AULA 15

Busca em vetor ordenado

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

PF 7.1 a 7.8

http://www.ime.usp.br/~pf/algoritmos/aulas/bubi2.html

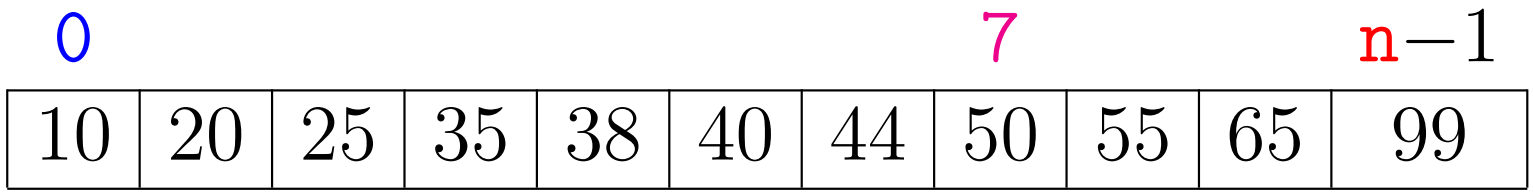
Busca em vetor ordenado

Um vetor $v[0\,..\,n-1]$ é **crescente** se

$$v[0] \le v[1] \le v[2] \le \cdots \le v[n-1].$$

Problema: Dado um número x e um vetor crescente $v[0\,..\,n-1]$, encontrar um índice m tal que $v[m]==x$.

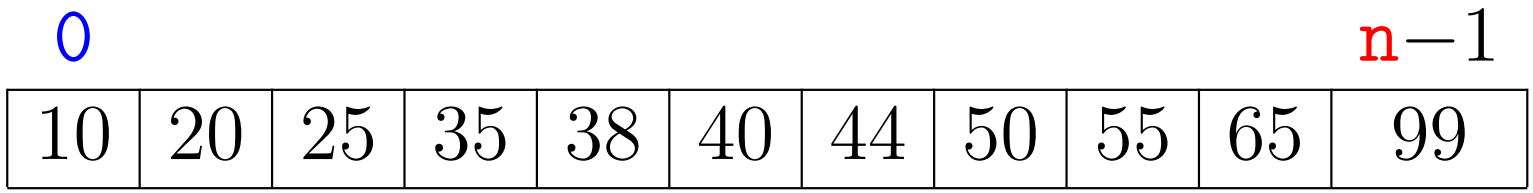
Entra: $x == 50$



Sai: $m == 7$

Busca em vetor ordenado

Entra: `x == 57`



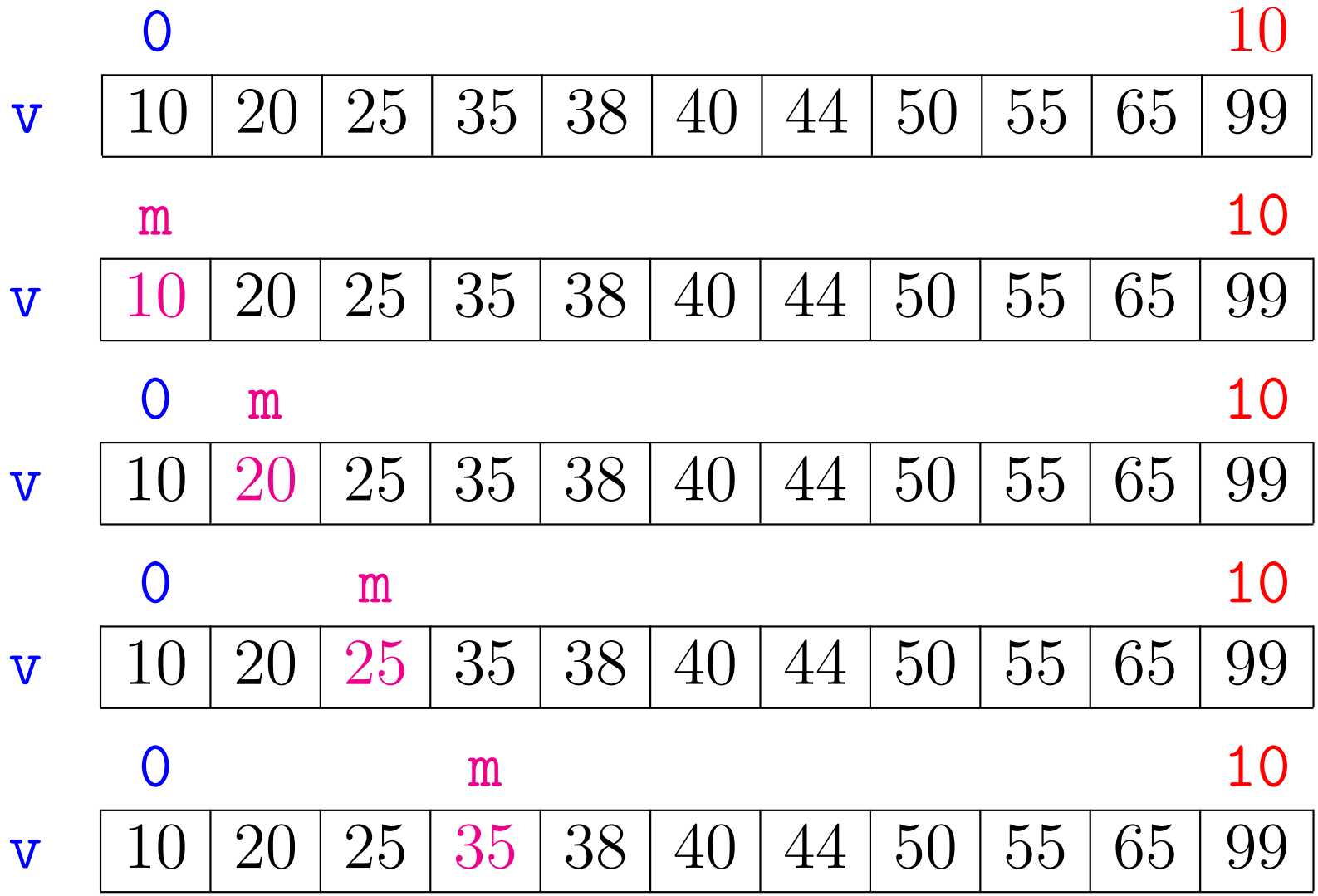
Sai: `m == -1` (`x` não está em `v`)

Busca sequencial

```
int buscaSequencial(int x, int n, int v[])
 {
1   int m = 0;
2   while (/*1*/ m < n && v[m] < x) m++;
3   if (m < n && v[m] == x)
4     return m;
5   return -1;
 }
```

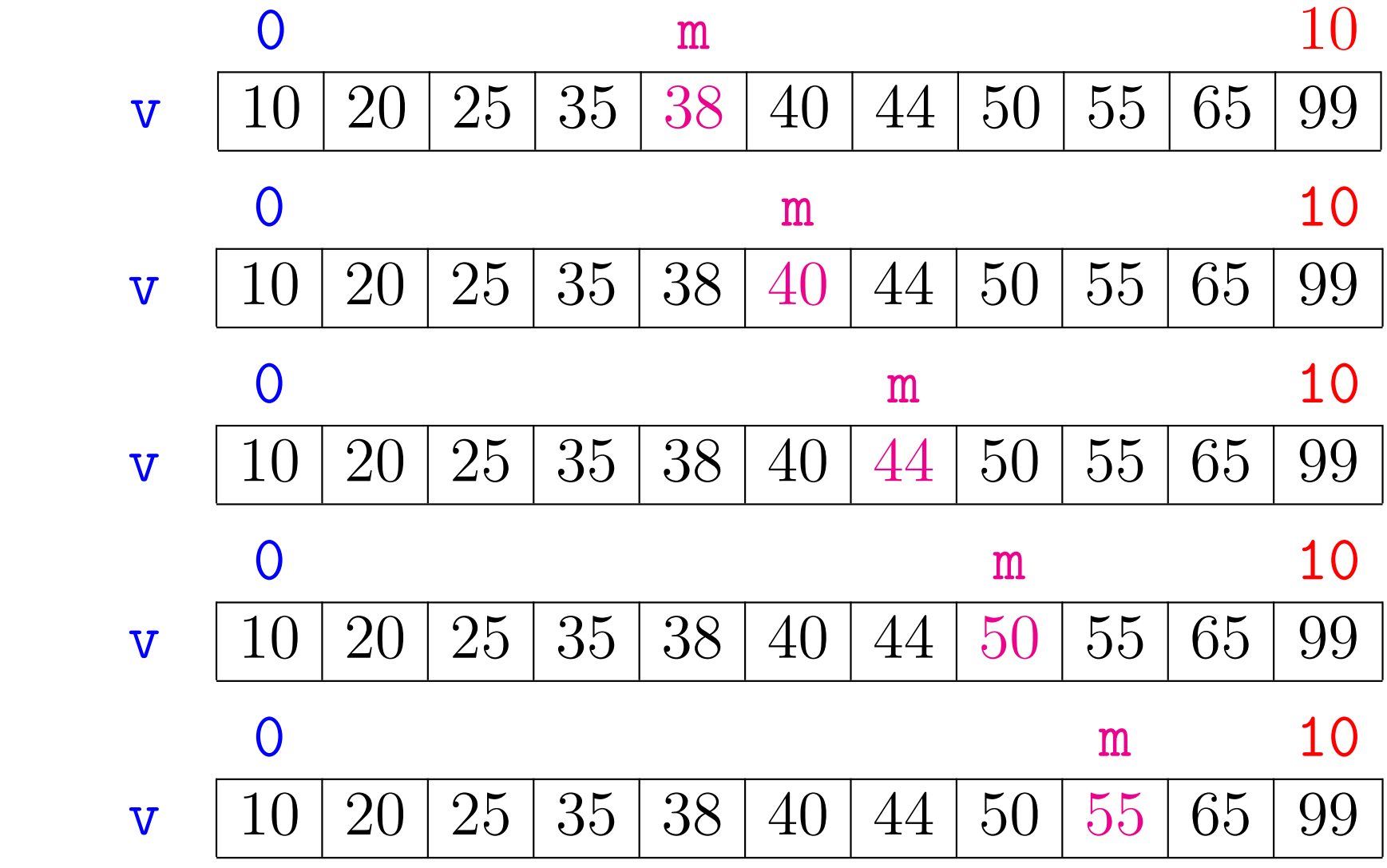
Exemplo

`x == 55`


0
10
v
10 20 25 35 38 40 44 50 55 65 99
m
10
v
10 20 25 35 38 40 44 50 55 65 99
0 m
10
v
10 20 25 35 38 40 44 50 55 65 99
0 m
10
v
10 20 25 35 38 40 44 50 55 65 99
0 m
10
v
10 20 25 35 38 40 44 50 55 65 99

Exemplo

`x == 55`


0
m
10
v
10 20 25 35 38 40 44 50 55 65 99
0
m
10
v
10 20 25 35 38 40 44 50 55 65 99
0
m
10
v
10 20 25 35 38 40 44 50 55 65 99
0
m
10
v
10 20 25 35 38 40 44 50 55 65 99
0
m
10
v
10 20 25 35 38 40 44 50 55 65 99

Consumo de tempo `buscaSequencial`

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

linha	todas as execuções da linha	
1	$= 1$	$= 1$
2	$\leq n + 1$	$\approx n$
3	$= 1$	$= 1$
4	≤ 1	≤ 1
5	≤ 1	≤ 1
total	$\leq n + 3$	$= O(n)$

Conclusão

O consumo de tempo do algoritmo `buscaSequencial` no pior caso é proporcional a `n`.

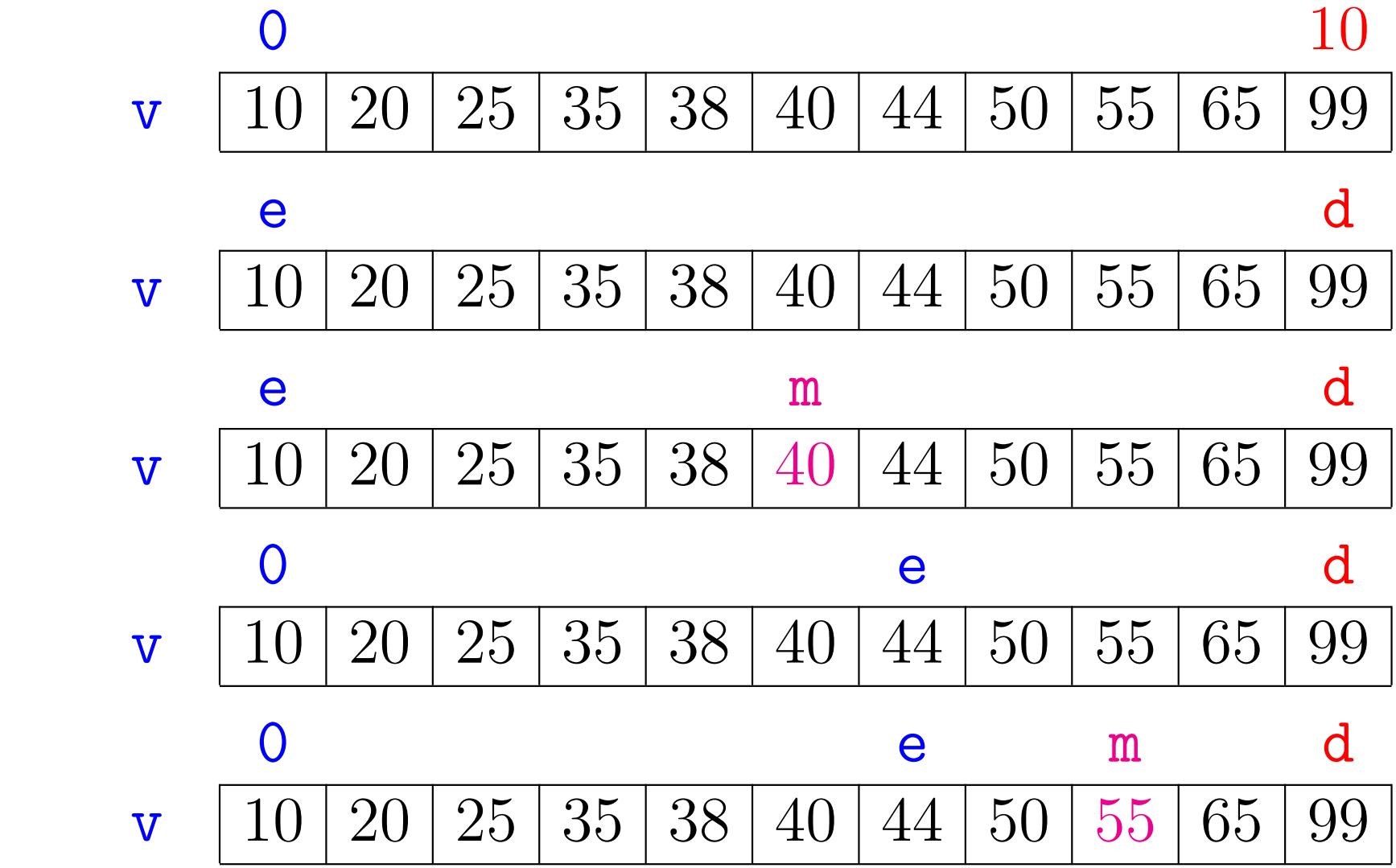
O consumo de tempo do algoritmo `buscaSequencial` é $\mathrm{O}(\mathtt{n})$.

Busca binária

```
int buscaBinaria(int x, int n, int v[]) {
    int e, m, d;
1   e = 0; d = n-1;
2   while (/*1*/ e <= d) {
3      m = (e + d)/2;
4      if (v[m] == x) return m;
5      if (v[m] < x) e = m + 1;
6      else d = m - 1;
    }
7   return -1;
 }
```

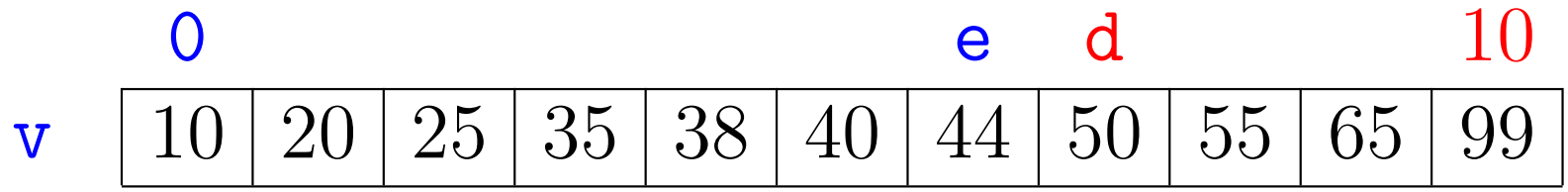
Exemplo

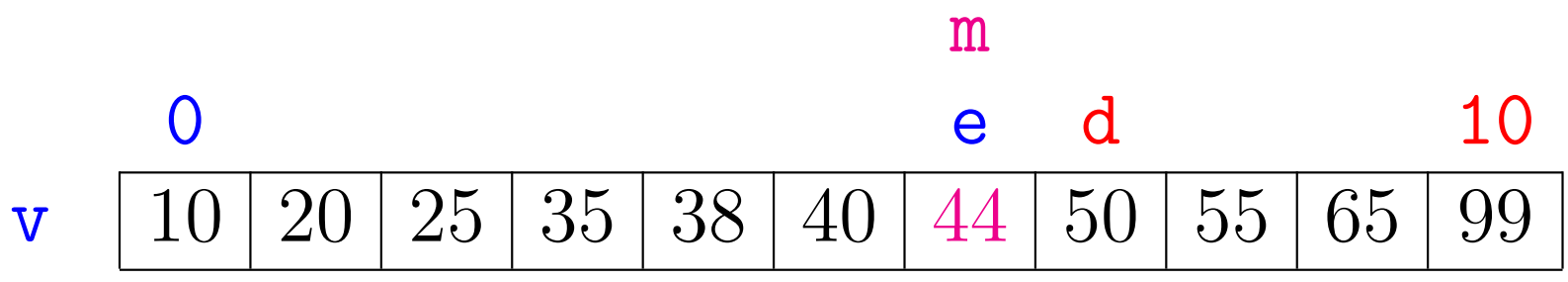
`x == 48`

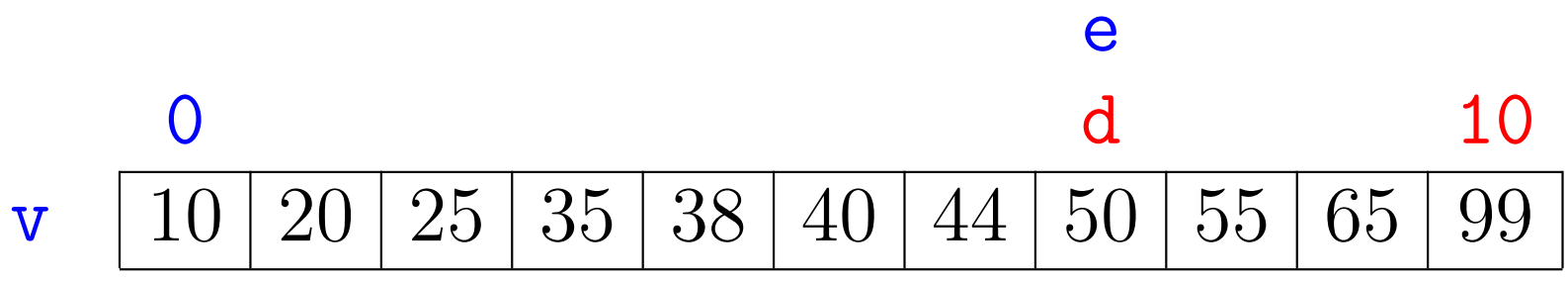

0
10
v
10 20 25 35 38 40 44 50 55 65 99
e
d
v
10 20 25 35 38 40 44 50 55 65 99
e
m
d
v
10 20 25 35 38 40 44 50 55 65 99
0
e
d
v
10 20 25 35 38 40 44 50 55 65 99
0
e
m
d
v
10 20 25 35 38 40 44 50 55 65 99

Exemplo

`x == 48`

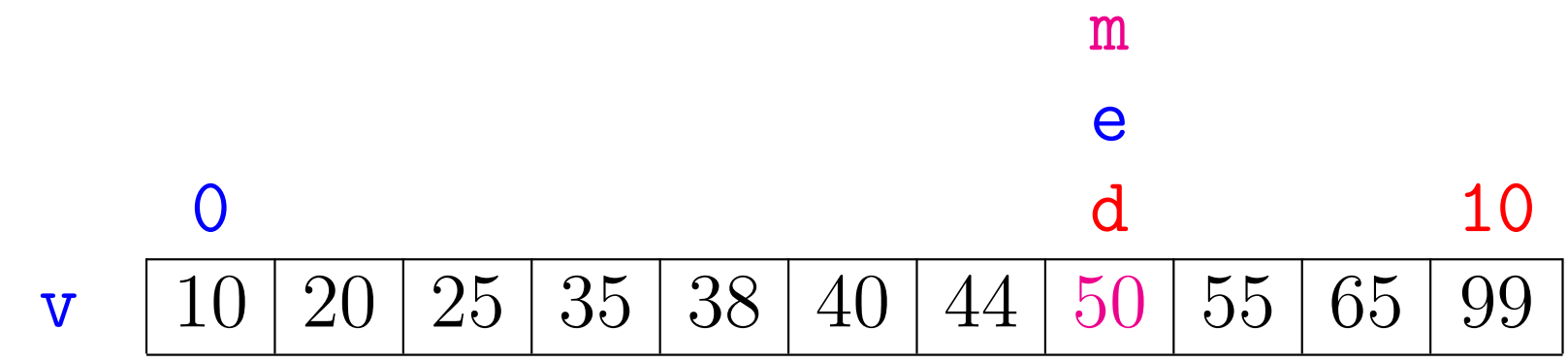

0
e
d
10
v
10
20
25
35
38
40
44
50
55
65
99

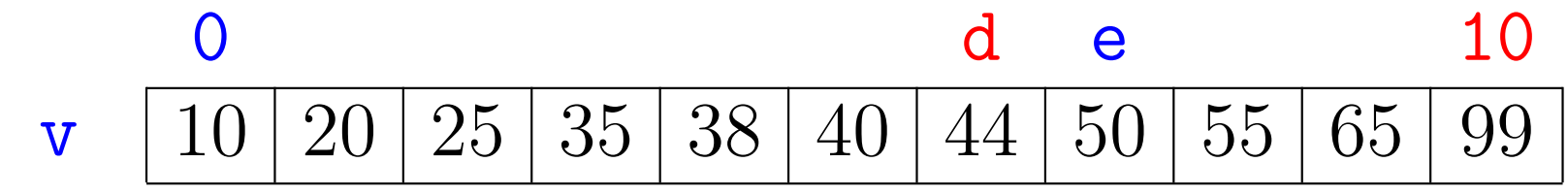

m
0
e
d
10
v
10
20
25
35
38
40
44
50
55
65
99


e
0
d
10
v
10
20
25
35
38
40
44
50
55
65
99

Exemplo

`x == 48`


m
e
0
d
10
v
10 20 25 35 38 40 44 50 55 65 99


0
d
e
10
v
10 20 25 35 38 40 44 50 55 65 99

Consumo de tempo `buscaBinaria`

O consumo de tempo da função `buscaBinaria` é proporcional ao número `k` de iterações do `while`.

No início da 1a. iteração tem-se que $\mathtt{d} - \mathtt{e} = \mathtt{n} - 1 \approx \mathtt{n}$.

Sejam

$$(\mathtt{e}_0, \mathtt{d}_0), (\mathtt{e}_1, \mathtt{d}_1), \ldots, (\mathtt{e}_\mathtt{k}, \mathtt{d}_\mathtt{k}),$$

os valores das variáveis `e` e `d`
no início de cada uma das iterações.

No pior caso `x` não está em `v`.

Assim, $\mathtt{d}_{\mathtt{k}-1} - \mathtt{e}_{\mathtt{k}-1} \geq 0$ **e** $\mathtt{d}_\mathtt{k} - \mathtt{e}_\mathtt{k} < 0$.

Número iterações

Estimaremos o valor de k em função de $d - e$.

Note que $d_{i+1} - e_{i+1} \leq (d_i - e_i)/2$
para $i=1,2,\ldots,k-1$.

Desta forma tem-se que

$$
\begin{array}{lcllcl}
d_0 - e_0 & = & n - 1 & < & n & \\
d_1 - e_1 & \leq & (d_0 - e_0)/2 & < & n/2 & \\
d_2 - e_2 & \leq & (d_1 - e_1)/2 & < & (n/2)/2 & = n/2^2 \\
d_3 - e_3 & \leq & (d_2 - e_2)/2 & < & (n/2^2)/2 & = n/2^3 \\
d_4 - e_4 & \leq & (d_3 - e_3)/2 & < & (n/2^3)/2 & = n/2^4 \\
\vdots & & \vdots & & \vdots & \vdots
\end{array}
$$

Número iterações

Percebe-se que depois de cada iteração
o valor de $\mathtt{d} - \mathtt{e}$ é reduzido pela metade.

Seja $\mathtt{t}$ o número inteiro tal que

$$2^{\mathtt{t}} \leq \mathtt{n} < 2^{\mathtt{t}+1}.$$

Da primeira desigualdade temos que $\mathtt{t} \leq \lg \mathtt{n}$,
onde $\lg \mathtt{n}$ denota o logaritmo de $\mathtt{n}$ na base 2.

Portanto, o número $\mathtt{k}$ de iterações é não superior a

$$\mathtt{t} + 1 \leq \lg \mathtt{n} + 1.$$

Conclusão

O consumo de tempo do algoritmo `buscaBinaria` no pior caso é proporcional a $\lg n$.

O consumo de tempo do algoritmo `buscaBinaria` é $O(\lg n)$.

Número de iterações

buscaSequencial	buscaBinaria
n	$\lg n$
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
262144	18
1048576	20
⋮	⋮
4294967296	32

Versão recursiva da busca binária

Para formular uma versão recursiva,
é necessário generalizar um pouco o problema
trocando $v[0\,..\,n-1]$ por $v[e\,..\,d]$.

```
   int buscaBinaria(int x, int n, int v[]) {
1      return buscaBinariaR(x, 0, n-1, v);
   }
```

Versão recursiva da busca binária

Recebe um vetor crescente $v[e\,..\,d]$ e devolve um índice m tal que $v[m] == x$. Se tal m não existe, devolve -1.

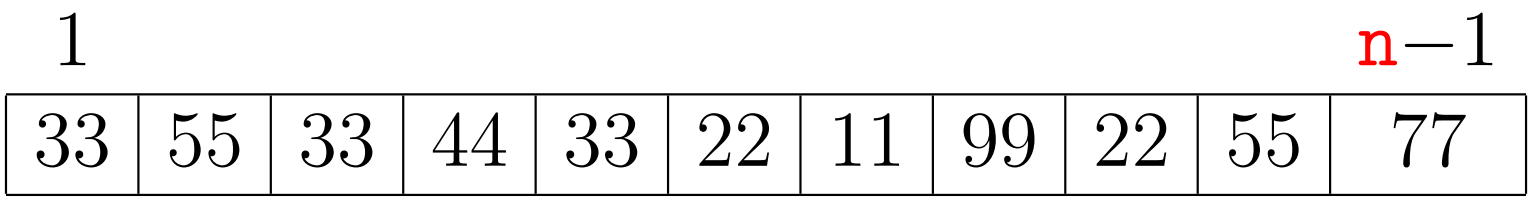
```
 int buscaBinariaR(int x, int e, int d, int v[]) {
    int m;
1   if (d < e) return -1;
2   m = (e + d)/2;
3   if (v[m] == x) return m;
4   if (v[m] < x)
5      return buscaBinariaR(x, m+1, d, v);
6   return buscaBinariaR(x, e, m-1, v);
 }
```

Ordenação

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

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

Entra:



Ordenação

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

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

Entra:

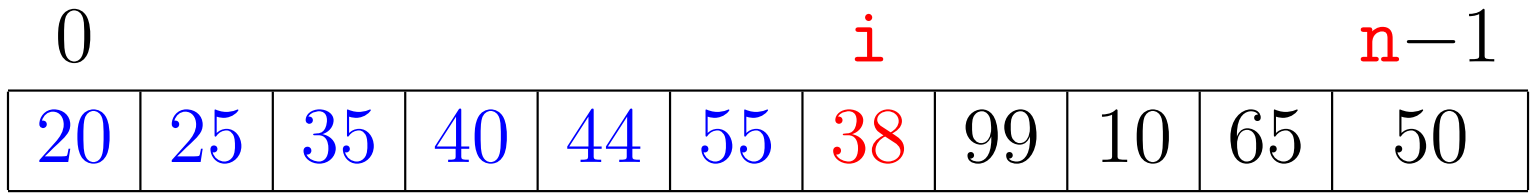
1										$\mathtt{n}-1$
33	55	33	44	33	22	11	99	22	55	77

Sai:

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

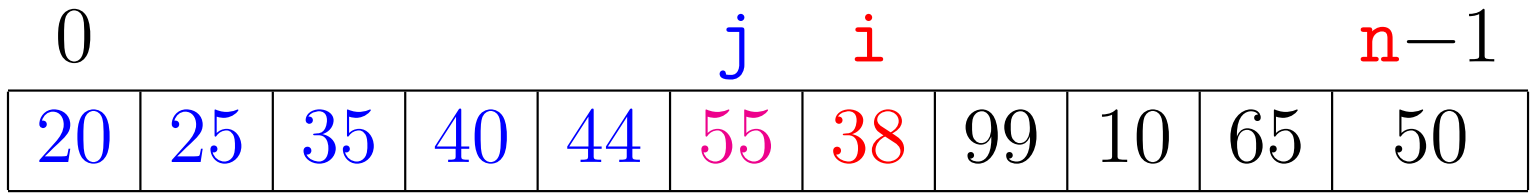
Ordenação por inserção (iteração)

$\mathtt{x} = 38$



Ordenação por inserção (iteração)

$\mathtt{x} = 38$



Ordenação por inserção (iteração)

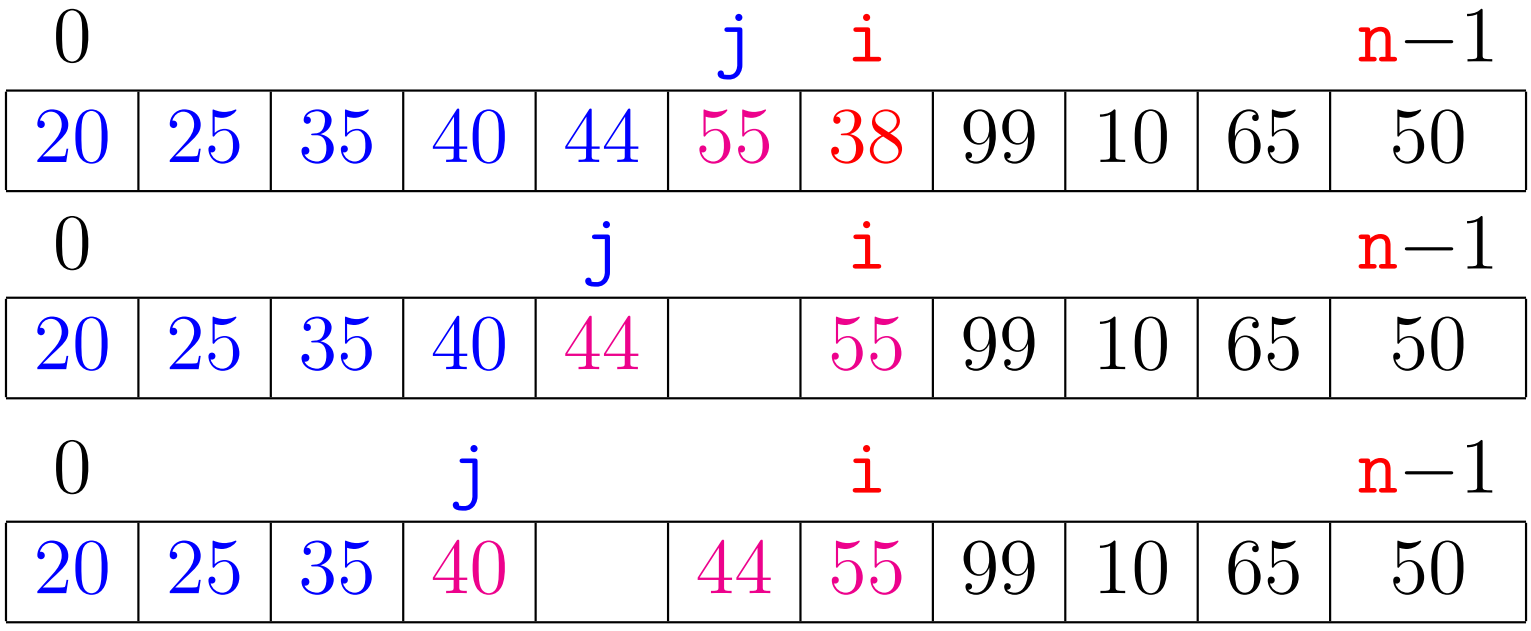
$\mathtt{x} = 38$

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

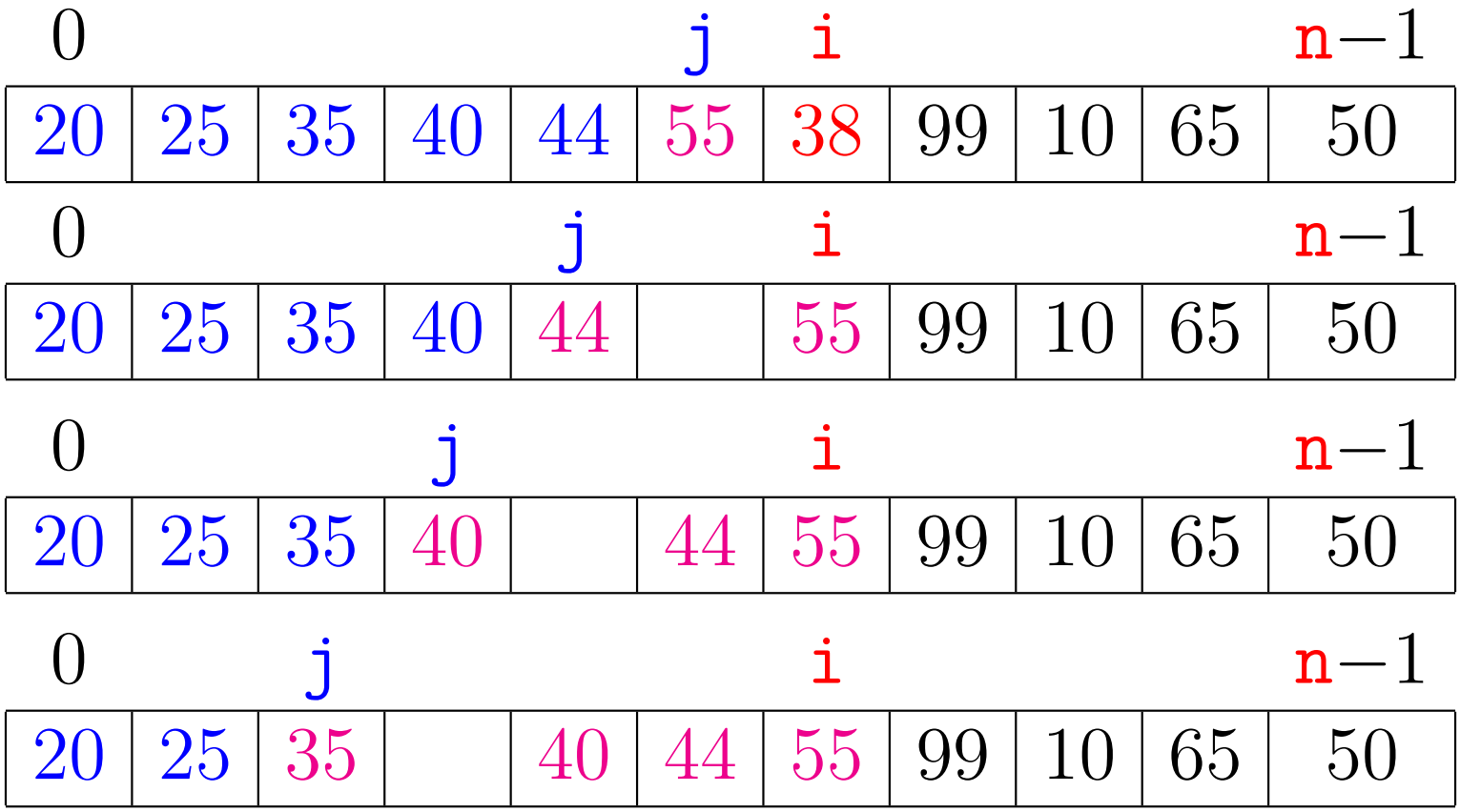
Ordenação por inserção (iteração)

$\mathtt{x} = 38$



Ordenação por inserção (iteração)

$\mathtt{x} = 38$



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

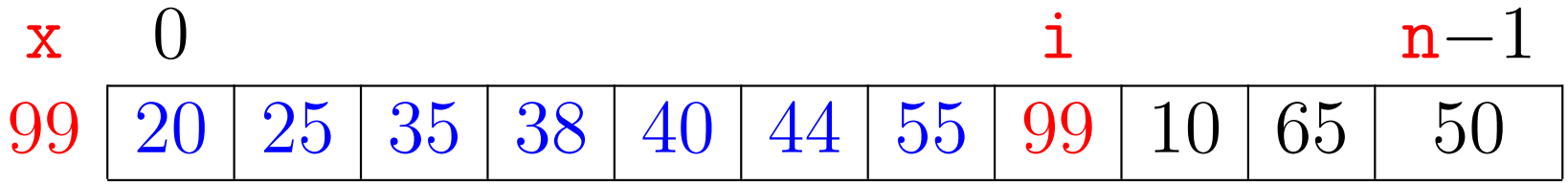
0		j				i				n−1
20	25	35		40	44	55	99	10	65	50

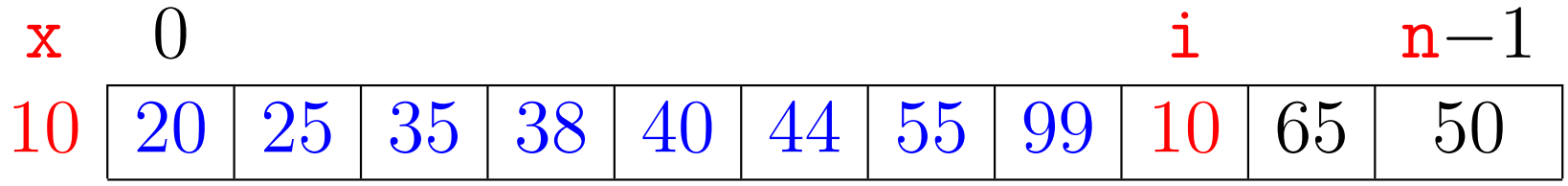
0		j				i				n−1
20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

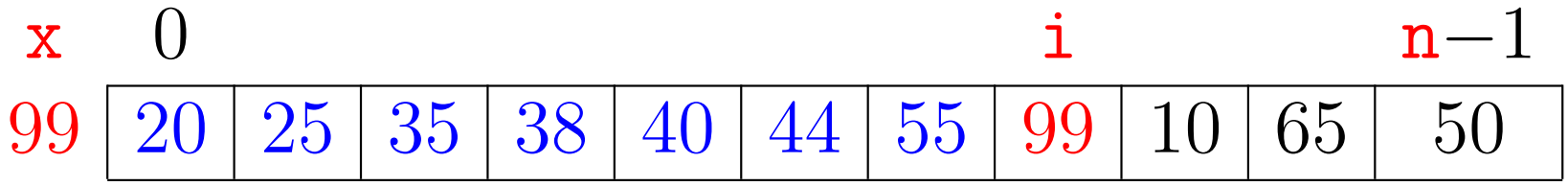
x	0							i			n−1
99	20	25	35	38	40	44	55	99	10	65	50

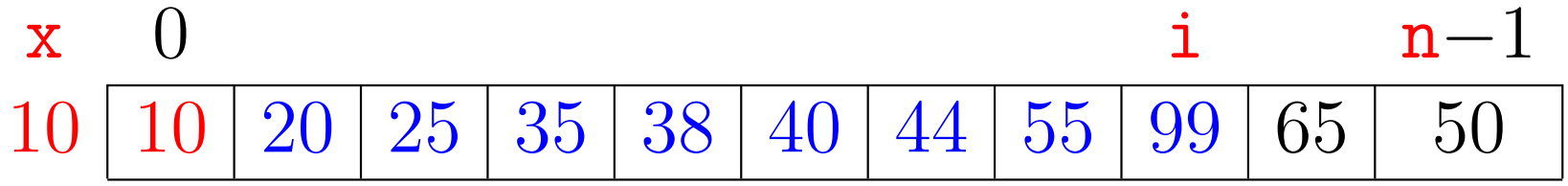
Ordenação por inserção



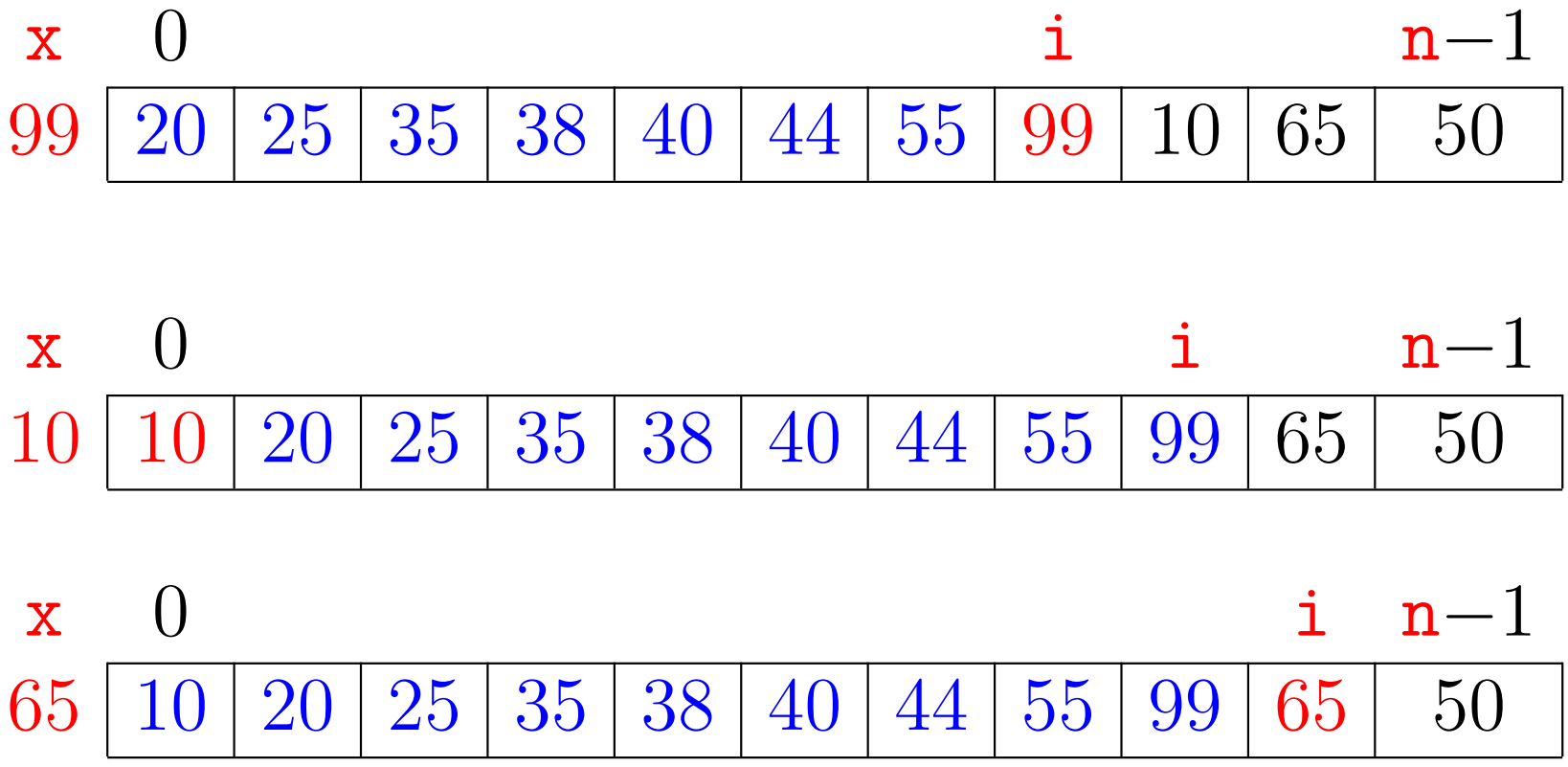


Ordenação por inserção

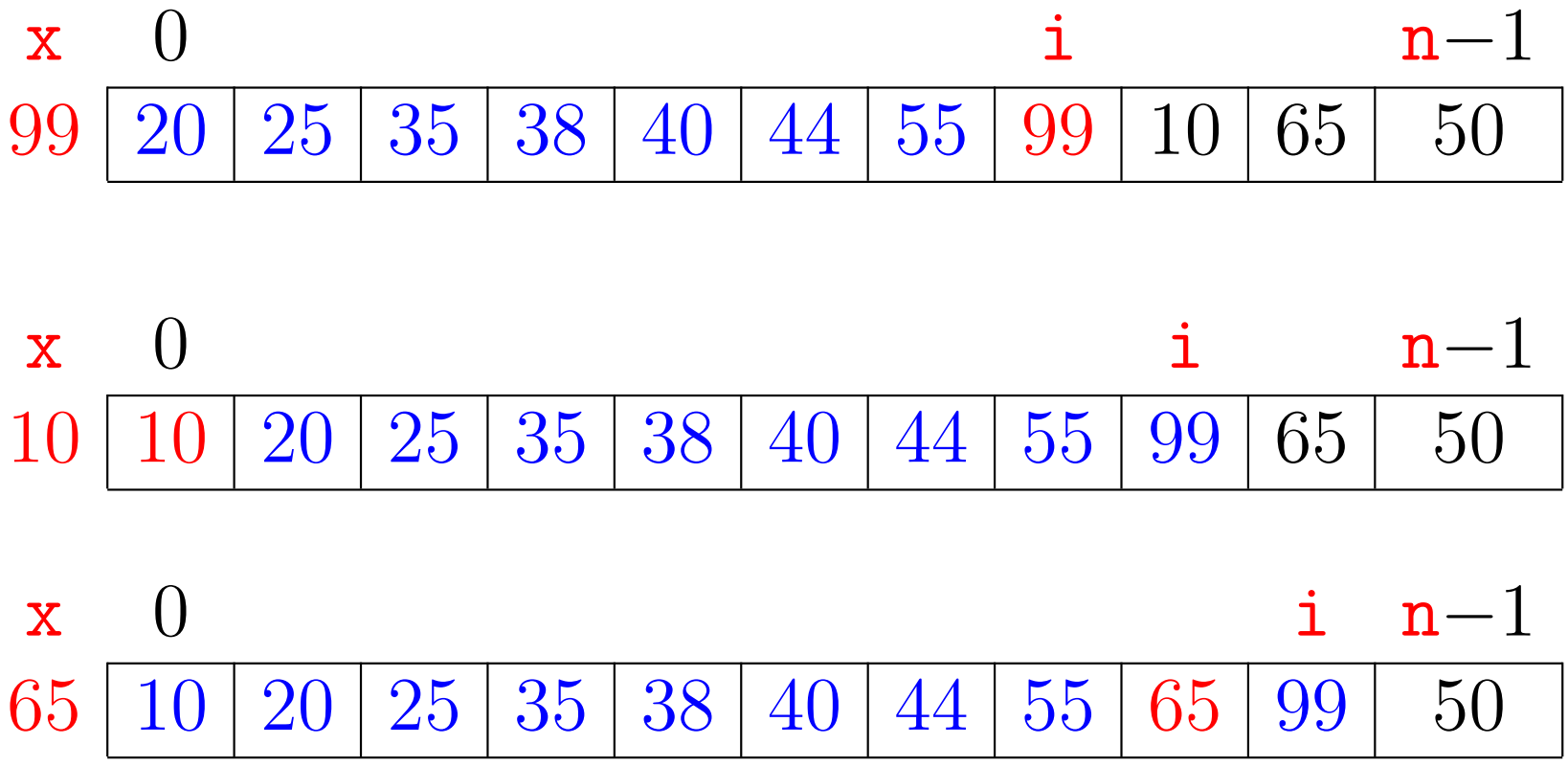




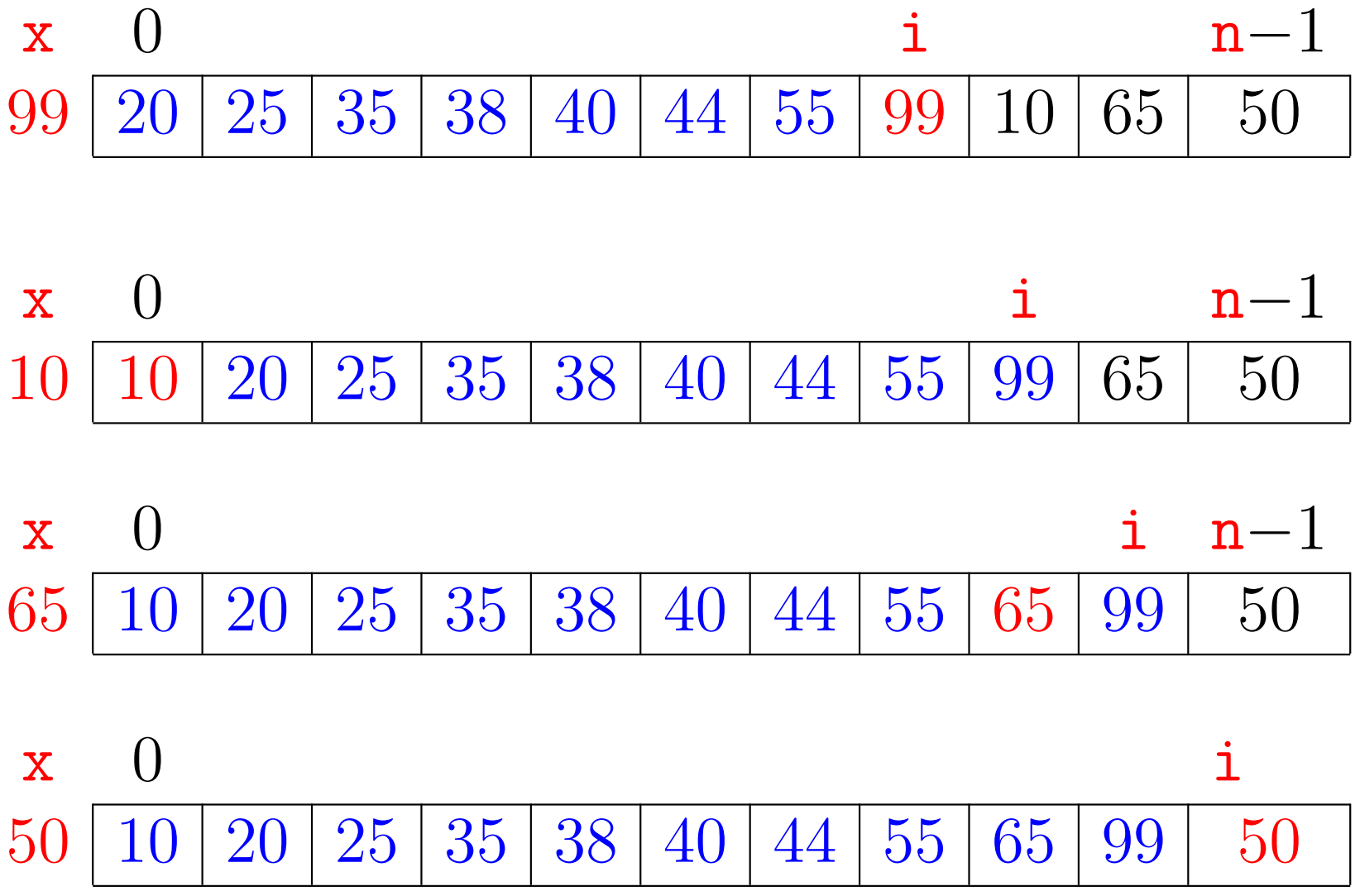
Ordenação por inserção



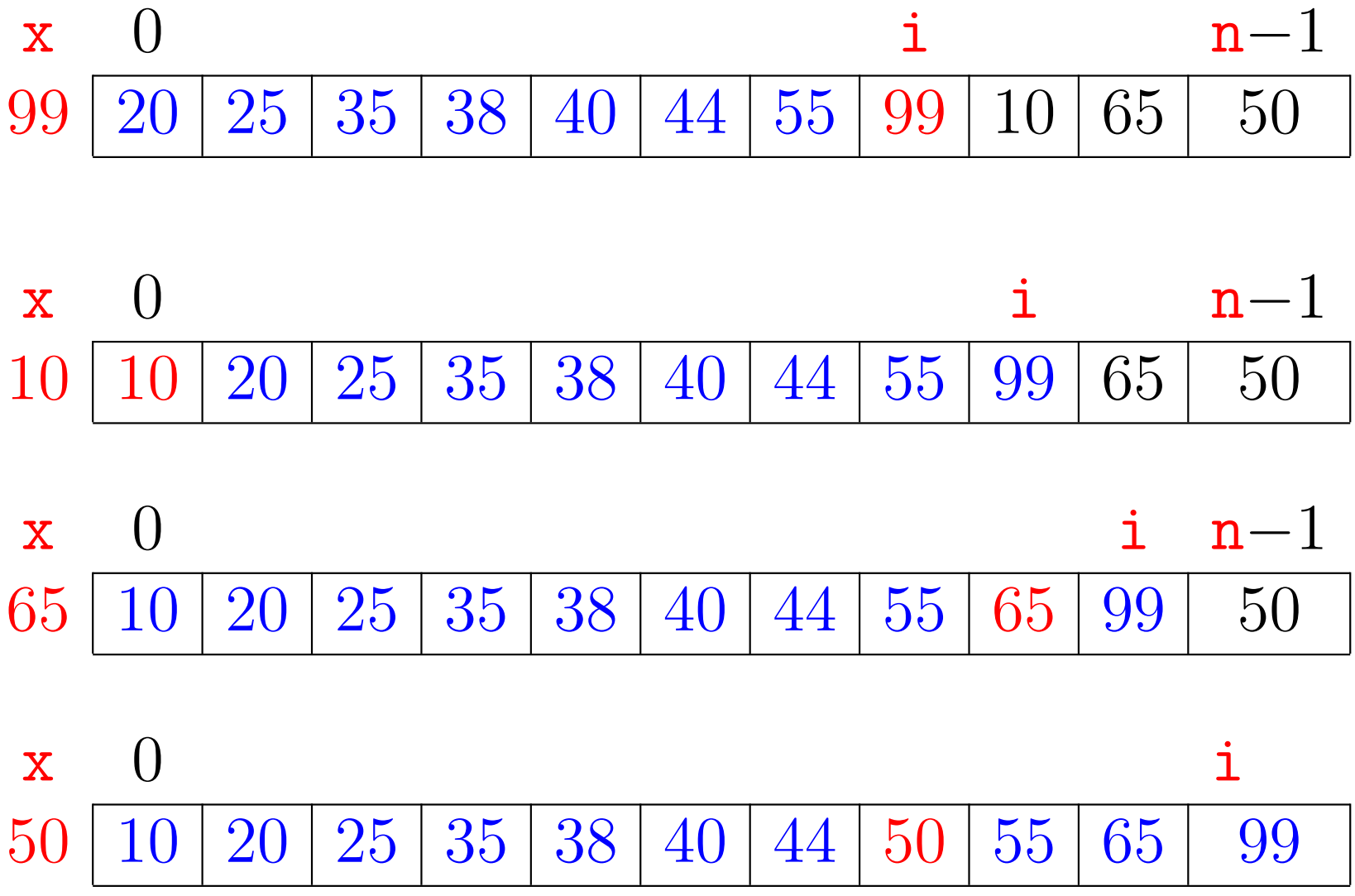
Ordenação por inserção



Ordenação por inserção



Ordenação por inserção



insercao

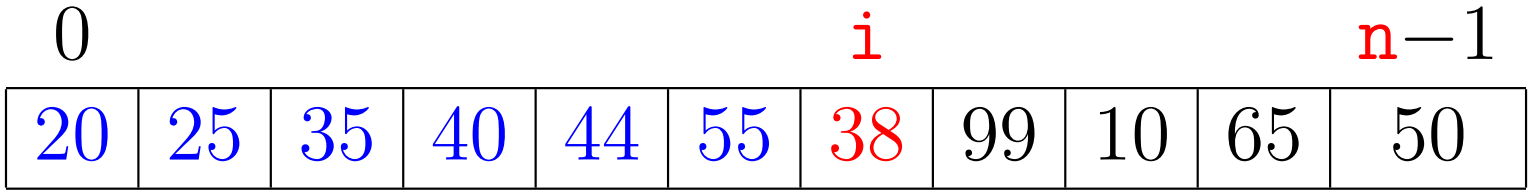
Função rearranja $v[0\,..\,n-1]$ em ordem crescente.

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

O algoritmo faz o que promete?

Relação **invariante** chave:

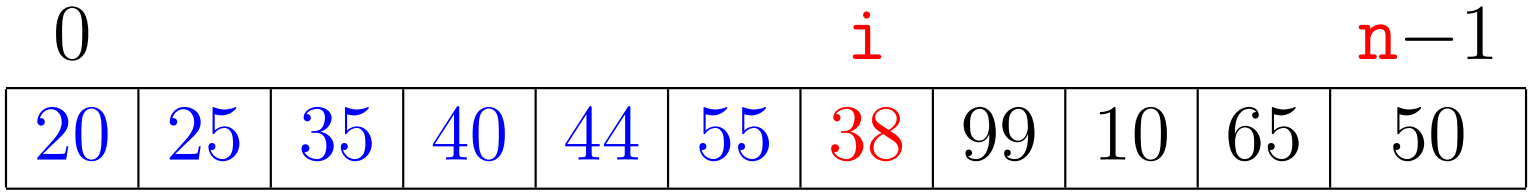
♡ (i0) Em `/*A*/` vale que: `v`$[0\,..\,$`i`$-1]$ é crescente.



O algoritmo faz o que promete?

Relação **invariante** chave:

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



Supondo que a relação invariante vale,
a correção do algoritmo é evidente.

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

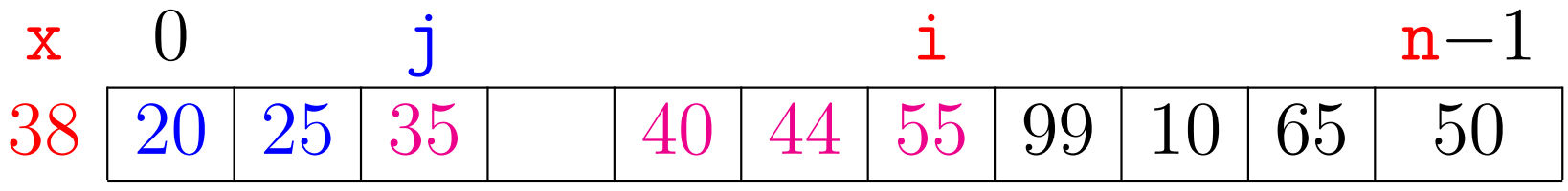
Mais invariantes

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

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



Invariantes (i1), (i2) e (i3)

+ condição de parada do `for` da linha 3

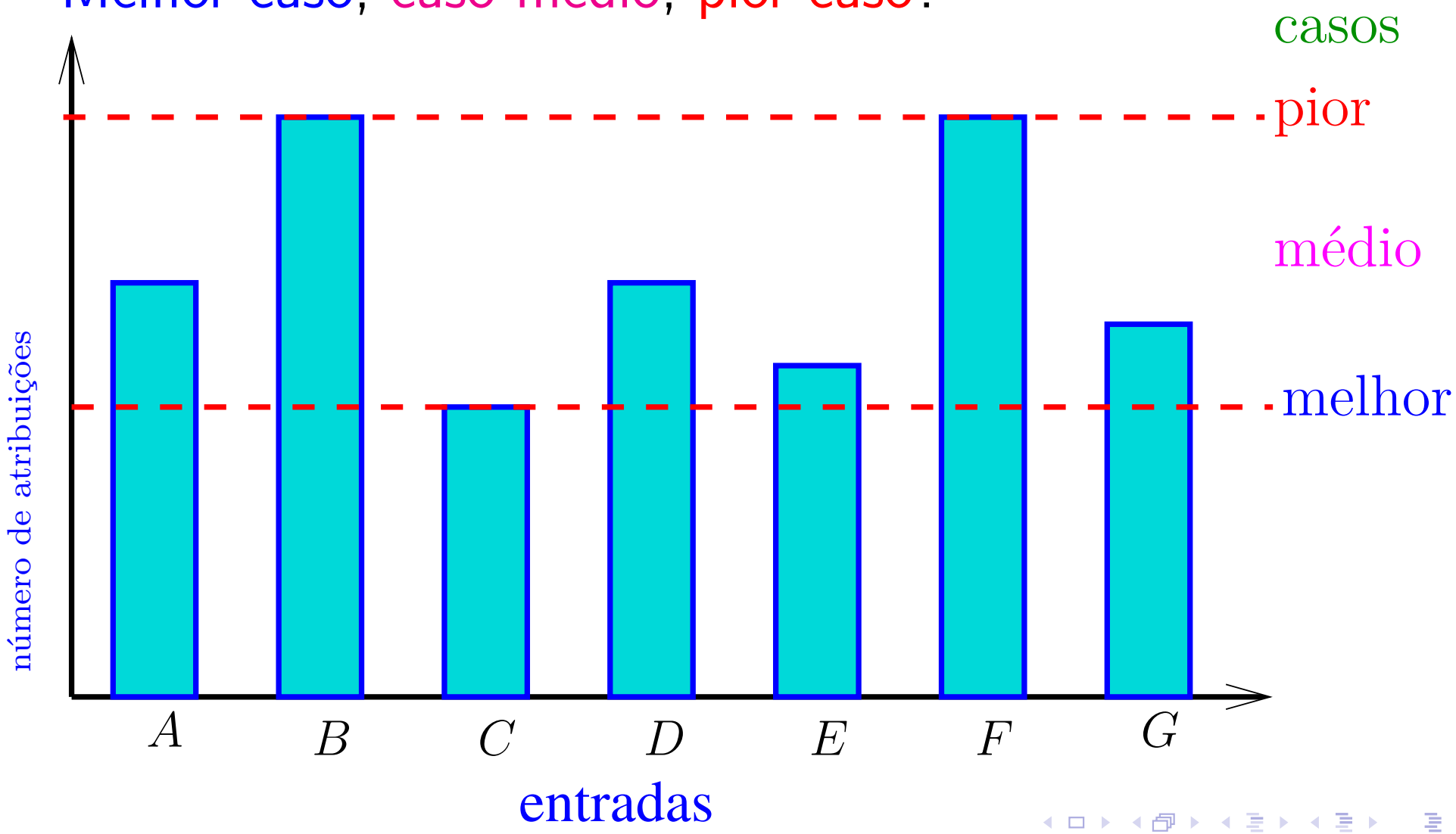
+ atribuição da linha 5 $\Rightarrow$ validade (i0)

Verifique!

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];
```

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

Quantas atribuições faz a função?

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

```
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	$\leq 1 + \texttt{i}$
4	?
total	?

Quantas atribuições faz a função?

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

```
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	$\leq 1 + \texttt{i}$
4	$\leq \texttt{i}$
total	$\leq 2\texttt{i} + 1 \leq 2\texttt{n}$

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	$= n$
2	$= n - 1$
3	$\leq 1 + 2 + \cdots + n = n(n+1)/2$
4	$\leq 1 + 2 + \cdots + (n-1) = (n-1)n/2$
5	$= n - 1$
total	$\leq n^2 + 3n - 2$

$n^2 + 3n - 2$ 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;
1   for (i = 1; /*A*/ i < n; i++){
2      x = v[i];
3      for (j = i-1; j >= 0 && v[j] > x;j--)
4         v[j+1] = v[j];
5      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	$=$	$\mathtt{n}$
2	$=$	$\mathtt{n} - 1$
3	$=$	$\mathtt{n} - 1$
4	$=$	0
5	$=$	$\mathtt{n} - 1$
total	$\leq$	$4\mathtt{n} - 3 = \mathrm{O}(\mathtt{n})$

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 n^2.

O consumo de tempo da função `insercao` melhor caso é proporcional a n.

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

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 $\mathtt{v}[0] \le \cdots \le \mathtt{v}[\mathtt{n}-1]$.

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

Entra:

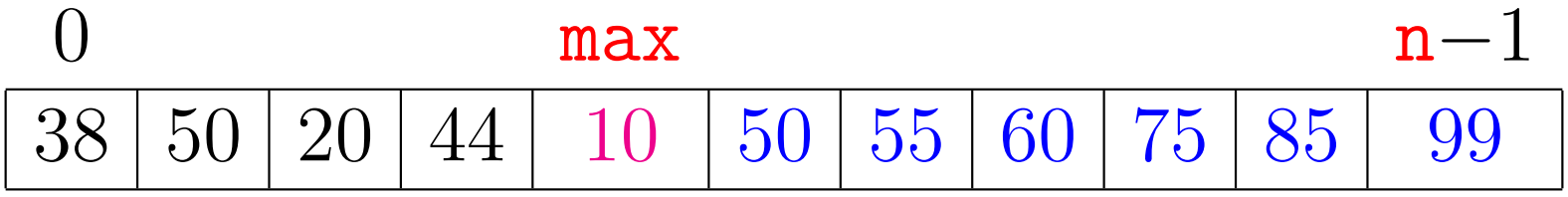
0										$\mathtt{n}-1$
33	55	33	44	33	22	11	99	22	55	77

Sai:

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

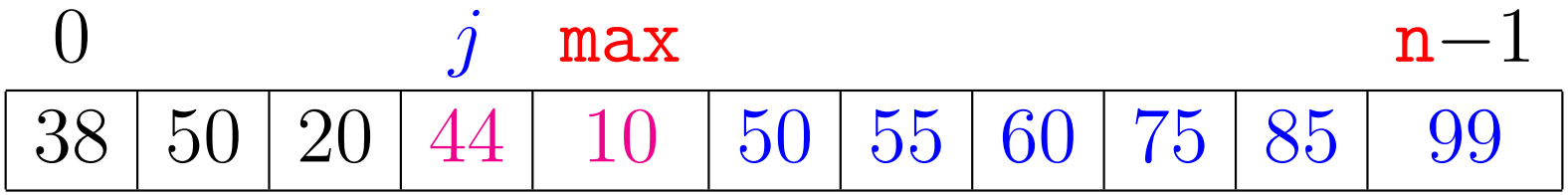
Ordenação por seleção (iteração)

$\texttt{i} = 5$



Ordenação por seleção (iteração)

$\texttt{i} = 5$



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

Ordenação por seleção (iteração)

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)

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

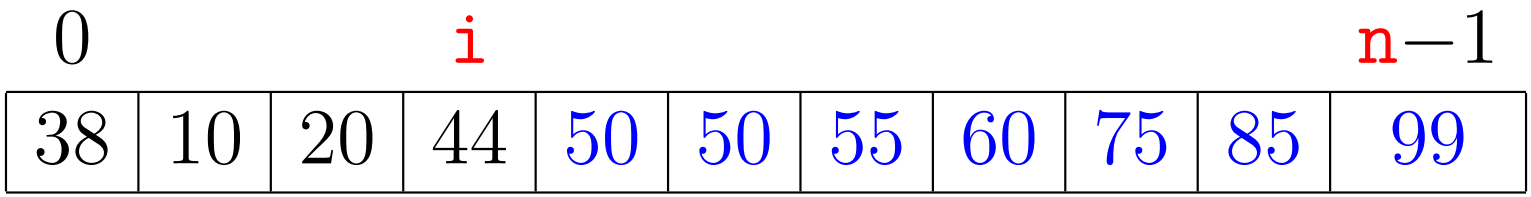
0		j	max						n−1
38	50	20	44	10	50	55	60	75	85	99

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

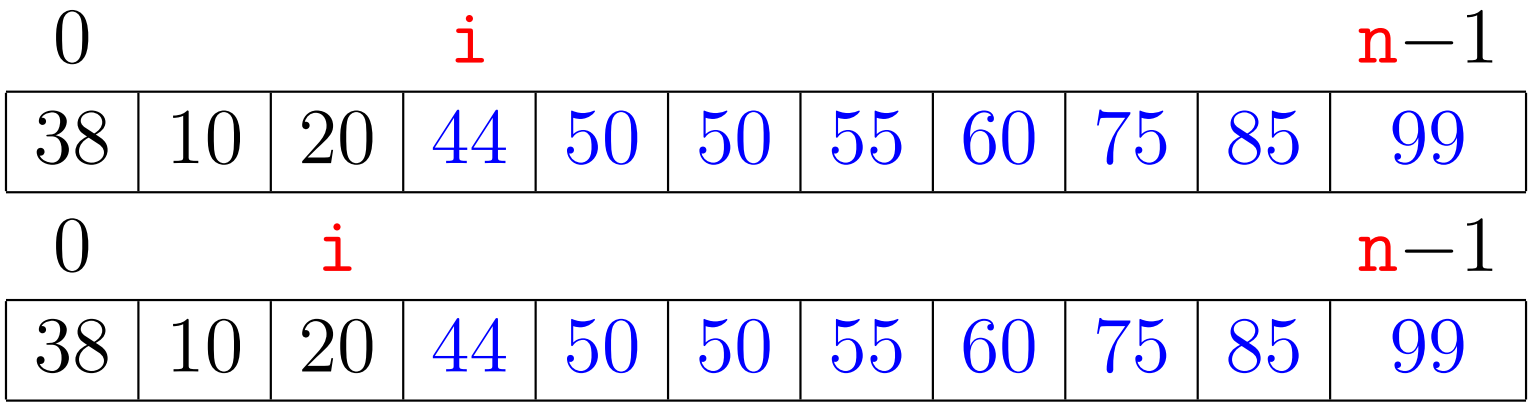
j	max								n−1
38	50	20	44	10	50	55	60	75	85	99

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

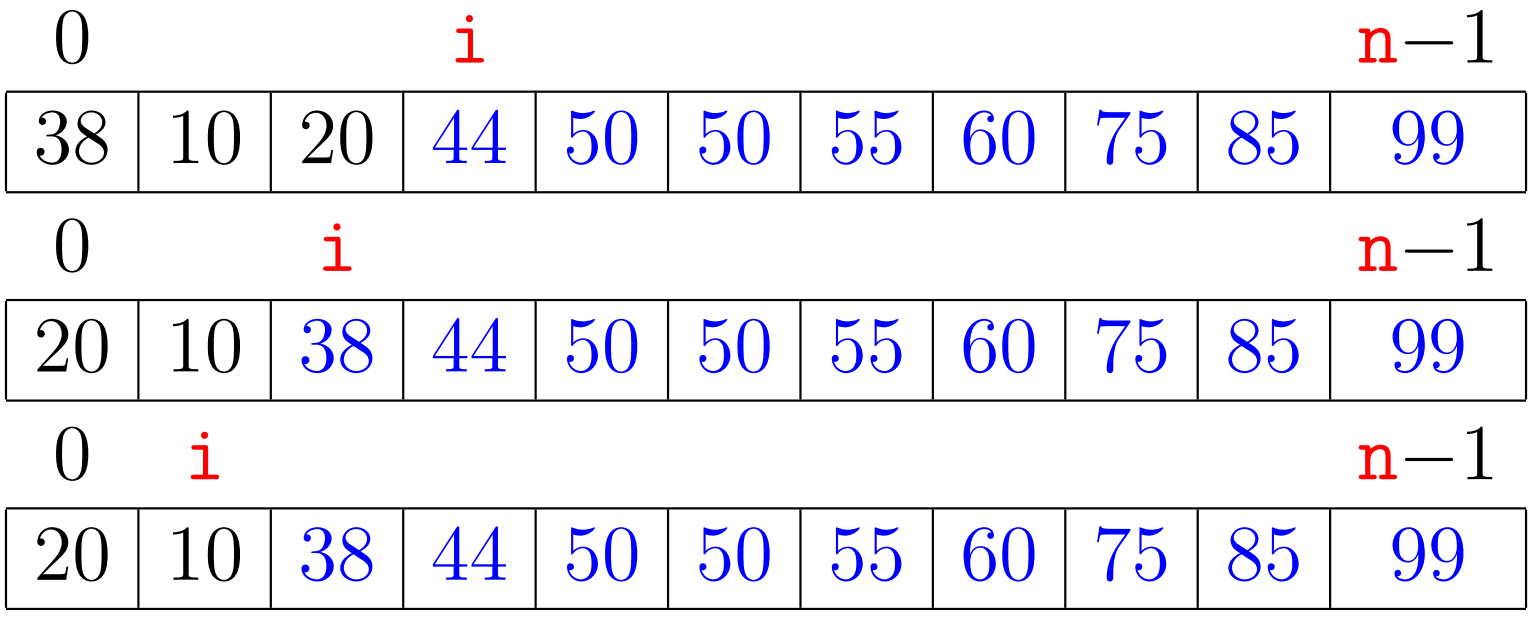
Ordenação por seleção



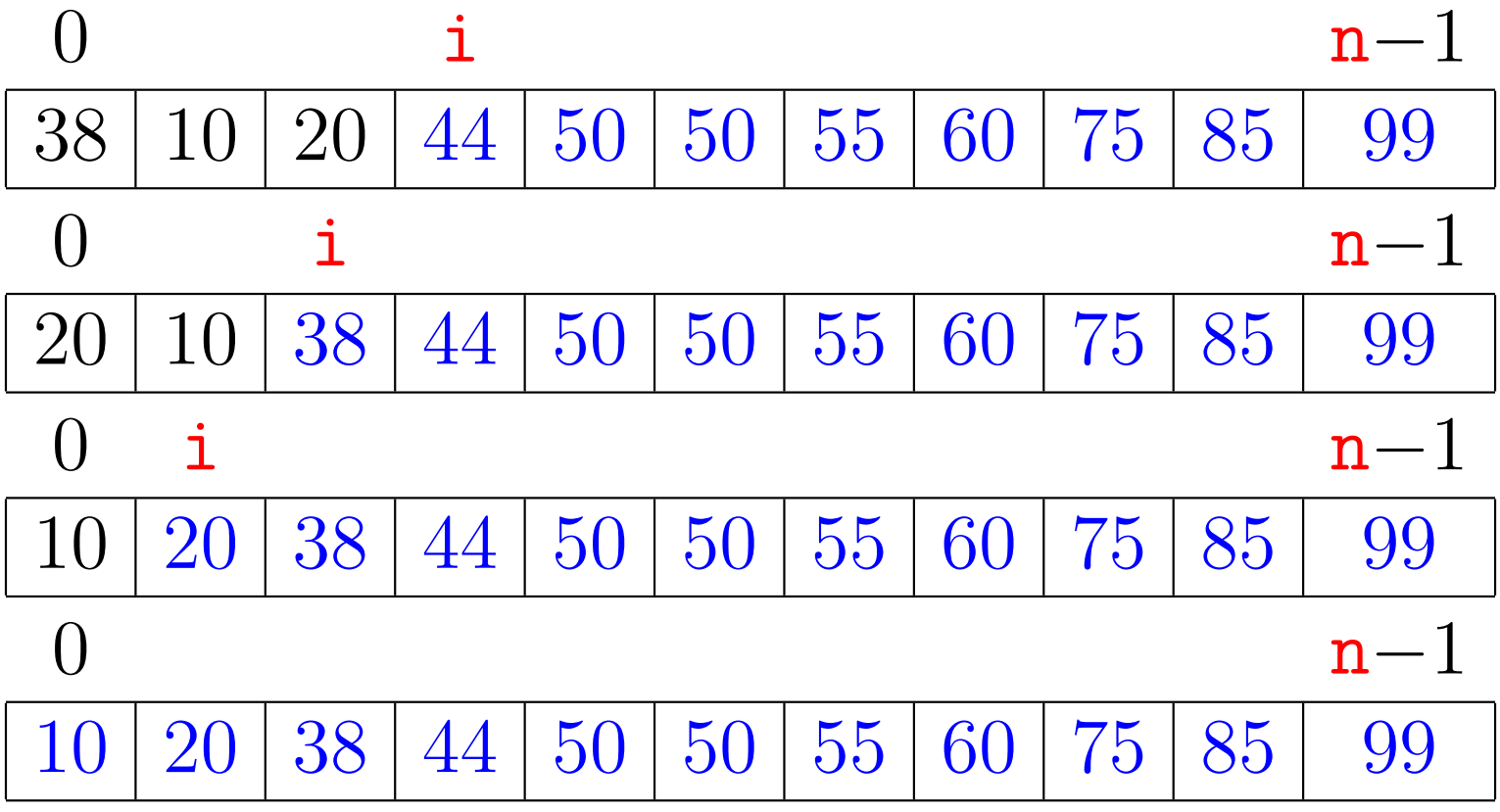
Ordenação por seleção



Ordenação por seleção



Ordenação por seleção



Função selecao

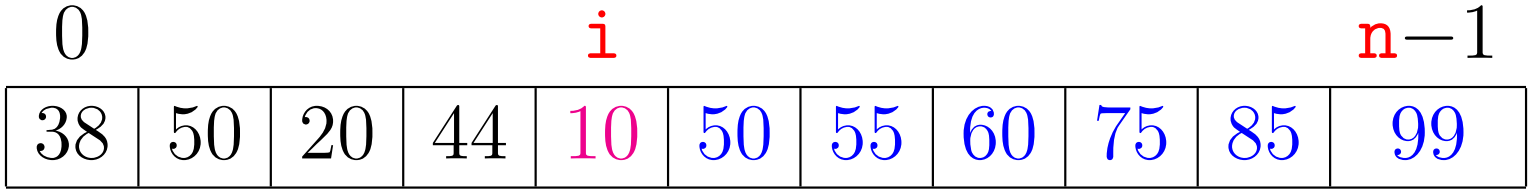
Algoritmo rearranja $v[0\,..\,n-1]$ em ordem crescente

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

Invariantes

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

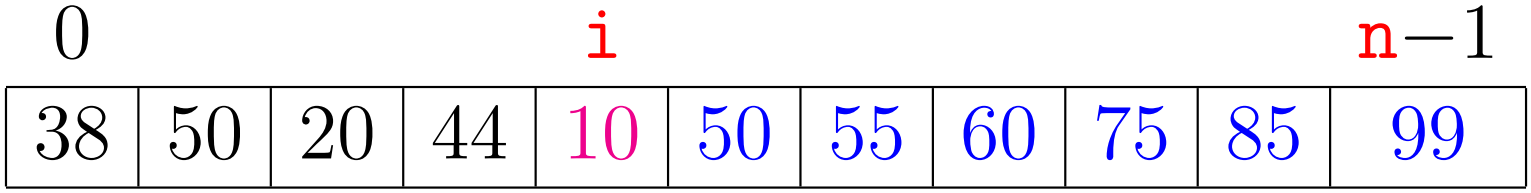
♡ (i0) $\mathtt{v}[\mathtt{i}+1\,..\,\mathtt{n}-1]$ é crescente e
$\mathtt{v}[0\,..\,\mathtt{i}] \le \mathtt{v}[\mathtt{i}+1\,..\,\mathtt{n}-1]$



Invariantes

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

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

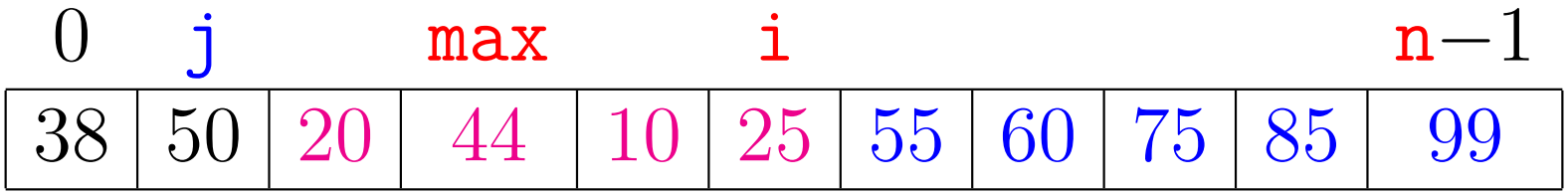


Supondo que a relação invariante vale,
a correção do algoritmo é evidente.

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

Mais invariantes

Na linha 1 vale que: (i1) $\texttt{v}[0\,..\,\texttt{i}] \le \texttt{v}[\texttt{i}+1]$;
Na linha 3 vale que: (i2) $\texttt{v}[\texttt{j}+1\,..\,\texttt{i}] \le \texttt{v}[\texttt{max}]$



Invariantes (i1) e (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 melhor caso é proporcional a $\mathtt{n}^2$.

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

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