

Programação Estruturada

Prof. Paulo Henrique Pisani

<http://professor.ufabc.edu.br/~paulo.pisani/>

Tópicos

- Ordenação:
 - Bubble sort
- Busca:
 - Busca linear/sequencial
 - Busca binária

Ordenação

- Ordenação é o processo de **rearranjar uma sequência de elementos em ordem ascendente ou descendente**, de acordo com a **chave** de cada elemento;
- Um dos principais objetivos de realizar a ordenação é **facilitar a recuperação** dos elementos por sua chave.

Algoritmos de ordenação

- Os algoritmos de ordenação podem ser divididos entre os baseados em comparação:
 - Bubble sort; ← Veremos esse algoritmo na aula de hoje.
 - Selection sort; } Vimos esses algoritmos na aula passada
 - Insertion sort; }
 - Merge sort;
 - Quick sort;
 - Heap sort.
- E os baseados em distribuição:
 - Count sort;
 - Radix sort;
 - Bucket sort.

Ordenação

- Faremos a **ordenação de elementos em vetores/arrays** nesta aula, mas podemos aplicar os algoritmos de ordenação para outras estruturas (e.g. listas ligadas).

6	9	40	3	5	16
---	---	----	---	---	----



Ordenação crescente

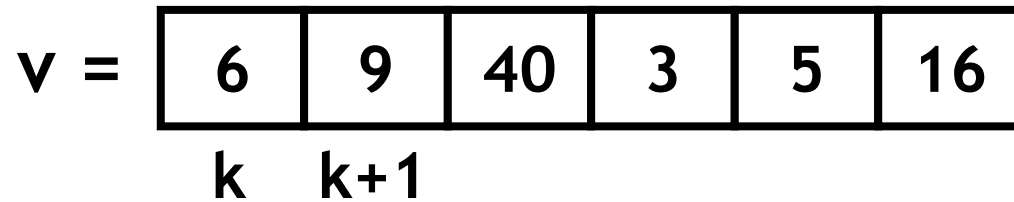
3	5	6	9	16	40
---	---	---	---	----	----

Bubble sort

Bubble sort

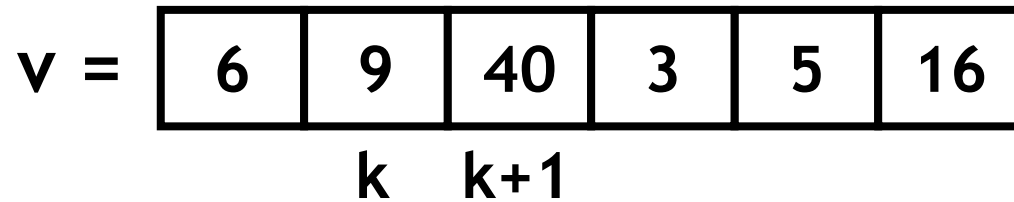
- Ideia geral:
 - Inicia no primeiro elemento e compara os elementos dois a dois;
 - Se elemento[k] > elemento[k+1], troca os dois elementos;
 - Repete o processo n - 1 vezes. Contudo, não é necessário ir até o fim do vetor nas demais iterações:
 - O processo aplicado garante que o maior elemento estará na última posição;
 - Na segunda iteração, o segundo maior elemento estará na penúltima posição, e assim por diante.

Bubble sort



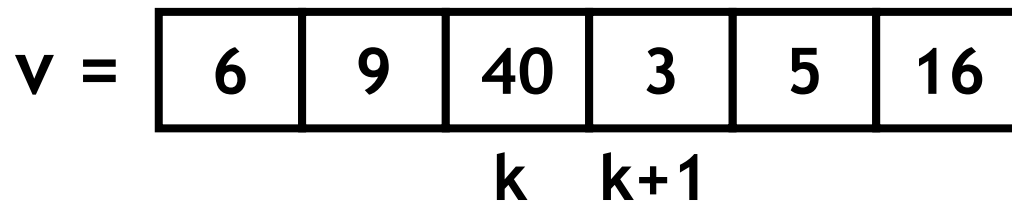
$v[k] > v[k+1]$? Não.

Bubble sort



$v[k] > v[k+1]$? Não.

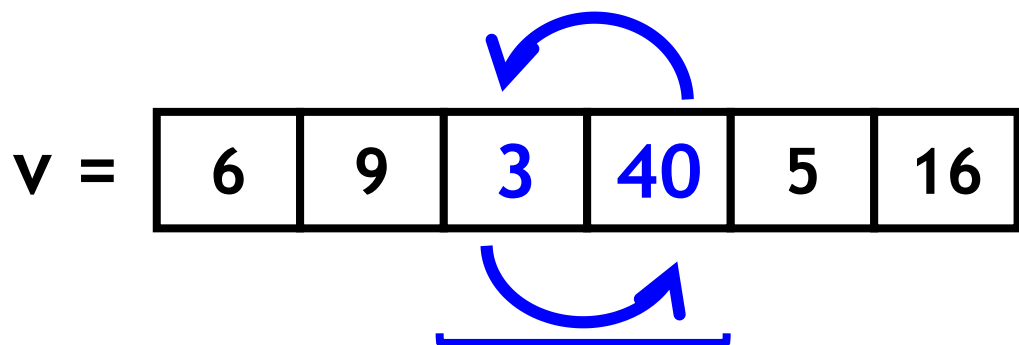
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

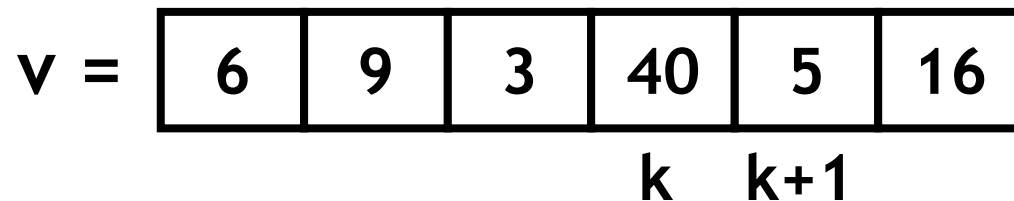
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

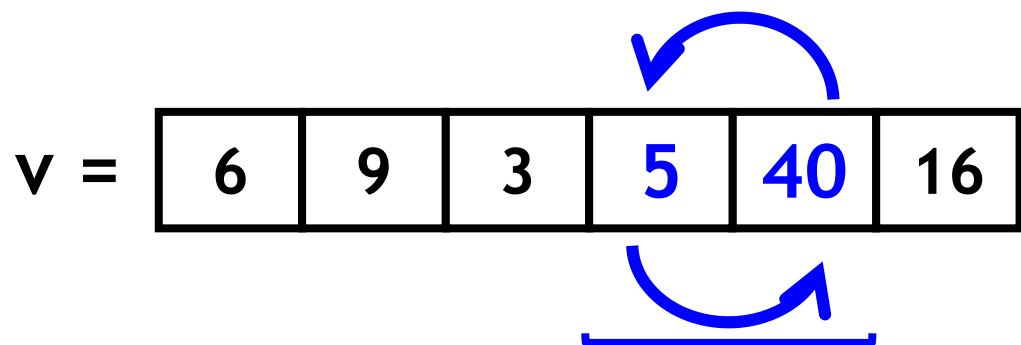
Bubble sort



$v[k] > v[k+1]$? **Sim.**

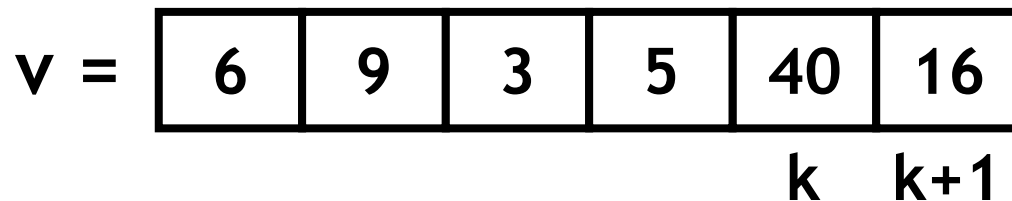
Troca os dois elementos.

Bubble sort



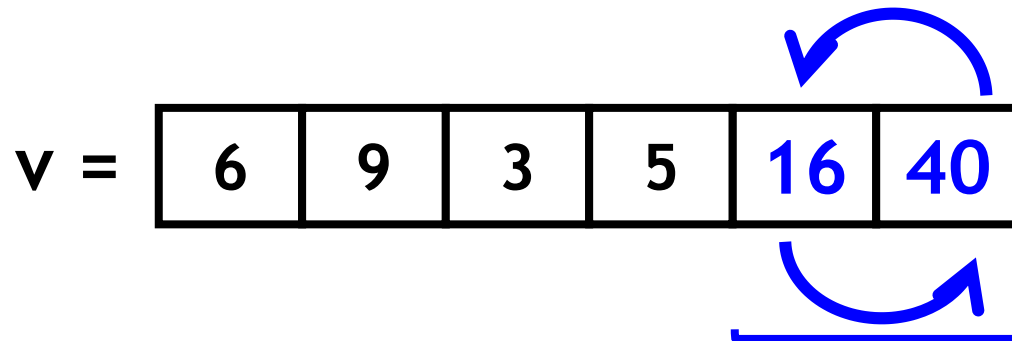
$v[k] > v[k+1]$? **Sim.**
Troca os dois elementos.

Bubble sort



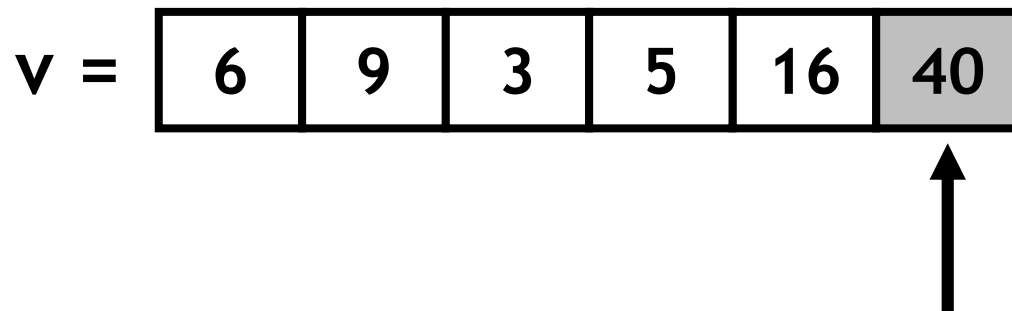
$v[k] > v[k+1]$? **Sim.**
Troca os dois elementos.

Bubble sort



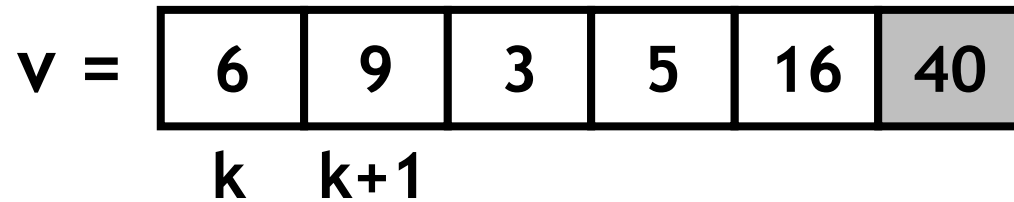
$v[k] > v[k+1]$? **Sim.**
Troca os dois elementos.

Bubble sort



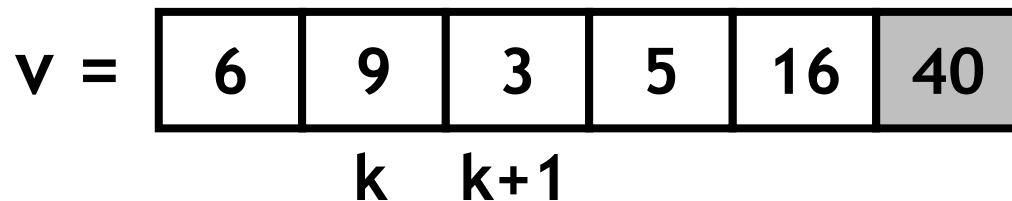
- Primeira iteração finalizada! Veja que o maior elemento está no final do vetor;
- Agora vamos repetir o processo, mas não precisamos ir até o último elemento.

Bubble sort



$v[k] > v[k+1]$? Não.

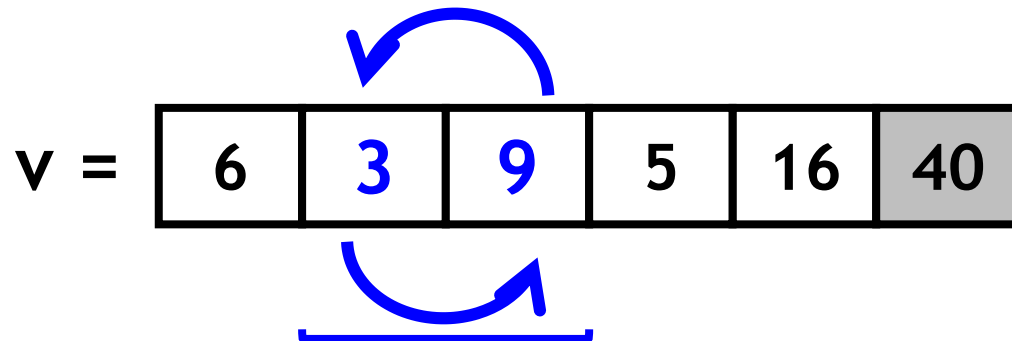
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

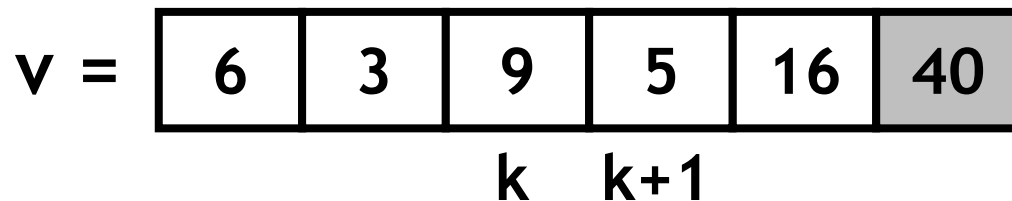
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

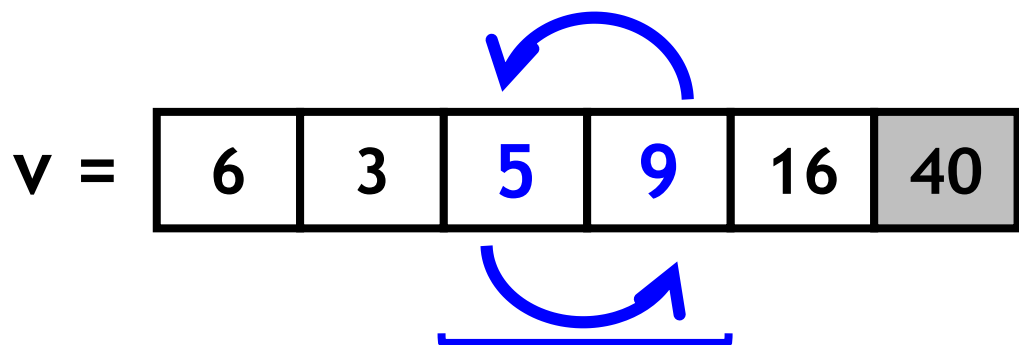
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

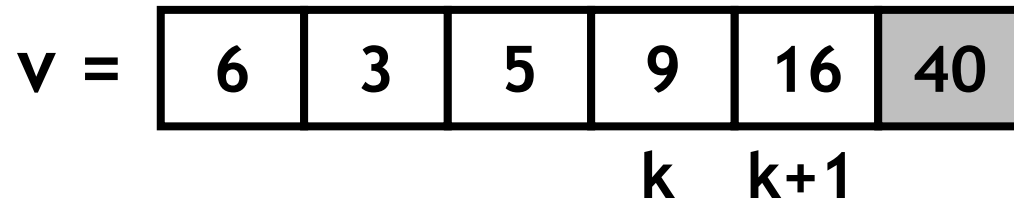
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

Bubble sort



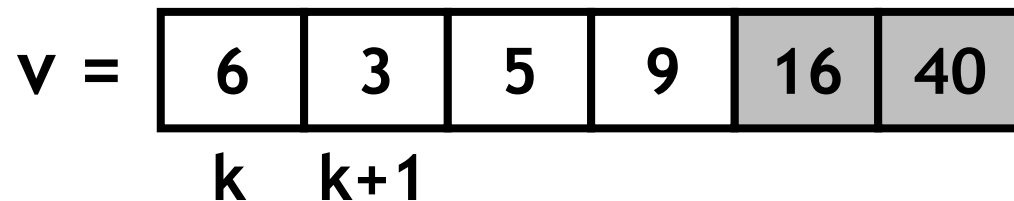
$v[k] > v[k+1]$? **Não.**

Bubble sort



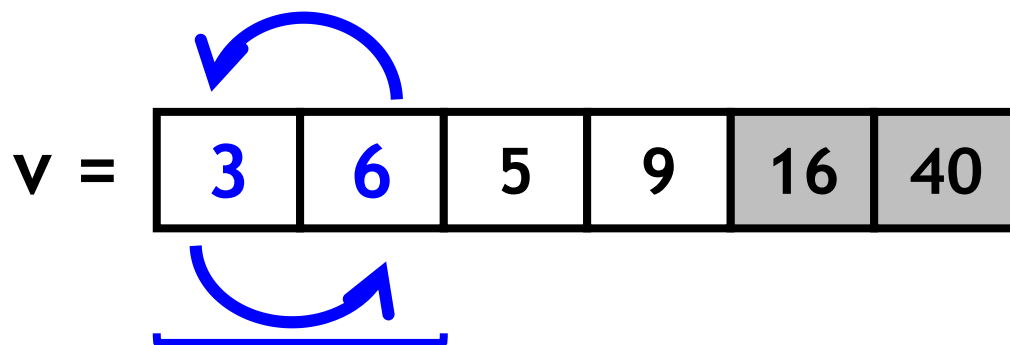
- Segunda iteração finalizada! Veja que o segundo maior elemento está no final do vetor;
- Agora vamos repetir o processo, mas não precisamos ir até o penúltimo elemento.

Bubble sort



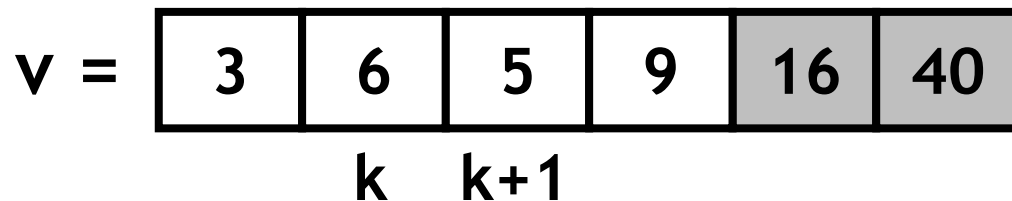
$v[k] > v[k+1]$? **Sim.**
Troca os dois elementos.

Bubble sort



$v[k] > v[k+1]$? **Sim.**
Troca os dois elementos.

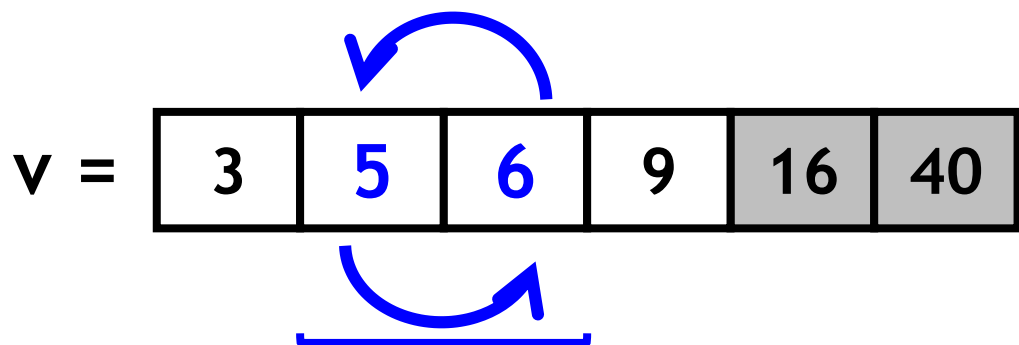
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

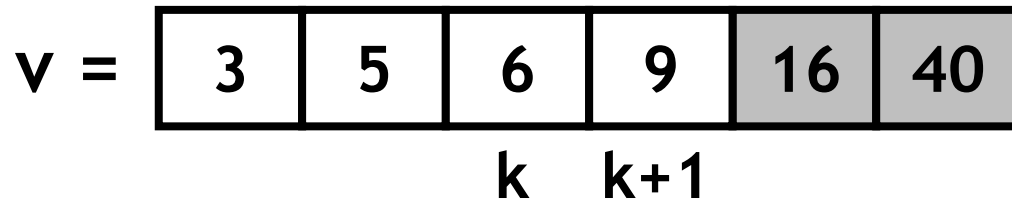
Bubble sort



$v[k] > v[k+1]$? **Sim.**

Troca os dois elementos.

Bubble sort



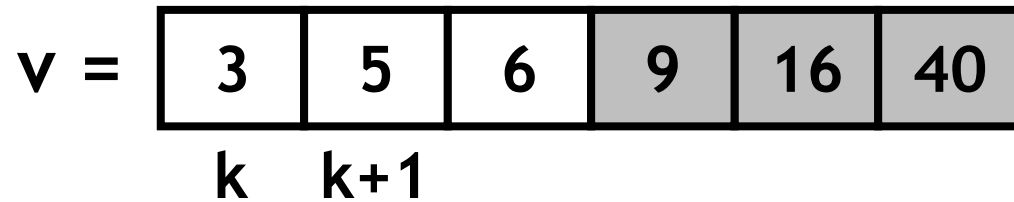
$v[k] > v[k+1]$? Não.

Bubble sort

$v =$

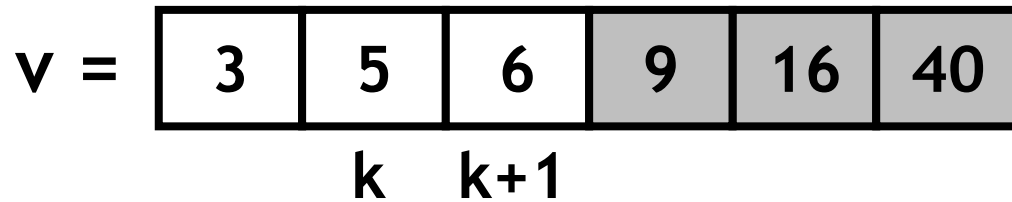
3	5	6	9	16	40
---	---	---	---	----	----

Bubble sort



$v[k] > v[k+1]$? Não.

Bubble sort



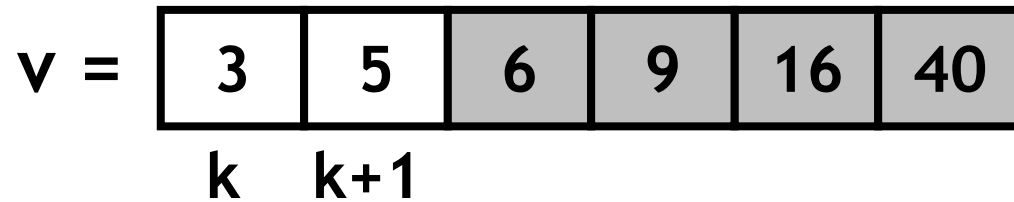
$v[k] > v[k+1]$? Não.

Bubble sort

v =

3	5	6	9	16	40
---	---	---	---	----	----

Bubble sort



$v[k] > v[k+1]$? Não.

Bubble sort

$v =$

3	5	6	9	16	40
---	---	---	---	----	----

Bubble sort

v =

3	5	6	9	16	40
---	---	---	---	----	----

Ordenação finalizada.

Ok, mas e para ordenar
em ordem decrescente?

Ok, mas e para ordenar
em ordem decrescente?

Apenas precisamos mudar a condição de
comparação de elementos de:

$$v[k] > v[k+1]$$

Para:

$$v[k] < v[k+1]$$

Quantas comparações de elementos do vetor realizamos nesse algoritmo?

```
void bubblesort(int *v, int n) {  
    int i, k;  
    for (i = 0; i < n - 1; i++)  
        for (k = 0; k < n - 1 - i; k++)  
            if (v[k] > v[k+1]) {  
                int tmp = v[k];  
                v[k] = v[k + 1];  
                v[k + 1] = tmp;  
            }  
}
```

A cada iteração,
percorre um elemento
a menos (-i).

Troca elementos
consecutivos se $v[k] > v[k+1]$

Análise do Bubble sort

- Número de comparações no pior caso:

$$\frac{n(n - 1)}{2}$$

- Número de comparações no melhor caso:

$$\frac{n(n - 1)}{2}$$

Será que podemos otimizar o Bubble sort?

Será que podemos otimizar o Bubble sort?

**Pense no caso que o vetor já está ordenado.
Há necessidade de continuar com as demais
iterações do algoritmo?**

Será que podemos otimizar o Bubble sort?

**Pense no caso que o vetor já está ordenado.
Há necessidade de continuar com as demais
iterações do algoritmo?**

NÃO!

**Ok, mas como sabemos que já está
ordenado?**

Será que podemos otimizar o Bubble sort?

**Pense no caso que o vetor já está ordenado.
Há necessidade de continuar com as demais
iterações do algoritmo?**

NÃO!

**Ok, mas como sabemos que já está
ordenado?**

**Se não foi realizada nenhuma troca na última
iteração, podemos parar a ordenação.**

Quantas comparações de elementos do vetor realizamos nesse algoritmo?

```
void bubblesort_es(int *v, int n) {  
    int i, k;  
    for (i = 0; i < n - 1; i++) {  
        int trocou = 0;  
        for (k = 0; k < n - 1 - i; k++) {  
            if (v[k] > v[k+1]) {  
                int tmp = v[k];  
                v[k] = v[k + 1];  
                v[k + 1] = tmp;  
                trocou = 1;  
            }  
        }  
        if (!trocou) break;  
    }  
}
```

A cada iteração,
percorre um elemento
a menos (-i).

Troca elementos
consecutivos se $v[k] > v[k+1]$

Outra implementação (sem usar o *break*)

```
void bubblesort_es(int *v, int n) {
    int i, k, trocou = 1;
    for (i = 0; i < n - 1 && trocou; i++) {
        trocou = 0;
        for (k = 0; k < n - 1 - i; k++)
            if (v[k] > v[k+1]) {
                int tmp = v[k];
                v[k] = v[k + 1];
                v[k + 1] = tmp;
                trocou = 1;
            }
    }
}
```

Análise do bubble sort (com a modificação que vimos)

- Número de comparações no pior caso:

$$\frac{n(n - 1)}{2}$$

- Número de comparações no melhor caso:

$$n - 1$$

Exemplos dos algoritmos

- Selection sort:
 - <https://visualgo.net/en/sorting?slide=7>
- Insertion sort:
 - <https://visualgo.net/en/sorting?slide=8>
- Bubble sort:
 - <https://visualgo.net/en/sorting?slide=6>
- Bubble sort (early stopping):
 - <https://visualgo.net/en/sorting?slide=6-2>

Busca

Busca

- **Problema de busca:** dados um vetor v e um valor x , verificar se o elemento x está em v . Se estiver, retornar o índice i da posição de x em v . Caso contrário, retorne -1 .

Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento x está em v .

$$x = 8$$

$v =$

6	9	40	3	8	16
---	---	----	---	---	----



$$6 == x?$$

Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento x está em v .

$x = 8$

$v =$

6	9	40	3	8	16
---	---	----	---	---	----



$9 == x?$

Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento x está em v .

$x = 8$

$v =$

6	9	40	3	8	16
---	---	----	---	---	----



$40 == x?$

Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento x está em v .

$$x = 8$$

$v =$

6	9	40	3	8	16
---	---	----	---	---	----



$$3 == x?$$

Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento x está em v .

$x = 8$

$v =$

6	9	40	3	8	16
---	---	----	---	---	----



$8 == x?$ Sim
Retorne 4 (índice do elemento no vetor v)

Busca linear (ou sequencial)

- Algoritmo de busca linear:

```
int busca_linear(int *v, int n, int x) {  
    int i;  
    for (i = 0; i < n; i++) | Percorre todo o vetor.  
        if (v[i] == x) | Compara cada elemento com x.  
            return i;  
    return -1;  
}
```


Quantas comparações de elementos do vetor realizamos nesse algoritmo?

Busca linear (ou sequencial)

- Algoritmo de busca (versão sem *return* no for):

```
int busca_linear2(int *v, int n, int x) {  
    int i, indice_encontrado = -1;  
    for (i = 0; i < n && indice_encontrado == -1; i++)  
        if (v[i] == x) | Compara cada elemento com x.  
            indice_encontrado = i;  
    return indice_encontrado;  
}
```

Percorre todo o vetor.



Análise da busca linear

- Número de comparações no pior caso (x é o último elemento ou não está presente no vetor):

n

- Número de comparações no melhor caso (x é o primeiro elemento):

1

Busca binária

Busca binária

- Algoritmo de busca mais eficiente, mas requer que o vetor esteja ordenado;
- A busca é realizada dividindo o vetor, até finalizar a busca.

Busca binária

$x = 32$

	0	1	2	3	4	5	6	7	8	9
$v =$	3	6	9	10	18	25	28	32	38	40



esq



$$meio = \lfloor (esq + dir) / 2 \rfloor = 4$$

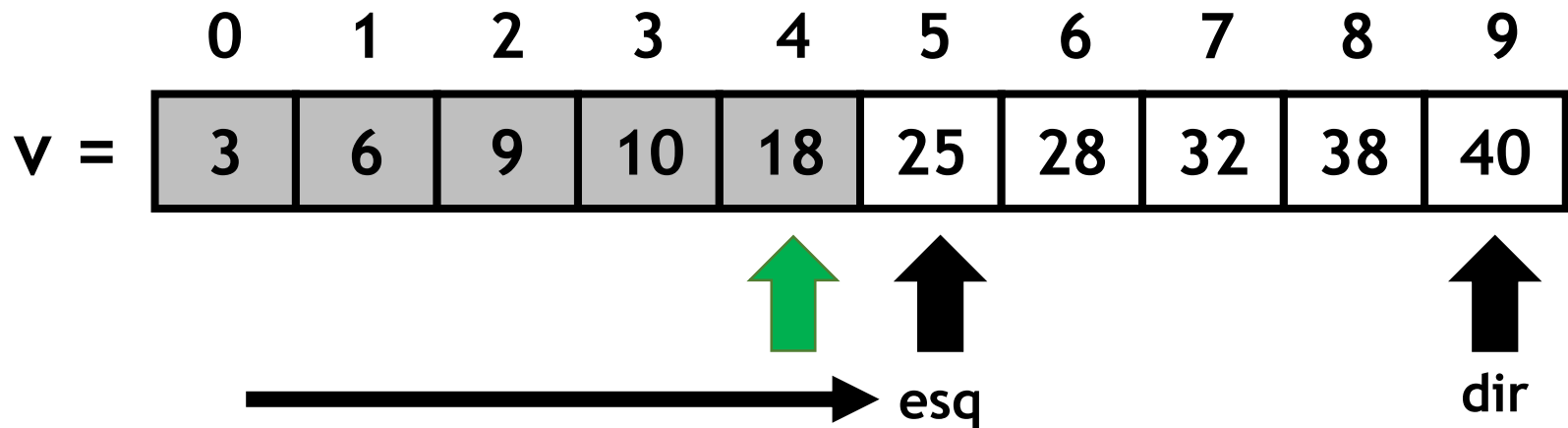
$v[meio] == 32?$ Não.



dir

Busca binária

$x = 32$

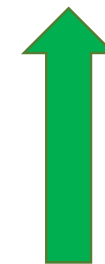


Busca binária

$x = 32$

	0	1	2	3	4	5	6	7	8	9
$v =$	3	6	9	10	18	25	28	32	38	40

↑
esq



↑
dir

$$meio = \lfloor (esq + dir) / 2 \rfloor = 7$$

$v[meio] == 32?$ Sim.

Retorna o índice 7

Busca binária (outro exemplo)

$x = 8$

	0	1	2	3	4	5	6	7	8	9
v =	3	6	9	10	18	25	28	32	38	40



esq



$$meio = \lfloor (esq + dir) / 2 \rfloor = 4$$

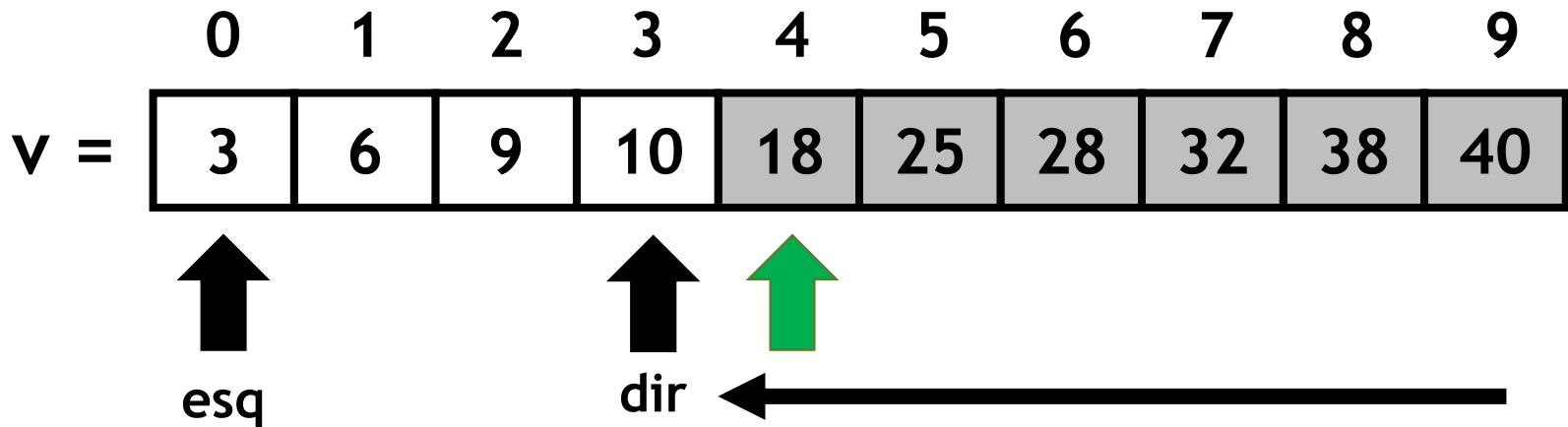
$v[meio] == 8$? Não.



dir

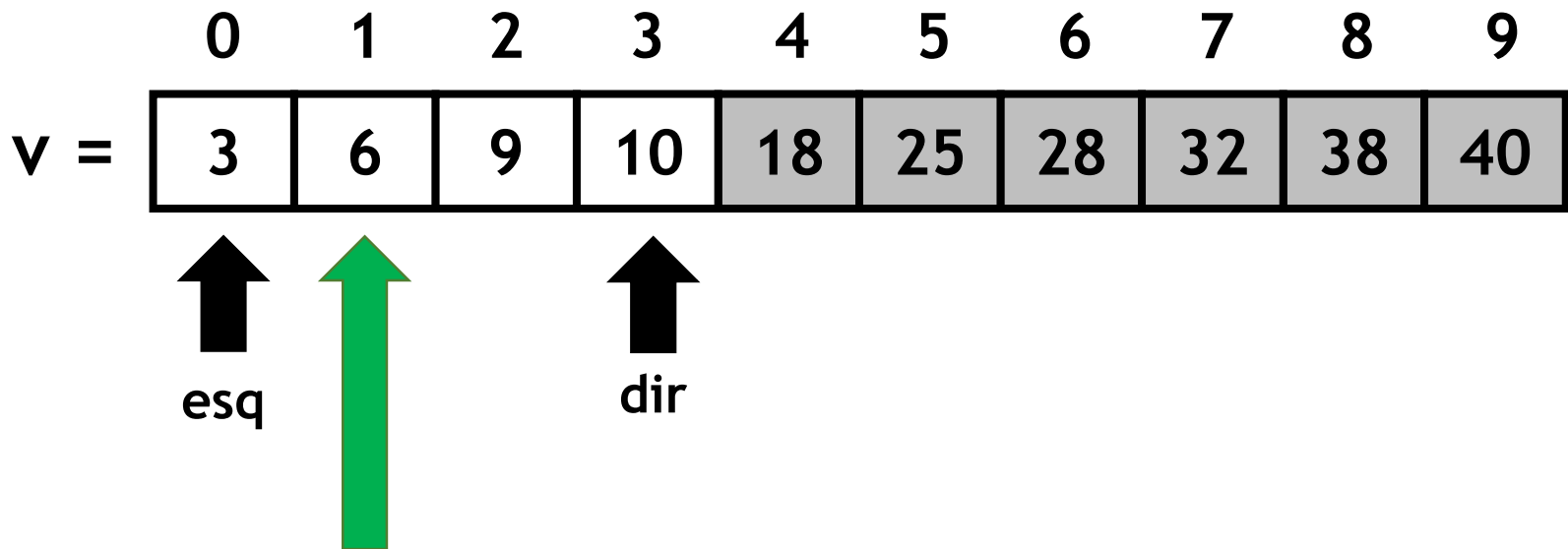
Busca binária

$x = 8$



Busca binária

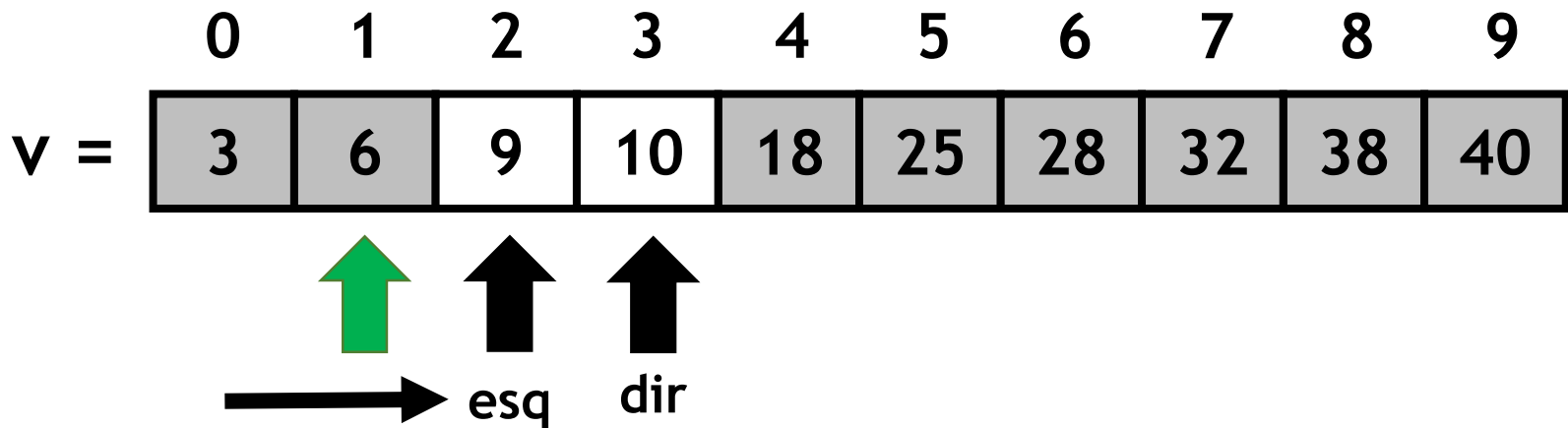
$x = 8$



$meio = \lfloor (esq + dir) / 2 \rfloor = 1$
 $v[meio] == 8?$ Não.

Busca binária

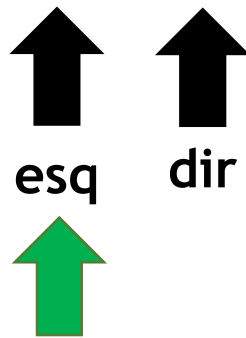
$x = 8$



Busca binária

$x = 8$

	0	1	2	3	4	5	6	7	8	9
v =	3	6	9	10	18	25	28	32	38	40

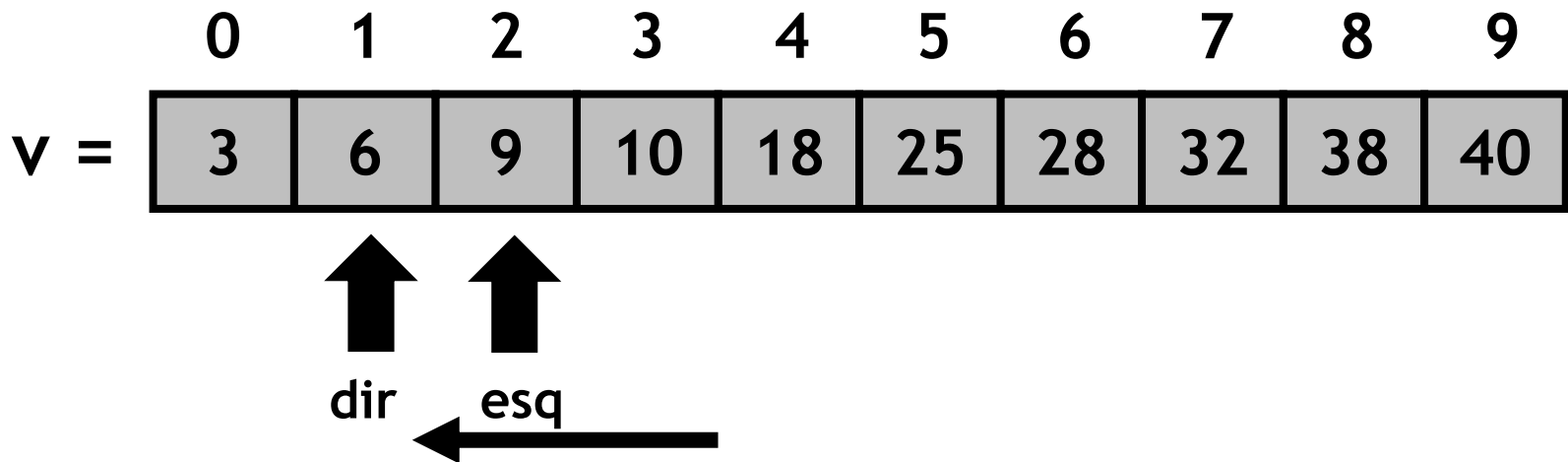


$$meio = \lfloor (esq + dir) / 2 \rfloor = 2$$

$v[meio] == 8?$ Não.

Busca binária

$x = 8$



$dir < esq \rightarrow$ Fim da busca. Elemento não foi encontrado! Retorne -1.

Busca binária

- Algoritmo de busca binária:

```
int busca_binaria(int *v, int n, int x) {
    int esq = 0, dir = n-1;
    while (esq <= dir) {
        int meio = (esq + dir) / 2;
        if (v[meio] == x)
            return meio;
        else if (v[meio] < x)
            esq = meio + 1;
        else
            dir = meio - 1;
    }
    return -1;
}
```

Exercício 1

- Crie vetor de Aluno (struct), que possui os membros RA e Nota;
- Implemente uma função para ordenar o vetor de struct, que possui um parâmetro **modo**:
 - modo = 0 → ordenação crescente pelo RA;
 - modo = 1 → ordenação decrescente pelo RA;
 - modo = 2 → ordenação crescente pela Nota;
 - modo = 3 → ordenação decrescente pela Nota.

Exercício 2

- Escreva uma função para verificar se um vetor está em ordem crescente.

Exercício 3

- Implemente a versão recursiva de cada um dos algoritmos de ordenação e de busca que vimos:
 - Selection sort
 - Insertion sort
 - Bubble sort
 - Busca linear
 - Busca binária

Referências

- Slides do Prof. Monael Pinheiro Riberio:
 - <https://sites.google.com/site/aed2018q1/>
- Slides do Prof. Jesús P. Mena-Chalco:
 - <http://professor.ufabc.edu.br/~jesus.mena/courses/mcta028-3q-2017/>
- Visualising Data Structures and algorithms through animation:
 - <https://visualgo.net/en>

Bibliografia básica

- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª edição. São Paulo, SP: Prentice Hall, 2005.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2ª edição. Rio de Janeiro, RJ: Campus, 2002.

Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3a edição. Rio de Janeiro, RJ: LTC, 1994.
- TEWNENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.