Melhores momentos

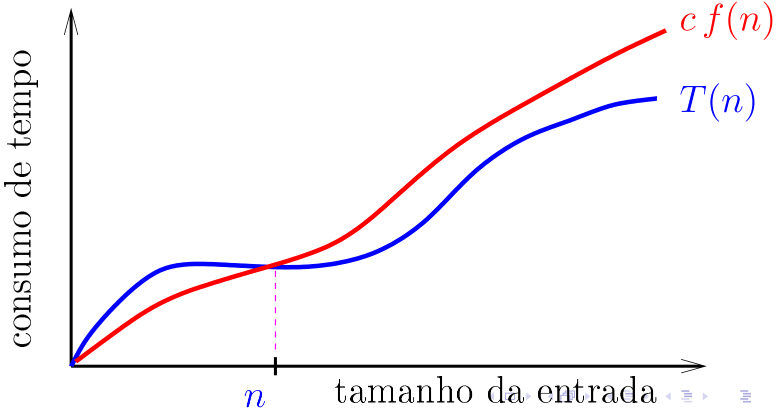
AULA 17

Notação assintótica (informal)

$\mathtt{T}(\mathtt{n})$ é $\mathrm{O}(\mathtt{f}(\mathtt{n}))$ se existe $\mathtt{c} > 0$ tal que

$$\mathtt{T}(\mathtt{n}) \ \leq \ \mathtt{c}\,\mathtt{f}(\mathtt{n})$$

para todo $\mathtt{n}$ suficientemente GRANDE.



AULA 18

Ordenação

Fonte: `http://xkcd.com/1185/`

PF 8

http://www.ime.usp.br/~pf/algoritmos/aulas/ordena.html

Ordenação

$v[0\,..\,n{-}1]$ é crescente se $v[0] \leq \cdots \leq v[n{-}1]$.

Problema: Rearranjar um vetor $v[0\,..\,n{-}1]$ de modo que ele fique crescente.

Entra:

1										$n{-}1$
33	55	33	44	33	22	11	99	22	55	77

Ordenação

$\mathtt{v}[0\,..\,\mathtt{n}{-}1]$ é crescente se $\mathtt{v}[0] \leq \cdots \leq \mathtt{v}[\mathtt{n}{-}1]$.

Problema: Rearranjar um vetor $\mathtt{v}[0\,..\,\mathtt{n}{-}1]$ de modo que ele fique crescente.

Entra:

1										$\mathtt{n}{-}1$
33	55	33	44	33	22	11	99	22	55	77

Sai:

0										$\mathtt{n}{-}1$
11	22	22	33	33	33	44	55	55	77	99

Ordenação por inserção (iteração)

$x = 38$

0						i				n−1
20	25	35	40	44	55	38	99	10	65	50

Ordenação por inserção (iteração)

$\mathtt{x} = 38$

0					j	i				n−1
20	25	35	40	44	55	38	99	10	65	50

Ordenação por inserção (iteração)

$x = 38$

0					j	i				n−1
20	25	35	40	44	55	38	99	10	65	50

0				j		i				n−1
20	25	35	40	44		55	99	10	65	50

Ordenação por inserção (iteração)

x = 38

0					j	i				n−1
20	25	35	40	44	55	38	99	10	65	50

0				j		i				n−1
20	25	35	40	44		55	99	10	65	50

0			j			i				n−1
20	25	35	40		44	55	99	10	65	50

Ordenação por inserção (iteração)

$x = 38$

0					j	i				n−1
20	25	35	40	44	55	38	99	10	65	50

0				j		i				n−1
20	25	35	40	44		55	99	10	65	50

0			j			i				n−1
20	25	35	40		44	55	99	10	65	50

0		j				i				n−1
20	25	35		40	44	55	99	10	65	50

Ordenação por inserção (iteração)

$x = 38$

0					j	i				n−1
20	25	35	40	44	55	38	99	10	65	50

0				j		i				n−1
20	25	35	40	44		55	99	10	65	50

0			j			i				n−1
20	25	35	40		44	55	99	10	65	50

0		j				i				n−1
20	25	35		40	44	55	99	10	65	50

0		j				i				n−1
20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

x	0							i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

x	0						i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

x	0							i		n−1
10	20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

x	0							i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

x	0								i		n−1
10	10	20	25	35	38	40	44	55	99	65	50

Ordenação por inserção

x	0						i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

x	0							i		n−1
10	10	20	25	35	38	40	44	55	99	65	50

x	0								i	n−1
65	10	20	25	35	38	40	44	55	99	65	50

Ordenação por inserção

x	0							i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

x	0								i		n−1
10	10	20	25	35	38	40	44	55	99	65	50

x	0									i	n−1
65	10	20	25	35	38	40	44	55	65	99	50

Ordenação por inserção

x	0						i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

x	0							i		n−1
10	10	20	25	35	38	40	44	55	99	65	50

x	0								i	n−1
65	10	20	25	35	38	40	44	55	65	99	50

x	0									i
50	10	20	25	35	38	40	44	55	65	99	50

Ordenação por inserção

x	0							i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

x	0								i		n−1
10	10	20	25	35	38	40	44	55	99	65	50

x	0									i	n−1
65	10	20	25	35	38	40	44	55	65	99	50

x	0										i
50	10	20	25	35	38	40	44	50	55	65	99

insercao

Função rearranja $v[0\,..\,n-1]$ em ordem crescente.

```
void insercao (int n, int v[])
{
   int i, j, x;
   for (i = 1; /*A*/ i < n; i++){
      x = v[i];
      for (j=i-1; j >= 0 && v[j] > x;j--)
         v[j+1] = v[j];
      v[j+1] = x;
   }
}
```

O algoritmo faz o que promete?

Relação **invariante** chave:

♡ (i0) Em /*A*/ vale que: $v[0\,..\,i-1]$ é crescente.

0						i				n−1
20	25	35	40	44	55	38	99	10	65	50

O algoritmo faz o que promete?

Relação **invariante** chave:

♡ (i0) Em `/*A*/` vale que: $v[0 \,.\,.\, i-1]$ é crescente.

0						i				n−1
20	25	35	40	44	55	38	99	10	65	50

Supondo que a invariante vale.
Correção do algoritmo é evidente.

No início da última iteração tem-se que $i = n$.
Da invariante conclui-se que $v[0 \,.\,.\, n-1]$ é crescente.

Mais invariantes

Na linha 3, antes de "`j >= 0...`", vale que:

(i1) $v[0\,..\,j]$ e $v[j{+}2\,..\,i]$ são crescentes

(i2) $v[0\,..\,j] \leq v[j{+}2\,..\,i]$

(i3) $v[j{+}2\,..\,i] > x$

x	0		j				i				n−1
38	20	25	35		40	44	55	99	10	65	50

Mais invariantes

Na linha 3, antes de "`j >= 0...`", vale que:

(i1) $\mathtt{v}[0\,..\,\mathtt{j}]$ e $\mathtt{v}[\mathtt{j}{+}2\,..\,\mathtt{i}]$ são crescentes

(i2) $\mathtt{v}[0\,..\,\mathtt{j}] \leq \mathtt{v}[\mathtt{j}{+}2\,..\,\mathtt{i}]$

(i3) $\mathtt{v}[\mathtt{j}{+}2\,..\,\mathtt{i}] > \mathtt{x}$

x	0		j				i				n−1
38	20	25	35		40	44	55	99	10	65	50

invariantes (i1),(i2) e (i3)
+ condição de parada do `for` da linha 3
+ atribuição da linha 5 $\Rightarrow$ validade (i0)

Verifique!

Correção de algoritmos iterativos

Estrutura "típica" de demonstrações da correção de algoritmos iterativos através de suas relações invariantes consiste em:

1. verificar que a relação vale no início da primeira iteração;
2. demonstrar que

   se a relação vale no início da iteração,
   então ela vale no final da iteração
   (com os papéis de alguns atores
   possivelmente trocados);

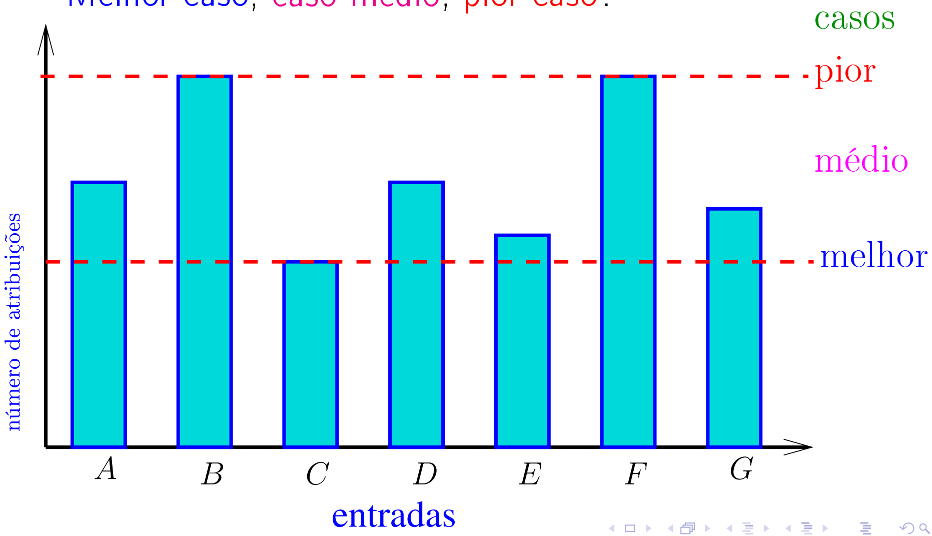
3. concluir que, se relação vale no início da última iteração, então a a relação junto com a condição de parada implicam na correção do algoritmo.

Quantas atribuições faz a função?

Quantas atribuições faz a função?

Número mínimo, médio ou máximo?

Melhor caso, caso médio, pior caso?



Quantas atribuições faz a função?

LINHAS 2-4 (v, i, x)

```
2       x = v[i];
3       for (j=i-1; j >= 0 && v[j] > x;j--)
4           v[j+1] = v[j];
```

Quantas atribuições faz a função?

LINHAS 2-4 (v, i, x)

```
2        x = v[i];
3        for (j=i-1; j >= 0 && v[j] > x;j--)
4            v[j+1] = v[j];
```

linha	atribuições (número máximo)
2	?
3	?
4	?
total	?

Quantas atribuições faz a função?

LINHAS 2-4 (v, i, x)

```
2    x = v[i];
3    for (j=i-1; j >= 0 && v[j] > x;j--)
4        v[j+1] = v[j];
```

linha	atribuições (número máximo)
2	$= 1$
3	$\leq 1 + i$
4	?

total ?

Quantas atribuições faz a função?

LINHAS 2-4 $(\mathtt{v}, \mathtt{i}, \mathtt{x})$

```
2       x = v[i];
3       for (j=i-1; j >= 0 && v[j] > x;j--)
4           v[j+1] = v[j];
```

linha	atribuições (número máximo)
2	$= 1$
3	$\leq 1 + \mathtt{i}$
4	$\leq \mathtt{i} - 1$

total $\leq 2\mathtt{i} + 1 \leq 2\mathtt{n}$

Quantas atribuições faz a função?

```
void insercao (int n, int v[]) {
    int i, j, x;
1   for (i = 1; /*A*/ i < n; i++){
2       LINHAS 2-4 (v, i, x)
5       v[j+1] = x;
    }
}
```

linha	atribuições (número máximo)
1	?
2–4	?
5	?

total ?

Quantas atribuições faz a função?

```
void insercao (int n, int v[]) {
    int i, j, x;
1   for (i = 1; /*A*/ i < n; i++){
2       LINHAS 2-4 (v, i, x)
5       v[j+1] = x;
    }
}
```

linha	atribuições (número máximo)
1	$= n$
2–4	$\leq (n-1)2n$
5	$= n-1$

total $\leq 2n^2 - 1$

Análise mais fina

linha	atribuições (número máximo)
1	?
2	?
3	?
4	?
5	?
total	?

Análise mais fina

linha	atribuições (número máximo)
1	$= \mathtt{n}$
2	$= \mathtt{n} - 1$
3	$\leq 1 + 2 + \cdots + \mathtt{n} = \mathtt{n}(\mathtt{n}+1)/2$
4	$\leq 1 + 2 + \cdots + (\mathtt{n}-1) = (\mathtt{n}-1)\mathtt{n}/2$
5	$= \mathtt{n} - 1$

total $\leq \mathtt{n}^2 + 3\mathtt{n} - 2$

$\mathtt{n}^2 + 3\mathtt{n} - 2$ versus $\mathtt{n}^2$

$\mathtt{n}$	$\mathtt{n}^2 + 3\mathtt{n} - 2$	$\mathtt{n}^2$
1	2	1
2	8	4

$\mathtt{n}^2 + 3\mathtt{n} - 2$ versus $\mathtt{n}^2$

$\mathtt{n}$	$\mathtt{n}^2 + 3\mathtt{n} - 2$	$\mathtt{n}^2$
1	2	1
2	8	4
3	16	9
10	128	100

$\mathtt{n}^2 + 3\mathtt{n} - 2$ versus $\mathtt{n}^2$

$\mathtt{n}$	$\mathtt{n}^2 + 3\mathtt{n} - 2$	$\mathtt{n}^2$
1	2	1
2	8	4
3	16	9
10	128	100
100	10298	10000
1000	1002998	1000000

$n^2 + 3n - 2$ versus n^2

n	$n^2 + 3n - 2$	n^2
1	2	1
2	8	4
3	16	9
10	128	100
100	10298	10000
1000	1002998	1000000
10000	100029998	100000000
100000	10000299998	10000000000

n^2 domina os outros termos

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo, qual o consumo total?

```
void insercao (int n, int v[])
{
   int i, j, x;
   for (i = 1; /*A*/ i < n; i++){
      x = v[i];
      for (j= i-1; j>= 0 && v[j] > x;j--)
         v[j+1] = v[j];
      v[j+1] = x;
   }
}
```

Consumo de tempo no pior caso

linha	todas as execuções da linha
1	$= \quad n$
2	$= \quad n - 1$
3	$\leq \quad 2 + 3 + \cdots + n \; = \; (n-1)(n+2)/2$
4	$\leq \quad 1 + 2 + \cdots + (n-1) \; = \; n(n-1)/2$
5	$= \quad n - 1$
total	$\leq \quad (3/2)n^2 + (7/2)n - 4 = O(n^2)$

Consumo de tempo no melhor caso

linha	todas as execuções da linha	
1	$=$	n
2	$=$	$n - 1$
3	$=$	$n - 1$
4	$=$	0
5	$=$	$n - 1$
total	$\leq$	$4n - 3 = O(n)$

Pior e melhor casos

O maior consumo de tempo da função `insercao` ocorre quando o vetor `v[0 . . n-1]` dado é decrescente. Este é o pior caso para a função `insercao`.

O menor consumo de tempo da função `insercao` ocorre quando o vetor `v[0 . . n-1]` dado é já é crescente. Este é o melhor caso para a função `insercao`.

Conclusão

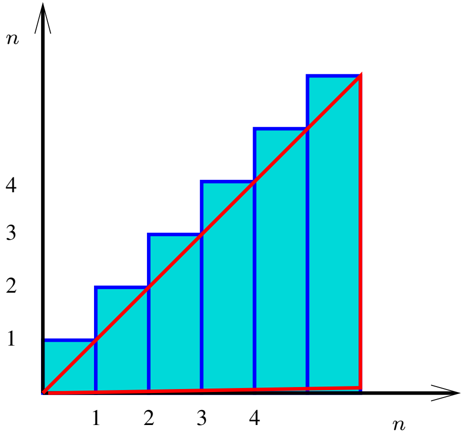
O consumo de tempo da função `insercao` no pior caso é proporcional a $\mathtt{n}^2$.

O consumo de tempo da função `insercao` melhor caso é proporcional a $\mathtt{n}$.

O consumo de tempo da função `insercao` é $O(\mathtt{n}^2)$.

$$1+2+\cdots+(n-1)+n=?$$

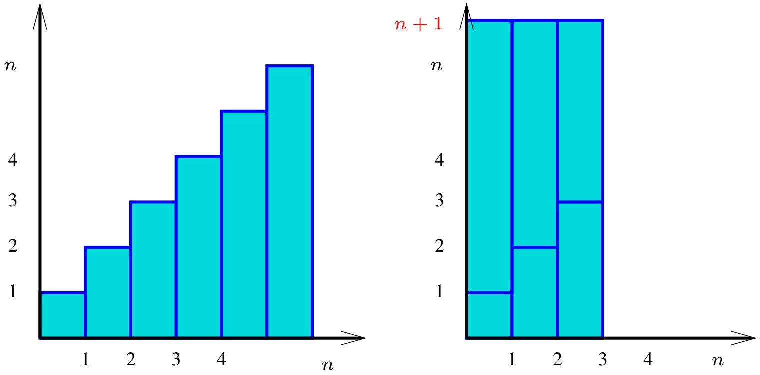
Carl Friedrich Gauss, 1787



$$\frac{n^2}{2}+\frac{n}{2}=\frac{n(n+1)}{2}$$

$$1 + 2 + \cdots + (\mathtt{n} - 1) + \mathtt{n} = ?$$

Carl Friedrich Gauss, 1787



$$(\mathtt{n} + 1) \times \frac{\mathtt{n}}{2} = \frac{\mathtt{n}(\mathtt{n} + 1)}{2}$$

Ordenação por inserção binária

Fonte: http://www.php5dp.com/

PF 7.3, 8.1 e 8.2

http://www.ime.usp.br/~pf/algoritmos/aulas/ordena.html

Busca binária

Esta função recebe um vetor crescente $v[0 \mathinner{.\,.\,.} n{-}1]$ com $n \geq 1$ e um inteiro x e devolve um índice j em $0 \mathinner{.\,.} n{-}1$ tal que $v[j] \leq x < v[j{+}1]$

```
int buscaBinaria (int x, int n, int v[]) {
    int e, m, d;
    e = -1; d = n;
    while (/*A*/e < d-1) {
        m = (e + d)/2;
        if (v[m] <= x) e = m;
        else d = m;
    }
    return e;
}
```

Relações invariantes

A relação invariante chave da função `buscaBinaria` é

(i0) Em /*A*/ vale que $\texttt{v}[\texttt{e}] \leq \texttt{x} < \texttt{v}[\texttt{d}]$

A correção da função segue facilmente dessa relação e da condição de parada do `while`.

Busca binária: recordação

O consumo de tempo da função `buscaBinaria` é proporcional a $\lg \mathtt{n}$.

O consumo de tempo da função `buscaBinaria` é $O(\lg \mathtt{n})$.

inserca o

Função rearranja $v[0\,.\,.\,n{-}1]$ em ordem crescente.

```
void insercao (int n, int v[])
{
    int i, j, x;
    for (i = 1; /*A*/ i < n; i++){
        x = v[i];
        for (j=i-1; j >= 0 && v[j] > x;j--)
            v[j+1] = v[j];
        v[j+1] = x;
    }
}
```

insercaoBinaria

Função rearranja $v[0\,..\,n-1]$ em ordem crescente.

```
void insercaoBinaria (int n, int v[])
{
    int i, j, k, x;
    for (i = 1; /*A*/ i < n; i++){
        x = v[i];
        j = buscaBinaria(x,i,v);
        for (k = i; k > j+1; k--)
            v[k] = v[k-1];
        v[j+1] = x;
    }
}
```

Pior e melhor casos

O maior consumo de tempo da função `insercaoBinaria` ocorre quando o vetor `v[0 . . n-1]` dado é decrescente. Este é o pior caso para a função `insercaoBinaria`.

O menor consumo de tempo da função `insercaoBinaria` ocorre quando o vetor `v[0 . . n-1]` dado é já é crescente. Este é o melhor caso para a função `insercaoBinaria`.

Consumo de tempo no pior caso

linha	consumo de tempo (proporcional a)
1	$= \mathtt{n}$
2	$= \mathtt{n}$
3	$= \lg 1 + \lg 2 + \cdots + \lg \mathtt{n} \leq \mathtt{n} \lg \mathtt{n}$
4	$\leq 2 + 2 + \cdots + \mathtt{n} = (\mathtt{n} - 1)(\mathtt{n} + 2)/2$
5	$\leq 1 + 2 + \cdots + (\mathtt{n}-1) = \mathtt{n}(\mathtt{n} - 1)/2$
6	$= \mathtt{n}$

total $\leq \mathtt{n}^2 + \mathtt{n} \lg \mathtt{n} + 3\mathtt{n} - 1 = \mathrm{O}(\mathtt{n}^2)$

Consumo de tempo no melhor caso

linha	consumo de tempo (proporcional a)
1	$= n$
2	$= n$
3	$= \lg 1 + \lg 2 + \cdots + \lg n \leq n \lg n$
4	$= 1 + 1 + \cdots + 1 = n$
5	$= 0$
6	$= n$
total	$= n \lg n + 4n = O(n \lg n)$

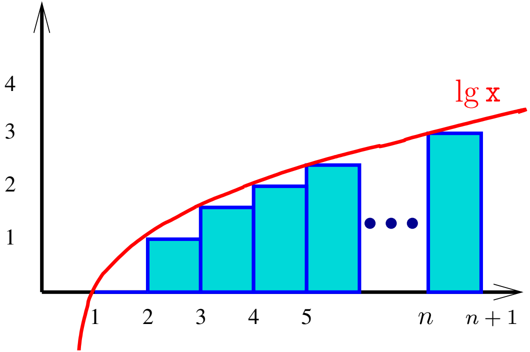
Conclusões

O consumo de tempo da função `insercaoBinaria` no pior caso é proporcional a n^2.

O consumo de tempo da função `insercaoBinaria` no melhor caso é proporcional a $n \lg n$.

O consumo de tempo da função `insercaoBinaria` é $O(n^2)$.

$$\lg 1 + \lg 2 + \cdots + \lg n = O(n \lg n)$$



$$\lg 1 + \lg 2 + \cdots + \lg n \leq \int_1^{n+1} \lg x \, dx$$

$$\lg \mathtt{1} + \lg \mathtt{2} + \cdots + \lg \mathtt{n} = O(\mathtt{n} \lg \mathtt{n})$$

$$\begin{aligned}
\int_1^{\mathtt{n}+1} \lg \mathtt{x}\, d\mathtt{x} &= \Big(\int_1^{\mathtt{n}+1} \ln \mathtt{x}\, d\mathtt{x}\Big) / \ln 2 \\
&= (\mathtt{x} \ln \mathtt{x} - \mathtt{x}]_0^{\mathtt{n}+1}) / \ln 2 \\
&= ((\mathtt{n}+1)\ln(\mathtt{n}+1) - (\mathtt{n}+1)) / \ln 2 \\
&= (\mathtt{n}+1)\lg(\mathtt{n}+1) - (\mathtt{n}+1) / \ln 2 \\
&< (\mathtt{n}+1)\lg(\mathtt{n}+1) \\
&= O(\mathtt{n} \lg \mathtt{n})
\end{aligned}$$

Ordenação por seleção

Fonte: http://www.exacttarget.com/

PF 8.3

http://www.ime.usp.br/~pf/algoritmos/aulas/ordena.html

Ordenação

$\mathtt{v}[0\,..\,\mathtt{n}-1]$ é crescente se $\mathtt{v}[0] \leq \cdots \leq \mathtt{v}[\mathtt{n}-1]$.

Problema: Rearranjar um vetor $\mathtt{v}[0\,..\,\mathtt{n}-1]$ de modo que ele fique crescente.

Entra:

0										$\mathtt{n}-1$
33	55	33	44	33	22	11	99	22	55	77

Sai:

0										$\mathtt{n}-1$
11	22	22	33	33	33	44	55	55	77	99

Ordenação por seleção (iteração)

$\mathtt{i} = 5$

0				max						n−1
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção (iteração)

i = 5

0			j	max						n−1
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção (iteração)

$\mathtt{i} = 5$

0			j	max						n−1
38	50	20	44	10	50	55	60	75	85	99

0		j	max							n−1
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção (iteração)

$\texttt{i} = 5$

0			j	max						n−1
38	50	20	44	10	50	55	60	75	85	99

0		j	max							n−1
38	50	20	44	10	50	55	60	75	85	99

0	j		max							n−1
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção (iteração)

$\mathtt{i} = 5$

0			j	max						n−1
38	50	20	44	10	50	55	60	75	85	99

0		j	max							n−1
38	50	20	44	10	50	55	60	75	85	99

0	j		max							n−1
38	50	20	44	10	50	55	60	75	85	99

j	max									n−1
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção (iteração)

$\texttt{i} = 5$

0			j	max						n−1
38	50	20	44	10	50	55	60	75	85	99

0		j	max							n−1
38	50	20	44	10	50	55	60	75	85	99

0	j		max							n−1
38	50	20	44	10	50	55	60	75	85	99

j	max									n−1
38	50	20	44	10	50	55	60	75	85	99

0	max									n−1
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

0			i							n−1
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

0			i							n−1
38	10	20	44	50	50	55	60	75	85	99

0		i								n−1
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

0			i							n−1
38	10	20	44	50	50	55	60	75	85	99

0		i								n−1
20	10	38	44	50	50	55	60	75	85	99

0	i									n−1
20	10	38	44	50	50	55	60	75	85	99

Ordenação por seleção

0			i							n−1
38	10	20	44	50	50	55	60	75	85	99

0		i								n−1
20	10	38	44	50	50	55	60	75	85	99

0	i									n−1
10	20	38	44	50	50	55	60	75	85	99

0										n−1
10	20	38	44	50	50	55	60	75	85	99

Função selecao

Algoritmo rearranja $v[0\,..\,n-1]$ em ordem crescente

```
void selecao (int n, int v[])
{
    int i, j, max, x;
    for (i = n-1; /*A*/ i > 0; i--) {
        max = i;
        for (j = i-1; j >= 0; j--)
            if (v[j] > v[max]) max = j;
        x=v[i]; v[i]=v[max]; v[max]=x;
    }
}
```

Invariantes

Relações **invariantes** chaves dizem que em /*A*/ vale que:

♡ (i0) $v[i+1 \mathinner{..} n-1]$ é crescente e
$v[0 \mathinner{..} i] \leq v[i+1 \mathinner{..} n-1]$

0				i						n−1
38	50	20	44	10	50	55	60	75	85	99

Invariantes

Relações **invariantes** chaves dizem que em /*A*/ vale que:

♡ (i0) $v[i+1\,..\,n-1]$ é crescente e
$v[0\,..\,i] \leq v[i+1\,..\,n-1]$

0				i						n−1
38	50	20	44	10	50	55	60	75	85	99

Supondo que a invariantes valem.
Correção do algoritmo é evidente.

No início da última iteração das linhas 1–5 tem-se que $i = 0$.
Da invariante conclui-se que $v[1\,..\,n-1]$ é crescente.
e que $v[0] \leq v[1\,..\,n-1]$.

Mais invariantes

Na linha 1 vale que: (i1) $v[0\,..\,i] \leq v[i+1]$;

Na linha 3 vale que: (i2) $v[j+1\,..\,i] \leq v[max]$

0	j		max		i					n−1
38	50	20	44	10	25	55	60	75	85	99

Mais invariantes

Na linha 1 vale que: (i1) $v[0\,..\,i] \le v[i{+}1]$;

Na linha 3 vale que: (i2) $v[j{+}1\,..\,i] \le v[max]$

0	j		max		i					n−1
38	50	20	44	10	25	55	60	75	85	99

invariantes (i1),(i2)

+ condição de parada do `for` da linha 3

+ troca linha 5 $\Rightarrow$ validade (i0)

Verifique!

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha	todas as execuções da linha
1	$= \ n$
2	$= \ n - 1$
3	$= \ n + (n-1) + \cdots + 1 \ = \ n(n+1)/2$
4	$= \ (n-1) + (n-2) + \cdots + 1 \ = \ (n-1)n/2$
5	$= \ n - 1$
total	$= \ n^2 + 3n - 2$

Conclusão

O consumo de tempo do algoritmo `selecao` no pior caso e no no melhor caso é proporcional a $\mathtt{n}^2$.

O consumo de tempo do algoritmo `selecao` é $O(\mathtt{n}^2)$.

Função selecao (versão min)

Algoritmo rearranja $v[0\,..\,n-1]$ em ordem crescente

```
void selecao (int n, intv[])
{
   int i, j, min, x;
   for (i = 0; i < n-1; i++) {
      min = i;
      for (j = i+1; j < n; j++)
         if (v[j] < v[min]) min = j;
      x=v[i]; v[i]=v[min]; v[min]=x;
   }
}
```

Ambiente experimental

A plataforma utilizada nos experimentos foi um computador rodando Ubuntu GNU/Linux 3.5.0-17

As especificações do computador que geraram as saídas a seguir são

```
model name: Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
cpu MHz    : 1596.000
cache size: 4096 KB

MemTotal   : 3354708 kB
```

Ambiente experimental

Os códigos foram compilados com o gcc 4.7.2 e com opções de compilação

-Wall -ansi -O2 -pedantic -Wno-unused-result

As implementações comparadas neste experimento são `bubble`, `selecao`, `insercao` e `insercaoBinaria`,

Ambiente experimental

A estimativa do tempo é calculada utilizando-se:

```
#include <time.h>
[...]
clock_t start, end;
double time;

start = clock();

[...implementação...]

end  = clock();
time = ((double)(end - start))/CLOCKS_PER_SEC;
```

Resultados experimentais: aleatórios

n	bubble	selecao	insercao	insercaoB
1024	0.00	0.00	0.00	0.00
2048	0.01	0.00	0.00	0.00
4096	0.03	0.01	0.00	0.00
8192	0.12	0.04	0.01	0.01
16384	0.51	0.17	0.05	0.03
32768	2.03	0.68	0.23	0.17
65536	8.12	2.70	0.90	0.69
131072	32.51	10.80	3.62	2.80
262144	130.05	43.14	14.49	11.26
524288	521.26	172.87	58.26	45.64

tempos em segundos

Resultados experimentais: crescente

n	bubble	selecao	insercao	insercaoB
1024	0.00	0.00	0.00	0.00
2048	0.00	0.00	0.00	0.00
4096	0.01	0.01	0.00	0.00
8192	0.03	0.04	0.00	0.00
16384	0.12	0.17	0.00	0.00
32768	0.48	0.67	0.00	0.00
65536	1.91	2.70	0.00	0.00
131072	7.67	10.77	0.00	0.00
262144	30.68	43.06	0.00	0.02
524288	123.11	172.57	0.00	0.02
1048576	500.89	696.91	0.00	0.06

tempos em segundos

Resultados experimentais: decrescente

n	bubble	selecao	insercao	insercaoB
1024	0.00	0.00	0.00	0.00
2048	0.01	0.00	0.00	0.00
4096	0.01	0.01	0.00	0.01
8192	0.04	0.04	0.03	0.01
16384	0.26	0.18	0.11	0.08
32768	1.12	0.72	0.45	0.34
65536	4.56	2.87	1.81	1.40
131072	18.23	11.47	7.24	5.64
262144	70.51	45.95	28.99	22.50
524288	203.44	183.87	116.93	92.19
1048576	754.52	742.56	493.33	405.10

tempos em segundos