INSTITUTO FEDERAL
Sudeste de Minas Gerais
Campus Manhuaçu

Estruturas de Dados I
Análise de Algoritmos

Prof. Leonardo C. R. Soares - *leonardo.soares@ifsudestemg.edu.br*

Instituto Federal do Sudeste de Minas Gerais

23 de agosto de 2023

Análise de algoritmos

Definição

Analisar um algoritmo é entender a quantidade de recursos que ele consome durante sua execução. Os recursos mais importantes referem-se a **tempo** e **espaço**.

Análise de algoritmos

Definição

Analisar um algoritmo é entender a quantidade de recursos que ele consome durante sua execução. Os recursos mais importantes referem-se a **tempo** e **espaço**.

Tempo de execução

É dado pelo número de instruções primitivas executadas pelo algoritmo.

Análise de algoritmos

Definição

Analisar um algoritmo é entender a quantidade de recursos que ele consome durante sua execução. Os recursos mais importantes referem-se a **tempo** e **espaço**.

Tempo de execução

É dado pelo número de instruções primitivas executadas pelo algoritmo.

Espaço

O espaço necessário para armazenar dados durante a execução do código do algoritmo.

Análise de algoritmos

Análise empírica

Uma forma de analisar algoritmos é configurar um ambiente em que as variáveis são controladas e executá-los com o objetivo de analisar o tempo de execução (neste caso, tempo relógio) e a qualidade das soluções

Análise de algoritmos

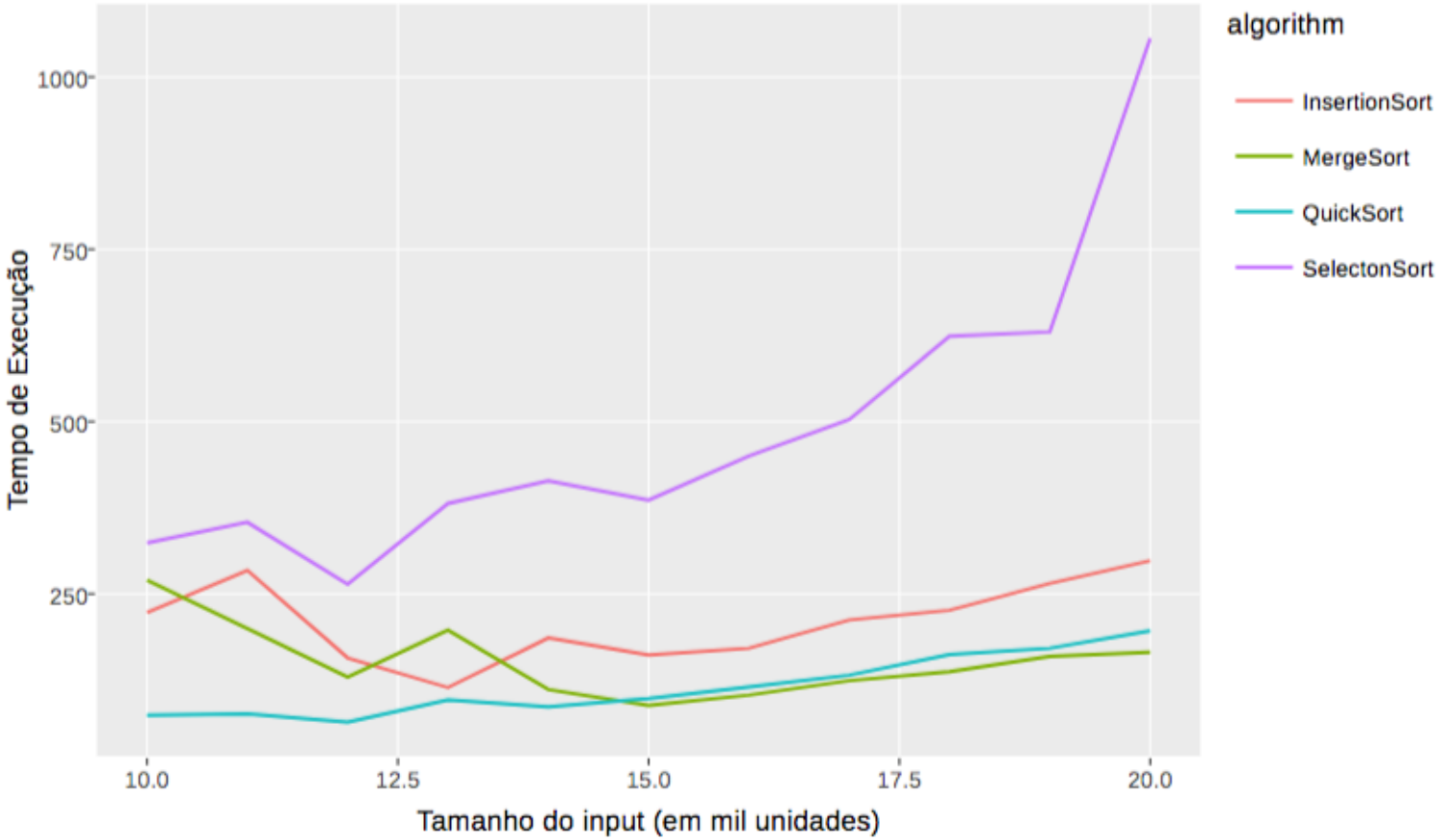


Figura: Comparação entre algoritmos de ordenação. Retirado de https://joaoarthurbm.github.io/eda/posts/introducao-a-analise/comparacao-ordenacao.jpeg

Análise de algoritmos

Embora seja uma alternativa válida, ela exige que todos os algoritmos sejam implementados antes que a análise possa ser conduzida, além disso, os resultados são diretamente dependente do *hardware* utilizado.

Análise de algoritmos

Dada as limitações da análise empírica, limitaremos nossas análise de tempo a contagem de operações primitivas.

Para isso, vamos considerar que toda operação primitiva (avaliação de expressões, atribuições, operações matemáticas, retorno de dados, leitura de dados) possuem tempo constante. Referenciado como $\mathcal{O}(1)$.

Contando operações primitivas

Identifique as operações primitivas realizadas pelo algoritmo abaixo:

```
1  int multiplicaRestoPorParteInteira(int i, int j)
       {
2      int resto = i % j;
3      int pInteira = i / j;
4      int resultado = resto * pInteira;
5      return resultado;
6  }
```

Contando operações primitivas

Temos as seguintes operações primitivas:

1. Atribuição da variável resto (resto =) -> c_1
2. Operação aritmética $i\%j$ -> c_2
3. Atribuição da variável pInteira (pInteira =) -> c_3
4. Operação aritmética i/j -> c_4
5. Atribuição da variável resultado (resultado=) -> c_5
6. Operação aritmética $(resto * pInteira)$ -> c_6
7. Retorno do método (return resultado) -> c_7

Contando operações primitivas

Temos as seguintes operações primitivas:

1. Atribuição da variável resto (resto =) -> c_1
2. Operação aritmética $i\%j$ -> c_2
3. Atribuição da variável pInteira (pInteira =) -> c_3
4. Operação aritmética i/j -> c_4
5. Atribuição da variável resultado (resultado=) -> c_5
6. Operação aritmética $(resto * pInteira)$ -> c_6
7. Retorno do método (return resultado) -> c_7

Quantas vezes cada operação primitiva é executada?

Contando operações primitivas

Temos as seguintes operações primitivas:

1. Atribuição da variável resto (resto =) -> c_1
2. Operação aritmética $i\%j$ -> c_2
3. Atribuição da variável pInteira (pInteira =) -> c_3
4. Operação aritmética i/j -> c_4
5. Atribuição da variável resultado (resultado=) -> c_5
6. Operação aritmética $(resto * pInteira)$ -> c_6
7. Retorno do método (return resultado) -> c_7

Quantas vezes cada operação primitiva é executada?
Para este algoritmo, cada instrução primitiva é executada apenas uma vez.

Custo total

O custo total do tempo de execução do algoritmo é a soma das execuções primitivas executadas. Neste caso, temos que a função que descreve o tempo de execução deste algoritmo é:

$$f(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7$$

Como consideramos que todas as operações primitivas consomem o mesmo tempo constante para ser executado, podemos simplificar a função acima para

$$f(n) = 7c$$

Custo total

O custo total do tempo de execução do algoritmo é a soma das execuções primitivas executadas. Neste caso, temos que a função que descreve o tempo de execução deste algoritmo é:

$$f(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7$$

Como consideramos que todas as operações primitivas consomem o mesmo tempo constante para ser executado, podemos simplificar a função acima para

$$f(n) = 7c$$

Estamos interessados em uma função que nos diga o tempo de execução em relação ao tamanho da entrada, representado por n. Como podemos ver na função acima, o custo dela não depende de n, por isso dizemos que esta função tem **custo constante**.

Contando operações primitivas

Identifique as operações primitivas realizadas pelo algoritmo abaixo:

```
1  double precisaNaFinal(double nota1, double nota2
       , double nota3) {
2      double media = (nota1 + nota2 + nota3) / 3;
3      if (media >= 7 || media < 4) {
4          return 0;
5      } else {
6          double mediaFinal = 5;
7          double pesoFinal = 0.4;
8          double pesoMedia = 0.6;
9          double precisa = (mediaFinal - pesoMedia
               * media) / pesoFinal;
10         return precisa;
11     }
12 }
```

Contando operações primitivas

Se contarmos todas as operações, temos:

$$f(n) = 13c$$

Mas o algoritmo possui uma condicional que altera significativamente (para o exemplo) o número de operações executadas. A análise precisa ser feita considerando as diferentes perspectivas existentes.

Perspectivas

Definição

Além do ambiente computacional, o comportamento de um algoritmo pode variar de acordo com o comportamento da entrada (tamanho, estrutura, etc.), o que gera diferentes perspectivas.

Perspectivas

Melhor Caso

A entrada está organizada de maneira que o algoritmo levará o tempo mínimo para resolver o problema.

Perspectivas

Melhor Caso

A entrada está organizada de maneira que o algoritmo levará o tempo mínimo para resolver o problema.

Pior Caso

A entrada está organizada de maneira que o algoritmo levará o tempo máximo para resolver o problema.

Perspectivas

Melhor Caso

A entrada está organizada de maneira que o algoritmo levará o tempo mínimo para resolver o problema.

Pior Caso

A entrada está organizada de maneira que o algoritmo levará o tempo máximo para resolver o problema.

Caso Médio

A entrada está organizada de maneira que o algoritmo levará um tempo médio para resolver o problema

Análises e perspectivas

As análises se concentram, geralmente, no pior caso e no caso médio.

- O pior caso nos dá uma ideia de quão ruim pode ser o comportamento do algoritmo. Em geral, o cálculo do pior caso é simples.
- O caso médio apresenta uma ideia geral de como o algoritmo se comportará para a maioria dos casos. Determinar o comportamento médio pode ser mais complexo.

Contando operações primitivas

Identifique as operações primitivas do algoritmo abaixo:

```
1  public static boolean contains(int[] v, int x) {
2      for (int i = 0; i < v.length; i++)
3          if (v[i] == x)
4              return true;
5      return false;
6  }
```

Contando operações primitivas

Consideremos como primitivas:

1. Atribuição (int i = 0) -> c_1
2. Avaliação de expressão booleana (i < v.length) -> c_2
3. Operação aritmética (i++) -> c_3
4. Avaliação de expressão booleana (v[i] == x) -> c_4
5. Retorno de método (return true) -> c_5
6. Retorno de método (return false) -> c_6

Contando operações primitivas

Consideremos como primitivas:

1. Atribuição (int i = 0) -> c_1
2. Avaliação de expressão booleana (i < v.length) -> c_2
3. Operação aritmética (i++) -> c_3
4. Avaliação de expressão booleana (v[i] == x) -> c_4
5. Retorno de método (return true) -> c_5
6. Retorno de método (return false) -> c_6

Qual o melhor caso para este algoritmo? Neste caso, quantas vezes cada primitiva será executada?

Contando operações primitivas

Consideremos como primitivas:

1. Atribuição (int i = 0) -> c_1
2. Avaliação de expressão booleana (i < v.length) -> c_2
3. Operação aritmética (i++) -> c_3
4. Avaliação de expressão booleana (v[i] == x) -> c_4
5. Retorno de método (return true) -> c_5
6. Retorno de método (return false) -> c_6

Qual o melhor caso para este algoritmo? Neste caso, quantas vezes cada primitiva será executada?

O melhor caso é o valor de x ser igual ao primeiro elemento do vetor. Neste caso, as cinco primeiras primitivas serão executas, uma vez.

Contando operações primitivas

Consideremos como primitivas:

1. Atribuição (int i = 0) -> c_1
2. Avaliação de expressão booleana (i < v.length) -> c_2
3. Operação aritmética (i++) -> c_3
4. Avaliação de expressão booleana (v[i] == x) -> c_4
5. Retorno de método (return true) -> c_5
6. Retorno de método (return false) -> c_6

Qual o melhor caso para este algoritmo? Neste caso, quantas vezes cada primitiva será executada?

O melhor caso é o valor de x ser igual ao primeiro elemento do vetor. Neste caso, as cinco primeiras primitivas serão executas, uma vez.

E o pior caso?

Contando operações primitivas

O pior caso para o algoritmo anterior ocorre quando o valor de x não está no vetor. Neste caso:

1. c_1 é executada apenas uma vez.
2. c_2 é executada (n+1) vezes. Exemplo: se n=5, temos as seguintes verificações: $0 < 5$, $1 < 5$; $2 < 5$, $3 < 5$, $4 < 5$ e $5 < 5$, quando encerra-se o loop. Ou seja, 6 verificações.
3. c_3 é executada n vezes. Exemplo: se n=5, temos os seguintes incrementos em i: 1, 2, 3, 4 e 5, quando encerra-se o loop.
4. c_4 é executada n vezes.
5. c_5 não é executada.
6. c_6 é executada apenas uma vez.

Custo total

A função que descreve o tempo de execução deste algoritmo, para o pior caso é:

$$f(n) = c_1 + c_2 * (n + 1) + c_3 * n + c_4 * n + c_6$$

ou

$$f(n) = 3 * c * n + 3 * c$$

ou ainda, considerando que o tempo de cada operação constante é igual a 1:

$$f(n) = 3 * n + 3$$

Esta função está diretamente ligada ao tamanho de n. De fato, o tempo de execução do pior caso cresce linearmente ao tamanho de n.

Análise assintótica

Definição

A análise assintótica é feita observando-se a ordem de crescimento do tempo de execução. O interesse da análise assintótica repousa na taxa de crescimento quando o tamanho da entrada for grande o suficiente para que constantes possam ser ignoradas.

Análise assintótica

Definição

A análise assintótica é feita observando-se a ordem de crescimento do tempo de execução. O interesse da análise assintótica repousa na taxa de crescimento quando o tamanho da entrada for grande o suficiente para que constantes possam ser ignoradas.

Assintoticamente, a função dedo algoritmo anterior pertence à classe de funções lineares, ou $\mathcal{O}(n)$.

$$f(n) = 3 * n + 3$$
$$f(n) = \cancel{3} * n + \cancel{3}$$
$$f(n) = n$$
$$f(n) \in \mathcal{O}(n)$$

Funções tipicamente utilizadas

- k: constante em k, geralmente denotado como 1
  - Independe do tamanho de n.
- $log\ n$: logarítmico (geralmente a base é 2)
  - Menor do que uma constante grande;
  - n=1000, $log\ n$=10;
  - n=1.000.000, $log\ n$=20.
- n: linear
  - Aumenta no mesmo ritmo que n.
- $n\ log\ n$: linear logarítmico, ou apenas $n\ log\ n$
  - n=1000, $n\ log\ n$=10.000;
  - n=1.000.000, $n\ log\ n$=20.000.000.

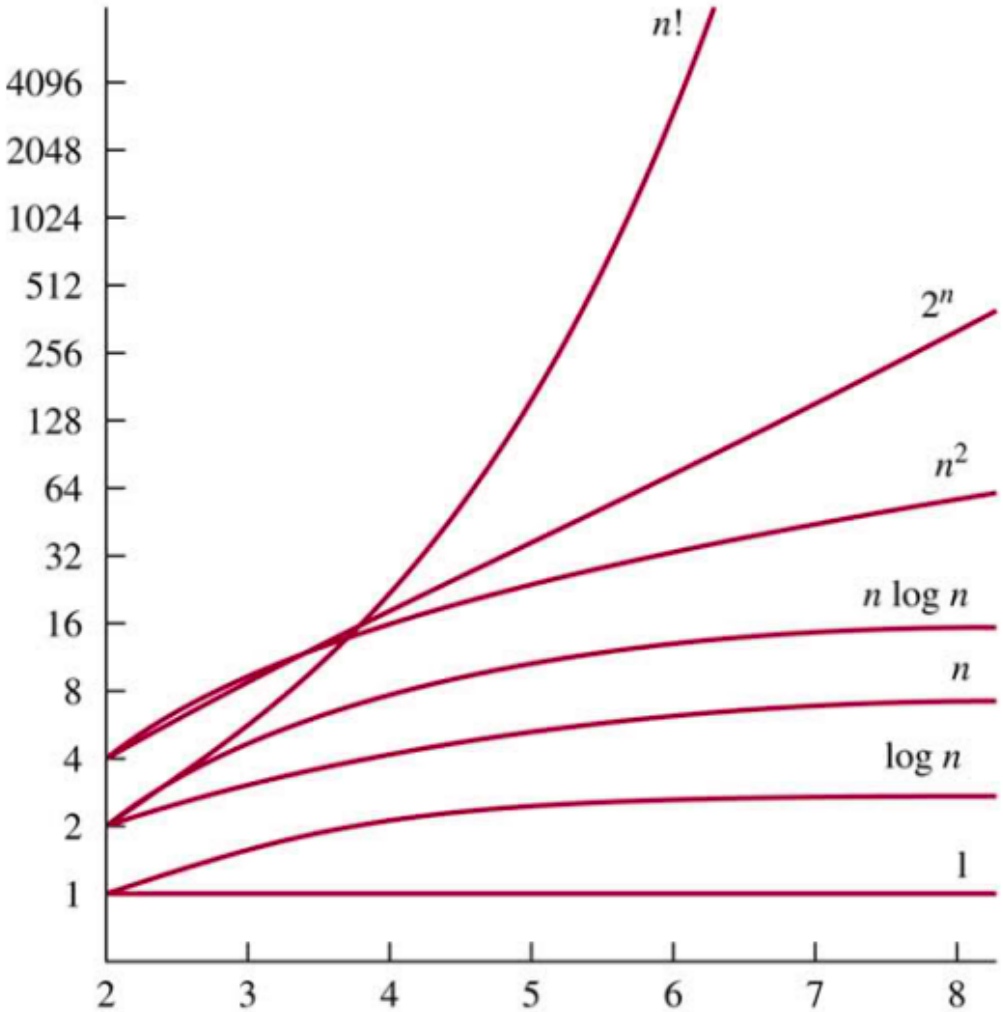
Figura: Funções tipicamente utilizadas. Retirado das notas de aulas do professor Marco A M Carvalho, UFOP.

Funções tipicamente utilizadas

- n^2: quadrático
  - n=1000, n^2=1.000.000.
- n^k: polinomial, proporcional à n elevado a k (constante);
- 2^n: exponencial, proporcional à 2 elevado a n
  - n=20, 2^n >1.000.000.
  - Quando n dobra, 2^n se eleva ao quadrado.
- $n!$: fatorial
  - n=10, $n!$= 3.628.800.
- n^n: sem nome formal, apenas n^n.

Figura: Funções tipicamente utilizadas. Retirado das notas de aulas do professor Marco A M Carvalho, UFOP.

Comparação da taxa de crescimento de funções



Taxa de crescimento de funções

Não afetam a taxa de crescimento

- Fatores constantes.
- Fatores de ordem mais baixa.

Exemplos

- $10^2n + 10^5$ é uma função linear;
- $10^5n^2 + 10^8n$ é uma função quadrática;
- $10^{-9}n^3 + 10^{20}n^2$ é uma função cúbica.

Análise assintótica

Descrição

Descreve como o tempo de execução cresce à medida que a entrada aumenta indefinidamente, o comportamento de **limite**.

É uma análise teórica, independente de *hardware* e que utiliza funções cujos domínios são o conjunto dos números naturais.

Os termos de mais baixa ordem e constantes são ignorados.

Utiliza três notações: $\mathcal{O}$, Θ e Ω.

Assumiremos que todas as funções apresentadas aqui são assintoticamente positivas, ou seja, são positivas para todo n suficientemente grande.

Notação Θ

Definição

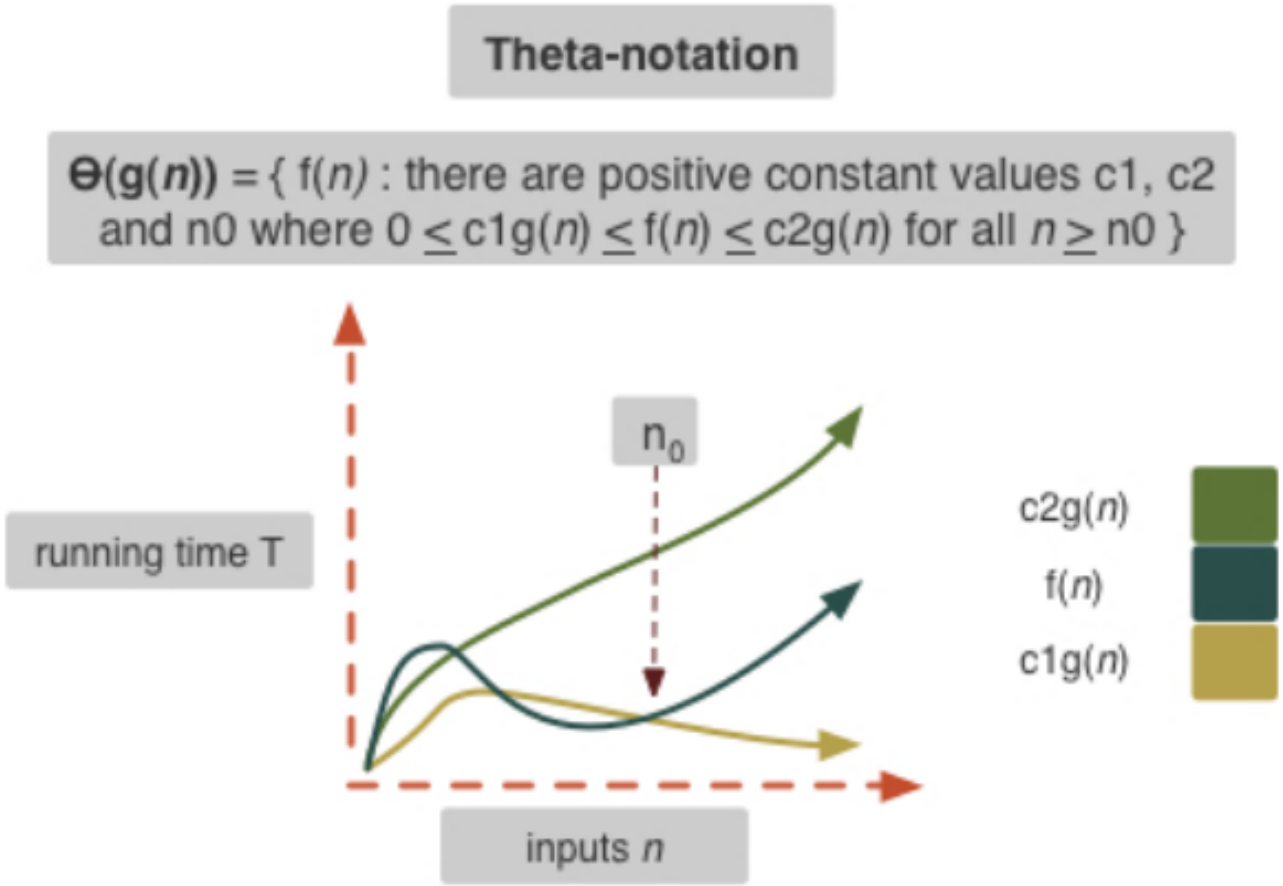
Formalmente, para uma função $g(n)$ denotamos por $\Theta(g(n))$ o conjunto de funções $\Theta(g(n)) = \{f(n)$: existam constantes positivas, c_1,c_2 e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0\}$.

Significado

Uma função $f(n)$ pertence a $g(n)$ caso existam constantes c_1 e c_2 tal que ela seja "imprensada"entre $c_1 g(n)$ e $c_2 g(n)$.

Poderíamos escrever $f(n) \in \Theta(g(n))$, porém, em um abuso de notação escrevemos $f(n) = \Theta(g(n))$

Notação Θ



Notação Θ

Exemplo

Vamos provar que $f(n) = 3n + 3 = \Theta(n)$.

Notação Θ

Exemplo

Vamos provar que $f(n) = 3n + 3 = \Theta(n)$.

Precisamos encontrar valores para c_1 e c_2 tais que a inequação $0 \leq c_1 * n \leq 3n + 3 \leq c_2 * n, \forall n \geq n_0$.

Notação Θ

Exemplo

Vamos provar que $f(n) = 3n + 3 = \Theta(n)$.

Precisamos encontrar valores para c_1 e c_2 tais que a inequação $0 \leq c_1 * n \leq 3n + 3 \leq c_2 * n, \forall n \geq n_0$.

Vamos testar com $c_1 = 1$ e $c_2 = 6$. Temos:

Notação Θ

Exemplo

Vamos provar que $f(n) = 3n + 3 = \Theta(n)$.

Precisamos encontrar valores para c_1 e c_2 tais que a inequação $0 \leq c_1 * n \leq 3n + 3 \leq c_2 * n, \forall n \geq n_0$.

Vamos testar com $c_1 = 1$ e $c_2 = 6$. Temos:

$0 \leq n \leq 3n + 3 \leq 6n, \forall n_0 \geq 0$.

Notação Θ

Exemplo

Vamos provar que $f(n) = 3n + 3 = \Theta(n)$.

Precisamos encontrar valores para c_1 e c_2 tais que a inequação $0 \leq c_1 * n \leq 3n + 3 \leq c_2 * n, \forall n \geq n_0$.

Vamos testar com $c_1 = 1$ e $c_2 = 6$. Temos:

$0 \leq n \leq 3n + 3 \leq 6n, \forall n_0 \geq 0$.

Se testarmos com $n = 1$, vemos que a inequação é verdadeira:

$0 \leq 1 \leq 6 \leq 6$.

Notação Θ

Exemplo

Vamos provar que $f(n) = 3n + 3 = \Theta(n)$.
Precisamos encontrar valores para c_1 e c_2 tais que a inequação $0 \leq c_1 * n \leq 3n + 3 \leq c_2 * n, \forall n \geq n_0$.
Vamos testar com $c_1 = 1$ e $c_2 = 6$. Temos:
$0 \leq n \leq 3n + 3 \leq 6n, \forall n_0 \geq 0$.
Se testarmos com $n = 1$, vemos que a inequação é verdadeira:
$0 \leq 1 \leq 6 \leq 6$.

Não é difícil notar que $\forall n > 1$ ela sempre será verdadeira. Assim, demonstramos que $f(n) \in \Theta(n)$, pois $g(n) = n$ limita inferior e superiormente $f(n)$. A taxa de crescimento é igual.

Notação $\mathcal{O}$

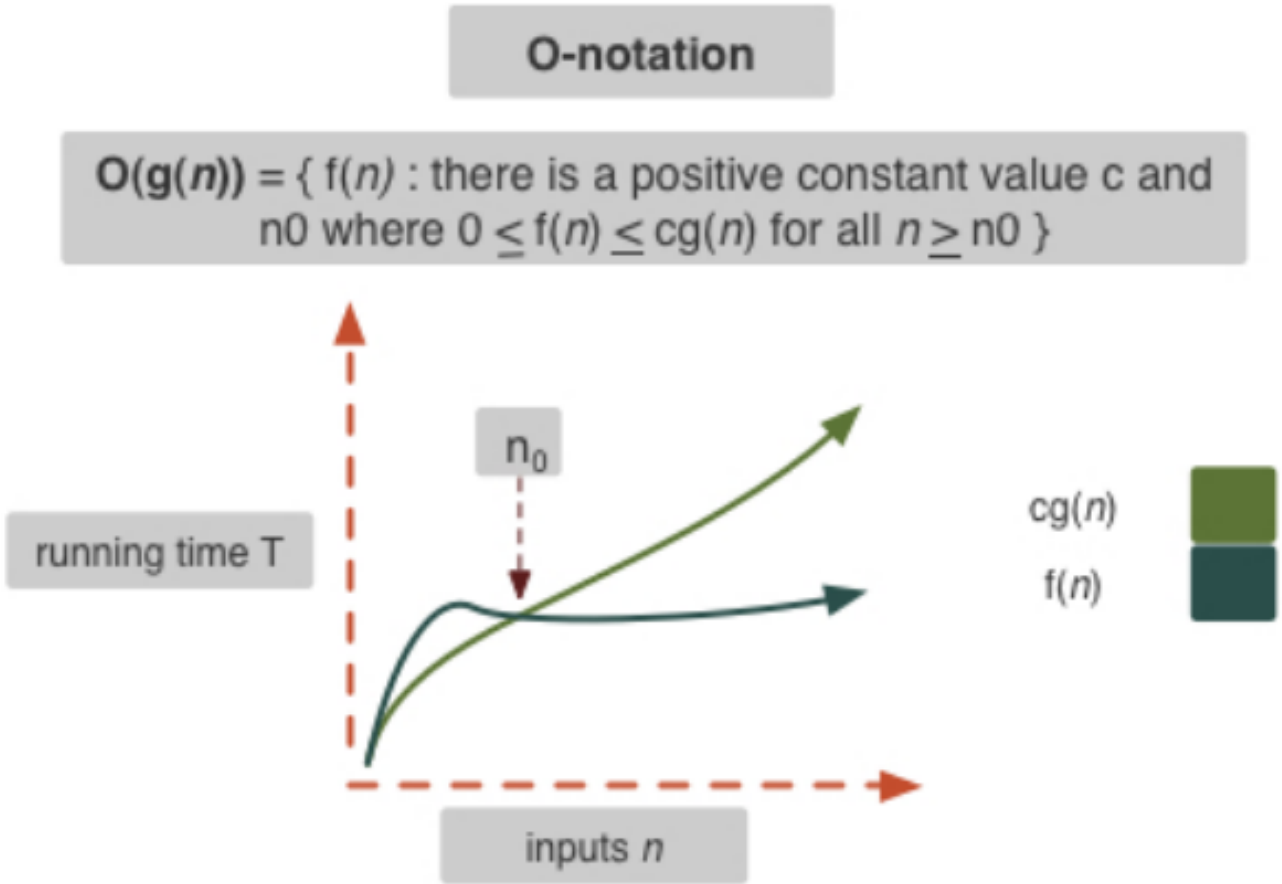
Definição

Formalmente, para uma função $g(n)$ denotamos por $\mathcal{O}(g(n))$ o conjunto de funções $\mathcal{O}(g(n)) = \{f(n)$: existam constantes positivas, c_1 e n_0 tais que $0 \leq f(n) \leq c_1 g(n) \forall n \geq n_0\}$.

Significado

Uma função $f(n)$ pertence a $g(n)$ caso exista uma constante c_1 tal que ela seja limitada superiormente por $c_1 g(n)$.

Notação $\mathcal{O}$



Notação $\mathcal{O}$

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \mathcal{O}(n^2)$.

Notação $\mathcal{O}$

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \mathcal{O}(n^2)$.
$0 \leq n^2 + 1 \leq c_1 * n^2, \forall n \geq n_0$.

Notação $\mathcal{O}$

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \mathcal{O}(n^2)$.
$0 \leq n^2 + 1 \leq c_1 * n^2, \forall n \geq n_0$.
Considere $c_1 = 2$ e $n_0 = 1$, temos:

Notação $\mathcal{O}$

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \mathcal{O}(n^2)$.

$0 \leq n^2 + 1 \leq c_1 * n^2, \forall n \geq n_0$.

Considere $c_1 = 2$ e $n_0 = 1$, temos:

$0 \leq n^2 + 1 \leq 2 * n^2$

$0 \leq 2 \leq 2$.

Notação $\mathcal{O}$

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \mathcal{O}(n^2)$.
$0 \leq n^2 + 1 \leq c_1 * n^2, \forall n \geq n_0$.
Considere $c_1 = 2$ e $n_0 = 1$, temos:
$0 \leq n^2 + 1 \leq 2 * n^2$
$0 \leq 2 \leq 2$.

Para $n \leq n_0$, $f(n)$ estará limitada por um fator constante de $g(n)$, ou seja, não será maior. A taxa de crescimento é no máximo igual.

Notação Ω

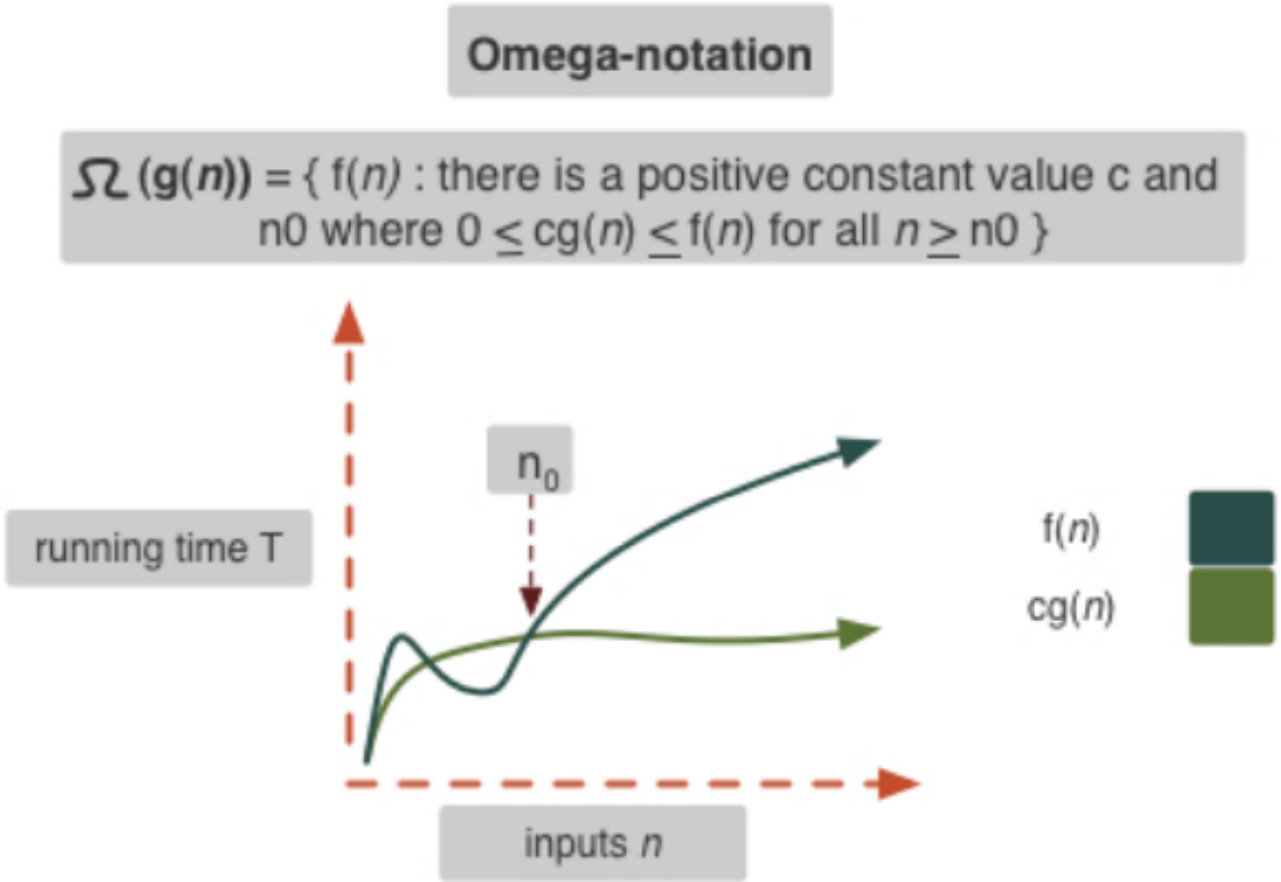
Definição

Formalmente, para uma função $g(n)$ denotamos por $\Omega(g(n))$ o conjunto de funções $\Omega(g(n)) = \{f(n)$: existam constantes positivas, c_1 e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \forall n \geq n_0\}$.

Significado

Uma função $f(n)$ pertence a $g(n)$ caso exista uma constante c_1 tal que ela seja limitada inferiormente por $c_1 g(n)$ para $n \geq n_0$.

Notação Ω



Notação Ω

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \Omega(n^2)$.

Notação Ω

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \Omega(n^2)$.
$0 \leq c_1 * n^2 \leq n^2 + 1, \forall n \geq n_0$.

Notação Ω

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \Omega(n^2)$.
$0 \leq c_1 * n^2 \leq n^2 + 1, \forall n \geq n_0$.
Considere $c_1 = 1$ e $n_0 = 1$, temos:

Notação Ω

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \Omega(n^2)$.

$0 \leq c_1 * n^2 \leq n^2 + 1, \forall n \geq n_0$.

Considere $c_1 = 1$ e $n_0 = 1$, temos:

$0 \leq 1 * n^2 \leq n^2 + 1$

$0 \leq 1 \leq 2$

Notação Ω

Exemplo

Vamos provar que $f(n) = n^2 + 1 = \Omega(n^2)$.
$0 \leq c_1 * n^2 \leq n^2 + 1, \forall n \geq n_0$.
Considere $c_1 = 1$ e $n_0 = 1$, temos:
$0 \leq 1 * n^2 \leq n^2 + 1$
$0 \leq 1 \leq 2$

Para $n \geq n_0$, $f(n)$ estará limitada por um fator constante de $g(n)$, ou seja, não será menor. A taxa de crescimento é no mínimo igual.

Análise assintótica e perspectivas

A notação Ω é utilizada para expressar o **melhor** caso de um algoritmo, ou seja, para definir o limite inferior para o tempo de execução deste algoritmo.

A notação $\mathcal{O}$ é utilizada para expressar o **pior** caso de um algoritmo, ou seja, para definir o limite superior para o tempo de execução deste algoritmo.

Exercício

Qual a complexidade assintótica do algoritmo abaixo? Considere o pior caso.

```
1  public static int busca(int[] x, int valor) {
2      for (int i = 0; i < x.length; ++i) {
3        if (x[i] == valor)
4          return i;
5      }
6      return -1;
7    }
```

Exercício

Qual a complexidade assintótica do algoritmo abaixo? Considere o pior e o melhor caso.

```
1  public static int busca2(int[] x, int valor) {
2      int rand;
3      for (int i = 0; i < 10; ++i) {
4        rand = (int)(Math.random()* x.length - 1);
5        if (x[rand] == valor)
6          return rand;
7      }
8      return -1;
9   }
```

Exercício

Qual a complexidade assintótica do algoritmo abaixo?

```
1  public static long soma(int n) {
2      long soma =0;
3      for (int i = 0; i < n; ++i) {
4         soma+=i;
5
6         for (int j = 0; j < n; ++j){
7            soma+=j;
8         }
9      }
10     return soma;
11 }
```

Exercício

Escreva um algoritmo que retorne o maior e o menor número entre os elementos de um vetor com n posições. Qual a complexidade assintótica do algoritmo que você desenvolveu?

Exercício

Escreva um método, considerando um vetor ordenado de forma não decrescente, que diga se um determinado número existe ou não neste vetor. Este algoritmo deve possuir complexidade assintótica limitada por $O(logn)$

Referências

Para elaboração deste material, além dos livros previstos na ementa da disciplina, foram utilizadas as anotações de aula do professor Marco A. M. Carvalho, professor adjunto da UFOP e o site `https://joaoarthurbm.github.io/eda/posts/analise-assintotica/`.


thank you
danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
gracias
merci
go raibh maith agat
obrigado
dziękuję
bedankt
спасибо
Баярлалаа
dank je
tapadh leat
mauruuru
hvala
sukriya
kop khun krap
terima kasih
grazie
arigatō
dakujem
takk
taiku
sagolun
chnorakaloutioun
gratias ago
gracies
sulpay
mochchakkeram
djiere dieuf
tau
mamnun
дякую
хвала
asante
manana
obrigada
tenki
chokrane
murakoze
mahalo
paldies
grazzi
misaotra
matondo
vinaka
спасибі
blagodaram
faafetai lava
kiitos
dankie
dhanyavad
nanni
nandri
gracie
bayarlalaa
enkosi
koszönöm
akun
dankon
aciū
wetalin
tack
barka
mersi
kia ora
spas
sobodi
dekuji
mèsi
didi madloba
najis tuke
kam sah hamnida
rahmat
ありがとう
tanemirt
rahmet
ευχαριστώ
diolch
dhanyavadagalu
shukriya
merce
мерси
xiexie
감사합니다
তোমাকে ধন্যবাদ