes, and Graziela Santos de Araújo. "Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++." São Paulo: Perarson Prentice Halt 3 (2010).