# Elementos de Análise Assintótica

Marcelo Keese Albertini
Faculdade de Computação
Universidade Federal de Uberlândia

23 de Março de 2018

# Aula de hoje

**Nesta aula veremos:**

- Elementos de Análise Assintótica
- Análise do algoritmo Insertion Sort

# Análise de custos

- Uso de algoritmo em um "computador" idealizado
  - Random Access Machine
- Acessa qualquer região da memória em tempo constante, operações aritméticas $+$,-,/,* são executadas em tempo constante, potência de 2 para exponentes pequenos também
- Não executa operações paralelamente, não tem threads

# Insertion sort: ordenação por inserção

```java
void insertionSort(int[] v) {

  int n = v.length, j = 1;

  for (j = 1; j < n; j++) {
    int c = v[j];
    int i = j - 1;

     // procura lugar de insercao e desloca números
    while (i >= 0 && v[i] > c) {
        v[i+1] = v[i];
        i = i - 1;
    }
    v[i+1] = c;
  }
}
```

# Algoritmo: ordenação por inserção

```
insertionSort(int[] v){

int n = v.length, j = 1;
 for (j = 1; j < n; j++) {
   int c = v[j];
   int i = j - 1;

   // desloca números
   while (i >=0 && v[i]>c){
     v[i+1] = v[i];
     i = i - 1;
   }

   //insere chave
   v[i+1] = c;
 }
}
```

*chave* = 5

| | | | | | |
|---|---|---|---|---|---|
| 6 | 5 | 2 | 1 | 9 | 6 |
| 6 | 5 | 2 | 1 | 9 | 6 |
| 6 | 6 | 2 | 1 | 9 | 6 |
| 5 | 6 | 2 | 1 | 9 | 6 |

*chave* = 6

| | | | | | |
|---|---|---|---|---|---|
| 5 | 6 | 2 | 1 | 9 | 6 |

*chave* = 2

| | | | | | |
|---|---|---|---|---|---|
| 5 | 6 | 2 | 1 | 9 | 6 |
| 5 | 6 | 6 | 1 | 9 | 6 |
| 5 | 5 | 6 | 1 | 9 | 6 |
| 2 | 5 | 6 | 1 | 9 | 6 |

*chave* = 1

| | | | | | |
|---|---|---|---|---|---|
| 2 | 5 | 6 | 1 | 9 | 6 |
| 2 | 5 | 6 | 6 | 9 | 6 |
| 2 | 5 | 5 | 6 | 9 | 6 |
| 2 | 2 | 5 | 6 | 9 | 6 |
| 1 | 2 | 5 | 6 | 9 | 6 |

*chave* = 9

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 5 | 6 | 9 | 6 |

*chave* = 6

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 5 | 6 | 9 | 6 |
| 1 | 2 | 5 | 6 | 9 | 9 |
| 1 | 2 | 5 | 6 | 6 | 9 |

# Insertion Sort: análise de tempo

http://www.facom.ufu.br/~albertini/ada/ExemploFor.java

| Código | Custo | Repetições |
|---|---|---|
| `insertionSort(int[] v){` | | |
| `  int n = v.length, j = 1; {` | $c_1$ | 1 |
| `  for (j = 1;` | $c_2$ | 1 atribuição |
| `       j < n;` | $c_3$ | $n$ testes |
| `       j++) {` | $c_4$ | $n-1$ incrementos |
| `    int c = v[j];` | $c_5$ | $n-1$ |
| `    int i = j - 1;` | $c_6$ | $n-1$ |
| | | |
| `    // desloca números` | | $t_j$ é núm. iterações do while |
| `    while (i >=0 && v[i]>c){` | $c_7$ | $\sum_{j=1}^{n-1} t_j$ |
| `      v[i+1] = v[i];` | $c_8$ | $\sum_{j=1}^{n-1}(t_j-1)$ |
| `      i = i - 1;` | $c_9$ | $\sum_{j=1}^{n-1}(t_j-1)$ |
| `    }` | | |
| `    v[i+1] = c;` | $c_{10}$ | $n-1$ |
| `  }` | | |
| `}` | | |

# Insertion Sort: custo total

$$
\begin{aligned}
T(n) = {} & c_1 + c_2 + nc_3 + (n-1)c_4 + (n-1)c_5 + c_6(n-1) + \\
& c_7 \sum_{j=1}^{n-1} t_j + c_8 \sum_{j=1}^{n-1} (t_j - 1) + c_9 \sum_{j=1}^{n-1} (t_j - 1) + c_{10}(n-1)
\end{aligned}
$$

- Depende do número $t_j$ de iterações do while para cada valor de $j$.

# Insertion Sort: custos iguais

- Suposição simplificadora: custos $c_1 \approx c_2 \approx \ldots \approx c$

$$T(n) = 5nc - 2c + c\sum_{j=1}^{n-1} t_j + 2c\sum_{j=1}^{n-1}(t_j - 1)$$

$$T(n) = 5nc - 2c + c\sum_{j=1}^{n-1} t_j - 2c(n-1) + 2c\sum_{j=1}^{n-1} t_j$$

$$T(n) = 5nc + c\sum_{j=1}^{n-1} t_j - (n) + 2c\sum_{j=1}^{n-1} t_j$$

$$T(n) = 5c(n-1) + 3c\sum_{j=1}^{n-1} t_j$$

# Insertion Sort: melhor caso

$$T(n) = 5c(n-1) + 3c\sum_{j=1}^{n-1} t_j$$

- Tempo de execução depende do número de iterações do while
- Se vetor estiver previamente ordenado, para qualquer $j$, $t_j = 1$, temos o melhor caso.

**Melhor caso**

$$T(n) = 5c(n-1) + 3c(n-1) = 8c(n-1)$$

# Insertion Sort: pior caso

$$T(n) = 5c(n-1) + 3c\sum_{j=1}^{n-1} t_j$$

- Qual é o maior valor de $t_j$: $\max(t_j)$=? Quando $\max(t_j) = j$ ?

$$T(n) = 5c(n-1) + 3c\sum_{j=1}^{n-1} j =$$

$$5c(n-1) + 3c(1+2+3+\cdots+(n-3)+(n-2)+(n-1)) =$$

$$5c(n-1) + 3c\frac{(n-1)}{2}n = 5c(n-1) - 3cn/2 + 3cn^2/2$$

**Pior caso**

$$T(n) = 3.5c(n-1) + 1.5cn^2$$

# Análise: Insertion Sort

**Aspectos**

- Corretude
- Melhor caso, limites $\Omega$ e $O$
- Pior caso, limites $\Omega$ e $O$
- Caso médio

# Insertion Sort: análise de corretude

- Invariante: subvetor atual está ordenado.
- A cada iteração: inserir elemento no subvetor e mantê-lo ordenado
- Início de cada iteração: subvetor com $k$ elementos está ordenado
- Manutenção do invariante: um elemento novo é inserido e todo elemento maior que o elemento sendo inserido deve ser deslocado
- Depois de uma iteração: subvetor com $k+1$ elementos está ordenado
- Depois de todas iterações: (sub)vetor com $k = n$ elementos está ordenado

# Complexidade de tempo

### Custo do algoritmo de multiplicação de números

A função que descreve o custo do pior caso do nosso algoritmo é $T(n) = 3.5c(n-1) + 1.5cn^2$.

### Complexidade de tempo

Para descrever a complexidade usamos somente o custo dominante: $1.5cn^2$.

### Porquê?

Existe um número $n = a$ a partir do qual o custo $1.5ca^2$ é sempre superior a $3.5c(a-1)$, tornando-o pouco importante.

### Ordem de crescimento

Dizemos que a complexidade *big-Oh* do nosso algoritmo é $O(n^2)$.

# Exemplos: ordem de crescimento

## Exemplo 1

- $f(n) = n^2 + n + c$
- $f(n) = O(n^2)$
- $n^2$ domina assintoticamente os outros termos

## Exemplo 2

- $f(n) = n + \log(n) + c$
- $f(n) = O(n)$
- $n$ domina assintoticamente os outros termos

## Exemplo 3

- $f(n) = n! + n^2 + c$
- $f(n) = O(n!)$
- $n!$ domina assintoticamente os outros termos

# Limitante assintótico superior: notação “big O” $O(\cdot)$

## Limitante superior: notação *big O*

$f(n) \in O(g(n))$ representa conjunto de funções positivas $f(n)$ tal que “$f(n) \leq cg(n)$”, para $n > N$, $n \to \infty$ e uma constante $c > 0$

- $n \in O(n)$
- $n = O(n)$
- $n = O(n^2)$
- $\log n = O(n^2)$
- $\log_2 n = O(\log n)$
- $1 \in O(n)$
- $5 \in O(1)$
- $n^3 \notin O(n^2)$
- $O(n \log n) = O(\log n!)$
- $O(n) = O(n^2)$
- $O(n^2) \neq O(n)$

# Limitante assintótico inferior: notação *big-omega* de Knuth $\Omega(\cdot)$

$f(n) \in \Omega(g(n))$ representa "$f(n) \geq cg(n)$"

- $1 \in \Omega(1)$
- $n = \Omega(n)$
- $n = \Omega(1)$
- $n^2 = \Omega(n)$
- $n! = \Omega(2^n)$
- $\log n = \Omega(\log \log n)$
- $\sqrt{n} = \Omega(\log n)$
- $n^2 = \Omega(n \log n)$
- $n^{1.5} = \Omega(n)$
- $n^n \in \Omega(n!)$

# Limitante estrito: notação *big-theta* $\Theta(\cdot)$

Se e somente se $f(n) \in \Theta(g(n))$ então "$f(n) \in \Omega(g(n))$" e "$f(n) \in O(g(n))$"

- $n = \Theta(n)$
- $n^2 - 2n \in \Theta(n^2)$

# Limitante estritamente superior: notação little-o $o(\cdot)$

$f(n) \in o(g(n))$ representa "$f(n) < cg(n)$":

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

- $n \neq o(n)$
- $n = o(n^2)$
- $1/n = o(1)$

# Limitante estritamente inferior: notação little-omega $\omega(\cdot)$

$f(n) \in \omega(g(n))$ representa "$f(n) > cg(n)$":

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

- $n = \omega(1)$
- $n \notin \omega(n)$

# Limitante estrito: notação “ordem de” $\sim$

$f(n) \sim g(n)$ representa $g(n) = f(n) + o(f(n))$, ou seja, que $f(n)$ e $g(n)$ são assintóticamente iguais:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

- $n \nsim n^2$
- $2n^2 \nsim 10n^2$
- $n^2 \sim n^2 + n^{1.5}$
- $\log n! \sim n \log n$

# Caso médio

## Caso médio ou esperado

O conjunto de entradas $n$ é descrito por uma distribuição de probabilidades.
Qual é a função de custo, considerando a probabilidade de cada entrada de tamanho $n$?

# Classes de complexidade

$$O(1) \in O(\log(n)) \in O(n) \in O(n \log(n)) \in O(n^2) \in O(n^3) \in O(2^n)$$

- $O(1)$: ordem constante
- $O(\log(n))$: ordem logaritmica
- $O(n)$: ordem linear
- $O(n \log(n))$: ordem típica de soluções divisão-e-conquista
- $O(n^2)$: ordem quadrática. "um laço dentro de outro"
- $O(2^n)$: ordem exponencial. Impraticável.
- $O(n!)$: ordem fatorial. Pior que exponencial.

# Teoria de Complexidade vs. Análise de Complexidade

- Desprezar constantes
  - $f(n) = 100000000 * n + 999999999$
  - $f(n) = O(n)$
- Teoria: desprezar termos de menor grandeza para sermos concisos
- Análise: estimar termos para podermos fazer predições

Complexidade assintótica menor não indica que o algoritmo sempre é o melhor. Se as constantes forem significativas, na prática é melhor considerá-las.

# Insertion Sort: ordens assintóticas

**Melhor caso:** $T(n) = O(n)$, $T(n) = \Omega(n)$, $T(n) = \Theta(n)$, $T(n) = o(n^2)$

$$T(n) = 8c(n-1)$$

**Pior caso:** $T(n) = O(n^2)$, $T(n) = \Omega(n^2)$, $T(n) = \Theta(n^2)$

$$T(n) = 3.5c(n-1) + 1.5cn^2$$

# Insertion Sort: caso médio

- Análise empírica
- Análise matemática: necessário modelo realístico
  - Assumimos que $n$ números são sorteados de uma distribuição uniformemente aleatória de arranjos

# Insertion Sort: análise empírica

- Geração de permutações aleatórias
- Visualização de algoritmos para aleatorização: `https://bost.ocks.org/mike/shuffle/compare.html`
- Testar com: `http://www.facom.ufu.br/~albertini/ada/permutacoesAleatorias.R`

```java
static void troca(Item[] a, int i, int j) {
    Item t = a[i]; a[i] = a[j]; a[j] = t;
  }
//Knuth-Fisher-Yates
static void aleatorizar(Item[] a, int N) {
  for (int i = 1; i < N; i++) {
    // r = (int) [0,1) * (i+1)
    int r = (int) Math.random()*(i+1);
    troca(a, i, r);
  }
}
```

# Permutações aleatórias: WARNING

- Caso real: [link]

```
// NÃO USAR -- não produz distribuição uniforme
for (int i = 0; i < N; i++) {
  troca(cards, i, r.nextInt(N));
}
```

- Nesse algoritmo existem $N \times N \times \ldots N$ possibilidades de geração de permutações
- Mas só existe $N \times (N-1) \times (N-2) \times \ldots 1 = N!$ permutações distintas
- Algumas permutações terão maior probabilidade de serem sorteadas que outras

# Permutações aleatórias: WARNING

- Caso real: [link 1, link 2 ]

```
// NÃO USAR -- não produz distribuição uniforme
function RandomSort (a,b)
{
    return (0.5 - Math.random());
}
```

# Insertion Sort: análise empírica

- Exemplo [link: InsertionSort.java]
- Gerar permutações aleatórias
- Aumentar $N$ de dobro em dobro
- Estimar coeficientes com regressão linear em escala log-log

$$
\begin{aligned}
T_n &= an^b + c \\
\log_2(T_n) &= \log_2(an^b + c) \\
\log_2 T_n &= b \log_2 n + d
\end{aligned}
$$

- Sendo $d = \log_2(an^b + c) - \log_2 n$ uma constante

# Insertion Sort: análise empírica

- `java InsertionSort 10000000 10 > tempos.dat`
- `tempos.dat`:

```
"N" "tN"
1000 0.7999999999999999
2000 0.5
4000 1.7000000000000002
8000 5.8999999999999999
16000 22.6
32000 90.8
64000 362.9000000000003
128000 1455.9999999999998
256000 5875.099999999999
512000 23455.4
```

# Insertion Sort: análise empírica

- Obter $b$ em $\log_2 T_n = b \log_2 n + d$. Exemplo em gnuplot:

```
plot "tempos.dat" using log(1):log(2) w lp
f(x) = b*log(x)/log(2) + d
fit f(x) "tempos.dat" via b,d
```

- Obter coeficientes de equação quadrática $T_n = an^2 + c$. Exemplo em R:

```
d <- read.delim("tempos.dat", sep = " ");
lm(formula = tN ~ N + I(N^2), data = d)

Call:
lm(formula = tN ~ N + I(N^2), data = d)

Coefficients:
(Intercept)            N         I(N^2)
  4.558e-01    -2.505e-05     8.921e-08
```

# Análise empírica: caso médio

- Executar $K$ vezes o algoritmo com entrada de tamanho $N$
- Computar Média $\bar{x}$ com:

$$\bar{x} = \sum_{k=1}^{K} \left( \frac{x_k}{K} \right)$$

- Computar Desvio Padrão $\sigma$ com:
  - Para $j = 1, 2$ obter

$$s_j = \sum_{k=1}^{K} x_k^j$$

  -

$$\sigma = \frac{\sqrt{K s_2 - s_1^2}}{K - 1}$$

# Insertion Sort: análise matemática

- Tabela de inversões: número de inversões proporcional ao número de cópias do Insertion Sort

| JC | PD | CN | AA | MC | AB | JG | MS | EF | HF | JL | AL | JB | PL | HT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 14 | 4 | 1 | 12 | 2 | 10 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 9 | 14 | 4 | 1 | 12 | 2 | 10 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 4 | 9 | 14 | 1 | 12 | 2 | 10 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 4 | 9 | 14 | 12 | 2 | 10 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 4 | 9 | 12 | 14 | 2 | 10 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 2 | 4 | 9 | 12 | 14 | 10 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 2 | 4 | 9 | 10 | 12 | 14 | 13 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 2 | 4 | 9 | 10 | 12 | 13 | 14 | 5 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 2 | 4 | 5 | 9 | 10 | 12 | 13 | 14 | 6 | 11 | 3 | 8 | 15 | 7 |
| 1 | 2 | 4 | 5 | 6 | 9 | 10 | 12 | 13 | 14 | 11 | 3 | 8 | 15 | 7 |
| 1 | 2 | 4 | 5 | 6 | 9 | 10 | 11 | 12 | 13 | 14 | 3 | 8 | 15 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 9 | 10 | 11 | 12 | 13 | 14 | 8 | 15 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| AA | AB | AL | CN | EF | HF | HT | JB | JC | JG | JL | MC | MS | PD | PL |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 3 | 1 | 4 | 2 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 2 | 3 | 1 | 4 | 2 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 3 | 1 | 4 | 2 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 1 | 4 | 2 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 4 | 2 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figura: Tabelas de inversões: AdA 3ed. pág. 385

# Insertion Sort: análise matemática de caso médio

AoA3d Teorema 7.8 (Distribuição de Inversões)

$$[u^k] \prod_{1 \le k \le N} \frac{1-u^k}{1-u} = [u^k](1+u)(1+u+u^2)\ldots(1+u+\cdots+u^{N-1})$$

- Uma permutação de $N$ itens tem $N(N-1)/4$ inversões em média

  e desvio padrão de $\sqrt{N(N-1)(2N+5)/72} = o(N^2)$