Algoritmos de Ordenação

Quick Sort
("partiona")

(Fonte: Material adaptado dos Slides do prof. Monael.)

Alguns Algoritmos de Ordenação

- Exemplos de algoritmos O(n log n):
  - Merge Sort
    - "Intercalação"
  - → Quick Sort
    - "Particionamento"
  - Heap Sort

Particionamento

- Problema do Particionamento
  - Dado um vetor v de n posições e um índice p qualquer.
  - Desenvolva um procedimento que garanta que todos os elementos com índice menores que p são menores ou iguais a v[p] e todos os elementos com índice maiores que p são maiores que v[p]

$$V[0, \ldots, p-1] \leq V[p] < V[p+1, \ldots, n-1]$$

("pivô")

Particionamento

- Problema do Particionamento
  - Exemplo:

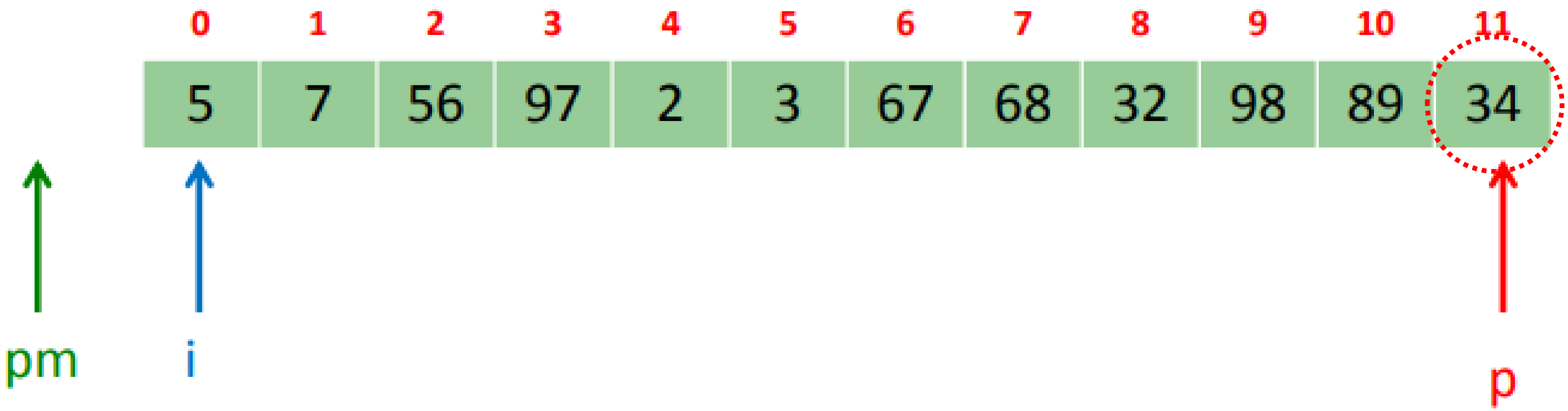
Entrada: V = [5,7,56,97,2,3,67,68,32,98,89,34]
p = 11

("pivô")

Portanto, V[p] = 34

Saída: V = [5,7,2,3,32,**34**,67,68,56,98,89,97]

Particionamento



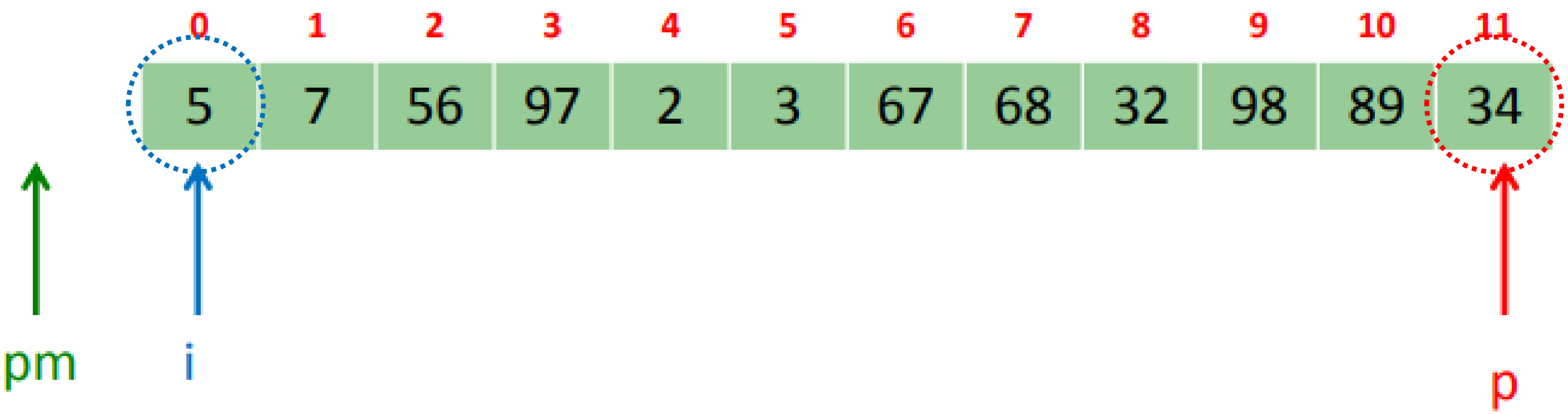
i=0

p=ultimo elemento do vetor

pm = -1

enquanto i < n-1 : Verdade

Particionamento



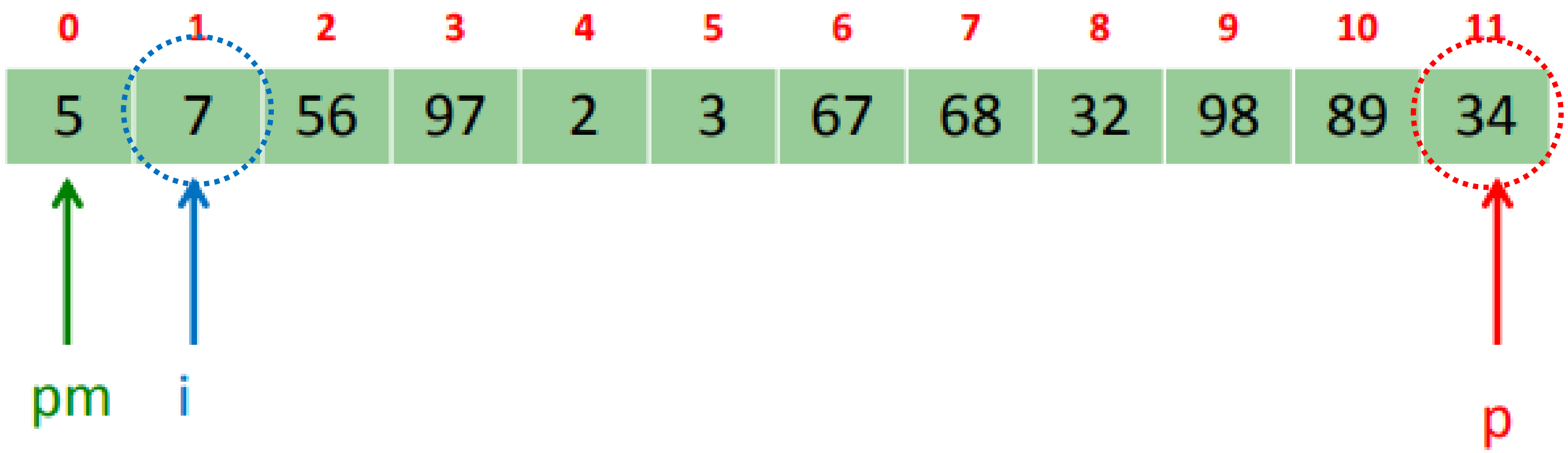
enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Sim

→ Então: pm = pm + 1 e troca v[i] com v[pm]

→ i = i + 1

Particionamento



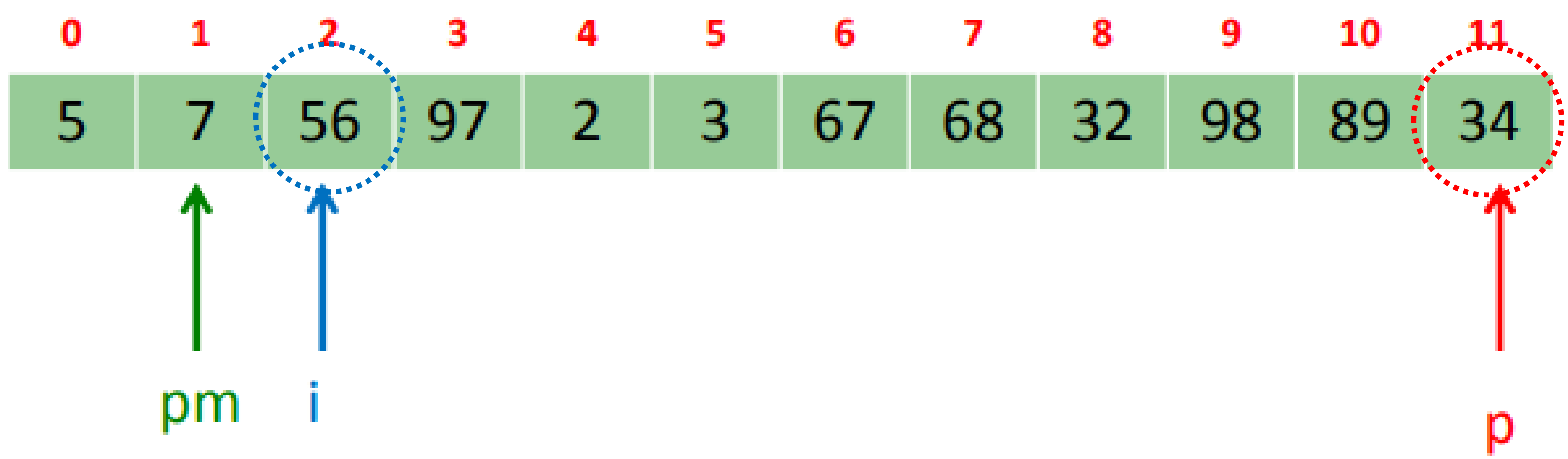
enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Sim

→ Então: pm = pm + 1 e troca v[i] com v[pm]

→ i = i + 1

Particionamento

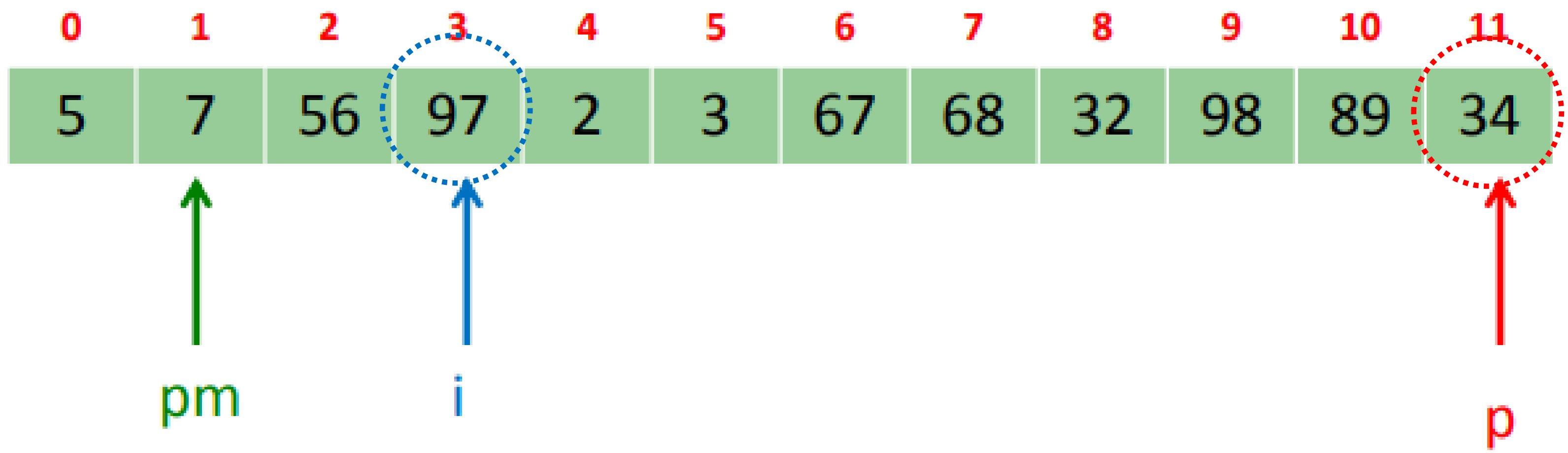


enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Não

→ Então: i = i + 1

Particionamento

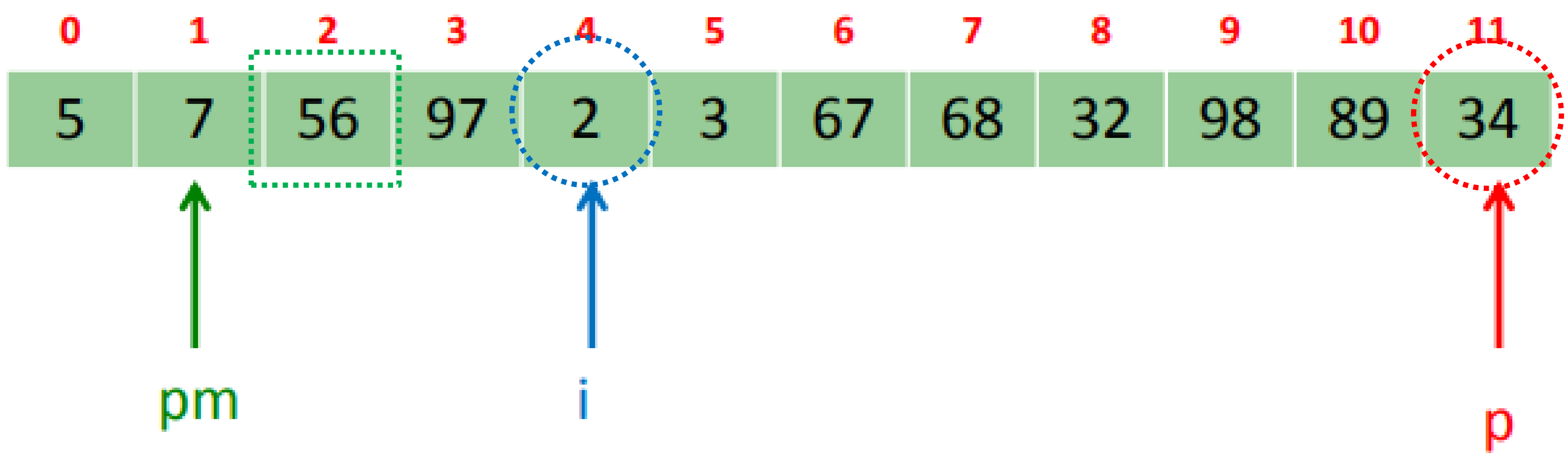


enquanto i < n-1 : Verdade

v[i] <= v[p] ? Não

Então: i = i + 1

Particionamento



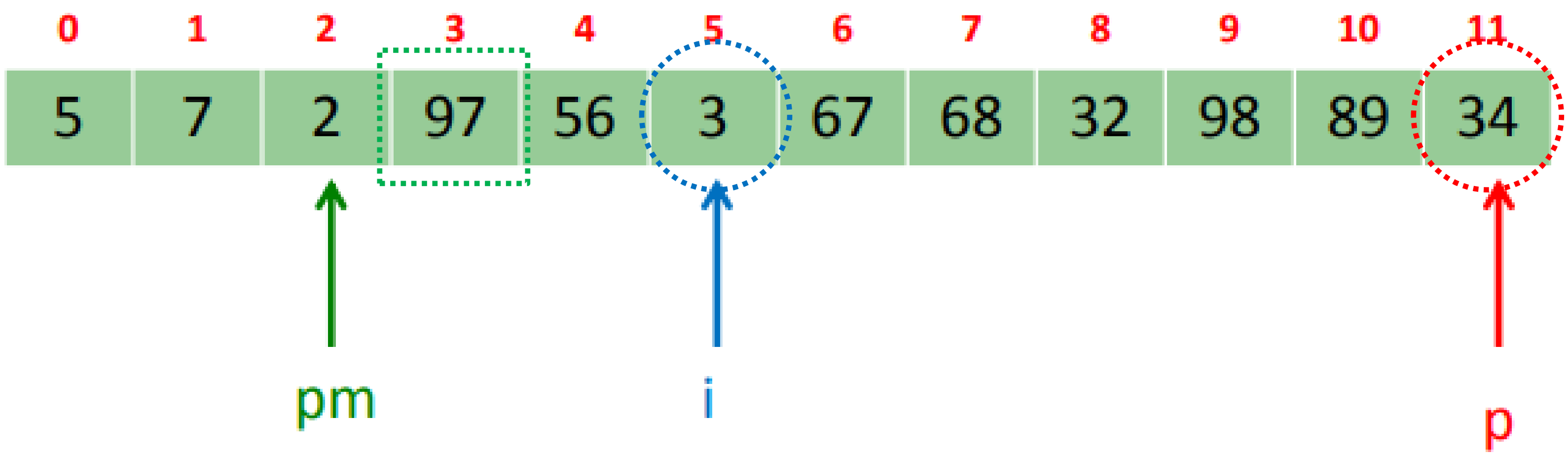
enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Sim

→ Então: pm = pm + 1 e troca v[i] com v[pm]

→ i = i + 1

Particionamento



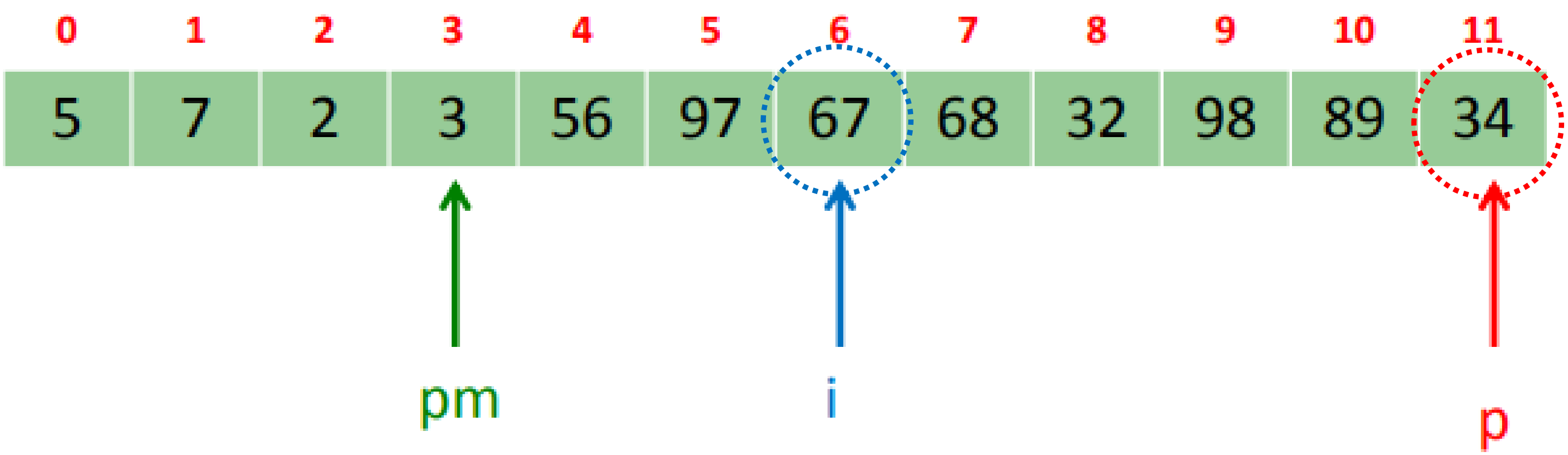
enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Sim

→ Então: pm = pm + 1 e troca v[i] com v[pm]

→ i = i + 1

Particionamento

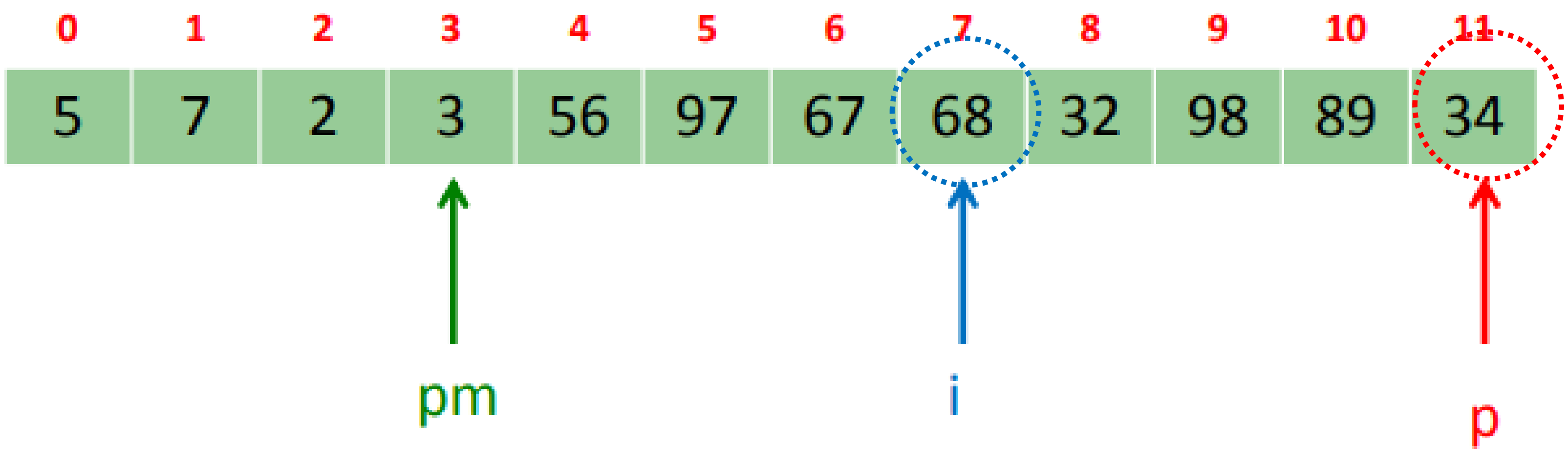


enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Não

→ Então: i = i + 1

Particionamento

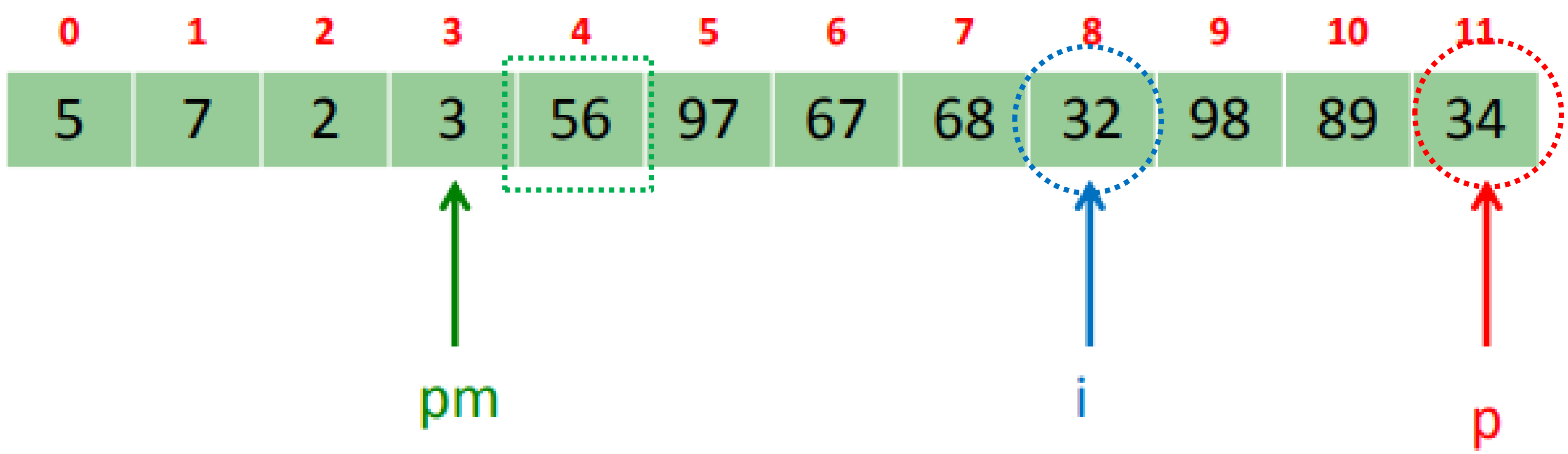


enquanto i < n-1 : Verdade

v[i] <= v[p] ? Não

Então: i = i + 1

Particionamento



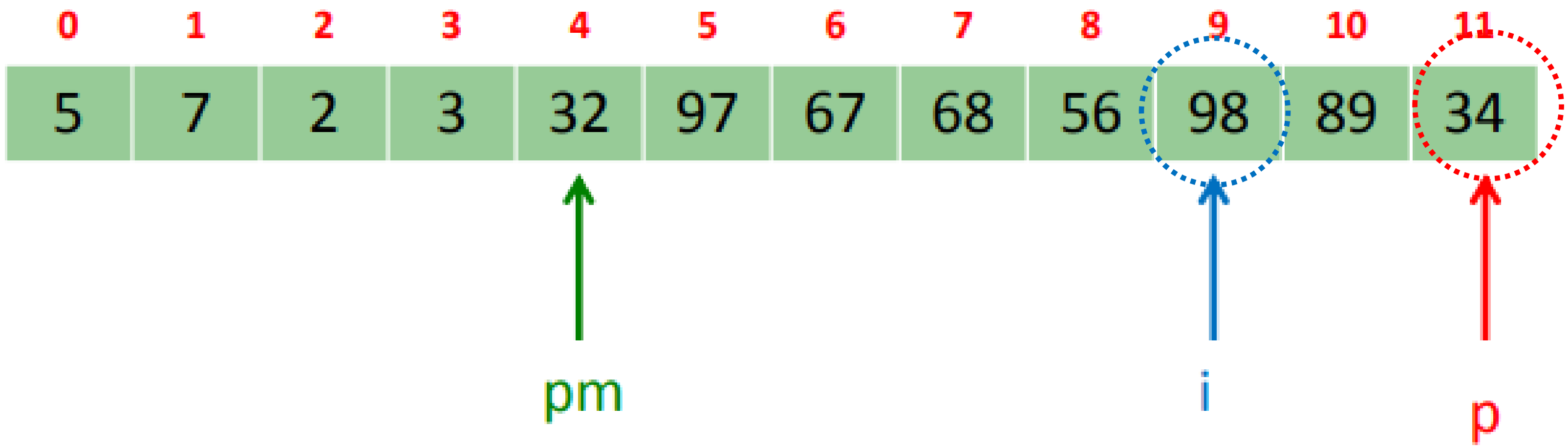
enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Sim

→ Então: pm = pm + 1 e troca v[i] com v[pm]

→ i = i + 1

Particionamento

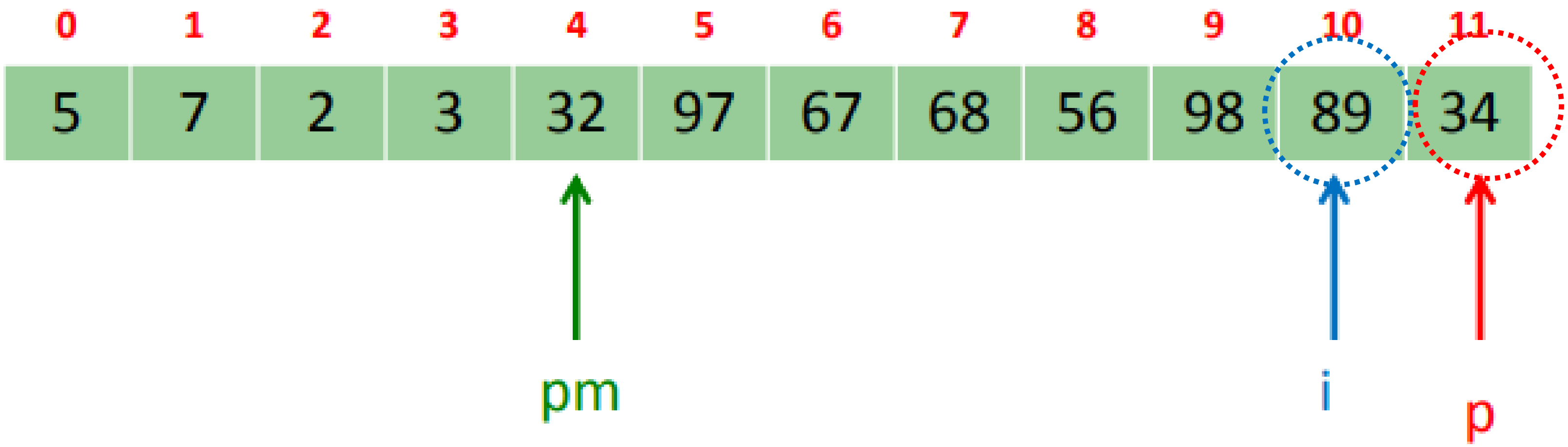


enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Não

→ Então: i = i + 1

Particionamento

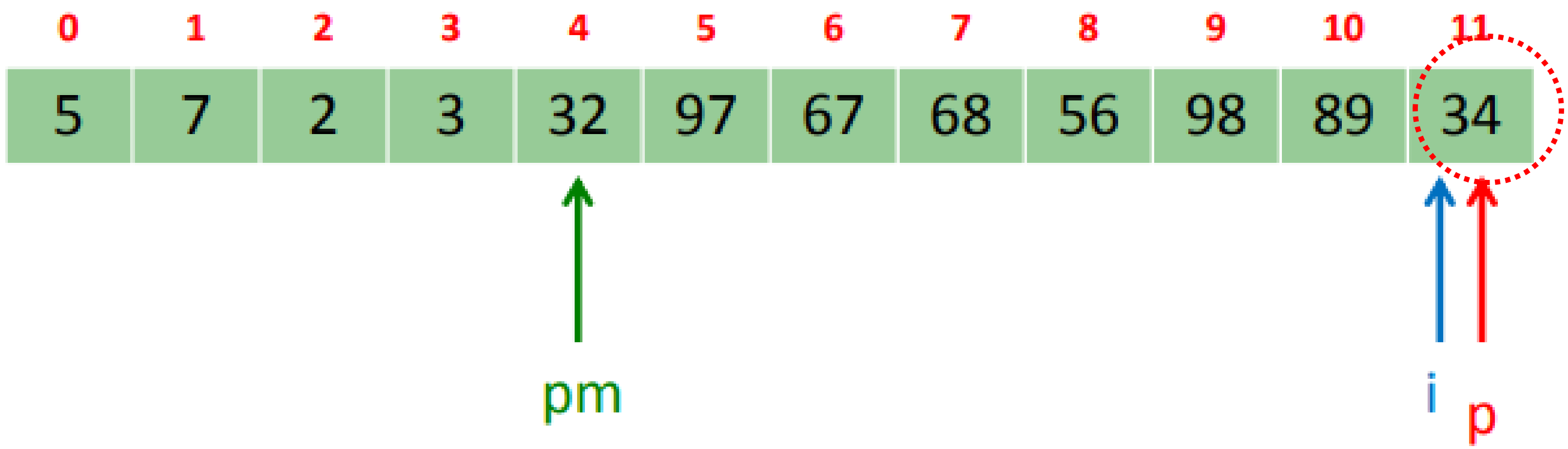


enquanto i < n-1 : Verdade

→ v[i] <= v[p] ? Não

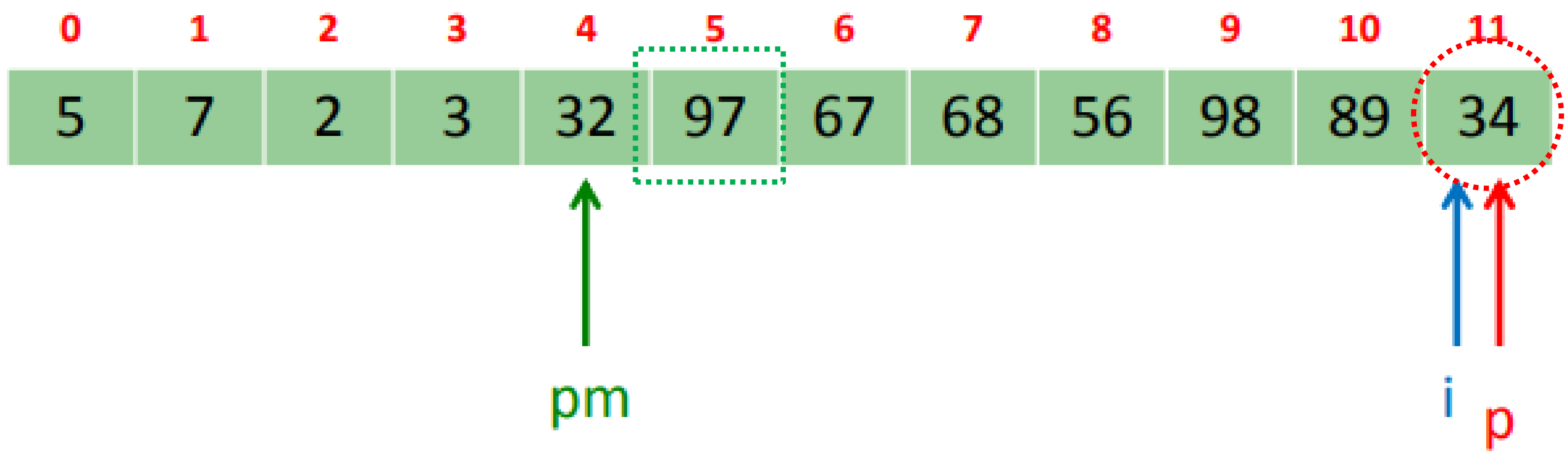
→ Então: i = i + 1

Particionamento



enquanto i < n-1 : Falso

Particionamento



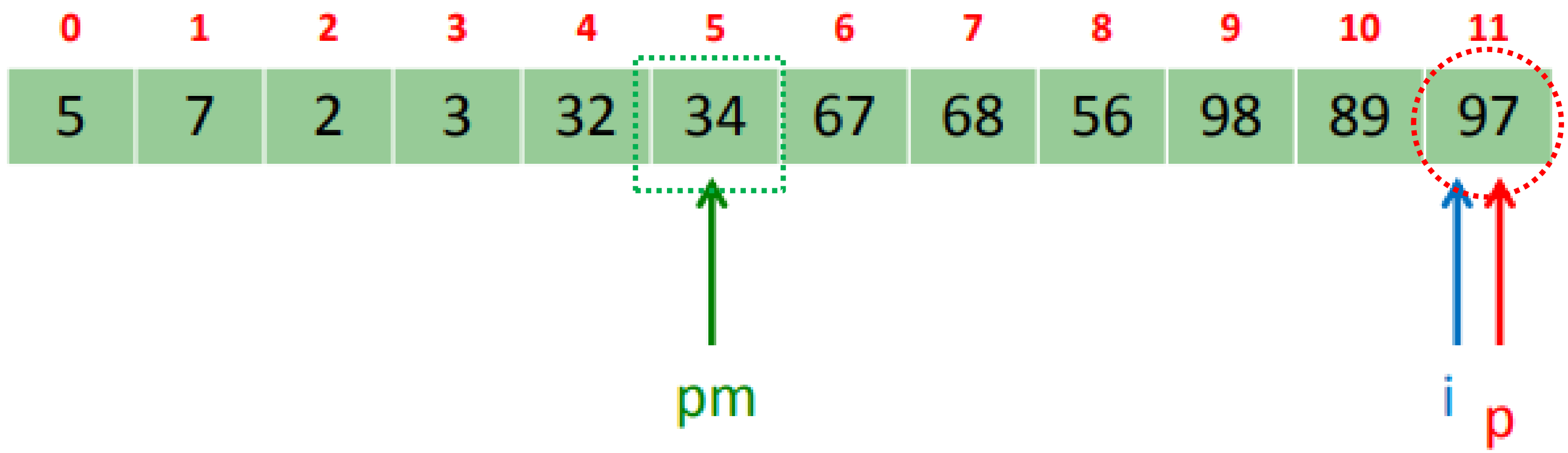
enquanto i < n-1 : Falso

pm = pm + 1

troca v[p] com v[pm]

retorne pm

Particionamento



enquanto i < n-1 : Falso

pm = pm + 1

troca v[p] com v[pm]

retorne pm

Particionamento

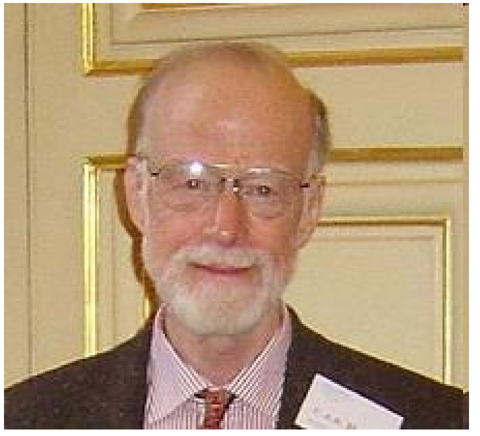
- Dado o vetor v antes e depois do particionamento, o que pode ser observado?

  Entrada: V = [05,07,56,97,02,03,67,68,32,98,89,34]
  Saída: V = [05,07,02,03,32,**34**,67,68,56,98,89,97]
  Ordenado: V = [02,03,05,07,32,**34**,56,67,68,89,97,98]

- É garantido que o elemento p ficou na sua posição correta do vetor ordenado.
- O que acontece se aplicar o procedimento recursivamente nas porções separadas por p ?
- Esse é o **Quick Sort** !

Quick Sort

- Proposto em 1962 por Charles Antony Richard Hoare no Computer Journal, 5, pp.10-15, 1962
  - É considerado o método de ordenação mais eficiente até os dias atuais;
  - Emprega a Divisão e Conquista;
  - O método consiste em:
    - Eleger um pivô
    - Garantir que todos os elementos a direita do pivô são menores ou iguais que ele e a esquerda são maiores
    - Repetir recursivamente na metade direita e na metade esquerda do arranjo (usando como referencia o pivô).

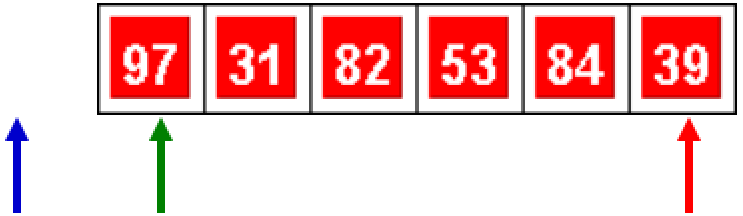


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Elemento pivô
Último menor elementos
Elemento atual
Troca

Quick Sort

Idéia

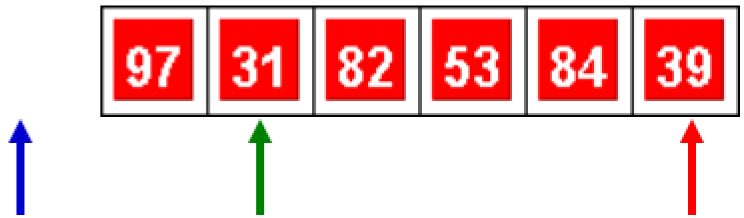


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

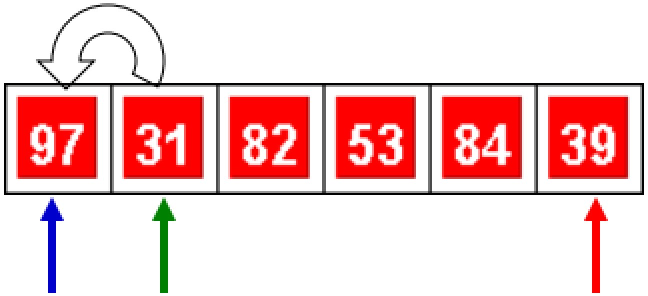


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

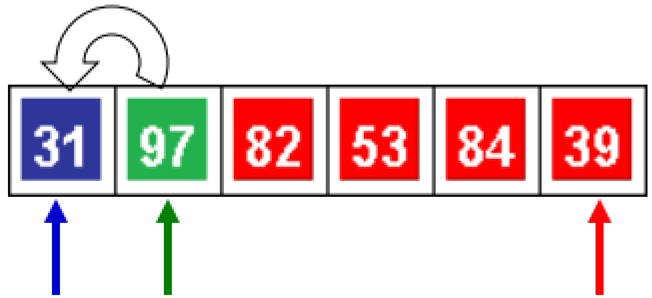


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Elemento pivô
Último menor elementos
Elemento atual
Troca

Quick Sort

Idéia

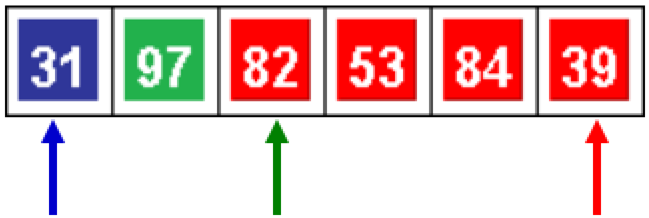


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Elemento pivô
Último menor elementos
Elemento atual
Troca

Quick Sort

Idéia



Elementos não verificados

Elementos menores que o pivô

Elementos maiores que o pivô

Elementos ordenados

Elemento pivô

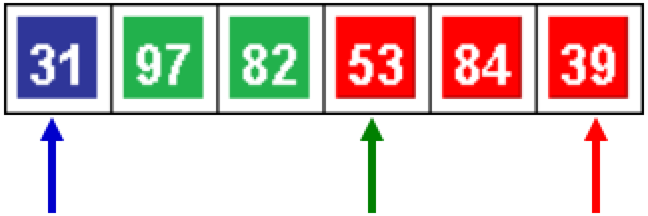
Último menor elementos

Elemento atual

Troca

Quick Sort

Idéia

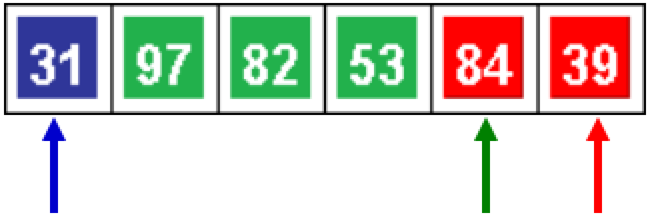


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

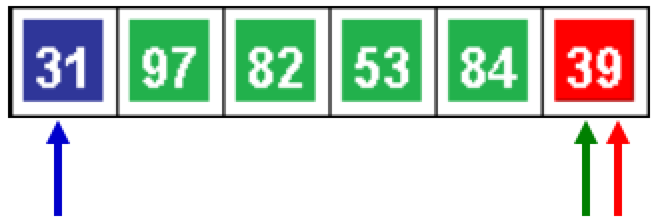


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

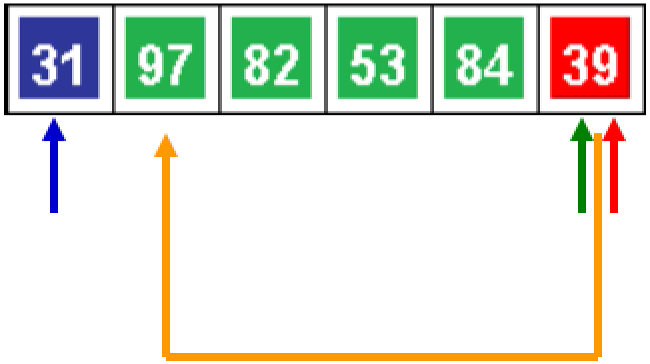
Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca



Quick Sort

Idéia



Elementos não verificados

Elementos menores que o pivô

Elementos maiores que o pivô

Elementos ordenados

Quick Sort

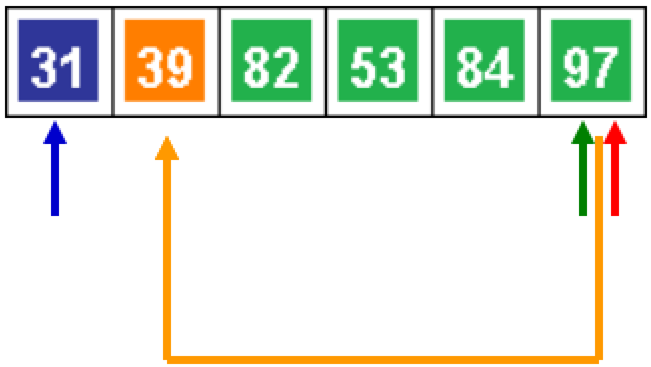
Elemento pivô

Último menor elementos

Elemento atual

Troca

Idéia

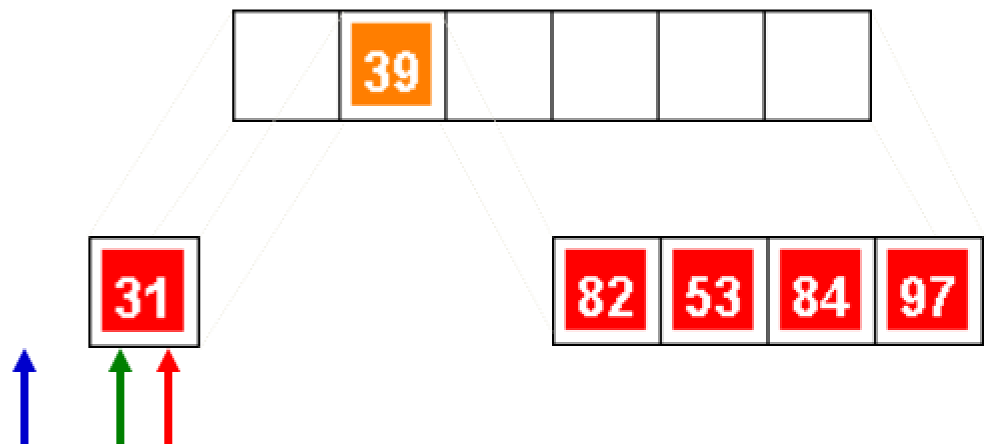


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca



Elementos não verificados

Elementos menores que o pivô

Elementos maiores que o pivô

Elementos ordenados

Quick Sort

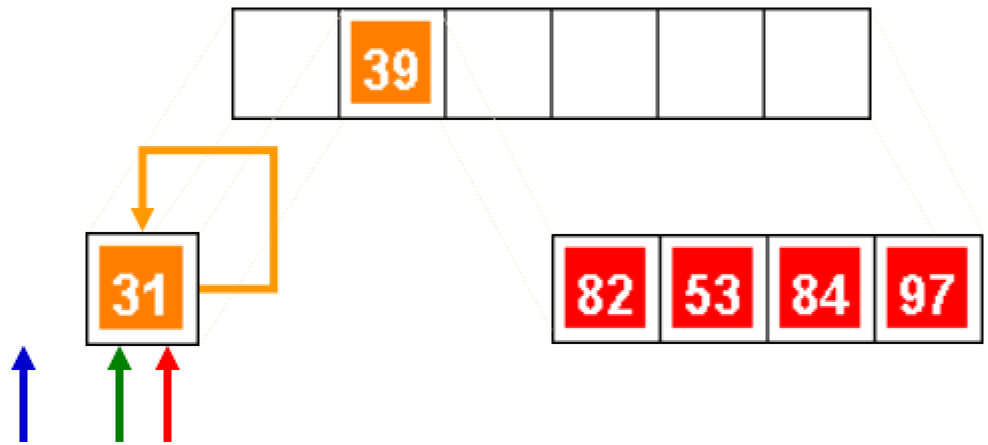
Idéia

Elemento pivô

Último menor elementos

Elemento atual

Troca

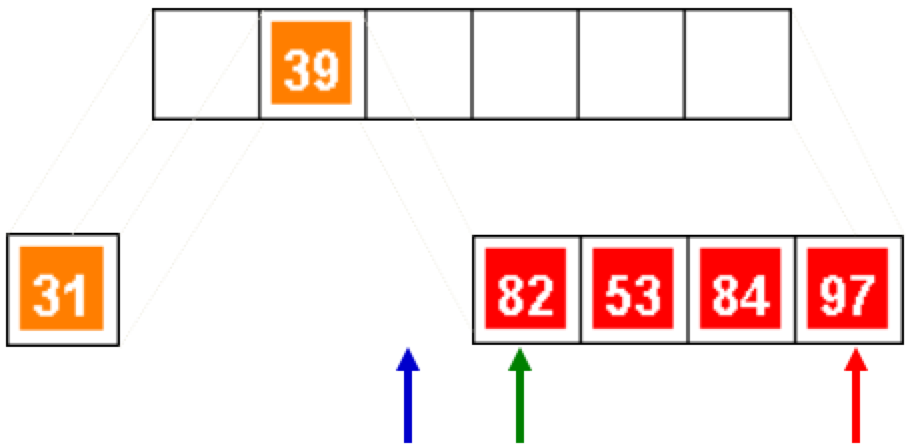


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

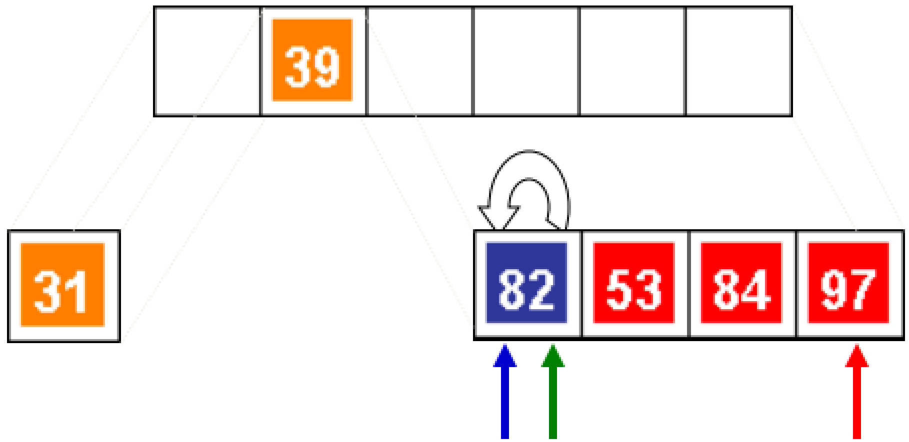


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

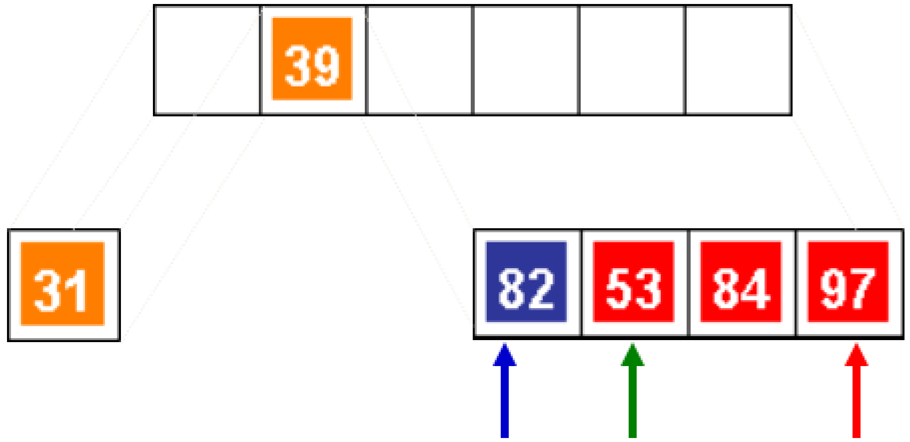


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Elemento pivô
Último menor elementos
Elemento atual
Troca

Idéia



Elementos não verificados

Elementos menores que o pivô

Elementos maiores que o pivô

Elementos ordenados

Quick Sort

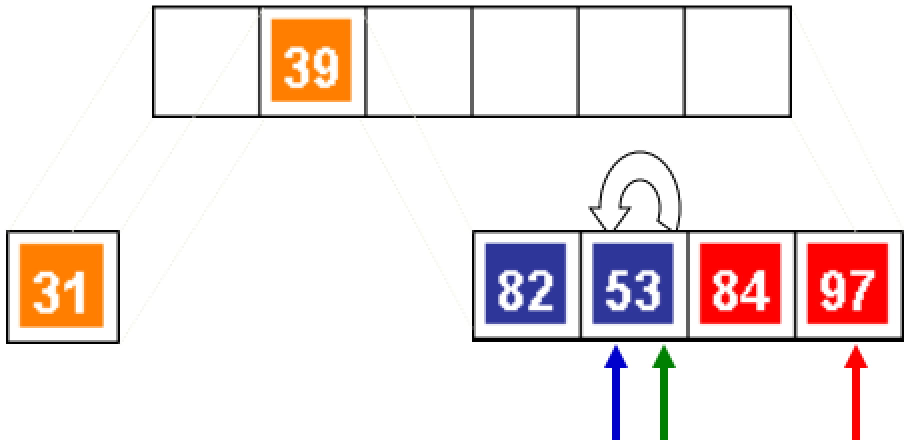
Elemento pivô

Último menor elementos

Elemento atual

Troca

Idéia

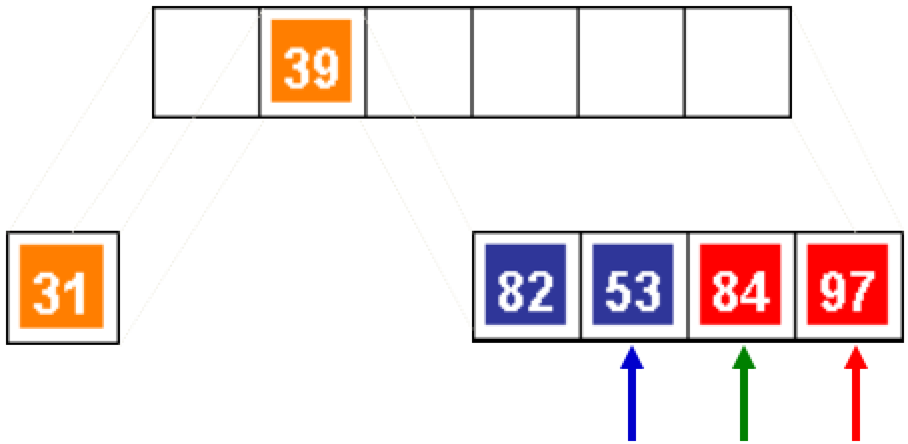


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

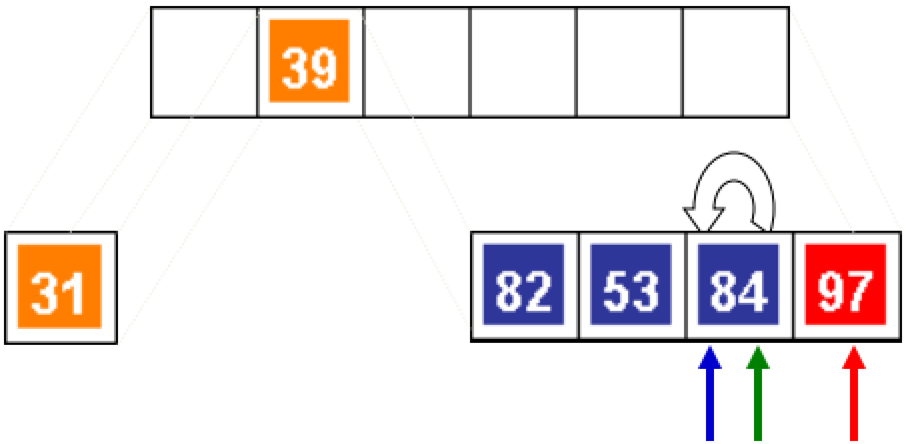


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

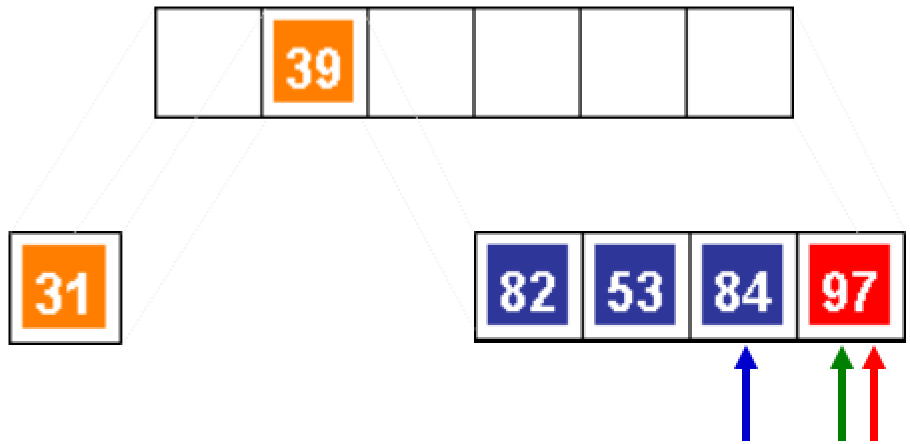


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Elemento pivô
Último menor elementos
Elemento atual
Troca

Idéia



Elementos não verificados

Elementos menores que o pivô

Elementos maiores que o pivô

Elementos ordenados

Quick Sort

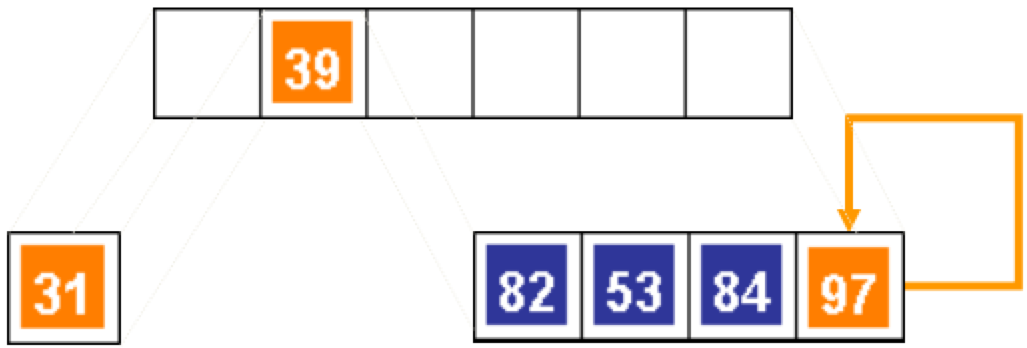
Elemento pivô

Último menor elementos

Elemento atual

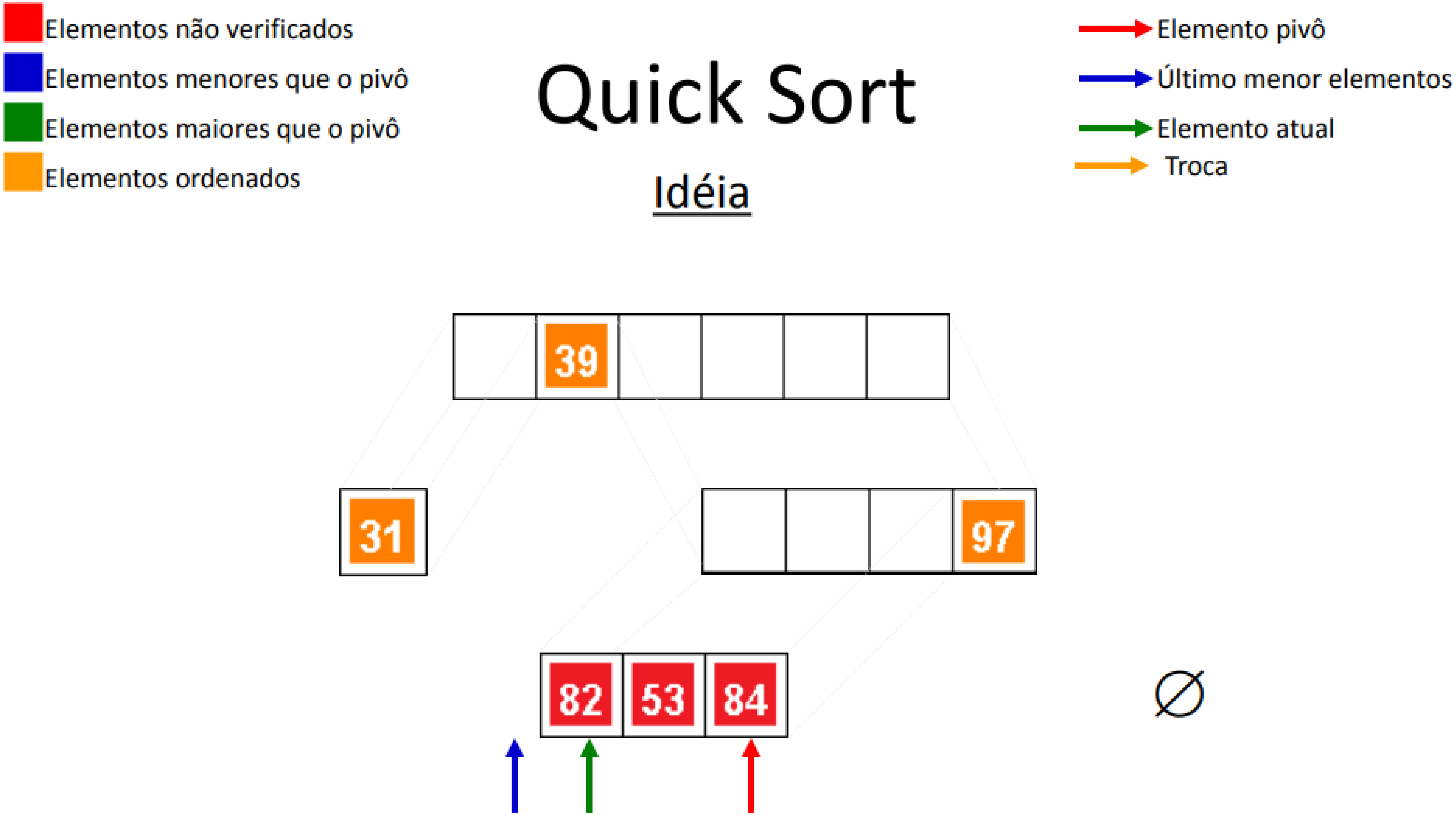
Troca

Idéia



Quick Sort

Idéia

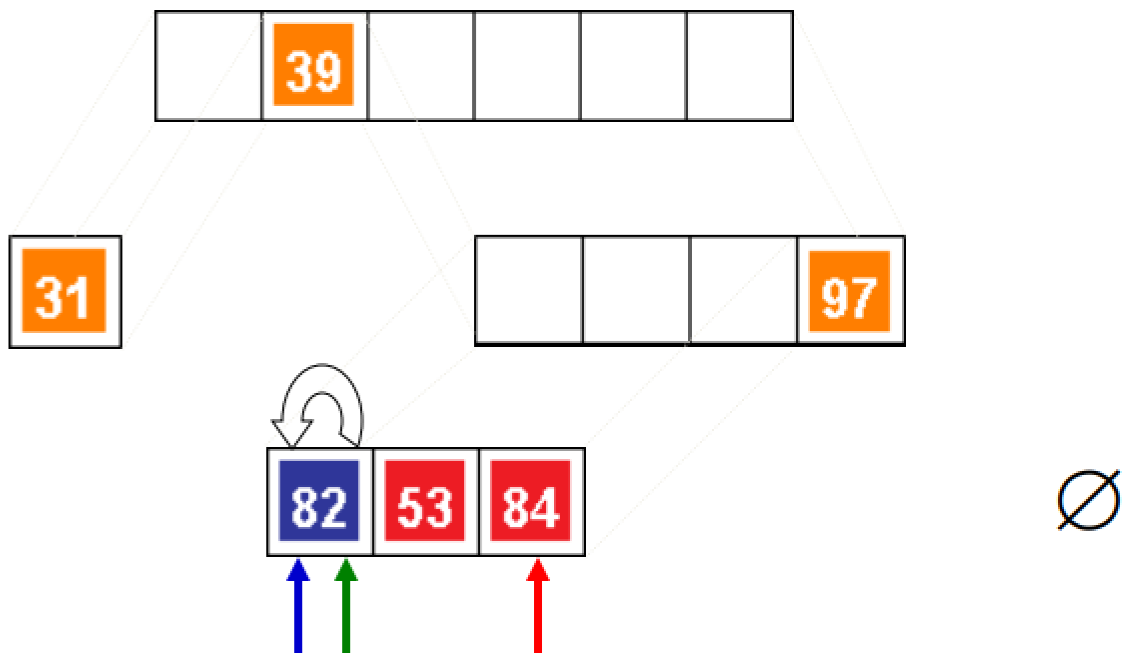


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

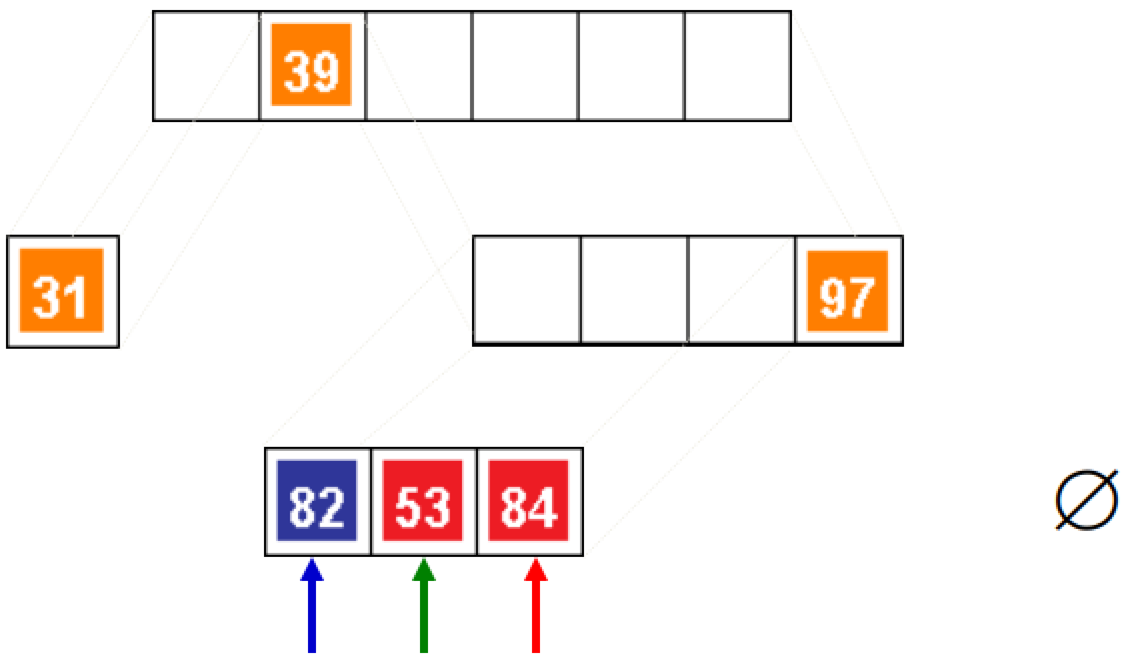


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

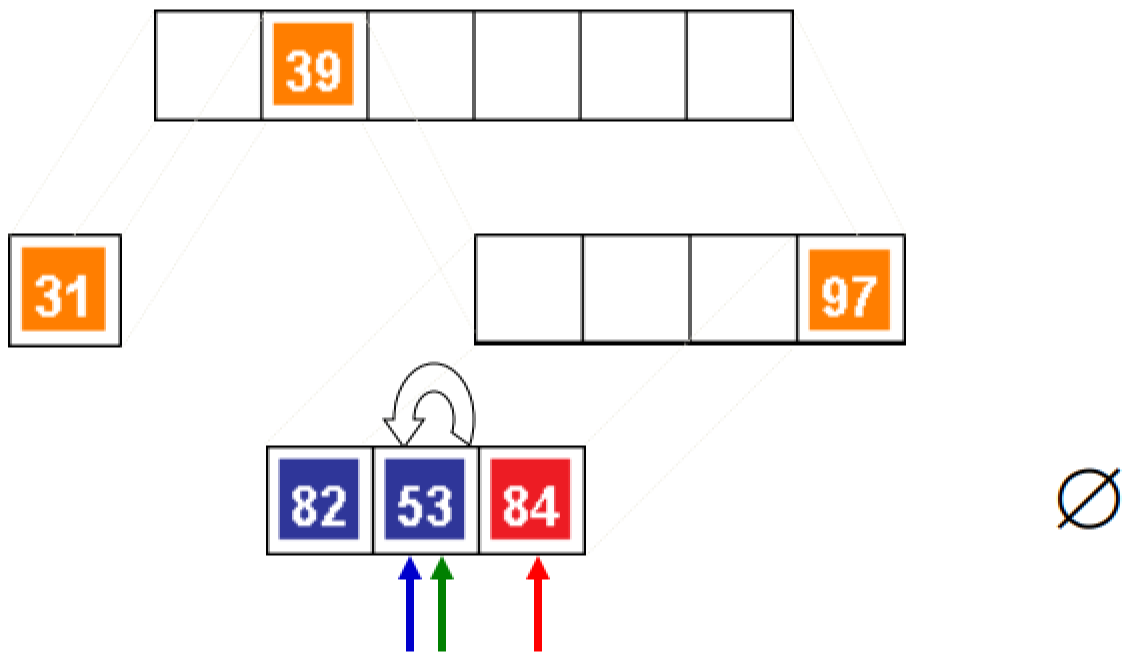


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

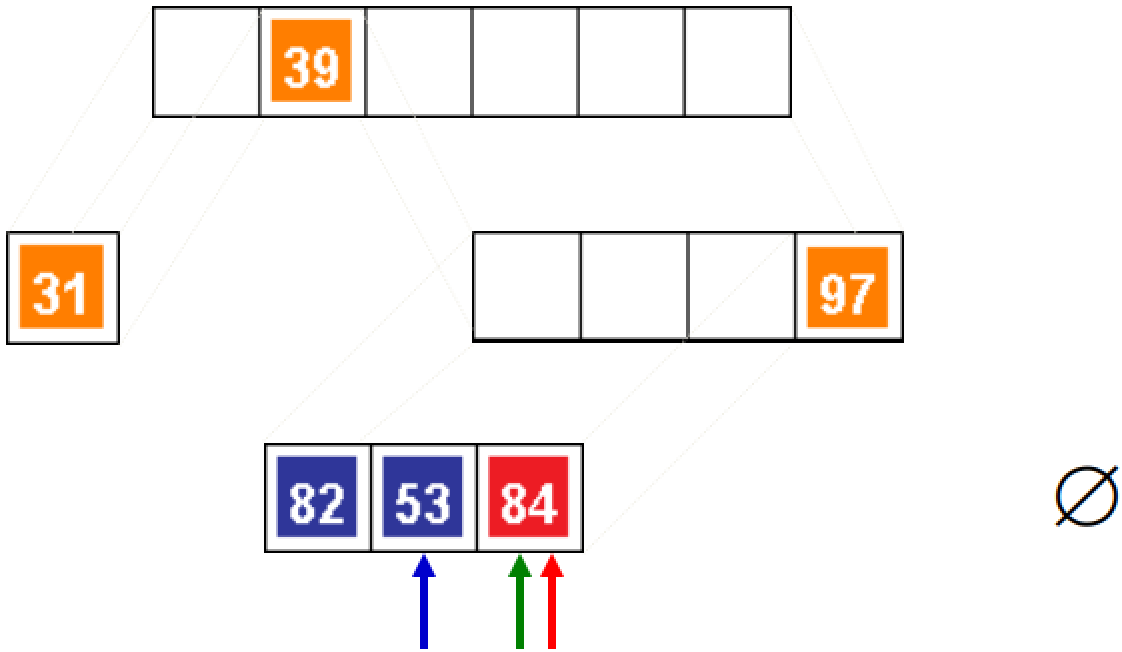


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Elemento pivô
Último menor elementos
Elemento atual
Troca

Quick Sort

Idéia

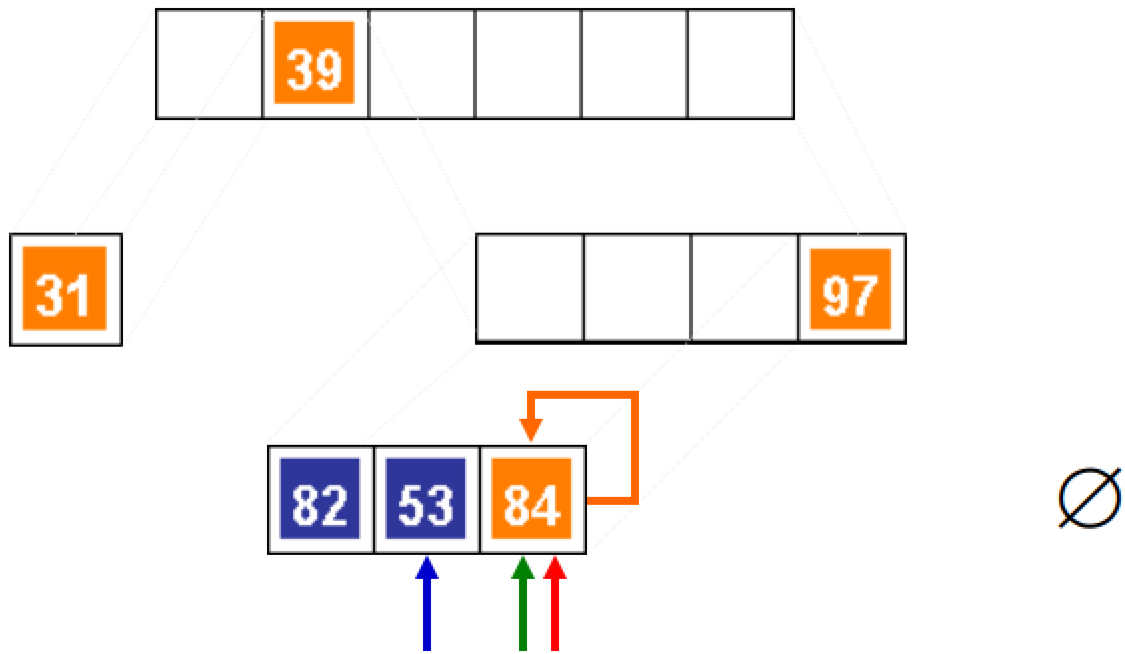


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

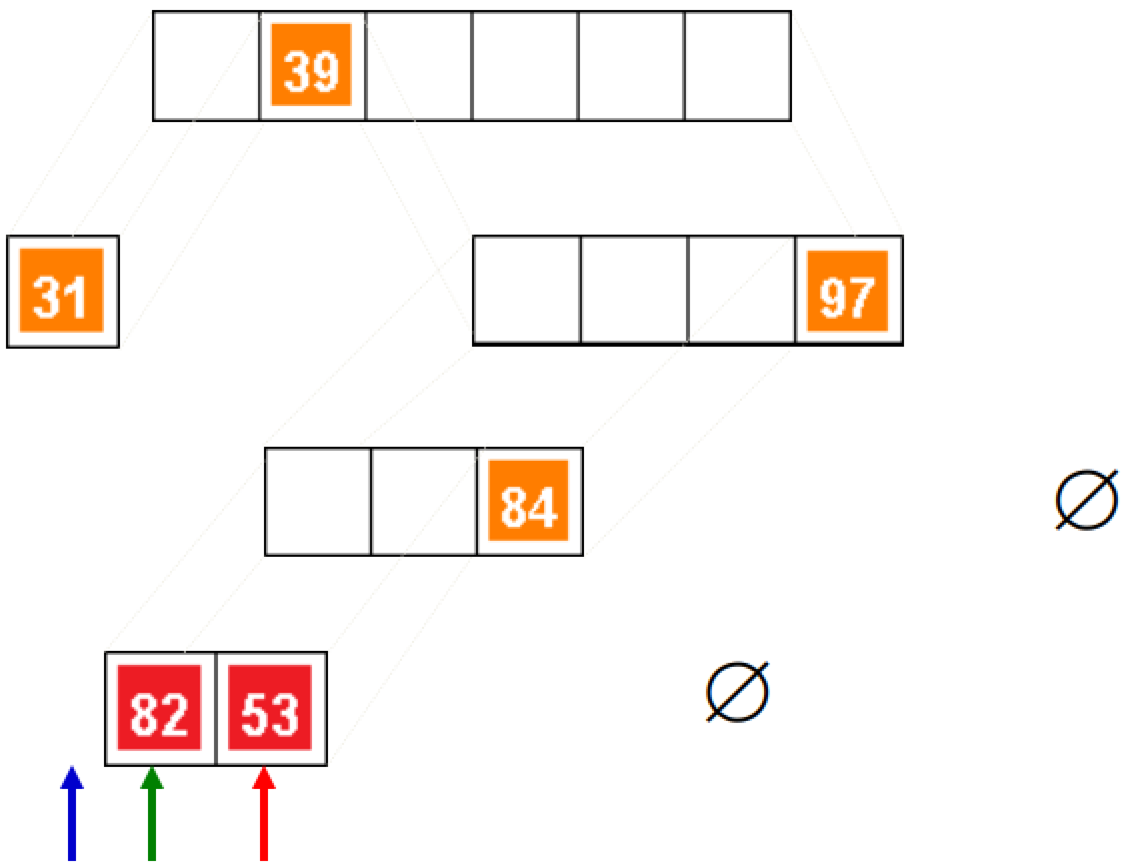


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Elemento pivô
Último menor elementos
Elemento atual
Troca

Idéia

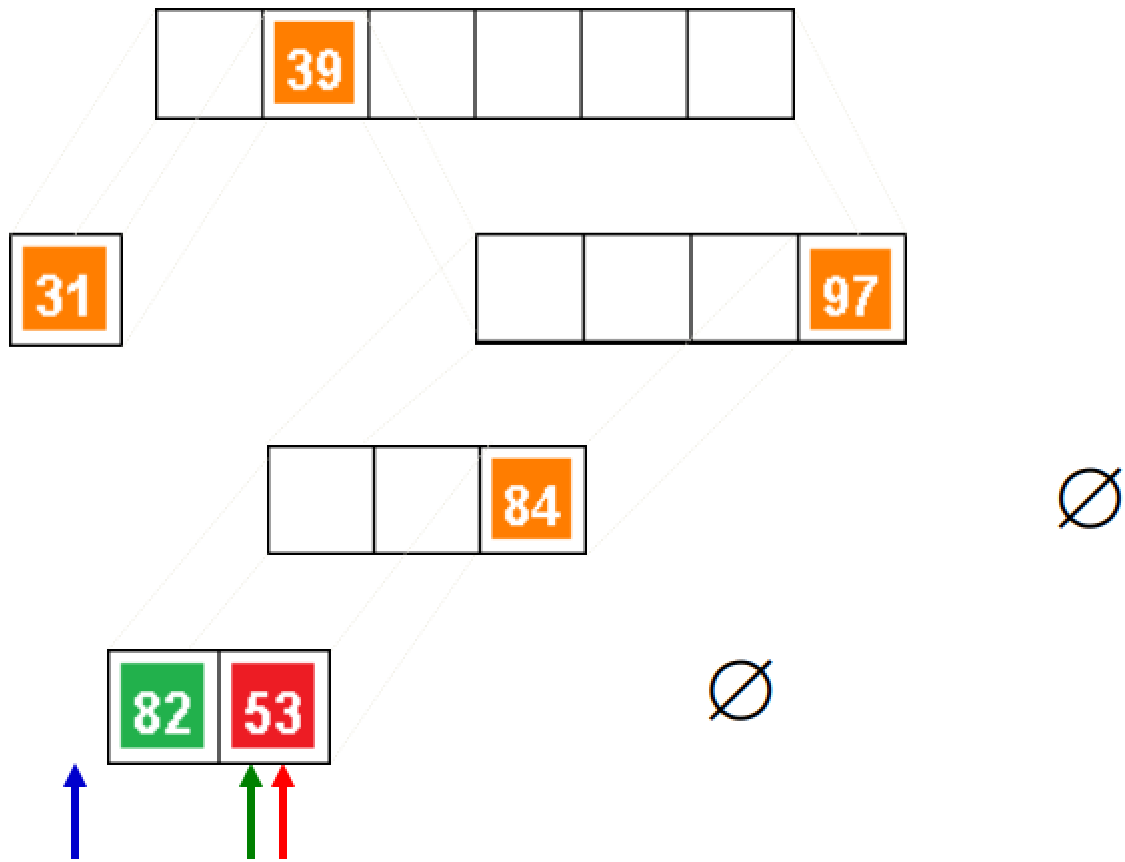


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

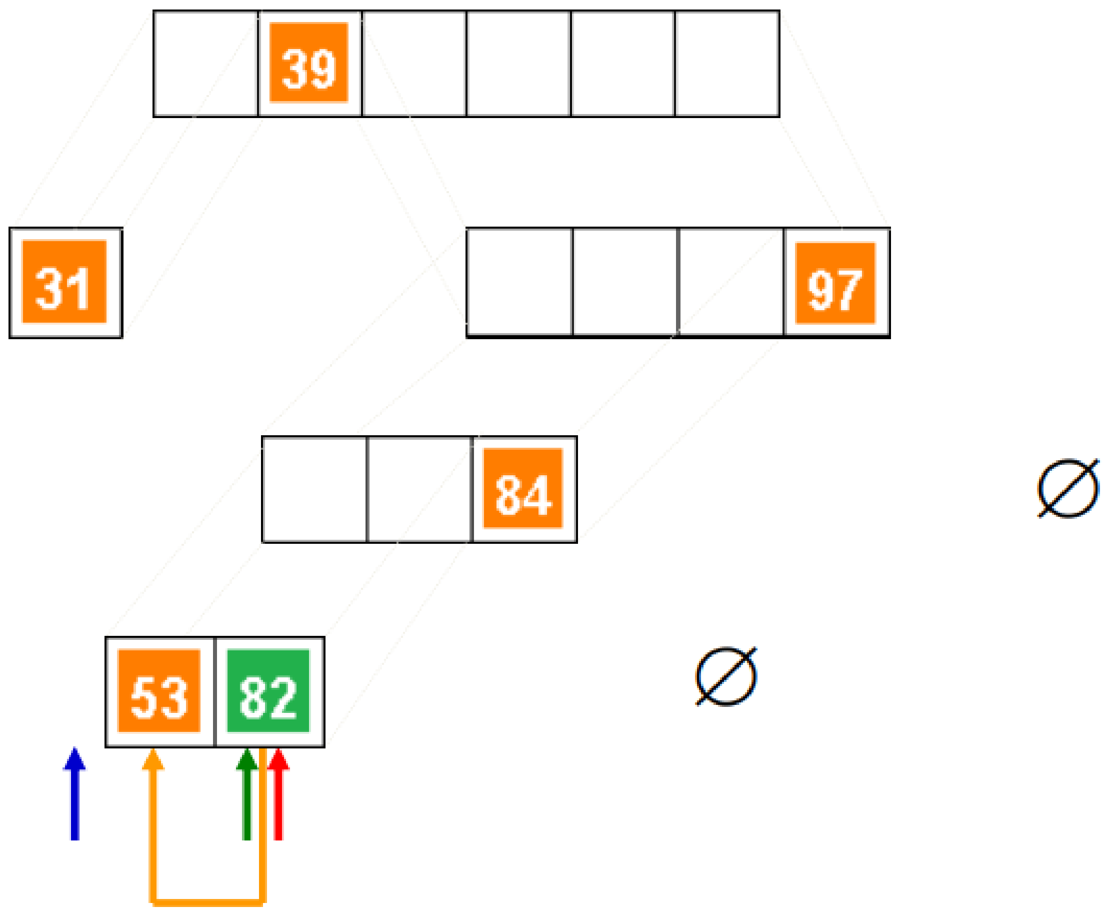


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

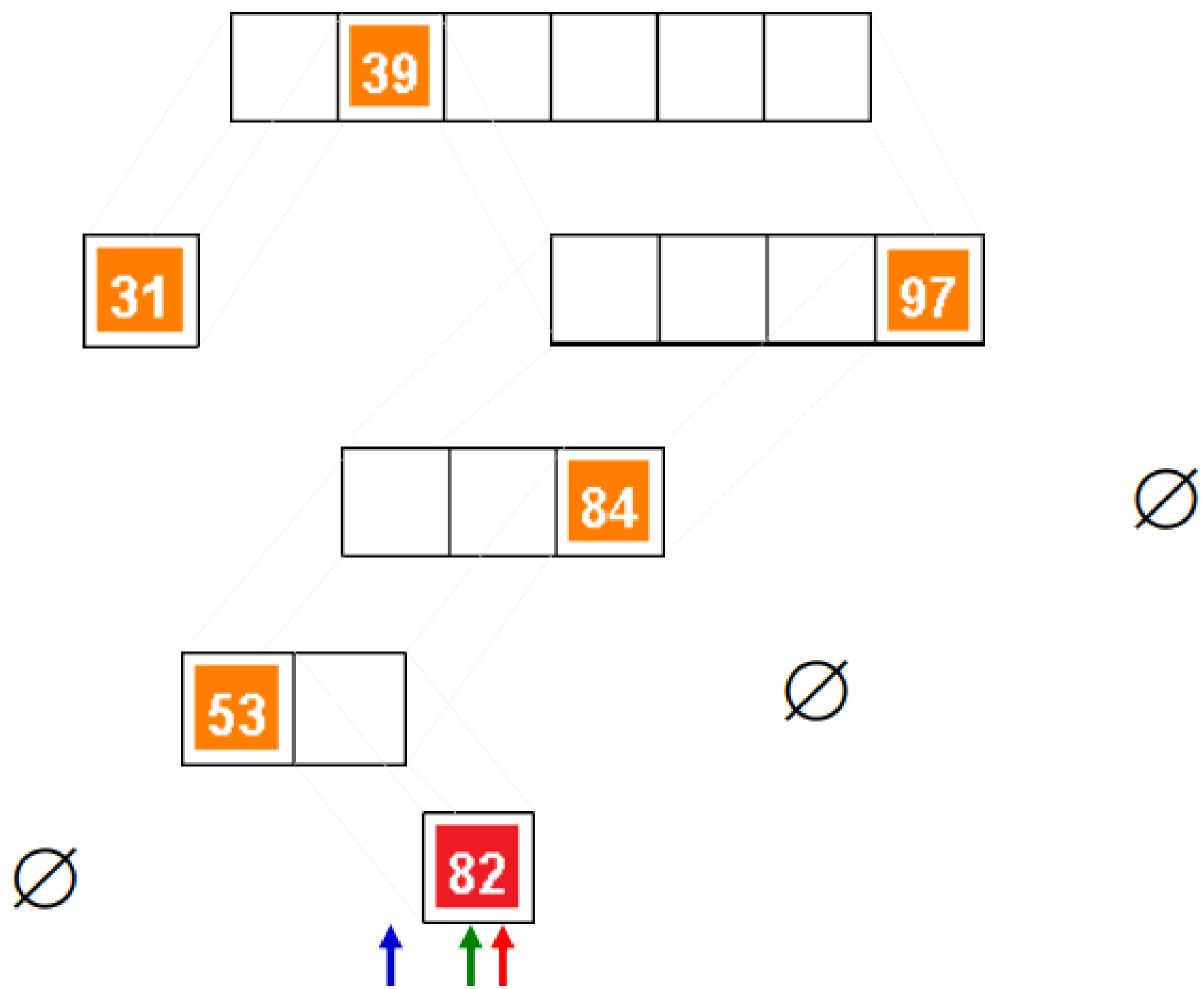


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Elemento pivô
Último menor elementos
Elemento atual
Troca

Idéia

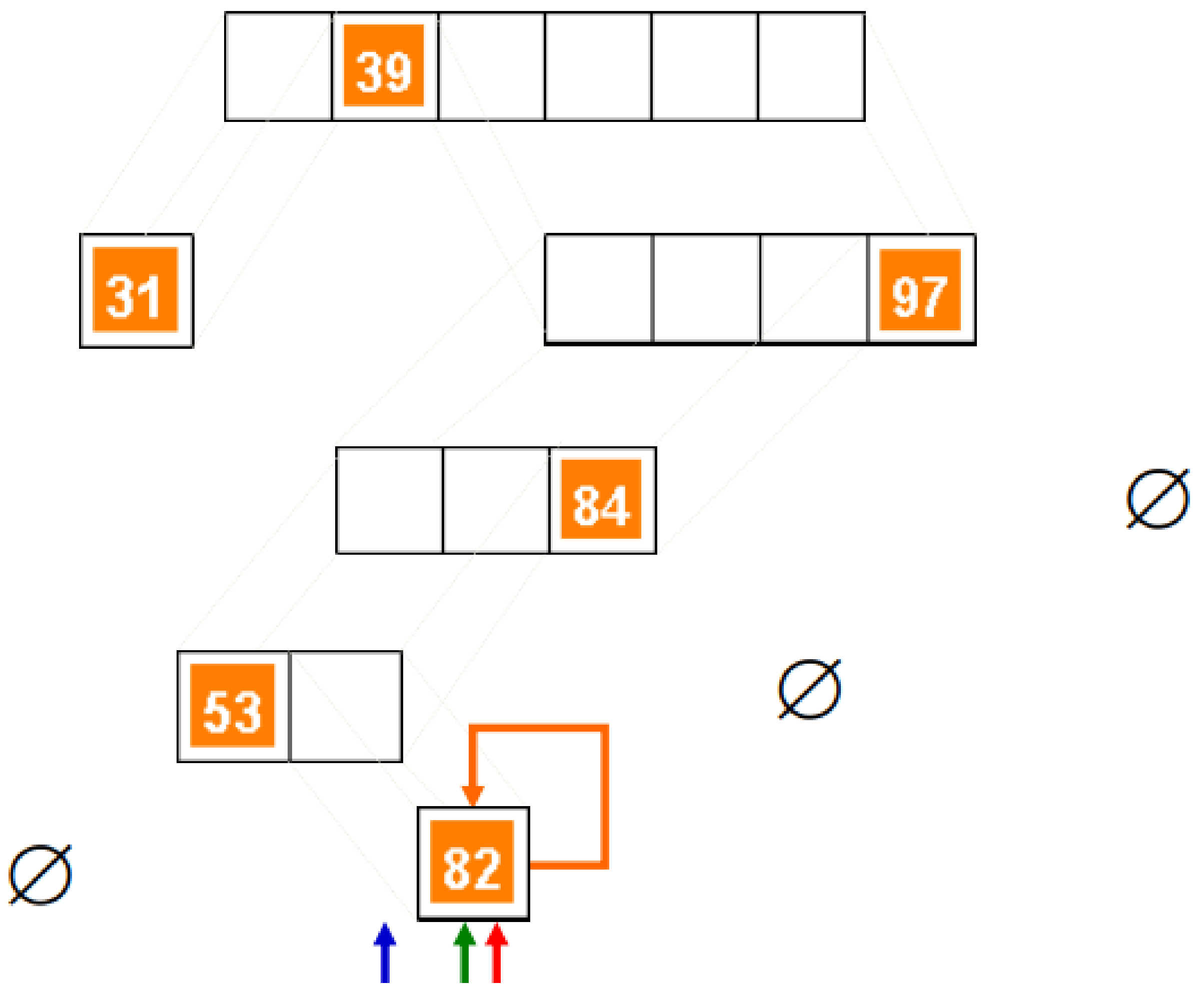


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

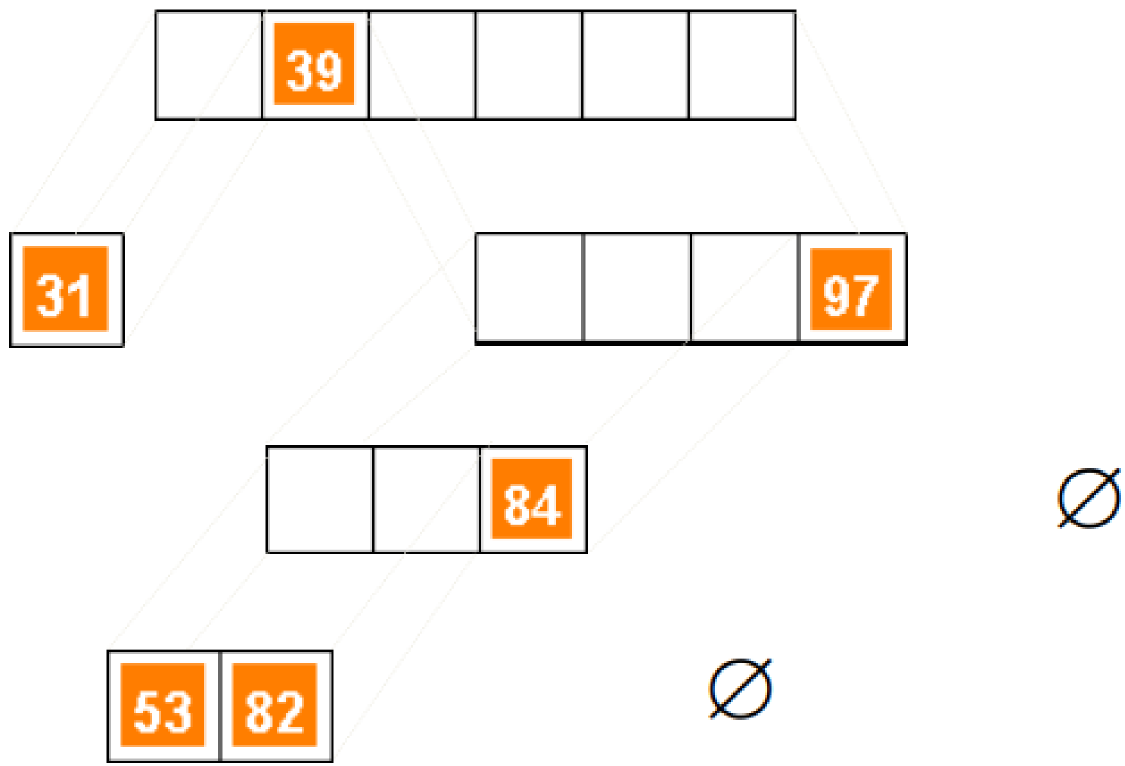


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

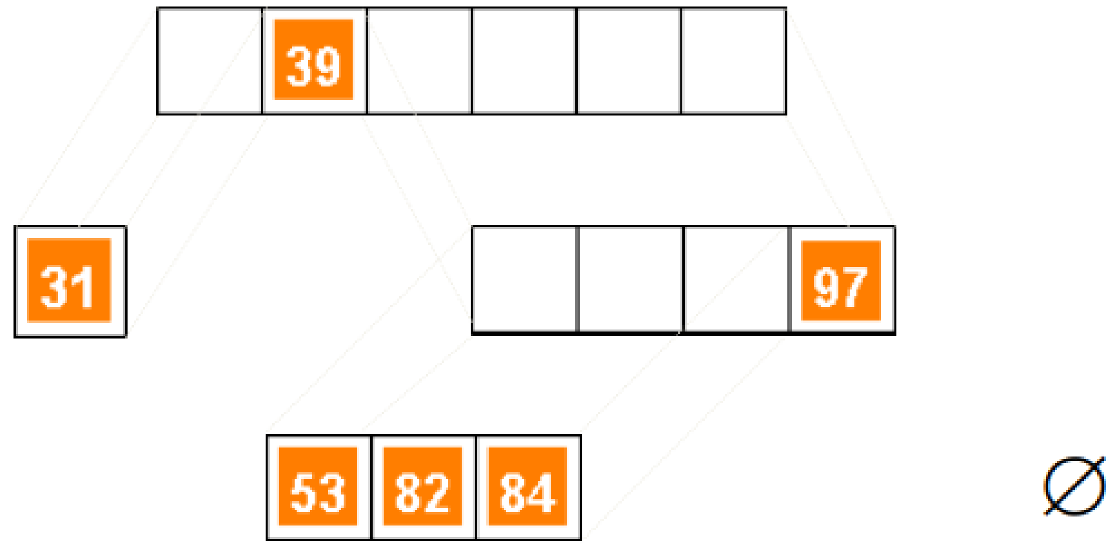


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Idéia

Elemento pivô
Último menor elementos
Elemento atual
Troca

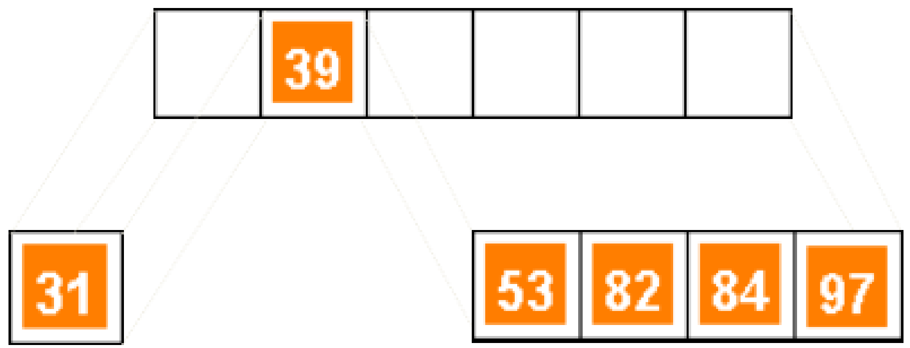


Elementos não verificados
Elementos menores que o pivô
Elementos maiores que o pivô
Elementos ordenados

Quick Sort

Elemento pivô
Último menor elementos
Elemento atual
Troca

Idéia



Elementos não verificados

Elementos menores que o pivô

Elementos maiores que o pivô

Elementos ordenados

Quick Sort

Idéia

Elemento pivô

Último menor elementos

Elemento atual

Troca

31	39	53	82	84	97

Quick Sort

```
void quickSort(int *v, int e, int d)  {
     int p;
     if(e < d)
     {
          p = particiona(v, e, d);
          quickSort(v, e, p-1);
          quickSort(v, p+1, d);
     }
}
```

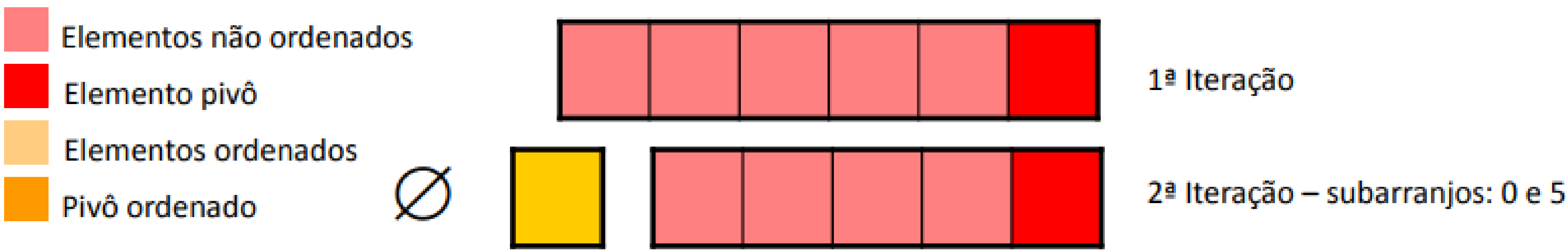
Quick Sort

```
                                              // Consumo de Tempo?
                                              // (Pior caso)
01.  void quickSort(int *v, int e, int d) {
02.       int p;                              // 02. O(1)
03.       if(e < d)                           // 03. O(1)
04.       {
05.            p = particiona(v, e, d);       // 05. O(n)
06.            quickSort(v, e, p-1);          // 06. ?
07.            quickSort(v, p+1, d);          // 07. ?
08.       }
09.  }                                        // Total: ?
```

Quick Sort

O pior caso possível do Quick Sort: Particiona Desbalanceado

A cada iteração pivô parte o arranjo em duas metades desbalanceadas.



Quick Sort

O pior caso possível do Quick Sort: Particiona Desbalanceado

A cada iteração pivô parte o arranjo em duas metades desbalanceadas.

Quick Sort

```
                                                  // Consumo de Tempo?
                                                  // (Pior caso)
01.  void quickSort(int *v, int e, int d) {
02.       int p;                                  // 02. O(1)
03.       if(e < d)                               // 03. O(1)
04.       {
05.            p = particiona(v, e, d);           // 05. O(n)
06.            quickSort(v, e, p-1);              // 06. T(1)
07.            quickSort(v, p+1, d);              // 07. T(n-1)
08.       }
09.  }                                            // Total: ?
```

```
// Total: T(n) = T(n-1) + O(n) + T(1) + 2*O(1)
//             = T(n-1) + O(n)
```

Quick Sort

O pior caso possível do Quick Sort: Particiona Desbalanceado

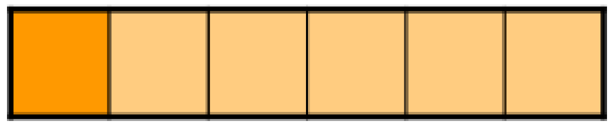
A cada iteração pivô parte o arranjo em duas metades desbalanceadas.

Elementos não ordenados

Elemento pivô

Elementos ordenados

Pivô ordenado



7ª Iteração – Ordenado após 6 chamadas

A cada iteração o vetor é particionado em n-1 elementos.

$$T(1) = 1$$

$$T(n) = T(n-1) + n$$

Quick Sort

- Análise de Pior Caso:

$$T(1) = 1$$

$$T(n) = T(n-1) + n$$

```
T(2) = T(1) + 2 = 1 + 2 = 3
T(3) = T(2) + 3 = 1 + 2 + 3 = 6
T(4) = T(3) + 4 = 1 + 2 + 3 + 4 = 10
...
T(n) = 1 + 2 + 3 + ... + (n-2) + (n-1) + n = ?
                                    (fórmula fechada?)
```

Quick Sort

- Análise de Pior Caso:

$$T(1) = 1$$

$$T(n) = T(n-1) + n$$

```
T(2) = T(1) + 2 = 1 + 2 = 3
T(3) = T(2) + 3 = 1 + 2 + 3 = 6
T(4) = T(3) + 4 = 1 + 2 + 3 + 4 = 10
...
T(n) = 1 + 2 + 3 + ... + (n-2) + (n-1) + n = (n+1)*n/2
                                           = O(n^2)
```

Quick Sort

```
                                                    // Consumo de Tempo?
                                                    // (Pior caso)
01.  void quickSort(int *v, int e, int d)  {
02.       int p;                                    // 02. O(1)
03.       if(e < d)                                 // 03. O(1)
04.       {
05.               p = particiona(v, e, d);          // 05. O(n)
06.               quickSort(v, e, p-1);             // 06. T(1)
07.               quickSort(v, p+1, d);             // 07. T(n-1)
08.       }
09.  }                                              // Total: ?
```

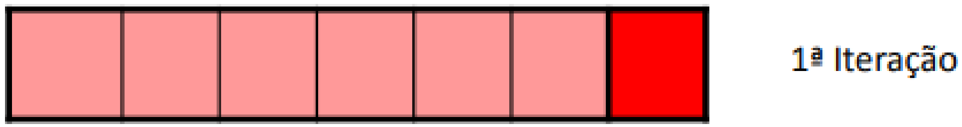
```
// Total: T(n) = T(n-1) + O(n) + T(1) + 2*O(1)
//             = T(n-1) + O(n) = O(n^2)
```

Quick Sort

O melhor caso do Quick Sort: Particiona Balanceado

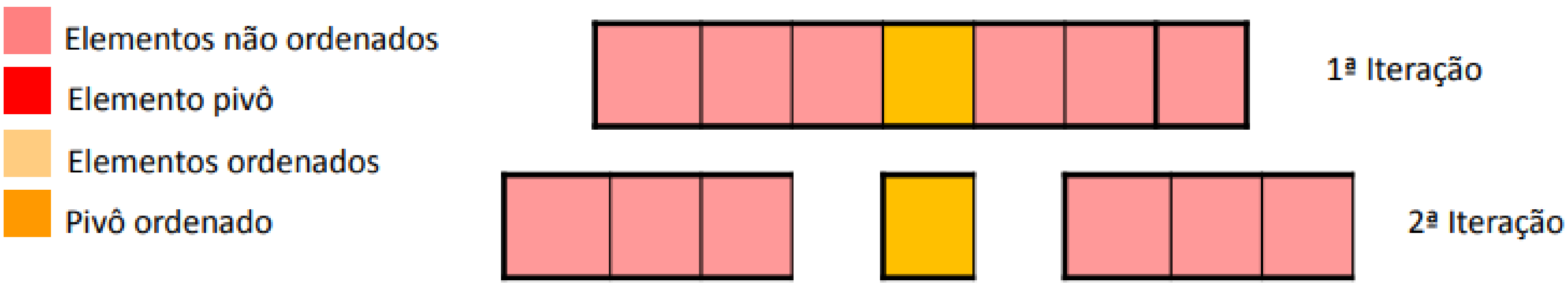
A cada iteração pivô parte o arranjo em duas metades iguais.

- Elementos não ordenados
- Elemento pivô
- Elementos ordenados
- Pivô ordenado



Quick Sort

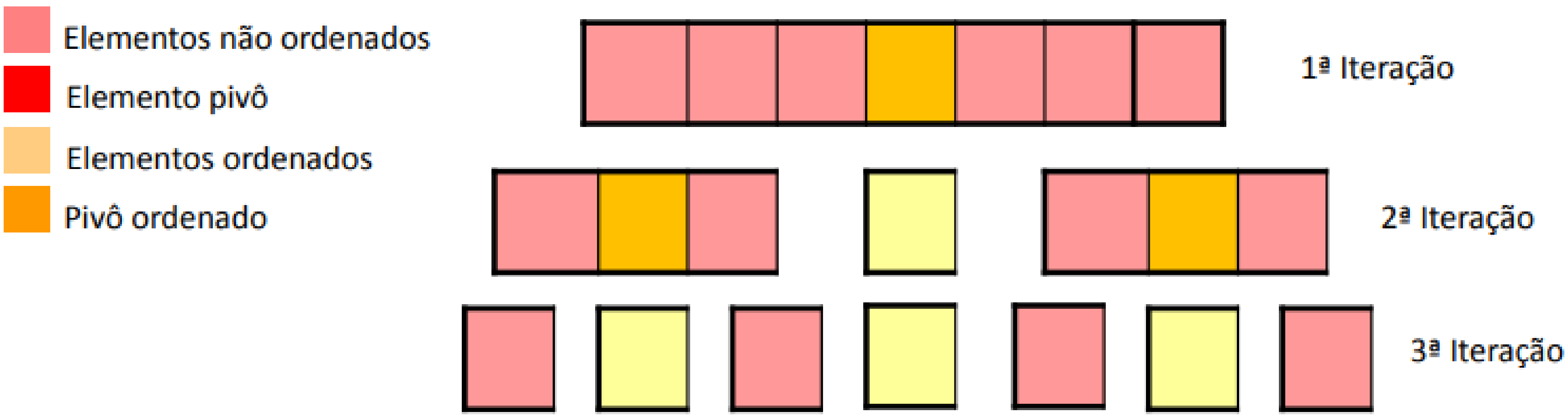
O melhor caso do Quick Sort: Particiona Balanceado
A cada iteração pivô parte o arranjo em duas metades iguais.



Quick Sort

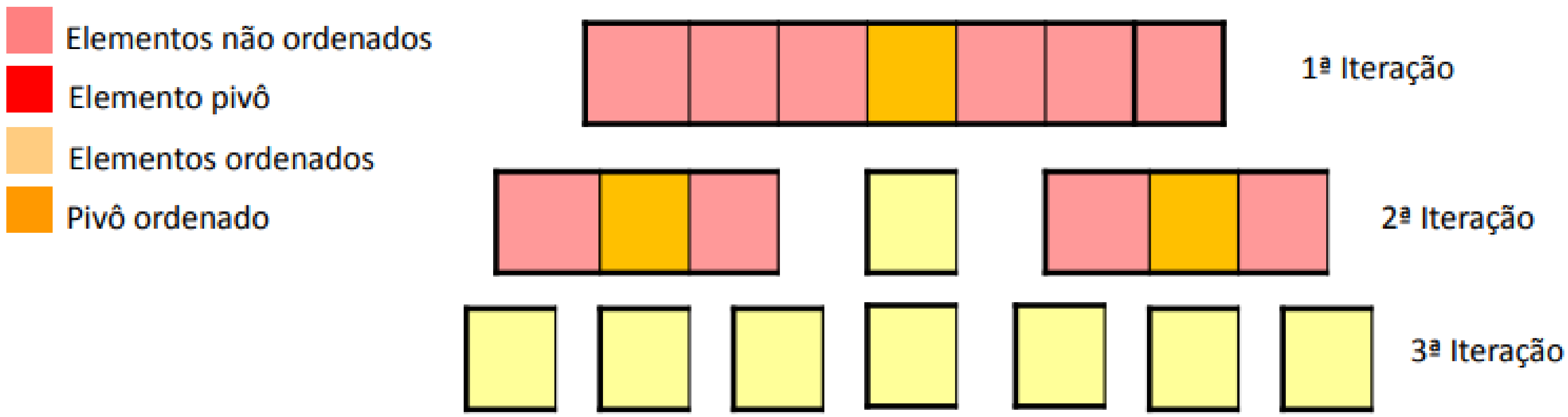
O melhor caso do Quick Sort: Particiona Balanceado

A cada iteração pivô parte o arranjo em duas metades iguais.



Quick Sort

O melhor caso do Quick Sort: Particiona Balanceado
A cada iteração pivô parte o arranjo em duas metades iguais.



A cada iteração o vetor é particionado em n/2 elementos.

Quick Sort

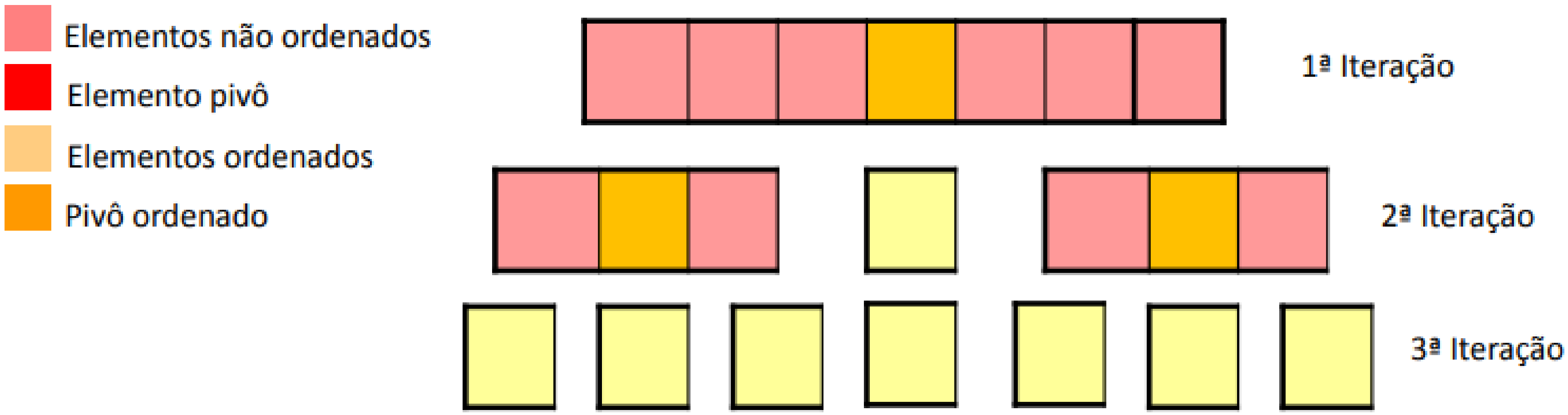
```
                                                 // Consumo de Tempo?
                                                 // (Melhor caso)
01.  void quickSort(int *v, int e, int d)        {
02.       int p;                                 // 02. O(1)
03.       if(e < d)                              // 03. O(1)
04.       {
05.            p = particiona(v, e, d);          // 05. O(n)
06.            quickSort(v, e, p-1);             // 06. T(n/2)
07.            quickSort(v, p+1, d);             // 07. T(n/2)
08.       }
09.  }
```

```
// Total: T(n) = 2*T(n/2) + O(n) + 2*O(1)
              = 2*T(n/2) + O(n)
              = O(n * log n)
```

Quick Sort

O melhor caso do Quick Sort: Particiona Balanceado

A cada iteração pivô parte o arranjo em duas metades iguais.



A cada iteração o vetor é particionado em n/2 elementos.

$$T(1) = 1$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

Exatamente igual ao Merge Sort

Portanto: **$O(n \log_2 n)$**

Quick Sort

- No Caso Médio o Quick Sort é $\mathbf{O(n \log_2 n)}$
- Prova: Sedgewick Cap. 7 Pg. 311
- No caso médio o número de comparações é cerca de **39%** maior que no melhor caso.

Quick Sort

- Consumo de Tempo
  - Pior caso: O(n^2)
  - Melhor caso: O(n * log n)
  - Caso Médio: O(n * log n)