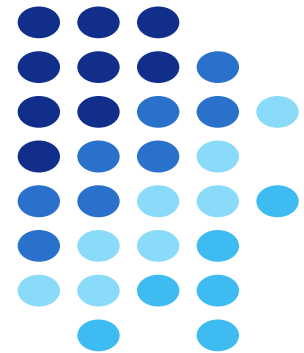
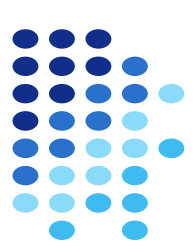


Universidade Federal de Sergipe
Departamento de Sistemas de Informação
SINF0007 – Estrutura de Dados II
Arquivos Sequenciais



3

Prof. Dr. Raphael Pereira de Oliveira
raphael.oliveira@academico.ufs.br



Relembrando

- Uma tabela ou arquivo é um conjunto de registros que possuem a mesma estrutura

Relembrando: Operações Usuais

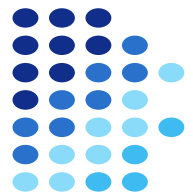
- **Criação**: alocação e inicialização da área de dados, assim como de seus descritores
- **Destruição**: liberação da área de dados e descritores usados na representação da tabela
- **Inserção**: inclusão de novo registro na tabela
- **Exclusão**: remoção de um registro da tabela
- **Alteração**: modificação dos valores de um ou mais atributos/campos da tabela
- **Consulta**: obtenção dos valores de todos os campos de um registro, dada uma chave de entrada

Consulta

- Dada uma tabela com **10 registros** e o **valor de uma chave**, como encontrar o registro correspondente na tabela?

Consulta

- Dada uma tabela com **10 registros** e o **valor de uma chave**, como encontrar o registro correspondente na tabela?
- **Solução:** busca sequencial dentro do arquivo
 - Recuperar o primeiro registro, testar a chave
 - Caso não seja a chave pesquisada, recuperar o segundo registro, testar, e assim por diante, até encontrar o registro ou chegar ao final do arquivo



Algoritmo Busca Sequencial

```
proc sequencial (tab: tabela; ch: chave; e: int)
```

```
{ENTRADA:
```

```
    tab: tabela onde será feita a pesquisa
```

```
    ch: chave que está sendo procurada
```

```
SAÍDA:
```

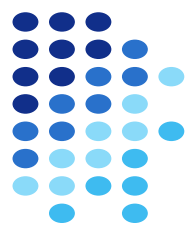
```
    e: endereço do registro procurado
```

```
    se e=0 não existe registro com a chave ch
```

```
}
```

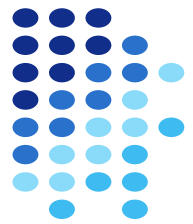
Algoritmo Busca Sequencial

```
proc sequencial (tab: tabela; ch: chave; e: int)
var n, i: int
begin
    n := #tab
    e := 0;
    for i:= 1 to n do
        if tab[i].chave = ch then
            begin
                e := i;
                escape;
            end;
    end; {sequencial}
```



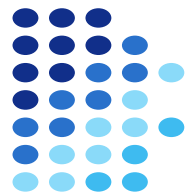
Simular a execução do Algoritmo para procurar a chave **201**

Chave	A	B	C	D
300	A1	B1	C1	D1
200	A2	B2	C2	D2
215	A3	B3	C3	D3
201	A4	B4	C4	D4
230	A5	B5	C5	D5
205	A6	B6	C6	D6
225	A7	B7	C7	D7
280	A8	B8	C8	D8



E para 1 milhão de registros?





Solução de Contorno

- As tabelas são mantidas ordenadas pela chave primária
- Deste modo não é necessário ir até o final do arquivo para saber que uma determinada chave não está no arquivo
- Assim que uma chave maior do que a chave que está sendo procurada for encontrada, sabe-se que a chave procurada não está lá

Algoritmo Busca Sequencial em Tabela Ordenada

```
proc seq_ordenado (tab: tabela; ch: chave; e: int)
```

```
{ENTRADA:
```

```
    tab: tabela onde será feita a pesquisa
```

```
    ch: chave que está sendo procurada
```

```
SAÍDA:
```

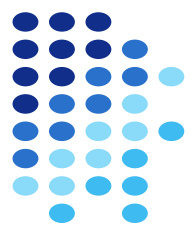
```
    e: endereço do registro procurado
```

```
        se e=0 não existe registro com a chave ch
```

```
}
```

Algoritmo Busca Sequencial em Tabela Ordenada

```
proc seq_ordenado (tab: tabela; ch: chave; e: int)
var n, i: int
begin
    n := #tab
    e := 0;
    i := 1;
    while (tab[i].chave < ch) and (i < n) do
        i := i + 1;
    if tab[i].chave = ch then
        e := i;
end; {seq_ordenado}
```



Simular a execução do Algoritmo para procurar a chave **202**

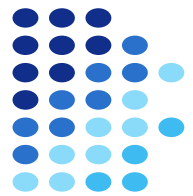
Chave	A	B	C	D
200	A1	B1	C1	D1
201	A2	B2	C2	D2
205	A3	B3	C3	D3
215	A4	B4	C4	D4
225	A5	B5	C5	D5
230	A6	B6	C6	D6
280	A7	B7	C7	D7
300	A8	B8	C8	D8

Métodos de Ordenação

- Vários métodos podem ser aplicados
 - Possível solução:
 - **métodos de ordenação em memória**
1. Ler arquivo e armazenar os dados num *array* em memória
 2. Ordenar o *array*
 3. Gravar novo arquivo com os dados ordenados

Convenção

- Os algoritmos que veremos assumem que todas as chaves do arquivo estão num **vetor A**
- Na prática isso será algo como **A[i].chave**



Exercício

- Construir um algoritmo para ordenar os seguintes valores

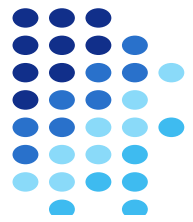
6	8	2	1	7	9	3
---	---	---	---	---	---	---

Ordenação por Inserção

- **Insertion Sort**
- Nome do método se deve ao fato de que no **i -ésimo** passo ele insere o **i -ésimo** elemento **$A[i]$** na posição correta entre **$A[1]$, $A[2]$, ..., $A[i-1]$** que já foram previamente ordenados

Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2



Funcionamento

1. **Assume que o primeiro valor já está ordenado**
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

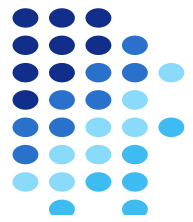
A

300

200

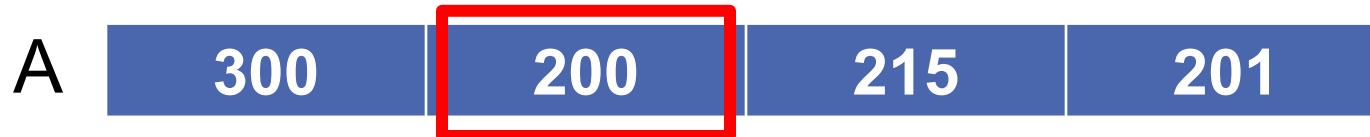
215

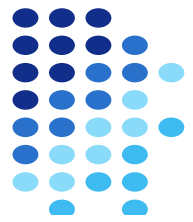
201



Funcionamento

1. Assume que o primeiro valor já está ordenado
2. **Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar**
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

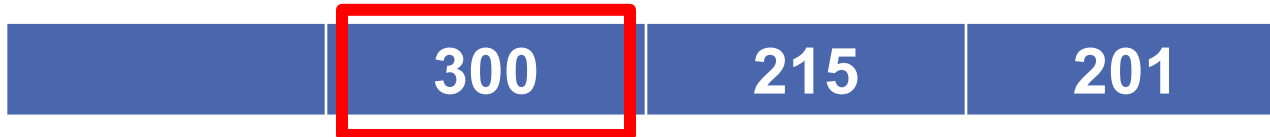


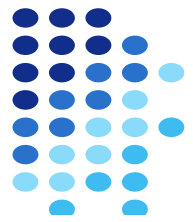


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. **Abre espaço no vetor para encaixar o valor na posição correta**
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

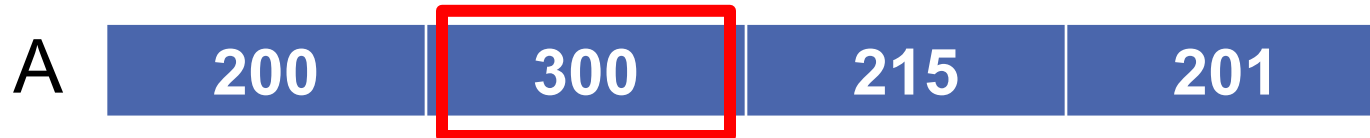
A

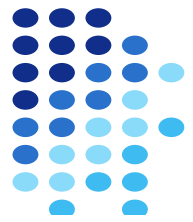




Funcionamento

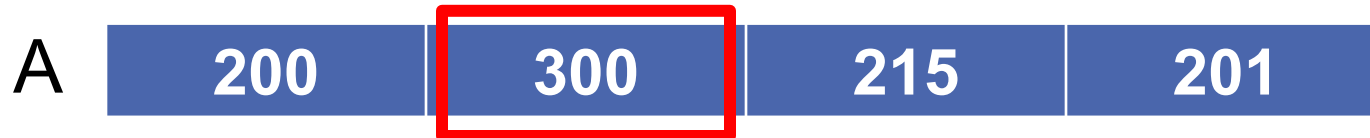
1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. **Encaixa o valor na posição correta**
5. Se vetor ainda não terminou, volta para o passo 2

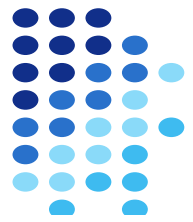




Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. **Se vetor ainda não terminou, volta para o passo 2**



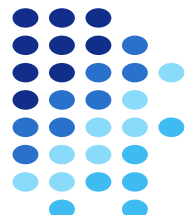


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. **Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar**
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

A

200	300	215	201
-----	-----	-----	-----

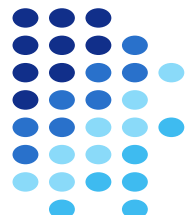


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. **Abre espaço no vetor para encaixar o valor na posição correta**
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

A



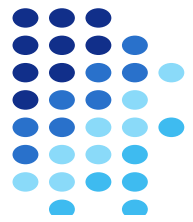


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. **Encaixa o valor na posição correta**
5. Se vetor ainda não terminou, volta para o passo 2

A



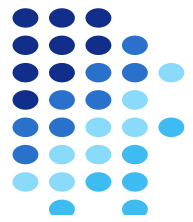


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. **Se vetor ainda não terminou, volta para o passo 2**

A



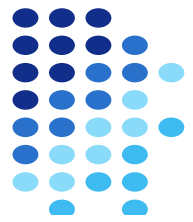


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. **Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar**
3. Abre espaço no vetor para encaixar o valor na posição correta
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

A

200	215	300	201
-----	-----	-----	-----

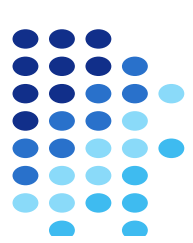


Funcionamento

1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. **Abre espaço no vetor para encaixar o valor na posição correta**
4. Encaixa o valor na posição correta
5. Se vetor ainda não terminou, volta para o passo 2

A

200		215	300
-----	--	-----	-----



Funcionamento

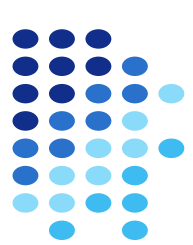
1. Assume que o primeiro valor já está ordenado
2. Pega o próximo valor, compara com os anteriores até descobrir em que posição ele deveria estar
3. Abre espaço no vetor para encaixar o valor na posição correta
4. **Encaixa o valor na posição correta**
5. Se vetor ainda não terminou, volta para o passo 2

A

200	201	215	300
-----	-----	-----	-----

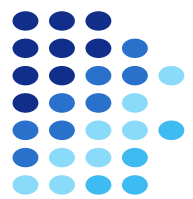
Ordenação por Inserção (**Insertion Sort**)

```
proc insertionSort(A: array, size: int)
{  ENTRADA:
    A: array com as chaves
    size: tamanho do array
  SAÍDA:
    A: array ordenado
}
begin
  for j := 2 to size do
  begin
    key := A[j];
    i := j - 1;
    while (i > 0) and (A[i] > key) do
    begin
      A[i+1] := A[i];
      i := i - 1;
    end;
    A[i+1] := key
  end;
end;
```



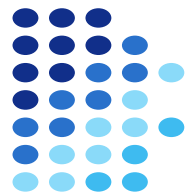
Complexidade do **Insertion Sort**

- Muito custoso:
 - Complexidade pior caso $O(n^2)$
 - Complexidade caso médio $O(n^2)$
 - Complexidade melhor caso $O(n)$



Ordenação com o **Bubble Sort**

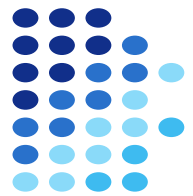
- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca



Ordenação com o **Bubble Sort**

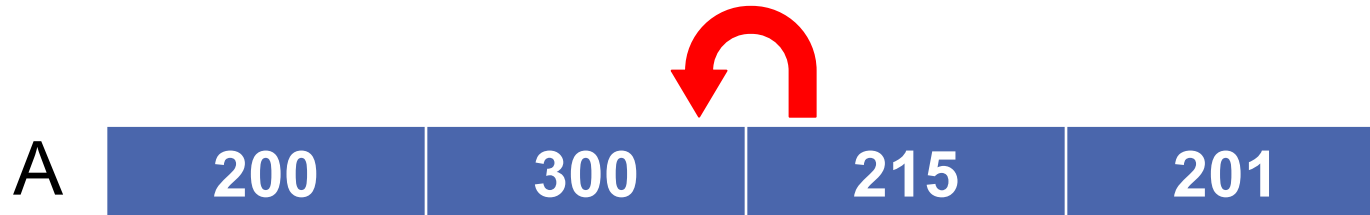
- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca

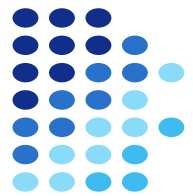




Ordenação com o **Bubble Sort**

- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca

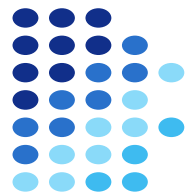




Ordenação com o **Bubble Sort**

- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca

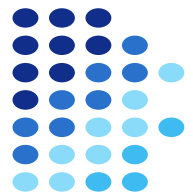




Ordenação com o **Bubble Sort**

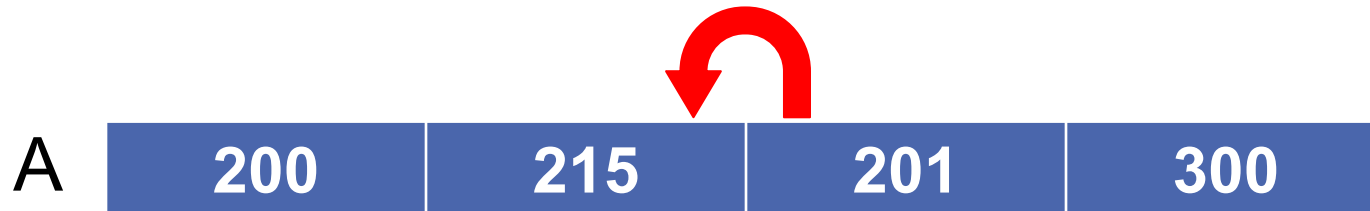
- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca

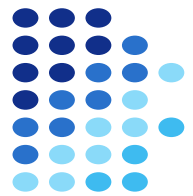




Ordenação com o **Bubble Sort**

- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca



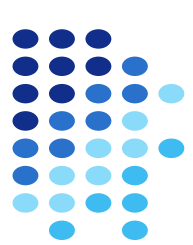


Ordenação com o **Bubble Sort**

- Comparar cada valor com o sucessor
- Se valor for menor que o sucessor, troca de posição
- Faz várias passadas
- Para quando não houver mais troca

A

200	201	215	300
-----	-----	-----	-----

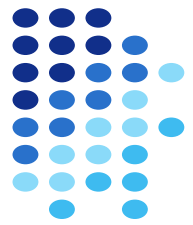


Ordenação com o **Bubble Sort**

```
proc bubbleSort(A: registro, size: int)
{ ENTRADA:
    A: registro com as chaves
    size: tamanho do registro
  SAÍDA:
    A: registro ordenado
}
```


Ordenação com o **Bubble Sort**

```
proc bubbleSort(A: registro, size: int)
begin
    houveTroca := true; # uma variável de controle
    m := size - 1;
    while houveTroca do
        begin
            houveTroca := false
            for i := 1 to m do
                begin
                    if (A[i] > A[i + 1]) then
                        begin
                            ch := A[i];
                            A[i] := A[i+1];
                            A[i+1] := ch;
                            k := i;
                            houveTroca := true;
                        end;
                    end;
                end;
            m := k; //local da última troca efetuada
        end;
    end;
```

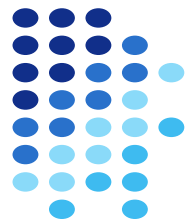


Complexidade do **Bubble Sort**

- Muito custoso:
 - Complexidade pior caso $O(n^2)$
 - Complexidade caso médio $O(n^2)$
 - Complexidade melhor caso $O(n)$

Ordenação com o **Quick Sort**

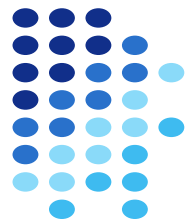
- Método **Dividir para Conquistar**
 1. Escolha um elemento da lista para ser o pivô (escolha o maior elemento entre os dois primeiros elementos mais à esquerda do vetor)
 2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**
 3. Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores



Funcionamento do Quick Sort

1. Escolha um elemento da lista para ser o pivô (escolha o maior elemento entre os dois primeiros elementos mais à esquerda do vetor)
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**
3. Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores



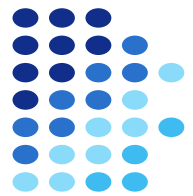


Funcionamento do Quick Sort

1. Escolha um elemento da lista para ser o pivô (escolha o maior elemento entre os dois primeiros elementos mais à esquerda do vetor)
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**
3. Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

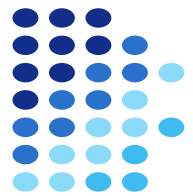
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



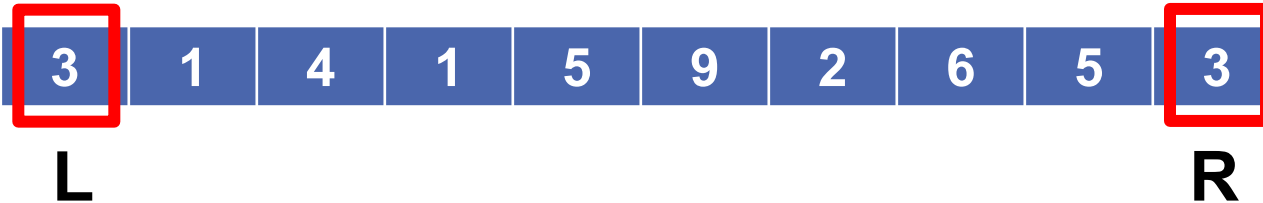
Funcionamento do Quick Sort

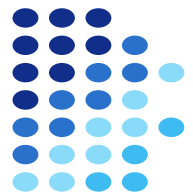
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3





Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

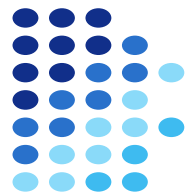
- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

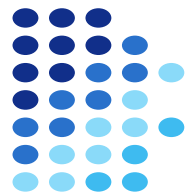
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R

Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

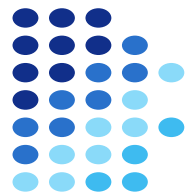
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

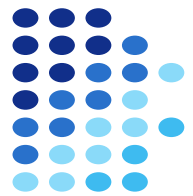
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

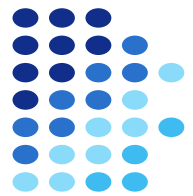
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

3	1	4	1	5	9	2	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



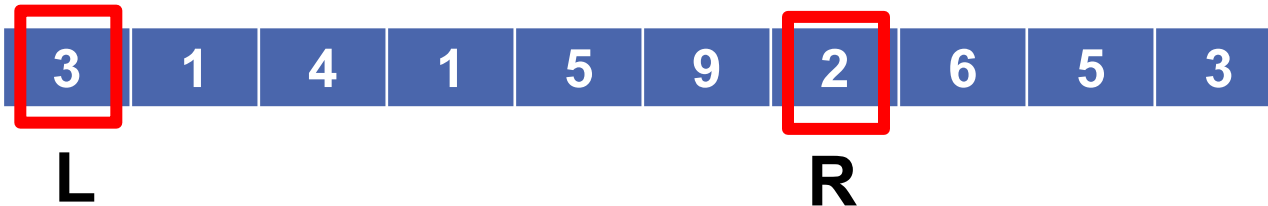
Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3



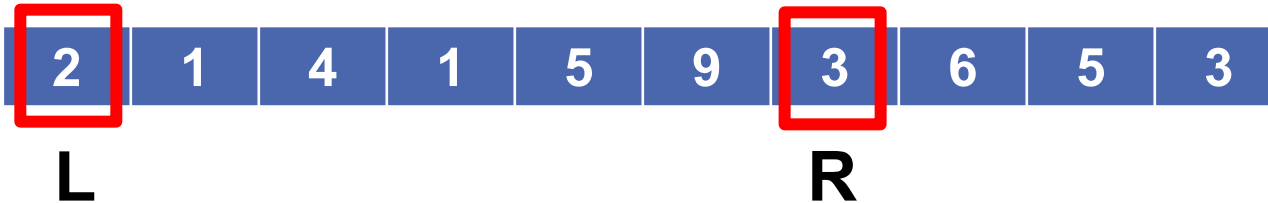
Funcionamento do Quick Sort

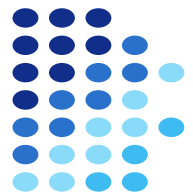
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3





Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

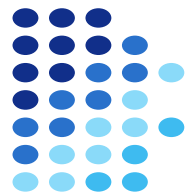
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

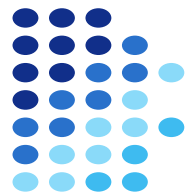
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

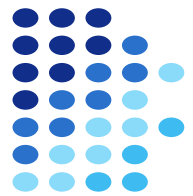
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

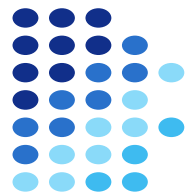
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

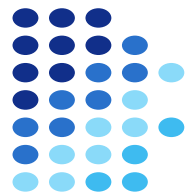
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

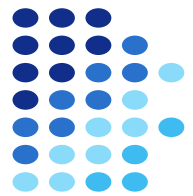
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

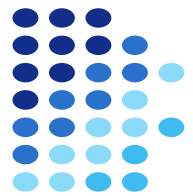
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 3

2	1	4	1	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L R



Funcionamento do Quick Sort

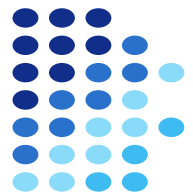
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3





Funcionamento do Quick Sort

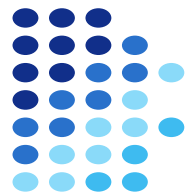
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3





Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

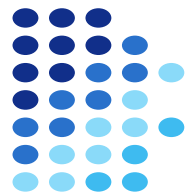
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	1	4	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

L R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

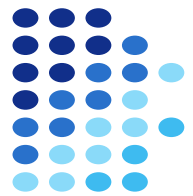
Repita até que o índice L seja maior que o índice R

- i. Troque o valor no índice L com o valor no índice R
- ii. Enquanto o valor no índice L for menor que o pivô, incremente L
- iii. Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	1	4	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

LR



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

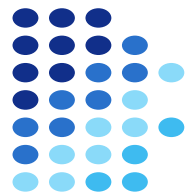
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 3

2	1	1	4	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

LR



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

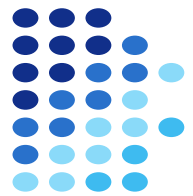
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 3

2	1	1	4	5	9	3	6	5	3
---	---	---	---	---	---	---	---	---	---

R L



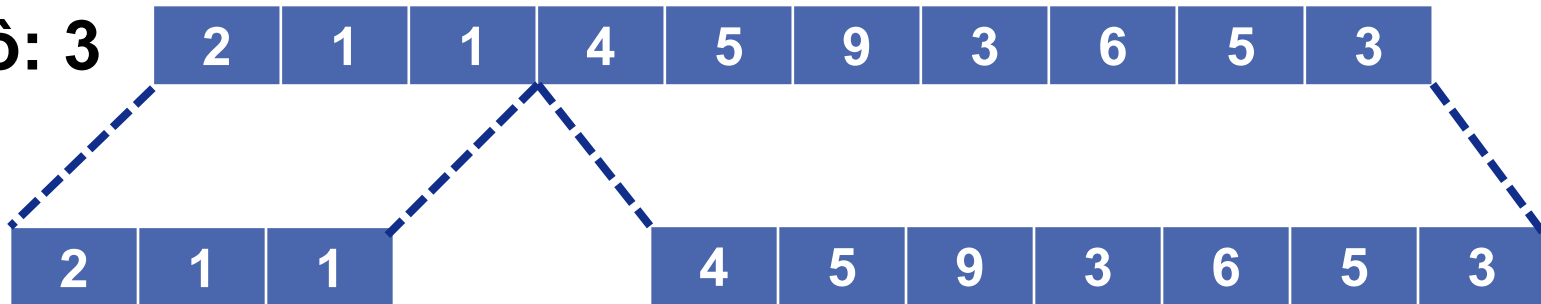
Funcionamento do Quick Sort

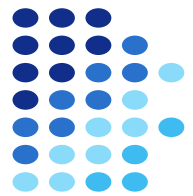
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

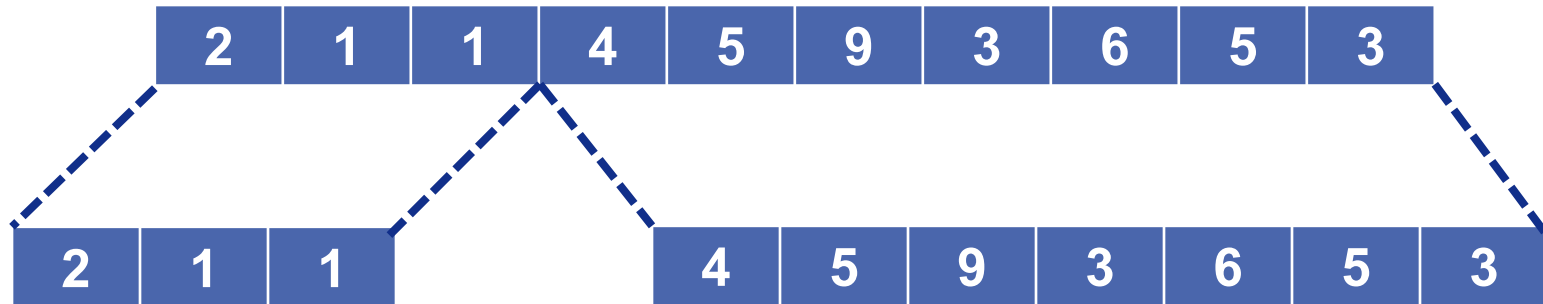
Pivô: 3





Funcionamento do Quick Sort

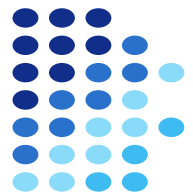
1. Escolha um elemento da lista para ser o pivô (escolha o maior elemento entre os dois primeiros elementos mais à esquerda do vetor)
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**
3. **Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores**



Funcionamento do Quick Sort

1. Escolha um elemento da lista para ser o pivô (escolha o maior elemento entre os dois primeiros elementos mais à esquerda do vetor)
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**
3. Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores





Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- i. Troque o valor no índice L com o valor no índice R
- ii. Enquanto o valor no índice L for menor que o pivô, incremente L
- iii. Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 2

2	1	1
---	---	---

L

R

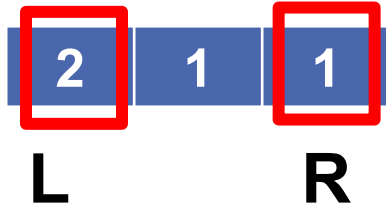
Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 2



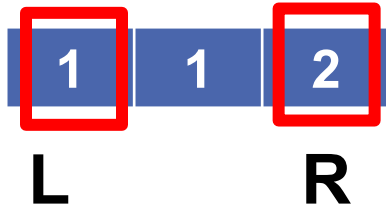
Funcionamento do Quick Sort

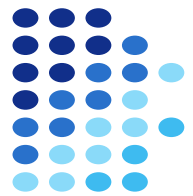
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R**
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 2





Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

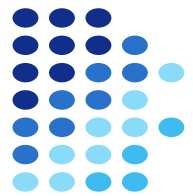
- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 2

1	1	2
---	---	---

L

R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

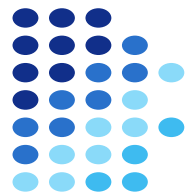
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrescente R

Pivô: 2



L R



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

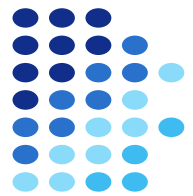
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L**
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 2



LR



Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decmente R

Pivô: 2



LR

Funcionamento do Quick Sort

2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**

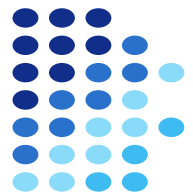
Repita até que o índice L seja maior que o índice R

- Troque o valor no índice L com o valor no índice R
- Enquanto o valor no índice L for menor que o pivô, incremente L
- Enquanto o valor no índice R for maior ou igual ao pivô, decrémente R

Pivô: 2



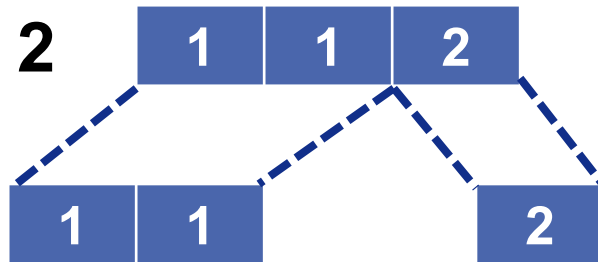
R L



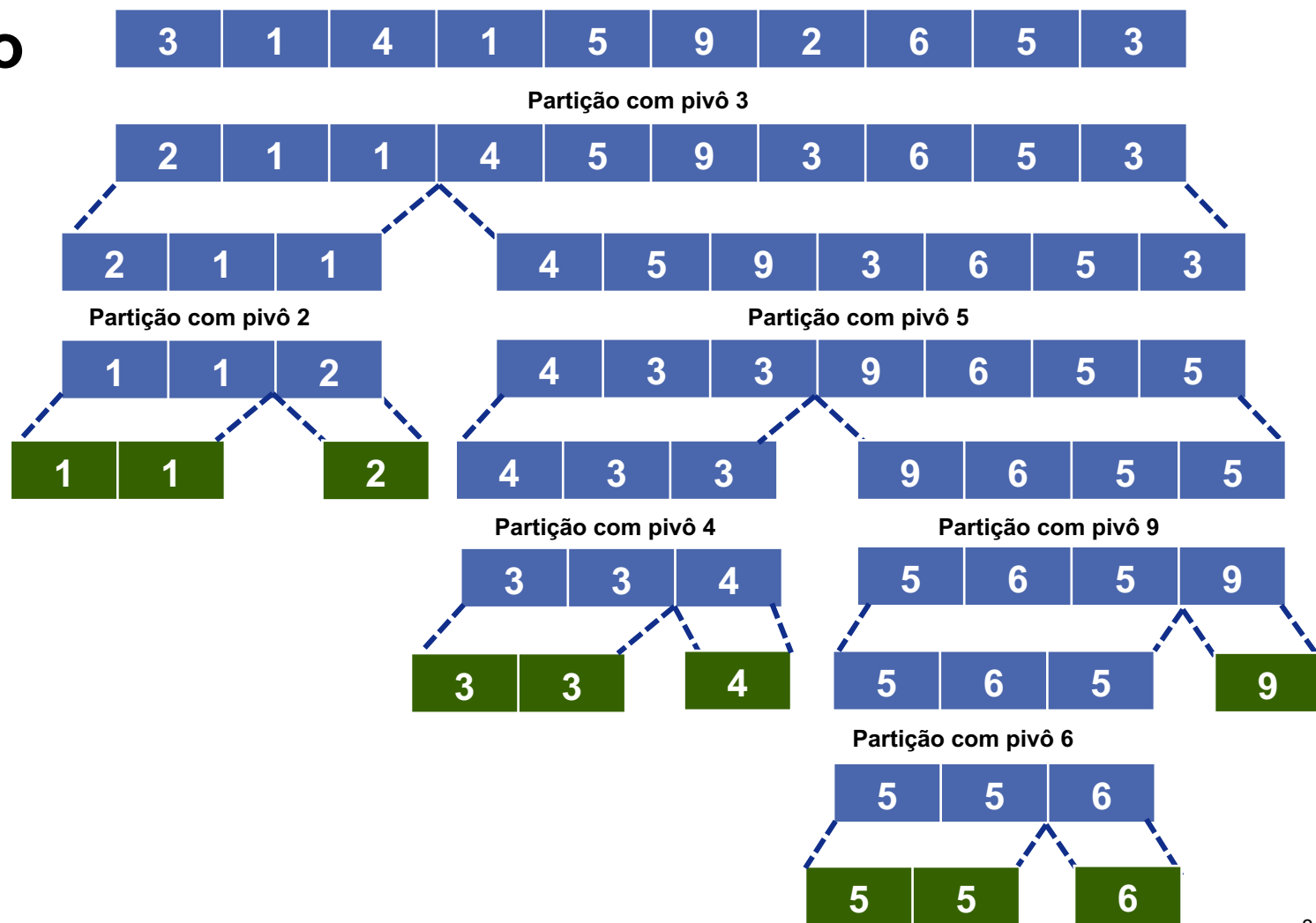
Funcionamento do Quick Sort

1. Escolha um elemento da lista para ser o pivô (escolha o maior elemento entre os dois primeiros elementos mais à esquerda do vetor)
2. Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores ou iguais a ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada **partição**
3. Recursivamente ordene a sublista dos elementos menores e a sublista dos elementos maiores

Pivô: 2



Seguindo o Algoritmo teremos:



Ordenação com o **Quick Sort**

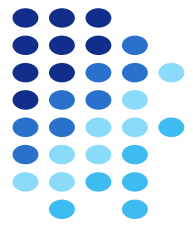
```
function findPivot (i,j: integer): integer
{retorna 0 se A[i], ..., A[j] possuem chaves idênticas caso
contrário retorna o índice da maior das duas chaves distintas
mais à esquerda }
var firstkey: keytype; {valor da primeira chave encontrada}
k: integer;
begin
    firstkey := A[i];
    for k:= i + 1 to j do {procura uma chave diferente}
        if A[k] > firstkey then {seleciona a maior chave}
            return (k)
        else if A[k] < firstkey then
            return (i);
    return (0); {não encontrou chaves diferentes}
end; {fim da função}
```

Ordenação com o Quick Sort

```
function partition (i,j: integer; pivot:keytype): integer
{particiona A[i], ..., A[j] tal que as chaves menores que o pivot
são colocadas à esquerda e as maiores ou iguais são colocadas à
direita. Retorna o início do grupo da direita }
var l,r: integer; {cursosores}
begin
    l := i;
    r := j;
    repeat
        swap (A[l],A[r]);
        while A[l] < pivot do
            l := l + 1;
        while A[r] >= pivot do
            r := r - 1;
    until l > r;
    return (l);
end; {fim da função}
```

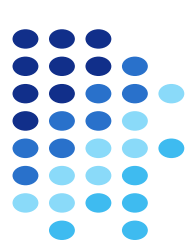
Ordenação com o Quick Sort

```
function quicksort (i,j: integer): integer
{ ordena os elementos A[i], ..., A[j] }
var pivot: keytype; {o valor do pivot }
pivotindex: integer; {o índice do elemento de A cuja chave é o pivot }
k: integer; {início do índice para o grupo de elementos >= pivot }
begin
    pivotindex := findpivot(i,j);
    if pivotindex <> 0 then
        begin {não faz nada se todas as chaves são iguais}
            pivot := A[pivotindex].key;
            k := partition(i,j,pivot);
            quicksort(i, k-1);
            quicksort(k,j);
        end;
    end; {fim da função}
```



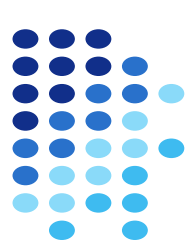
Complexidade do Quick Sort

- Muito custoso:
 - Complexidade pior caso $O(n^2)$
 - Complexidade caso médio $O(n \log n)$
 - Complexidade melhor caso $O(n \log n)$



Considerações

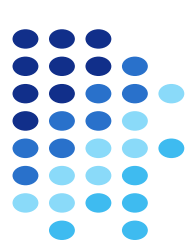
1. Todos os casos assumem que o arquivo cabe na memória
2. Se os arquivos são sequenciais, como atualizar o arquivo e mantê-lo ordenado?
 - Executar uma ordenação a cada atualização?



Exercício

- Simule a execução do algoritmo **Quick Sort** para a sequência de números

6	8	2	1	7	9	3
---	---	---	---	---	---	---



Referências

- Material baseado nos slides de **Vanessa Braganholo**, Disciplina de Estruturas de Dados e Seus Algoritmos. Instituto de Computação. Universidade Federal Fluminense (UFF), Niterói, Brasil.
- Schildt, H. C Completo e Total. Ed. McGraw-Hill.
- Alfred V. Aho, Murray Hill, John E. Hopcroft, Ithaca, D. Ullman, Data Structures and Algorithms. Addison-Wesley Professional; 1ª Edição