Noções de Análise de Algoritmos

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

1º semestre/2020

Objetivo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Estimar quanto tempo um programa será executado.
- Estimar o tamanho da maior entrada (instância) que pode ser dada ao programa.
- Comparar a eficiência de diferentes algoritmos.
- Escolher um algoritmo para uma aplicação.

Introdução

Análise de Algoritmos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- No desenvolvimento de algoritmos é importante ter a noção da eficiência de um algoritmo.

Análise de Algoritmos

- No desenvolvimento de algoritmos é importante ter a noção da eficiência de um algoritmo.
  - **Eficiência:** normalmente mede-se o tempo de execução ou espaço (memória) necessário à execução do algoritmo.

Análise de Algoritmos

- No desenvolvimento de algoritmos é importante ter a noção da eficiência de um algoritmo.
  - **Eficiência:** normalmente mede-se o tempo de execução ou espaço (memória) necessário à execução do algoritmo.
    - Um algoritmo que realiza uma tarefa em 15 horas é melhor que outro que realiza em 15 dias.

Análise de Algoritmos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- No desenvolvimento de algoritmos é importante ter a noção da eficiência de um algoritmo.
  - **Eficiência:** normalmente mede-se o tempo de execução ou espaço (memória) necessário à execução do algoritmo.
    - Um algoritmo que realiza uma tarefa em 15 horas é melhor que outro que realiza em 15 dias.
  - **Tempo de Execução** de um algoritmo varia com a entrada e normalmente aumenta com o tamanho da entrada.

Análise de Algoritmos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- No desenvolvimento de algoritmos é importante ter a noção da eficiência de um algoritmo.
  - **Eficiência:** normalmente mede-se o tempo de execução ou espaço (memória) necessário à execução do algoritmo.
    - Um algoritmo que realiza uma tarefa em 15 horas é melhor que outro que realiza em 15 dias.
  - **Tempo de Execução** de um algoritmo varia com a entrada e normalmente aumenta com o tamanho da entrada.
    - normalmente, olha-se para o pior caso possível de tempo de execução que é mais simples de analisar e crucial nas aplicações mais exigentes.

Análise de Algoritmos

- No desenvolvimento de algoritmos é importante ter a noção da eficiência de um algoritmo.
  - **Eficiência:** normalmente mede-se o tempo de execução ou espaço (memória) necessário à execução do algoritmo.
    - Um algoritmo que realiza uma tarefa em 15 horas é melhor que outro que realiza em 15 dias.
  - **Tempo de Execução** de um algoritmo varia com a entrada e normalmente aumenta com o tamanho da entrada.
    - normalmente, olha-se para o pior caso possível de tempo de execução que é mais simples de analisar e crucial nas aplicações mais exigentes.
    - Um algoritmo que usa 1MB de memória RAM é melhor que outro que usa 1GB.

Tempo de processamento

- Medir o tempo (em microssengundos, etc.) gasto por um algoritmo.
  - Em certos contextos, não é uma boa opção. **Por quê?**

Tempo de processamento

- Medir o tempo (em microssengundos, etc.) gasto por um algoritmo.
  - Em certos contextos, não é uma boa opção. **Por quê?**
  - Depende do compilador
    - Pode preferir algumas construções ou otimizar melhor.

Tempo de processamento

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Medir o tempo (em microssengundos, etc.) gasto por um algoritmo.
  - Em certos contextos, não é uma boa opção. **Por quê?**
  - Depende do compilador
    - Pode preferir algumas construções ou otimizar melhor.
  - Depende do hardware
    - GPU vs. CPU, desktop vc. smartphone.

Tempo de processamento

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Medir o tempo (em microssengundos, etc.) gasto por um algoritmo.
    - Em certos contextos, não é uma boa opção. **Por quê?**
    - Depende do compilador
        - Pode preferir algumas construções ou otimizar melhor.
    - Depende do hardware
        - GPU vs. CPU, desktop vc. smartphone.
    - Depende da linguagem de programação e habilidade do programador

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
// busca x no vetor v. Retorna indice de x se o encontrar;
// ou retorna -1 caso contrario.
int busca(int v[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (v[i] == x)
      return i;
  return -1;
}
```

Quantos segundos demora para executar a função acima?

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
// busca x no vetor v. Retorna indice de x se o encontrar;
// ou retorna -1 caso contrario.
int busca(int v[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (v[i] == x)
      return i;
  return -1;
}
```

Quantos segundos demora para executar a função acima?

Depende...

- do computador onde ele for rodado

Busca sequencial e consumo de tempo


```c
// busca x no vetor v. Retorna indice de x se o encontrar;
// ou retorna -1 caso contrario.
int busca(int v[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (v[i] == x)
      return i;
  return -1;
}
```

Quantos segundos demora para executar a função acima?

Depende...

- do computador onde ele for rodado
  - computador rápido vs lento

Busca sequencial e consumo de tempo


```c
// busca x no vetor v. Retorna indice de x se o encontrar;
// ou retorna -1 caso contrario.
int busca(int v[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (v[i] == x)
      return i;
  return -1;
}
```

Quantos segundos demora para executar a função acima?

Depende...

- do computador onde ele for rodado
  - computador rápido vs lento
- da posição de x no vetor

Busca sequencial e consumo de tempo

```
1 // busca x no vetor v. Retorna indice de x se o encontrar;
2 // ou retorna -1 caso contrario.
3 int busca(int v[], int n, int x) {
4   int i;
5   for (i = 0; i < n; i++)
6     if (v[i] == x)
7       return i;
8   return -1;
9 }
```

Quantos segundos demora para executar a função acima?

Depende...

- do computador onde ele for rodado
    - computador rápido vs lento
- da posição de x no vetor
    - no melhor caso, a linha 6 é executada 1 vez
    - no pior caso, a linha 6 é executada n vezes

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 // busca x no vetor v. Retorna indice de x se o encontrar;
2 // ou retorna -1 caso contrario.
3 int busca(int v[], int n, int x) {
4   int i;
5   for (i = 0; i < n; i++)
6     if (v[i] == x)
7       return i;
8   return -1;
9 }
```

Quantos segundos demora para executar a função acima?

Depende...

- do computador onde ele for rodado
  - computador rápido vs lento
- da posição de x no vetor
  - no melhor caso, a linha 6 é executada 1 vez
  - no pior caso, a linha 6 é executada n vezes
- do valor de n
  - $n = 10$ vs $n = 10.000$

Complexidade computacional e assintótica

Queremos analisar algoritmos:

- Independentemente do computador onde ele for rodado
- Em função do valor de n (a quantidade de dados)

Complexidade computacional e assintótica

Queremos analisar algoritmos:

- Independentemente do computador onde ele for rodado
- Em função do valor de n (a quantidade de dados)

Para evitar análises dependente de tempo, considera-se que um algoritmo é subdividido, ao invés de milissegundos, em uma quantidade finita de *passos*.

Complexidade computacional e assintótica

Queremos analisar algoritmos:

- Independentemente do computador onde ele for rodado
- Em função do valor de n (a quantidade de dados)

Para evitar análises dependente de tempo, considera-se que um algoritmo é subdividido, ao invés de milissegundos, em uma quantidade finita de *passos*.

- Um passo pode ser interpretado como uma instrução indivisível e de tempo constante, ou seja, independente de condições de entrada e processamento.
  - Exemplo: soma, multiplicação, atribuição, comparação.

Complexidade computacional e assintótica

Queremos analisar algoritmos:

- Independentemente do computador onde ele for rodado
- Em função do valor de n (a quantidade de dados)

Para evitar análises dependente de tempo, considera-se que um algoritmo é subdividido, ao invés de milissegundos, em uma quantidade finita de *passos*.

- Um passo pode ser interpretado como uma instrução indivisível e de tempo constante, ou seja, independente de condições de entrada e processamento.
  - Exemplo: soma, multiplicação, atribuição, comparação.
- A quantidade de passos necessários ao cumprimento de um algoritmo é denominada complexidade do algoritmo.

Pior caso, caso médio e melhor caso

Universidade Federal do Ceará
Campus Quixadá

Na análise da complexidade, definimos três casos de entrada para um algoritmo, como forma de mensurar o custo do algoritmo de resolver determinado problema diante de diferentes entradas.

- **Melhor caso:** o menor tempo de execução sobre todas as entradas de tamanho n.

Pior caso, caso médio e melhor caso

Na análise da complexidade, definimos três casos de entrada para um algoritmo, como forma de mensurar o custo do algoritmo de resolver determinado problema diante de diferentes entradas.

- **Melhor caso:** o menor tempo de execução sobre todas as entradas de tamanho n.
- **Pior caso:** maior tempo de execução sobre todas as entradas de tamanho n.

Pior caso, caso médio e melhor caso

Na análise da complexidade, definimos três casos de entrada para um algoritmo, como forma de mensurar o custo do algoritmo de resolver determinado problema diante de diferentes entradas.

- **Melhor caso:** o menor tempo de execução sobre todas as entradas de tamanho n.
- **Pior caso:** maior tempo de execução sobre todas as entradas de tamanho n.
- **Caso médio:** é a média dos tempos de execução de todas as entradas de tamanho n.

Pior caso, caso médio e melhor caso

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Em geral, queremos analisar o pior caso do algoritmo.

- A análise do melhor caso pode ser interesse, mas é rara.
- A análise do caso médio é mais difícil
  - É uma análise probabilística
  - Precisamos fazer suposições sobre os dados de entrada

Medindo a complexidade de Algoritmos

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
int busca(int v[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (v[i] == x)
      return i;
  return -1;
}
```

Busca sequencial e consumo de tempo

```
int busca(int v[], int n, int x) {
  int i;
  for (i = 0; i < n; i++)
    if (v[i] == x)
      return i;
  return -1;
}
```

Consumo de tempo por linha no pior caso:

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)

Busca sequencial e consumo de tempo

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

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
  - No pior caso, essa linha é executada $n + 1$ vezes

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
  - No pior caso, essa linha é executada $n + 1$ vezes
- Linha 4: tempo c_4 (acessos e comparação)

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
1 int busca(int v[], int n, int x) {
2   int i;
3   for (i = 0; i < n; i++)
4     if (v[i] == x)
5       return i;
6   return -1;
7 }
```

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
  - No pior caso, essa linha é executada $n + 1$ vezes
- Linha 4: tempo c_4 (acessos e comparação)
  - No pior caso, essa linha é executada n vezes

Busca sequencial e consumo de tempo

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

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
  - No pior caso, essa linha é executada $n + 1$ vezes
- Linha 4: tempo c_4 (acessos e comparação)
  - No pior caso, essa linha é executada n vezes
- Linha 5: tempo c_5 (acesso e `return`)

Busca sequencial e consumo de tempo


```c
1 int busca(int v[], int n, int x) {
2   int i;
3   for (i = 0; i < n; i++)
4     if (v[i] == x)
5       return i;
6   return -1;
7 }
```

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
  - No pior caso, essa linha é executada $n + 1$ vezes
- Linha 4: tempo c_4 (acessos e comparação)
  - No pior caso, essa linha é executada n vezes
- Linha 5: tempo c_5 (acesso e `return`)
- Linha 6: tempo c_6 (`return`)

Busca sequencial e consumo de tempo

```c
1 int busca(int v[], int n, int x) {
2   int i;
3   for (i = 0; i < n; i++)
4     if (v[i] == x)
5       return i;
6   return -1;
7 }
```

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
    - No pior caso, essa linha é executada $n + 1$ vezes
- Linha 4: tempo c_4 (acessos e comparação)
    - No pior caso, essa linha é executada n vezes
- Linha 5: tempo c_5 (acesso e `return`)
- Linha 6: tempo c_6 (`return`)

O tempo de execução é menor ou igual a

Busca sequencial e consumo de tempo

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

Consumo de tempo por linha no pior caso:

- Linha 2: tempo c_2 (declaração de variável)
- Linha 3: tempo c_3 (atribuições, acessos e comparação)
  - No pior caso, essa linha é executada $n+1$ vezes
- Linha 4: tempo c_4 (acessos e comparação)
  - No pior caso, essa linha é executada n vezes
- Linha 5: tempo c_5 (acesso e `return`)
- Linha 6: tempo c_6 (`return`)

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n + 1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 = c_2 + c_3 + c_5 + c_6 + (c_3 + c_4) \cdot n$$

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n + 1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$$\begin{aligned} c_2 + c_3 \cdot (n + 1) + c_4 \cdot n + c_5 + c_6 &= c_2 + c_3 + c_5 + c_6 + (c_3 + c_4) \cdot n \\ &= a + b \cdot n \end{aligned}$$

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$$\begin{aligned} c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 &= c_2 + c_3 + c_5 + c_6 + (c_3 + c_4) \cdot n \\ &= a + b \cdot n \leq a \cdot n + b \cdot n \end{aligned}$$

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$$\begin{aligned} c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 &= c_2 + c_3 + c_5 + c_6 + (c_3 + c_4) \cdot n \\ &= a + b \cdot n \leq a \cdot n + b \cdot n = d \cdot n \end{aligned}$$

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$$\begin{aligned} c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 &= c_2 + c_3 + c_5 + c_6 + (c_3 + c_4) \cdot n \\ &= a + b \cdot n \leq a \cdot n + b \cdot n = d \cdot n \end{aligned}$$

Isto é, o crescimento do tempo é linear em n

Busca sequencial e consumo de tempo

O tempo de execução é menor ou igual a

$$c_2 + c_3 \cdot (n + 1) + c_4 \cdot n + c_5 + c_6$$

Cada c_i não depende de n, depende apenas do computador

- Leva um tempo constante

Sejam $a := c_2 + c_3 + c_5 + c_6$, $b := c_3 + c_4$ e $d := a + b$

Se $n \geq 1$, temos que o tempo de execução é menor ou igual a

$$\begin{aligned} c_2 + c_3 \cdot (n + 1) + c_4 \cdot n + c_5 + c_6 &= c_2 + c_3 + c_5 + c_6 + (c_3 + c_4) \cdot n \\ &= a + b \cdot n \leq a \cdot n + b \cdot n = d \cdot n \end{aligned}$$

Isto é, o crescimento do tempo é linear em n

- Se n dobra, o tempo de execução praticamente dobra

Busca sequencial e consumo de tempo

Como vimos, existe uma constante d tal que, para $n \geq 1$,

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 \leq dn$$

Busca sequencial e consumo de tempo

Universidade Federal do Ceará
Campus Quixadá

Como vimos, existe uma constante d tal que, para $n \geq 1$,

$$c_2 + c_3 \cdot (n + 1) + c_4 \cdot n + c_5 + c_6 \leq dn$$

d não interessa tanto, depende apenas do computador...

Busca sequencial e consumo de tempo

Como vimos, existe uma constante d tal que, para $n \geq 1$,

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 \leq dn$$

d não interessa tanto, depende apenas do computador...

- Estamos preocupados em estimar

Busca sequencial e consumo de tempo

Como vimos, existe uma constante d tal que, para $n \geq 1$,

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 \leq dn$$

d não interessa tanto, depende apenas do computador...

- Estamos preocupados em estimar

O tempo do algoritmo é da ordem de n

Busca sequencial e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Como vimos, existe uma constante d tal que, para $n \geq 1$,

$$c_2 + c_3 \cdot (n+1) + c_4 \cdot n + c_5 + c_6 \leq dn$$

d não interessa tanto, depende apenas do computador...

- Estamos preocupados em estimar

O tempo do algoritmo é da ordem de n

- A ordem de crescimento do tempo é igual a de $f(n) = n$

Soma de matrizes

Sejam A e B duas matrizes quadradas de ordem n que devem ser somadas.

Soma de matrizes

Sejam A e B duas matrizes quadradas de ordem n que devem ser somadas.

```c
void soma_matrizes (int **A, int **B, int **C, int n){
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      C[i][j] = A[i][j] + B[i][j];
}
```

Soma de matrizes

Sejam A e B duas matrizes quadradas de ordem n que devem ser somadas.

```
void soma_matrizes (int **A, int **B, int **C, int n){
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      C[i][j] = A[i][j] + B[i][j];
}
```

- Qual o tempo de execução desse algoritmo?

Soma de matrizes

Sejam A e B duas matrizes quadradas de ordem n que devem ser somadas.

```
1 void soma_matrizes (int **A, int **B, int **C, int n){
2   for (int i = 0; i < n; i++)
3     for (int j = 0; j < n; j++)
4       C[i][j] = A[i][j] + B[i][j];
5 }
```

- Qual o tempo de execução desse algoritmo?
- A operação que predomina no algoritmo é a soma (realizada na linha 4).

Soma de matrizes

Sejam A e B duas matrizes quadradas de ordem n que devem ser somadas.

```
1 void soma_matrizes (int **A, int **B, int **C, int n){
2   for (int i = 0; i < n; i++)
3     for (int j = 0; j < n; j++)
4       C[i][j] = A[i][j] + B[i][j];
5 }
```

- Qual o tempo de execução desse algoritmo?
- A operação que predomina no algoritmo é a soma (realizada na linha 4).
- Logo, expressamos a complexidade como sendo o total de vezes que a soma acontece.

Soma de matrizes

Sejam A e B duas matrizes quadradas de ordem n que devem ser somadas.

```
1 void soma_matrizes (int **A, int **B, int **C, int n){
2   for (int i = 0; i < n; i++)
3     for (int j = 0; j < n; j++)
4       C[i][j] = A[i][j] + B[i][j];
5 }
```

- Qual o tempo de execução desse algoritmo?
- A operação que predomina no algoritmo é a soma (realizada na linha 4).
- Logo, expressamos a complexidade como sendo o total de vezes que a soma acontece.
- Associando-se os laços, contabilizam-se um total de $n \cdot n = n^2$ iterações. Logo, a complexidade é dada por $f(n) = n^2$.

Complexidade de melhor caso e pior caso

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Diferentemente do algoritmo para soma de matrizes, na maior parte dos algoritmos conhecidos a função de complexidade $f(n)$ não é geral, ou seja, pode mudar conforme características da entrada.

Complexidade de melhor caso e pior caso

Universidade Federal do Ceará
Campus Quixadá

- Diferentemente do algoritmo para soma de matrizes, na maior parte dos algoritmos conhecidos a função de complexidade $f(n)$ não é geral, ou seja, pode mudar conforme características da entrada.

- Nestes casos, a análise de complexidade consiste em avaliar o algoritmo em situações extremas:

Complexidade de melhor caso e pior caso

- Diferentemente do algoritmo para soma de matrizes, na maior parte dos algoritmos conhecidos a função de complexidade $f(n)$ não é geral, ou seja, pode mudar conforme características da entrada.
- Nestes casos, a análise de complexidade consiste em avaliar o algoritmo em situações extremas:
  - Determinar que funções de complexidade descrevem os casos de melhor e de pior entrada.

Máximo e Mínimo elementos de um vetor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
void max_min(int v[], int n, int *max, int *min) {
    *max = *min = 0;
    for (int k = 1; k < n; k++)
        if (v[k] < v[*min])
            *min = k;
        else if (v[k] > v[*max])
            *max = k;
}
```

Máximo e Mínimo elementos de um vetor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
void max_min(int v[], int n, int *max, int *min) {
    *max = *min = 0;
    for (int k = 1; k < n; k++)
        if (v[k] < v[*min])
            *min = k;
        else if (v[k] > v[*max])
            *max = k;
}
```

- Qual a complexidade de melhor caso desse algoritmo?

Máximo e Mínimo elementos de um vetor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void max_min(int v[], int n, int *max, int *min) {
2     *max = *min = 0;
3     for (int k = 1; k < n; k++)
4         if (v[k] < v[*min])
5             *min = k;
6         else if (v[k] > v[*max])
7             *max = k;
8 }
```

- Qual a complexidade de melhor caso desse algoritmo?
- No melhor caso, todos os testes da linha 4 obtêm êxito, impedindo a execução dos testes da linha 6; fazendo assim um total de $n-1$ testes. Logo, $f(n) = n - 1$ no melhor caso.

Máximo e Mínimo elementos de um vetor


```
1 void max_min(int v[], int n, int *max, int *min) {
2     *max = *min = 0;
3     for (int k = 1; k < n; k++)
4         if (v[k] < v[*min])
5             *min = k;
6         else if (v[k] > v[*max])
7             *max = k;
8 }
```

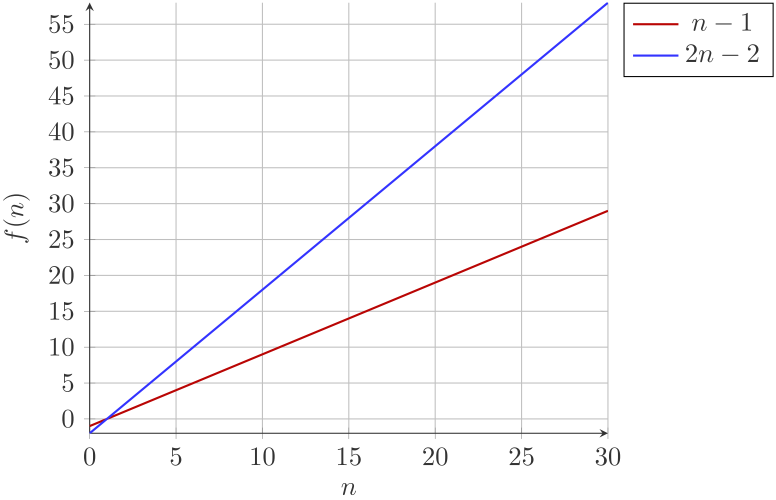
- Qual a complexidade de melhor caso desse algoritmo?
- No melhor caso, todos os testes da linha 4 obtêm êxito, impedindo a execução dos testes da linha 6; fazendo assim um total de $n-1$ testes. Logo, $f(n) = n - 1$ no melhor caso.
- Qual a complexidade de pior caso desse algoritmo?

Máximo e Mínimo elementos de um vetor


```
1 void max_min(int v[], int n, int *max, int *min) {
2     *max = *min = 0;
3     for (int k = 1; k < n; k++)
4         if (v[k] < v[*min])
5             *min = k;
6         else if (v[k] > v[*max])
7             *max = k;
8 }
```

- Qual a complexidade de melhor caso desse algoritmo?
- No melhor caso, todos os testes da linha 4 obtêm êxito, impedindo a execução dos testes da linha 6; fazendo assim um total de $n - 1$ testes. Logo, $f(n) = n - 1$ no melhor caso.
- Qual a complexidade de pior caso desse algoritmo?
- No pior caso, todos os testes da linha 4 devem falhar, fazendo com que todos os testes da linha 6 ocorram. Logo, $f(n) = 2(n - 1)$ no pior caso.

Comparando as funções $n - 1$ e $2(n - 1)$



Ordem de crescimento assintótico

Comportamento assintótico

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Motivação: Determinar a complexidade de tempo exata de um algoritmo é muito difícil e frequentemente não faz muito sentido.

Comportamento assintótico

- Motivação: Determinar a complexidade de tempo exata de um algoritmo é muito difícil e frequentemente não faz muito sentido.

- Para valores suficientemente pequenos de n, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes.
  - Escolha de um algoritmo não é um problema crítico.

Comportamento assintótico

- Motivação: Determinar a complexidade de tempo exata de um algoritmo é muito difícil e frequentemente não faz muito sentido.

- Para valores suficientemente pequenos de n, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes.
    - Escolha de um algoritmo não é um problema crítico.

- Logo, analisamos algoritmos para grandes valores de n.
    - Estudamos o comportamento assintótico das funções de complexidade de um programa (comportamento para grandes valores de n).

Comparando funções

Queremos comparar duas funções f e g

Comparando funções

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Queremos comparar duas funções f e g

- Queremos entender a velocidade de crescimento de f

Comparando funções

Queremos comparar duas funções f e g

- Queremos entender a velocidade de crescimento de f
- Queremos dizer que f cresce mais lentamente ou igual a g

Comparando funções

Queremos comparar duas funções f e g

- Queremos entender a velocidade de crescimento de f
- Queremos dizer que f cresce mais lentamente ou igual a g

f pode ser o tempo de execução do algoritmo e g uma função mais simples

Comparando funções

Universidade Federal do Ceará
Campus Quixadá

Queremos comparar duas funções f e g

- Queremos entender a velocidade de crescimento de f
- Queremos dizer que f cresce mais lentamente ou igual a g

f pode ser o tempo de execução do algoritmo e g uma função mais simples

- $f(n) = 3n^2 + 10 \lg n$ e $g(n) = n^2$

Comparando funções

Universidade Federal do Ceará
Campus Quixadá

Queremos comparar duas funções f e g

- Queremos entender a velocidade de crescimento de f
- Queremos dizer que f cresce mais lentamente ou igual a g

f pode ser o tempo de execução do algoritmo e g uma função mais simples

- $f(n) = 3n^2 + 10 \lg n$ e $g(n) = n^2$

f e g podem ser os tempos de execução de dois algoritmos

Comparando funções

Queremos comparar duas funções f e g

- Queremos entender a velocidade de crescimento de f
- Queremos dizer que f cresce mais lentamente ou igual a g

f pode ser o tempo de execução do algoritmo e g uma função mais simples

- $f(n) = 3n^2 + 10 \lg n$ e $g(n) = n^2$

f e g podem ser os tempos de execução de dois algoritmos

- $f(n) = dn$ e $g(n) = c + c \lg n$

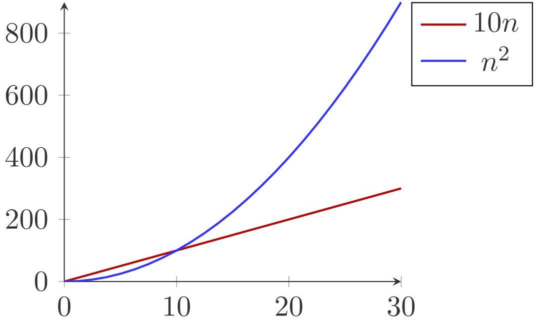
Primeira Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para todo n

Primeira Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para todo n

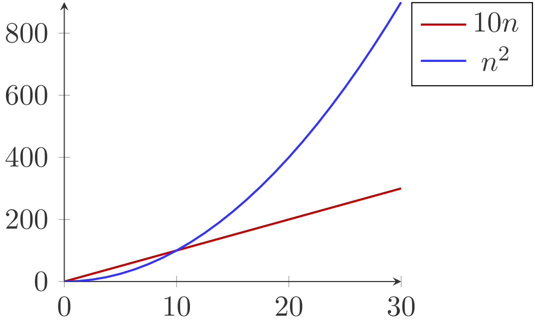
Problema: $10n > n^2$ para $n < 10$



Primeira Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para todo n

Problema: $10n > n^2$ para $n < 10$

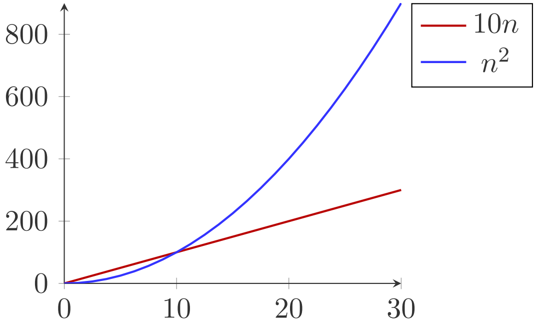


Solução: Ao invés de comparar todo n, comparar apenas n suficientemente grande

Primeira Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para todo n

Problema: $10n > n^2$ para $n < 10$



Solução: Ao invés de comparar todo n, comparar apenas n suficientemente grande

- Para todo $n \geq n_0$ para algum n_0

Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Problema: $n + 5 > n$ para todo n

Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Problema: $n + 5 > n$ para todo n

- Mas a velocidade de crescimento das funções é o mesmo

Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Universidade Federal do Ceará
Campus Quixadá

Problema: $n + 5 > n$ para todo n

- Mas a velocidade de crescimento das funções é o mesmo
- Constantes dependem da máquina onde executamos

Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Problema: $n + 5 > n$ para todo n

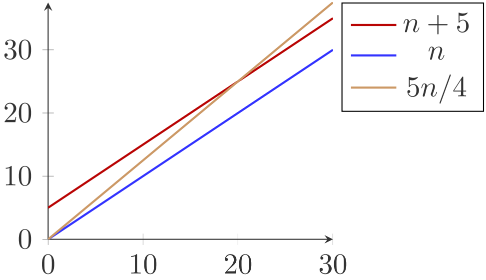
- Mas a velocidade de crescimento das funções é o mesmo
- Constantes dependem da máquina onde executamos
- Vamos ignorar constantes e termos menos importantes

Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Problema: $n + 5 > n$ para todo n

- Mas a velocidade de crescimento das funções é o mesmo
- Constantes dependem da máquina onde executamos
- Vamos ignorar constantes e termos menos importantes

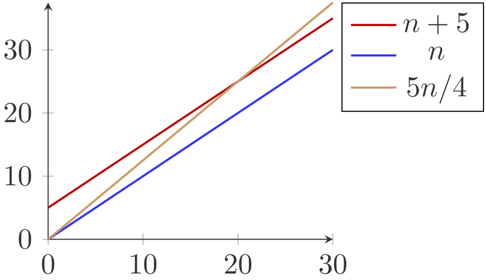


Segunda Ideia

Comparar funções verificando se $f(n) \leq g(n)$ para $n \geq n_0$

Problema: $n + 5 > n$ para todo n

- Mas a velocidade de crescimento das funções é o mesmo
- Constantes dependem da máquina onde executamos
- Vamos ignorar constantes e termos menos importantes



Solução: Ao invés de comparar f com g, comparar com $c \cdot g$, onde c é uma constante

Notação Assintótica

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Denomina-se notação assintótica a forma matemática de representação simplificada de uma função $f(n)$ levando em conta as componentes de f que crescem mais rapidamente quando o valor de n *tende ao infinito*.

Notação Assintótica

- Denominaa-se notação assintótica a forma matemática de representação simplificada de uma função $f(n)$ levando em conta as componentes de f que crescem mais rapidamente quando o valor de n *tende ao infinito*.

- Veremos duas dessas importantes notações:
  - Notação O
  - Notação Ω (Ômega)

Notação O

Notação Assintótica – Notação O

Dada uma função $f(n)$, dizemos que $f(n) = O(g(n))$ se

- existem constantes positivas c e n_0, tais que

Notação Assintótica – Notação O

Dada uma função $f(n)$, dizemos que $f(n) = O(g(n))$ se

- existem constantes positivas c e n_0, tais que

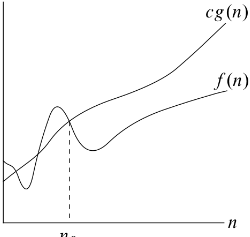
$$0 \leq f(n) \leq c \cdot g(n), \qquad \text{para todo } n \geq n_0.$$

Notação Assintótica – Notação O

Dada uma função $f(n)$, dizemos que $f(n) = O(g(n))$ se

- existem constantes positivas c e n_0, tais que

$$0 \leq f(n) \leq c \cdot g(n), \qquad \text{para todo } n \geq n_0.$$



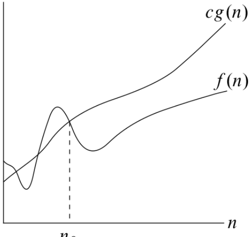
$f(n) = O(g(n))$

Notação Assintótica – Notação O

Dada uma função $f(n)$, dizemos que $f(n) = O(g(n))$ se

- existem constantes positivas c e n_0, tais que

$$0 \leq f(n) \leq c \cdot g(n), \qquad \text{para todo } n \geq n_0.$$



$f(n) = O(g(n))$

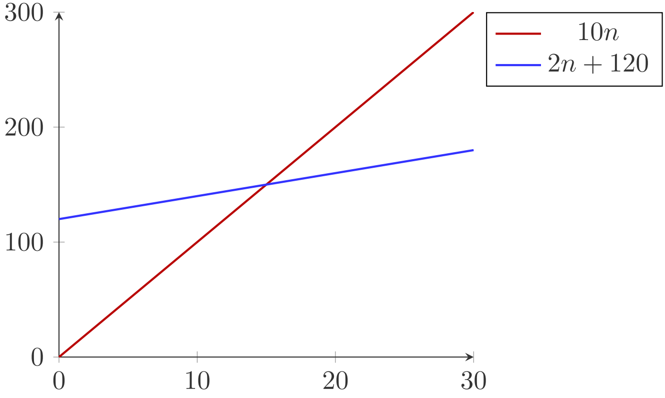
$f(n) = O(g(n))$ se, para todo n suficientemente grande, $f(n)$ é menor ou igual a um múltiplo de $g(n)$

Exemplo: $2n + 120 = O(n)$

Universidade Federal do Ceará
Campus Quixadá

Exemplo: $2n + 120 = O(n)$

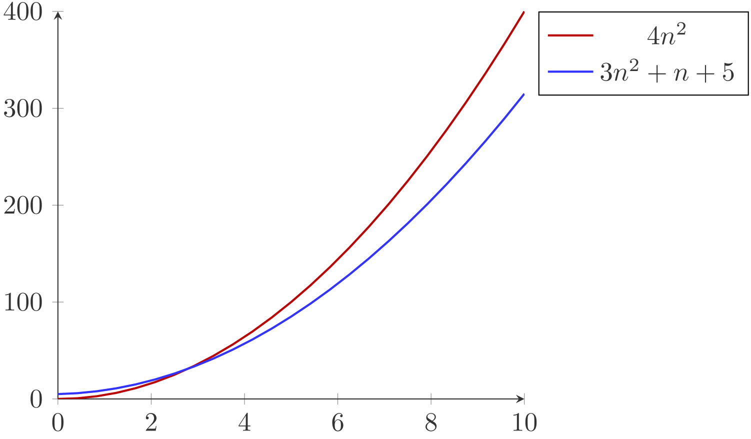
Basta escolher, por exemplo, $c = 10$ e $n_0 = 15$



Exemplo: $3n^2 + n + 5 = O(n^2)$

Exemplo: $3n^2 + n + 5 = O(n^2)$

Basta escolher, por exemplo, $c = 4$ e $n_0 = 4$



Outros exemplos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

$$1 = O(1)$$

Outros exemplos

$$1 = O(1)$$

$$1.000.000 = O(1)$$

Outros exemplos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

$$1 = O(1)$$

$$1.000.000 = O(1)$$

$$5n + 2 = O(n)$$

Outros exemplos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

$$1 = O(1)$$

$$1.000.000 = O(1)$$

$$5n + 2 = O(n)$$

$$5n^2 + 5n + 2 = O(n^2)$$

Outros exemplos

Universidade Federal do Ceará
Campus Quixadá

$$1 = O(1)$$

$$1.000.000 = O(1)$$

$$5n + 2 = O(n)$$

$$5n^2 + 5n + 2 = O(n^2)$$

$$\log_2 n = O(\log_{10} n)$$

Outros exemplos

$$
\begin{aligned}
1 &= O(1) \\
1.000.000 &= O(1) \\
5n + 2 &= O(n) \\
5n^2 + 5n + 2 &= O(n^2) \\
\log_2 n &= O(\log_{10} n) \\
\log_{10} n &= O(\log_2 n)
\end{aligned}
$$

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(1)$: tempo constante
    - não depende de n

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)
  - Ex: acesso a uma posição de um vetor

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)
  - Ex: acesso a uma posição de um vetor

- $O(\lg n)$: logarítmico

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)
  - Ex: acesso a uma posição de um vetor
- $O(\lg n)$: logarítmico
  - $\lg$ indica $\log_2$

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)
  - Ex: acesso a uma posição de um vetor

- $O(\lg n)$: logarítmico
  - $\lg$ indica $\log_2$
  - quando n dobra, o tempo aumenta em uma constante

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)
  - Ex: acesso a uma posição de um vetor
- $O(\lg n)$: logarítmico
  - $\lg$ indica $\log_2$
  - quando n dobra, o tempo aumenta em uma constante
  - Ex: Busca binária

Nomenclatura e consumo de tempo

- $O(1)$: tempo constante
  - não depende de n
  - Ex: atribuição e leitura de uma variável
  - Ex: operações aritméticas: `+`, `-`, `*`, `/`
  - Ex: comparações (`<`, `<=`, `==`, `>=`, `>`, `!=`)
  - Ex: operadores booleanos (`&&`, `&`, `||`, `|`, `!`)
  - Ex: acesso a uma posição de um vetor

- $O(\lg n)$: logarítmico
  - $\lg$ indica $\log_2$
  - quando n dobra, o tempo aumenta em uma constante
  - Ex: Busca binária
  - Outros exemplos durante o curso

Nomenclatura e consumo de tempo

- $O(n)$: linear

Nomenclatura e consumo de tempo

- $O(n)$: linear
  - quando n dobra, o tempo dobra

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear

Nomenclatura e consumo de tempo

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores
- $O(n \lg n)$:

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores

- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores

- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos

Nomenclatura e consumo de tempo

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores
- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos
- $O(n^2)$: quadrático

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores
- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos
- $O(n^2)$: quadrático
  - quando n dobra, o tempo quadriplica

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores
- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos
- $O(n^2)$: quadrático
  - quando n dobra, o tempo quadriplica
  - Ex: BubbleSort, SelectionSort e InsertionSort

Nomenclatura e consumo de tempo

Universidade Federal do Ceará
Campus Quixadá

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores
- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos
- $O(n^2)$: quadrático
  - quando n dobra, o tempo quadriplica
  - Ex: BubbleSort, SelectionSort e InsertionSort
- $O(n^3)$: cúbico

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores
- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos
- $O(n^2)$: quadrático
  - quando n dobra, o tempo quadriplica
  - Ex: BubbleSort, SelectionSort e InsertionSort
- $O(n^3)$: cúbico
  - quando n dobra, o tempo octuplica

Nomenclatura e consumo de tempo

- $O(n)$: linear
  - quando n dobra, o tempo dobra
  - Ex: Busca linear
  - Ex: Encontrar o máximo/mínimo de um vetor
  - Ex: Produto interno de dois vetores

- $O(n \lg n)$:
  - quando n dobra, o tempo um pouco mais que dobra
  - Ex: algoritmos de ordenação que veremos

- $O(n^2)$: quadrático
  - quando n dobra, o tempo quadriplica
  - Ex: BubbleSort, SelectionSort e InsertionSort

- $O(n^3)$: cúbico
  - quando n dobra, o tempo octuplica
  - Ex: multiplicação de matrizes $n \times n$

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $f(n) = O(c^n)$: complexidade exponencial

Nomenclatura e consumo de tempo

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.
  - Não são úteis do ponto de vista prático.

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.
  - Não são úteis do ponto de vista prático.
    - Quando n é 20, $O(2^n)$ é um milhão.

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.
  - Não são úteis do ponto de vista prático.
    - Quando n é 20, $O(2^n)$ é um milhão.

- $f(n) = O(n!)$: complexidade exponencial

Nomenclatura e consumo de tempo

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.
  - Não são úteis do ponto de vista prático.
    - Quando n é 20, $O(2^n)$ é um milhão.

- $f(n) = O(n!)$: complexidade exponencial
  - Pior que $O(c^n)$

Nomenclatura e consumo de tempo

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.
  - Não são úteis do ponto de vista prático.
    - Quando n é 20, $O(2^n)$ é um milhão.

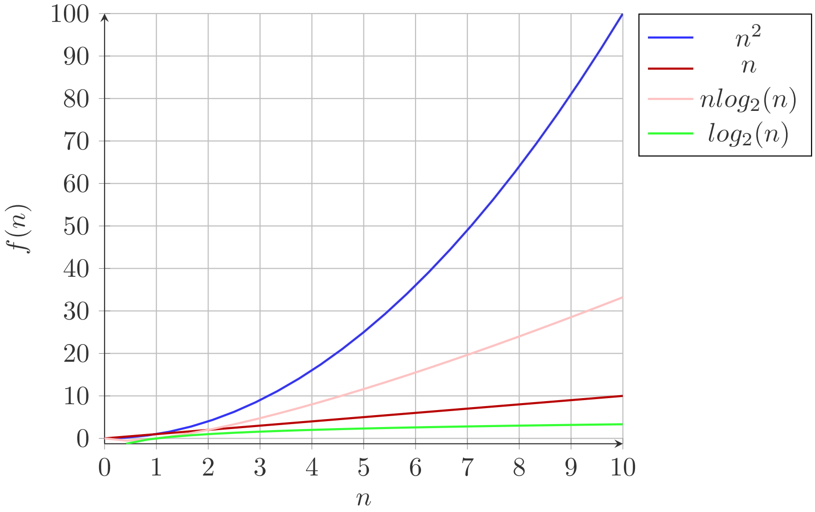
- $f(n) = O(n!)$: complexidade exponencial
  - Pior que $O(c^n)$
  - Não são úteis do pronto de vista prático.

Nomenclatura e consumo de tempo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- $f(n) = O(c^n)$: complexidade exponencial
  - Típico de algoritmos que fazem busca exaustiva (força bruta) para resolver um problema.
  - Não são úteis do ponto de vista prático.
    - Quando n é 20, $O(2^n)$ é um milhão.
- $f(n) = O(n!)$: complexidade exponencial
  - Pior que $O(c^n)$
  - Não são úteis do pronto de vista prático.
  - Quando n é 20, $O(n!)$ é maior que 2 quintilhões.

Comparando quatro funções



Comparação de funções de complexidade

Tamanho	Função de custo					
n	$\lg_2 n$	n	$n \lg_2 n$	n^2	n^3	2^n
10	3	10	30	100	1000	1000
100	6	100	664	10^4	10^6	10^{30}
1000	9	1000	9965	10^6	10^9	10^{300}
10^4	13	10^4	10^5	10^8	10^{12}	10^{3000}
10^5	16	10^5	10^6	10^{10}	10^{15}	10^{30000}
10^6	19	10^6	10^7	10^{12}	10^{18}	10^{300000}

1 semana $\approx 1,21 \cdot 10^6$ segundos
1 ano $\approx 3 \cdot 10^7$ segundos
1 século $\approx 3 \cdot 10^9$ segundos
1 milênio $\approx 3 \cdot 10^{10}$ segundos

Um cuidado

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

Um cuidado

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)

Um cuidado

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)
- O tempo é menor ou igual a um múltiplo de n^3

Um cuidado

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)
- O tempo é menor ou igual a um múltiplo de n^3

Pode ser que o tempo do algoritmo seja $2n^2$...

Um cuidado

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)
- O tempo é menor ou igual a um múltiplo de n^3

Pode ser que o tempo do algoritmo seja $2n^2$...

- $2n^2 = O(n^3)$, mas...

Um cuidado

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)
- O tempo é menor ou igual a um múltiplo de n^3

Pode ser que o tempo do algoritmo seja $2n^2$...

- $2n^2 = O(n^3)$, mas...
- $2n^2 = O(n^2)$

Um cuidado

Universidade Federal do Ceará
Campus Quixadá

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)
- O tempo é menor ou igual a um múltiplo de n^3

Pode ser que o tempo do algoritmo seja $2n^2$...

- $2n^2 = O(n^3)$, mas...
- $2n^2 = O(n^2)$

Ou seja, podemos ter feito uma análise "folgada"

Um cuidado

O que significa dizer que o tempo de um algoritmo é $O(n^3)$?

- Para instâncias grandes ($n \geq n_0$)
- O tempo é menor ou igual a um múltiplo de n^3

Pode ser que o tempo do algoritmo seja $2n^2$...

- $2n^2 = O(n^3)$, mas...
- $2n^2 = O(n^2)$

Ou seja, podemos ter feito uma análise "folgada"

- achamos que o algoritmo é muito pior do que é realmente

Notação Ω

Notação Assintótica – Notação Ω

Dada uma função $f(n)$, dizemos $f(n) = \Omega(g(n))$ se

- existem constantes positivas c e n_0, tais que:

Notação Assintótica – Notação Ω

Dada uma função $f(n)$, dizemos $f(n) = \Omega(g(n))$ se

- existem constantes positivas c e n_0, tais que:

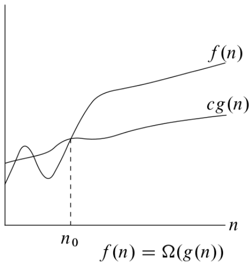
$$0 \leq c \cdot g(n) \leq f(n), \qquad \text{para todo } n \geq n_0.$$

Notação Assintótica – Notação Ω

Dada uma função $f(n)$, dizemos $f(n) = \Omega(g(n))$ se

- existem constantes positivas c e n_0, tais que:

$$0 \leq c \cdot g(n) \leq f(n), \qquad \text{para todo } n \geq n_0.$$

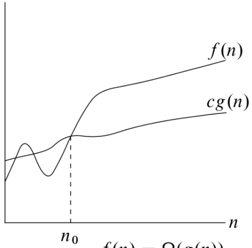


Notação Assintótica – Notação Ω

Dada uma função $f(n)$, dizemos $f(n) = \Omega(g(n))$ se

- existem constantes positivas c e n_0, tais que:

$$0 \leq c \cdot g(n) \leq f(n), \qquad \text{para todo } n \geq n_0.$$



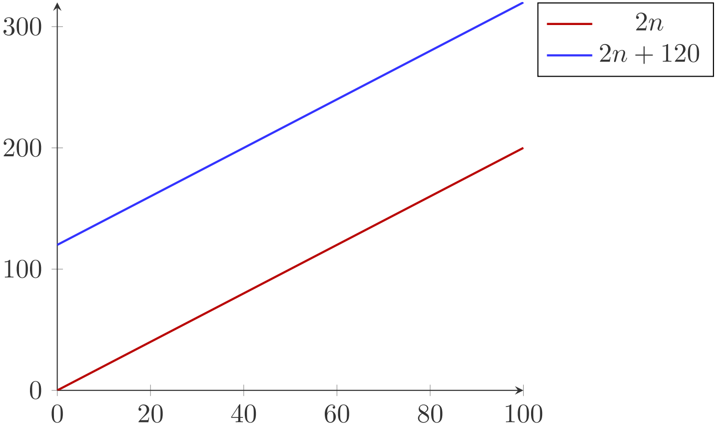
$f(n) = \Omega(g(n))$

$f(n) = \Omega(g(n))$ se, para todo n suficientemente grande, $f(n)$ é maior ou igual a um múltiplo de $g(n)$

Exemplo: $2n + 120 = \Omega(n)$

Exemplo: $2n + 120 = \Omega(n)$

Basta escolher, por exemplo, $c = 2$ e $n_0 = 0$

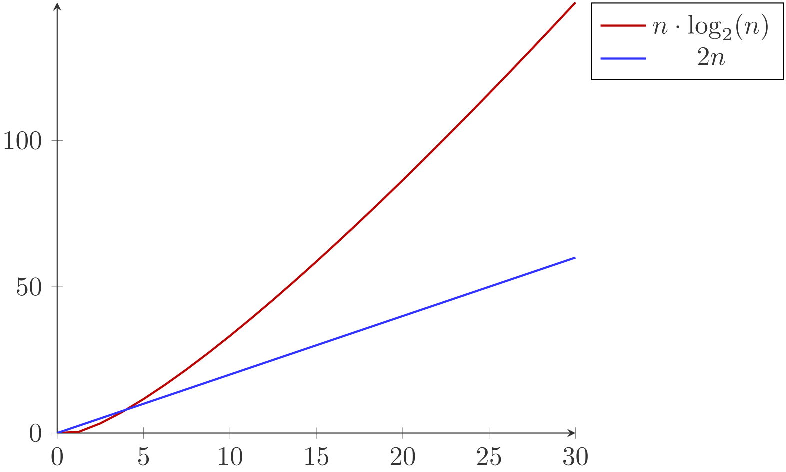


Exemplo: $n \log_2(n) = \Omega(n)$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Exemplo: $n \log_2(n) = \Omega(n)$

Basta escolher, por exemplo, $c = 2$ e $n_0 = 4$



Observação Final

- Também podemos usar limites para determinar a ordem das funções, devido às seguintes declarações de equivalência que relacionam limites com as definições de limites assintóticos:

$$f(n) \in O(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty \quad \text{(constante ou zero)}$$

$$f(n) \in \Omega(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} > 0 \quad \text{(constante } > 0 \text{ ou infinito)}$$

Conclusão

- A análise de algoritmos é útil para definir o algoritmo mais eficiente em determinados problemas.
- O objetivo final não é apenas fazer códigos que funcionem, mas que sejam também eficientes.

> "Um bom algoritmo, mesmo rodando em uma máquina lenta, sempre acaba derrotando (para instâncias grandes do problema) um algoritmo pior rodando em uma máquina rápida. Sempre."
>
> — S. S. Skiena, The Algorithm Design Manual

Exercícios

Exercício

Exercício: Proponha um limite superior para a função $f(n) = 3n^2 + 18$ juntamente com constantes c e n_0 válidas.

Exercício

Exercício: Proponha um limite superior para a função $f(n) = 3n^2 + 18$ juntamente com constantes c e n_0 válidas.

Solução: Como limite superior, propomos a função $g(n) = n^2$ e como constantes válidas citamos $c = 4$ e $n_0 = 5$.

Exercício

Exercício: Proponha um limite superior para a função $f(n) = 3n^2 + 18$ juntamente com constantes c e n_0 válidas.

Solução: Como limite superior, propomos a função $g(n) = n^2$ e como constantes válidas citamos $c = 4$ e $n_0 = 5$.

Vamos verificar essas constantes:

Exercício

Exercício: Proponha um limite superior para a função $f(n) = 3n^2 + 18$ juntamente com constantes c e n_0 válidas.

Solução: Como limite superior, propomos a função $g(n) = n^2$ e como constantes válidas citamos $c = 4$ e $n_0 = 5$.

Vamos verificar essas constantes:

$$
\begin{aligned}
c \cdot g(n) &\geq f(n) \\
4n^2 &\geq 3n^2 + 18 \\
n^2 &\geq 18 \Rightarrow \{n \leq \sqrt{18} \ \cup \ n \geq \sqrt{18}\}
\end{aligned}
$$

Exercício

Universidade Federal do Ceará
Campus Quixadá

Exercício: Proponha um limite superior para a função $f(n) = 3n^2 + 18$ juntamente com constantes c e n_0 válidas.

Solução: Como limite superior, propomos a função $g(n) = n^2$ e como constantes válidas citamos $c = 4$ e $n_0 = 5$.

Vamos verificar essas constantes:

$$
\begin{aligned}
c \cdot g(n) &\geq f(n) \\
4n^2 &\geq 3n^2 + 18 \\
n^2 &\geq 18 \Rightarrow \{n \leq \sqrt{18} \;\cup\; n \geq \sqrt{18}\}
\end{aligned}
$$

Como $c = 4$ e $n = 5 > 4.25 \approx \sqrt{18}$, então $3n^2 + 18 = O(n^2)$.

Exercício

Exercício: Proponha um limite inferior para a função $f(n) = n^2 - 3n$ juntamente com constantes c e n_0 válidas.

Exercício

Exercício: Proponha um limite inferior para a função $f(n) = n^2 - 3n$ juntamente com constantes c e n_0 válidas.

Solução: Como limite inferior, propomos a função $g(n) = n$ e como constantes válidas citamos $c = 1$ e $n_0 = 4$.

Exercício

Exercício: Proponha um limite inferior para a função $f(n) = n^2 - 3n$ juntamente com constantes c e n_0 válidas.

Solução: Como limite inferior, propomos a função $g(n) = n$ e como constantes válidas citamos $c = 1$ e $n_0 = 4$.

Vamos verificar essas constantes:

Exercício

Exercício: Proponha um limite inferior para a função $f(n) = n^2 - 3n$ juntamente com constantes c e n_0 válidas.

Solução: Como limite inferior, propomos a função $g(n) = n$ e como constantes válidas citamos $c = 1$ e $n_0 = 4$.

Vamos verificar essas constantes:

$$\begin{aligned} c \cdot g(n) &\leq f(n) \\ 1(n) &\leq n^2 - 3n \\ n^2 - 4n &\geq 0 \\ n(n-4) &\geq 0 \Rightarrow \{n \leq 0 \ \cup \ n \geq 4\} \end{aligned}$$

Exercício

Exercício: Proponha um limite inferior para a função $f(n) = n^2 - 3n$ juntamente com constantes c e n_0 válidas.

Solução: Como limite inferior, propomos a função $g(n) = n$ e como constantes válidas citamos $c = 1$ e $n_0 = 4$.

Vamos verificar essas constantes:

$$\begin{aligned} c \cdot g(n) &\leq f(n) \\ 1(n) &\leq n^2 - 3n \\ n^2 - 4n &\geq 0 \\ n(n-4) &\geq 0 \Rightarrow \{n \leq 0 \;\cup\; n \geq 4\} \end{aligned}$$

Como $c = 1$ e $n_0 = 4$, então $n^2 - 3n = \Omega(n)$.

Esse limite pode ser melhorado. Prove que $n^2 - 3n = \Omega(n^2)$.

Exercício

Exercício: Suponha $f(n) = 2n^2 + 30n + 400$ e $g(n) = n^2$. Mostre que $f = O(g)$.

Exercício

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Exercício: Suponha $f(n) = 2n^2 + 30n + 400$ e $g(n) = n^2$. Mostre que $f = O(g)$.

Solução: Para todo n positivo, temos:

$$\begin{aligned} f(n) &= 2n^2 + 30n + 400 \\ &\leq 2n^2 + 30n^2 + 400n^2 \\ &= 432n^2 \\ &= 432g(n). \end{aligned}$$

Resumindo, $f(n) \leq 432g(n)$ para todo $n \leq 1$. Além disso, note que $f(n)$ e $g(n)$ são assintoticamente não-negativas. Portanto, $f(n) = O(g(n))$.

Exercício

Exercício: Suponha $f(n) = \lceil n/2 \rceil + 10$ e $g(n) = n$. Mostre que $f(n) = O(g(n))$.

Exercício

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Exercício: Suponha $f(n) = \lceil n/2 \rceil + 10$ e $g(n) = n$. Mostre que $f(n) = O(g(n))$.

Solução: De fato, temos que:

$$\begin{aligned} f(n) &= \lceil n/2 \rceil + 10 \\ &\leq n/2 + 1 + 10 \\ &= n/2 + 11 \\ &\leq 20n \text{ para todo } n \geq 1. \end{aligned}$$

Portanto, $f(n) = O(g(n))$.

Exercício

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Exercício: Suponha $f(n) = 5n \lg n + 8 \lg^2 n - 11$ e $g(n) = n \lg n$. Mostre que $f(n) = O(g(n))$.

Exercício

Exercício: Suponha $f(n) = 5n \lg n + 8 \lg^2 n - 11$ e $g(n) = n \lg n$. Mostre que $f(n) = O(g(n))$.

Solução: Desta vez, vamos usar limites:

$$
\begin{aligned}
&\lim_{n\to\infty} \frac{5n \lg n + 8 \lg^2 n - 11}{n \lg n} \\
&= \lim_{n\to\infty} 5 + 8\frac{\lg n}{n} - 11\frac{1}{n \lg n} \\
&= 5 + 8(0) - 11(0) \\
&= 5.
\end{aligned}
$$

Logo, como o limte existe, então $f(n) = O(g(n))$.

Exercícios

1. É verdade que $\log_2 n = O(\log_3 n)$? É verdade que $\log_3 n = O(\log_2 n)$?

Exercícios

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

1. É verdade que $\log_2 n = O(\log_3 n)$? É verdade que $\log_3 n = O(\log_2 n)$?

2. Mostre que $15n = O(n \lg n)$ mas que $n \lg n \neq O(n)$

Exercícios

1. É verdade que $\log_2 n = O(\log_3 n)$? É verdade que $\log_3 n = O(\log_2 n)$?

2. Mostre que $15n = O(n \lg n)$ mas que $n \lg n \neq O(n)$
    - Essa análise é folgada, já que $15n = O(n)$

Exercícios

1. É verdade que $\log_2 n = O(\log_3 n)$? É verdade que $\log_3 n = O(\log_2 n)$?

2. Mostre que $15n = O(n \lg n)$ mas que $n \lg n \neq O(n)$
   - Essa análise é folgada, já que $15n = O(n)$

3. Mostre que $42n = O(n^2)$ mas que $n^2 \neq O(42n)$

Exercícios

1. É verdade que $\log_2 n = O(\log_3 n)$? É verdade que $\log_3 n = O(\log_2 n)$?

2. Mostre que $15n = O(n \lg n)$ mas que $n \lg n \neq O(n)$
   - Essa análise é folgada, já que $15n = O(n)$

3. Mostre que $42n = O(n^2)$ mas que $n^2 \neq O(42n)$
   - Essa análise é folgada, já que $42n = O(n)$

FIM