INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Estrutura de Dados Básicas I

Algoritmos de ordenação V

Prof. Eiji Adachi M. Barbosa

Objetivos

- Apresentar e implementar o algoritmo de ordenação por partição (*Quick sort*)

Referência extra

- Notas de aula “Análise de algoritmos”, prof. Paulo Feofiloff, IME-USP:
  - http://www.ime.usp.br/~pf/algoritmos/aulas/quick.html

ORDENAÇÃO POR PARTIÇÃO (*QUICK SORT*)

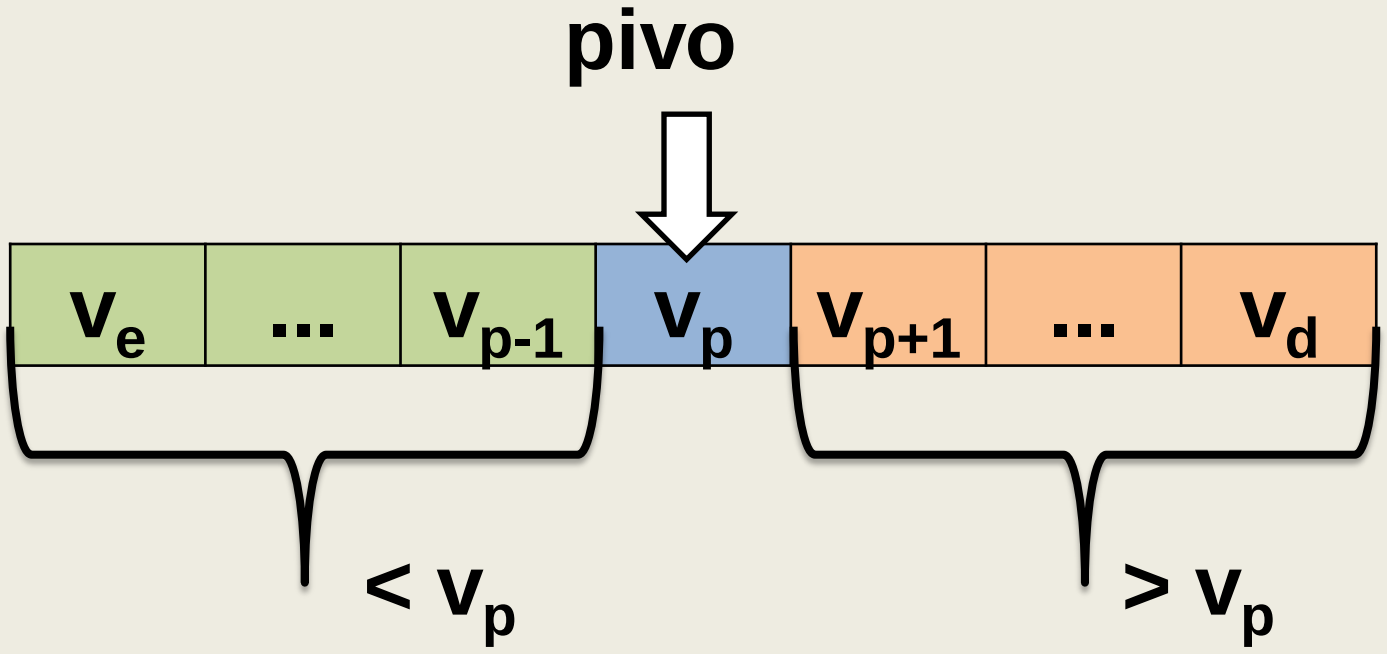
Ordenação por partição

- Estratégia “Dividir para conquistar”:

- Para ordenar um vetor v[e...d] é necessário:
  - Dividir:
    - Particionar o vetor v[e...d] ao redor do valor v[p] em dois sub-vetores (possivelmente vazios) v[e...p-i] e v[p+1, d] tais que:
      - Todo elemento em v[e...p-1] é menor que v[p]
      - Todo elemento em v[p+1, d] é maior que v[p]
  - Conquistar:
    - Os dois sub-vetores v[e...p-1] e v[p+1...d] são ordenados por chamadas recursivas

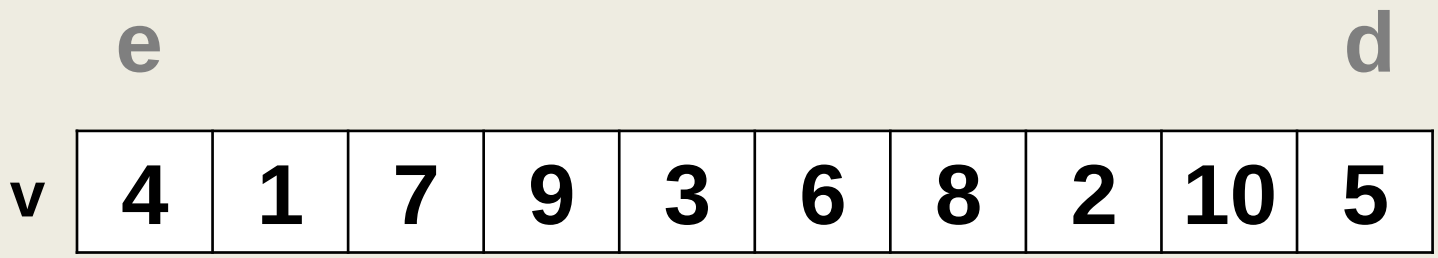
Partição

- Pós condição:

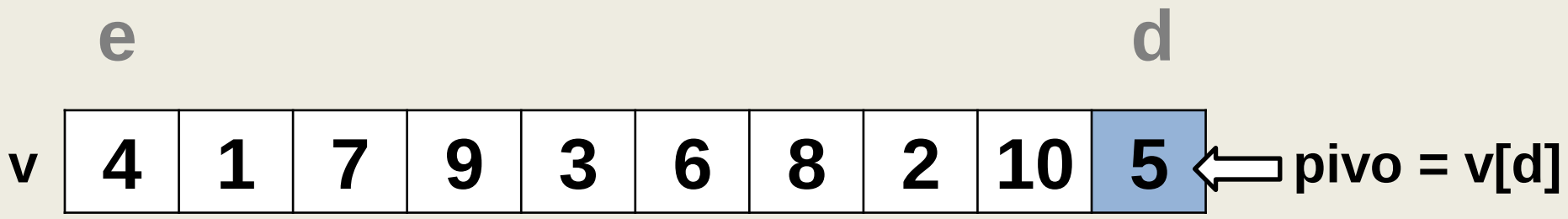


Pivô é o elemento ao redor do qual é feito o particionamento do vetor
(Nós quem definimos quem é o pivô!)

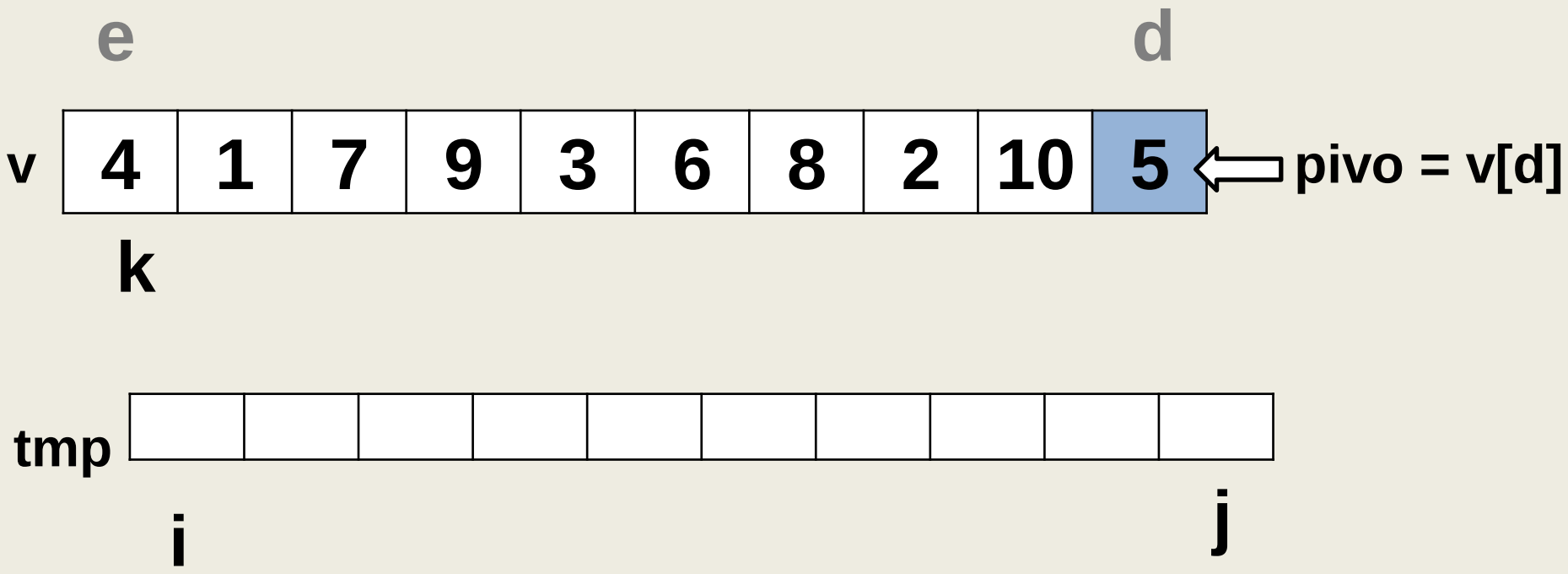
Partição – Versão simples



Partição – Versão simples

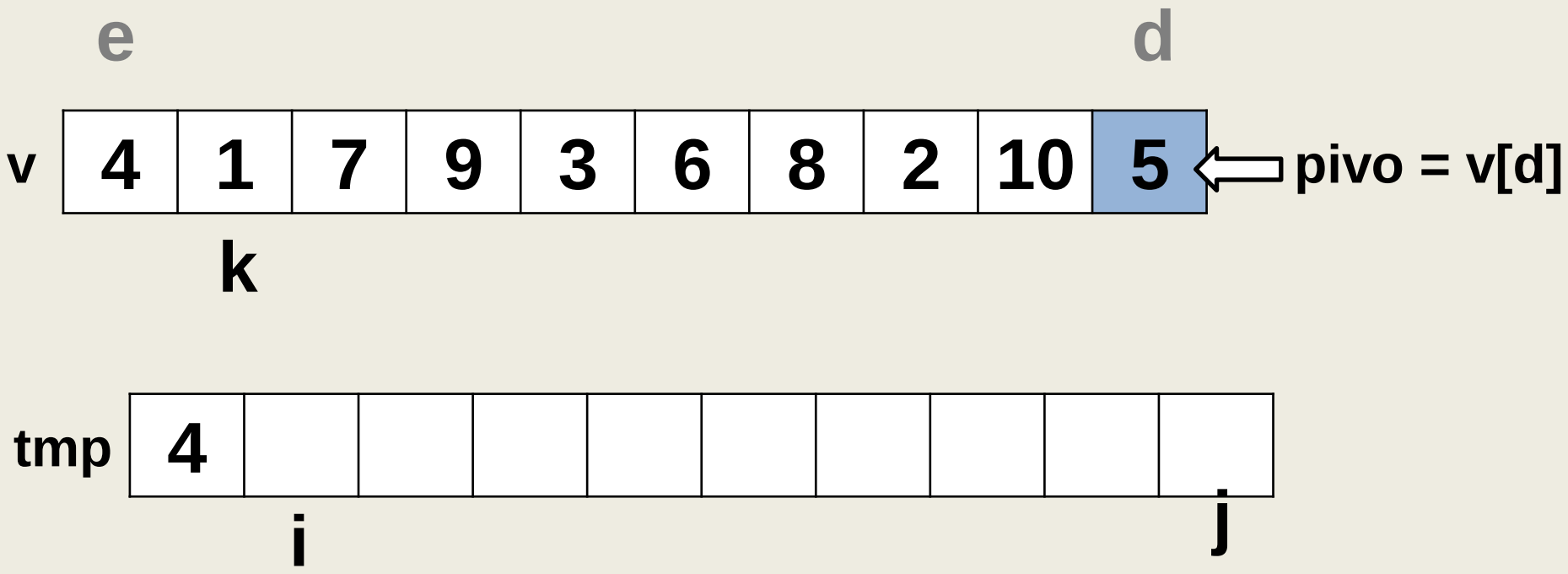


Partição – Versão simples



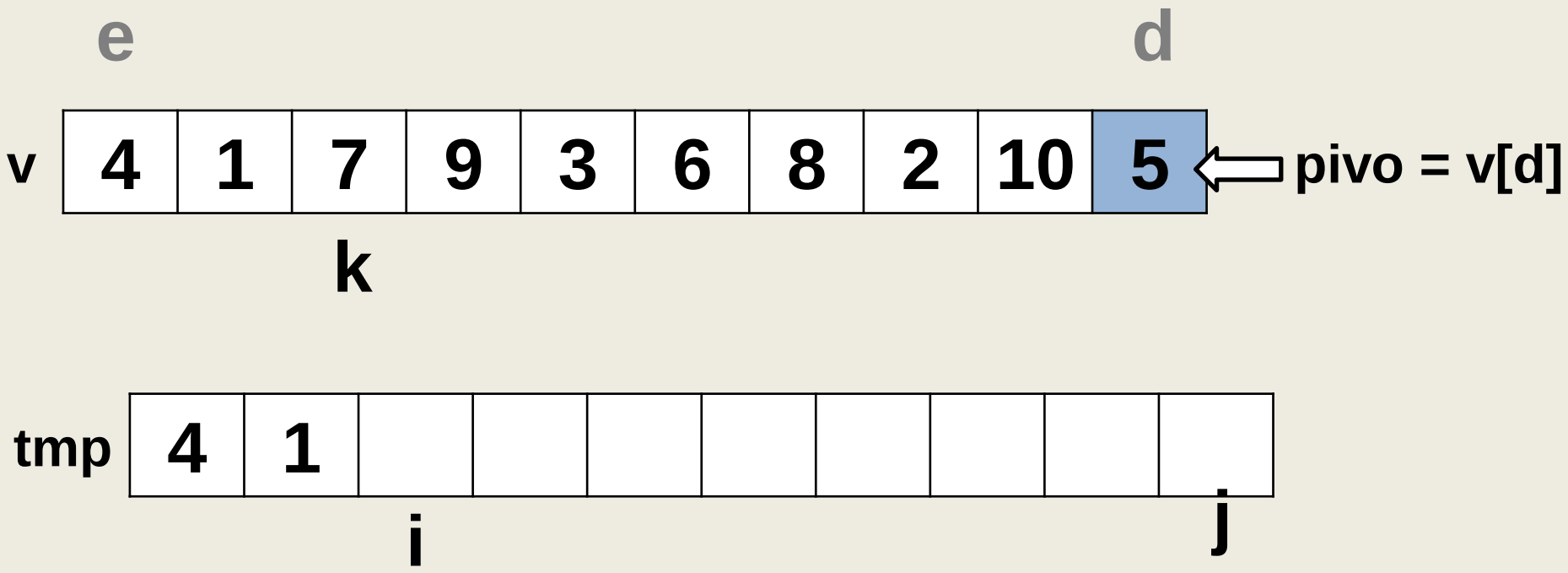

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



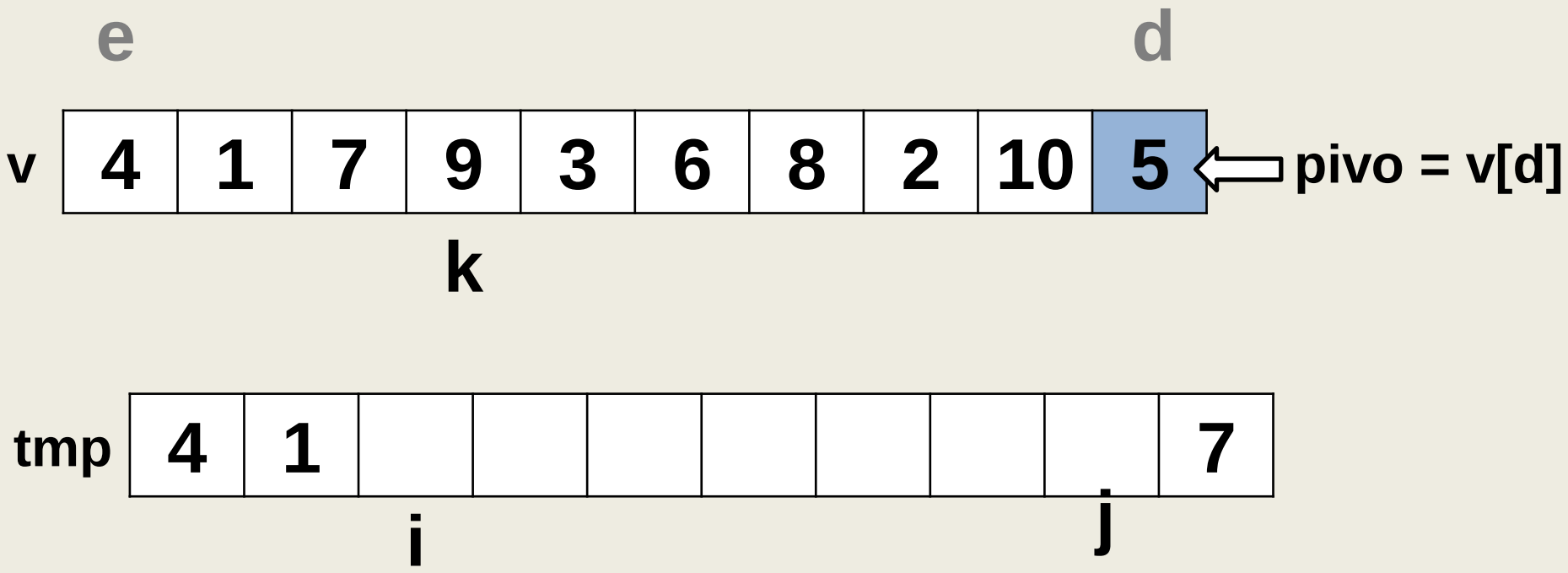

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



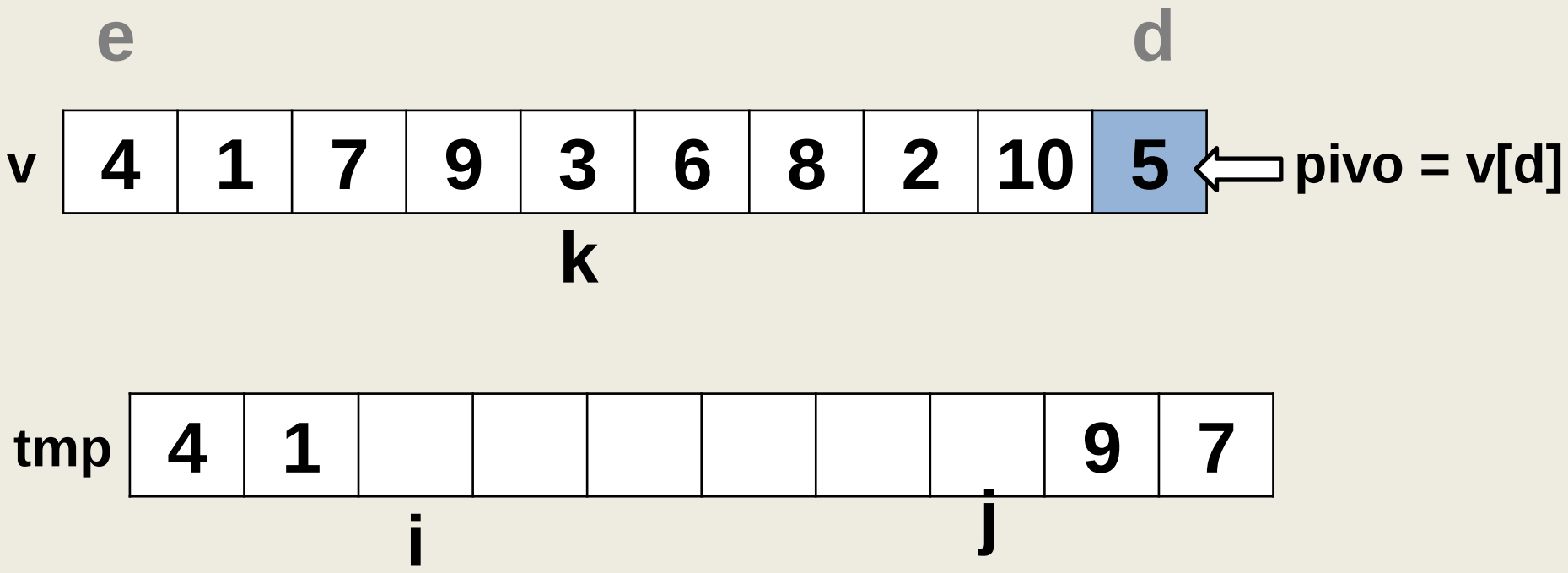

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



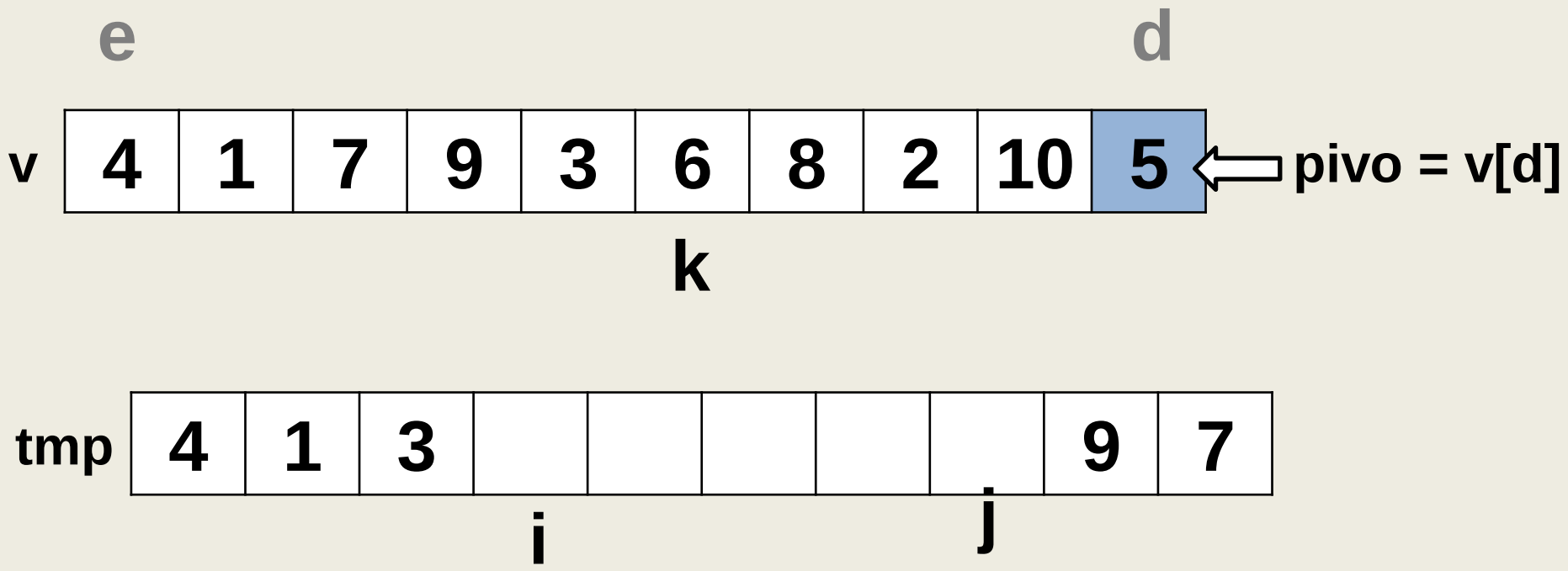

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



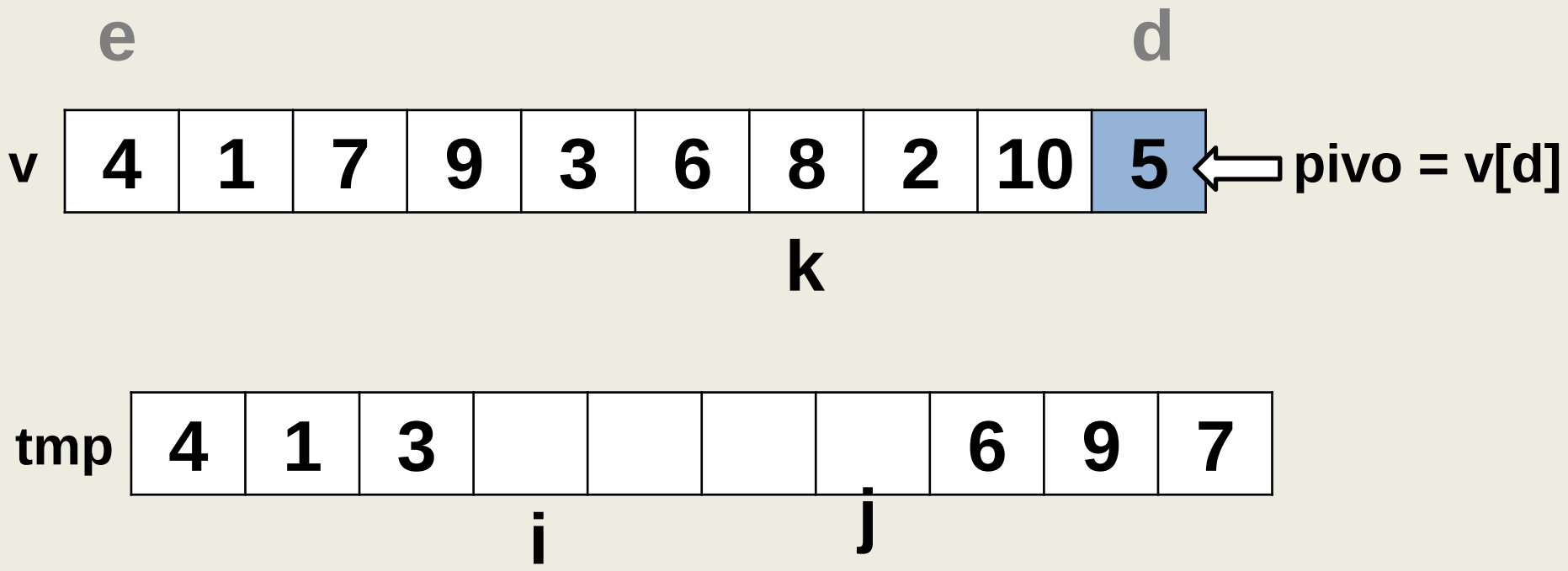

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



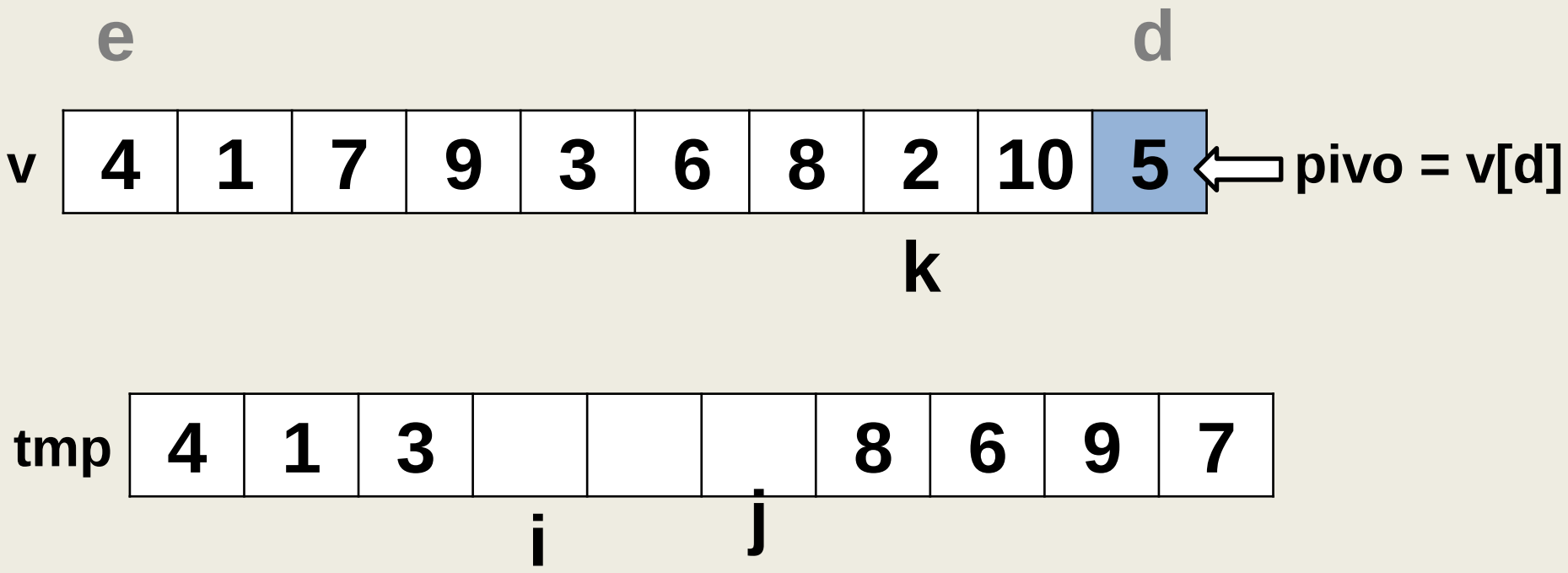

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



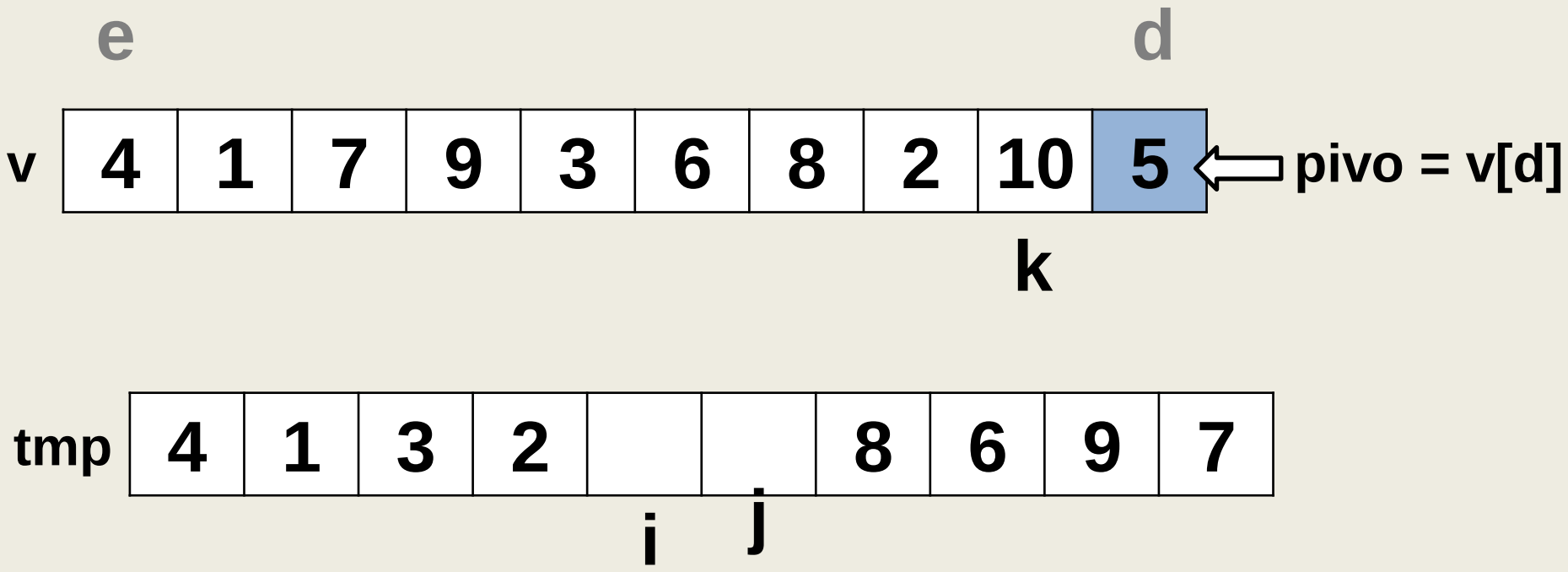

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples



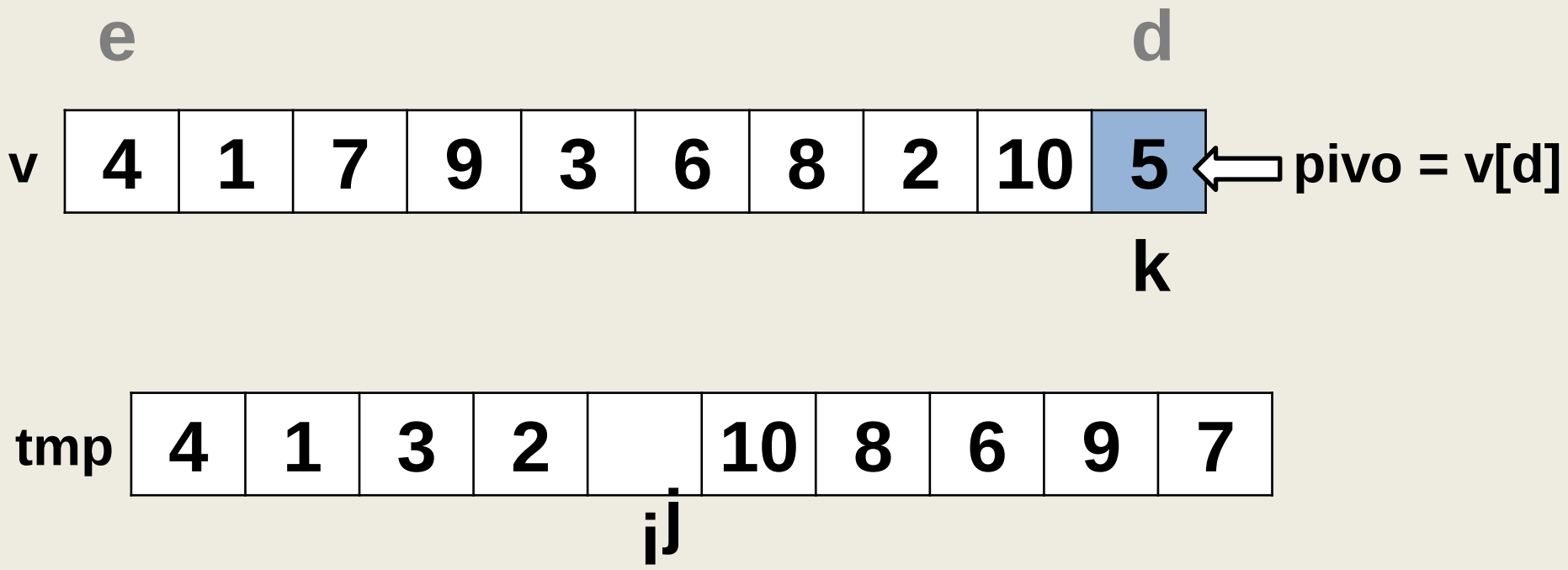

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

Partição – Versão simples




```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

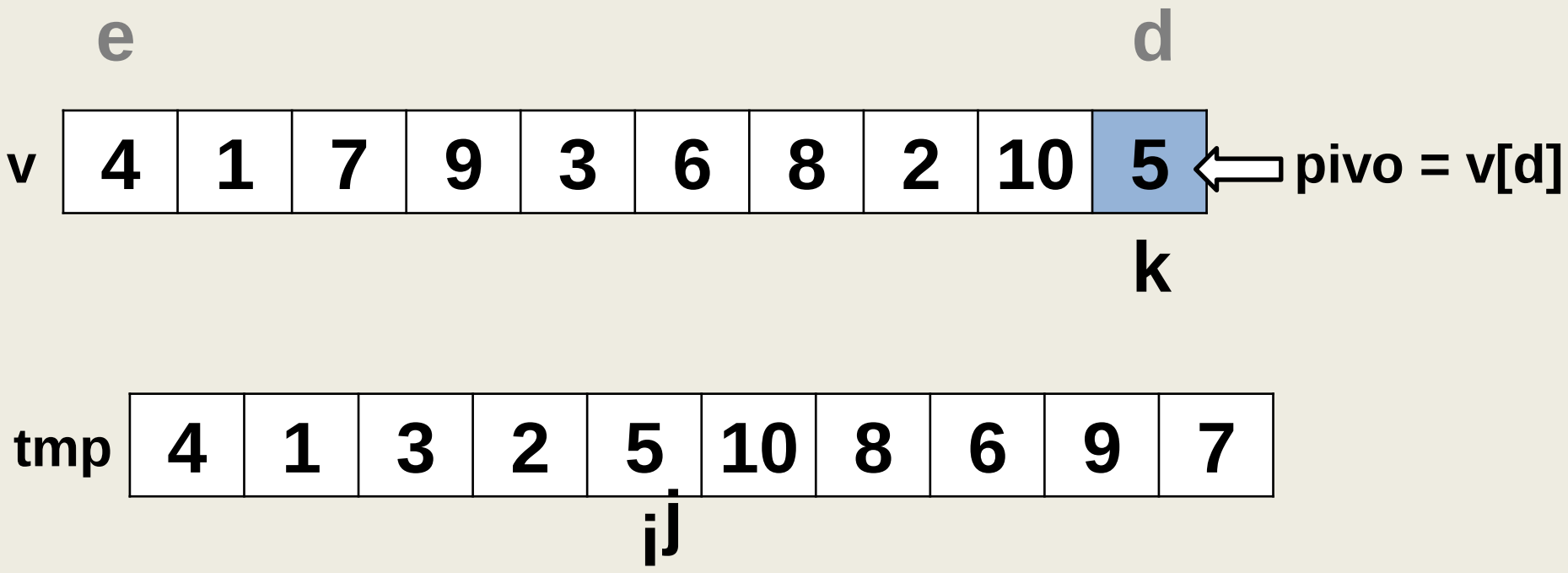
Partição – Versão simples




```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

```
tmp[i] = pivo
```

Partição – Versão simples




```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

```
tmp[i] = pivo
```

Partição – Versão simples

```
Particionar( v[n], esquerda, direita ) :
   pivo = v[direita]
   i = 0, j = 1+direita-esquerda, k = esquerda
   WHILE k < direita :
        IF v[k] < pivo :
               tmp[i] = v[k], i=i+1
        ELSE :
               tmp[j] = v[k], j=j-1
        END_IF
        k = k+1
   END_WHILE
   tmp[i] = pivo,
   COPY tmp in v
   RETORNE i
FIM
```

Qual a complexidade (pior e melhor caso)?

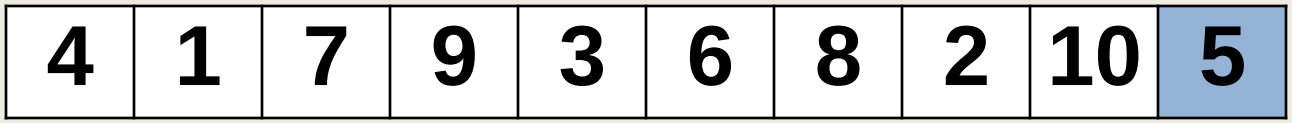
Partição – Versão simples

```
Particionar( v[n], esquerda, direita ) :
   pivo = v[direita]
   i = 0, j = 1+direita-esquerda, k = esquerda
   WHILE k < direita :
        IF v[k] < pivo :
               tmp[i] = v[k], i=i+1
        ELSE :
               tmp[j] = v[k], j=j-1
        END_IF
        k = k+1
   END_WHILE
   tmp[i] = pivo,
   COPY tmp in v
   RETORNE i
FIM
```

Qual a complexidade (pior e melhor caso)?

Já sei particionar, mas como **ordenar o vetor**?

Ordenação por partição – Ex.



Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7

Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						

Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						

Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						

Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1									

Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1									

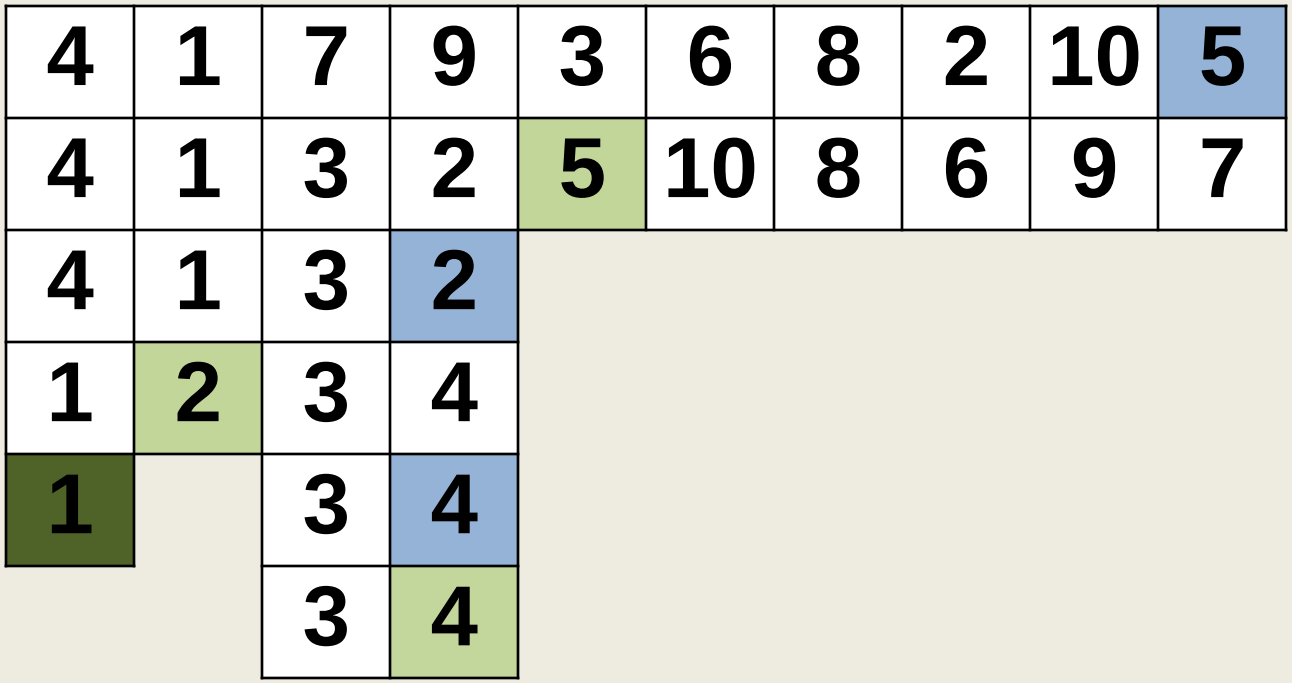
Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1		3	4						

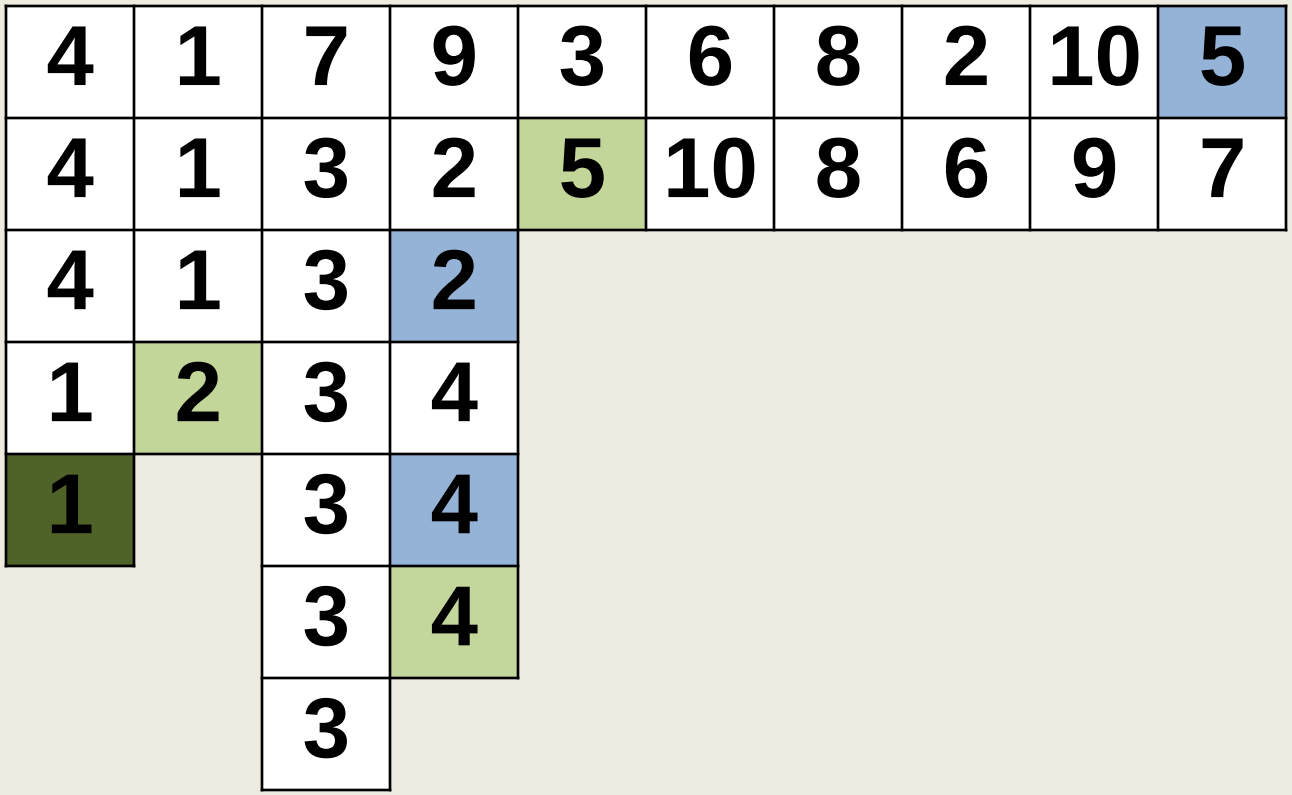
Ordenação por partição – Ex.

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1		3	4						

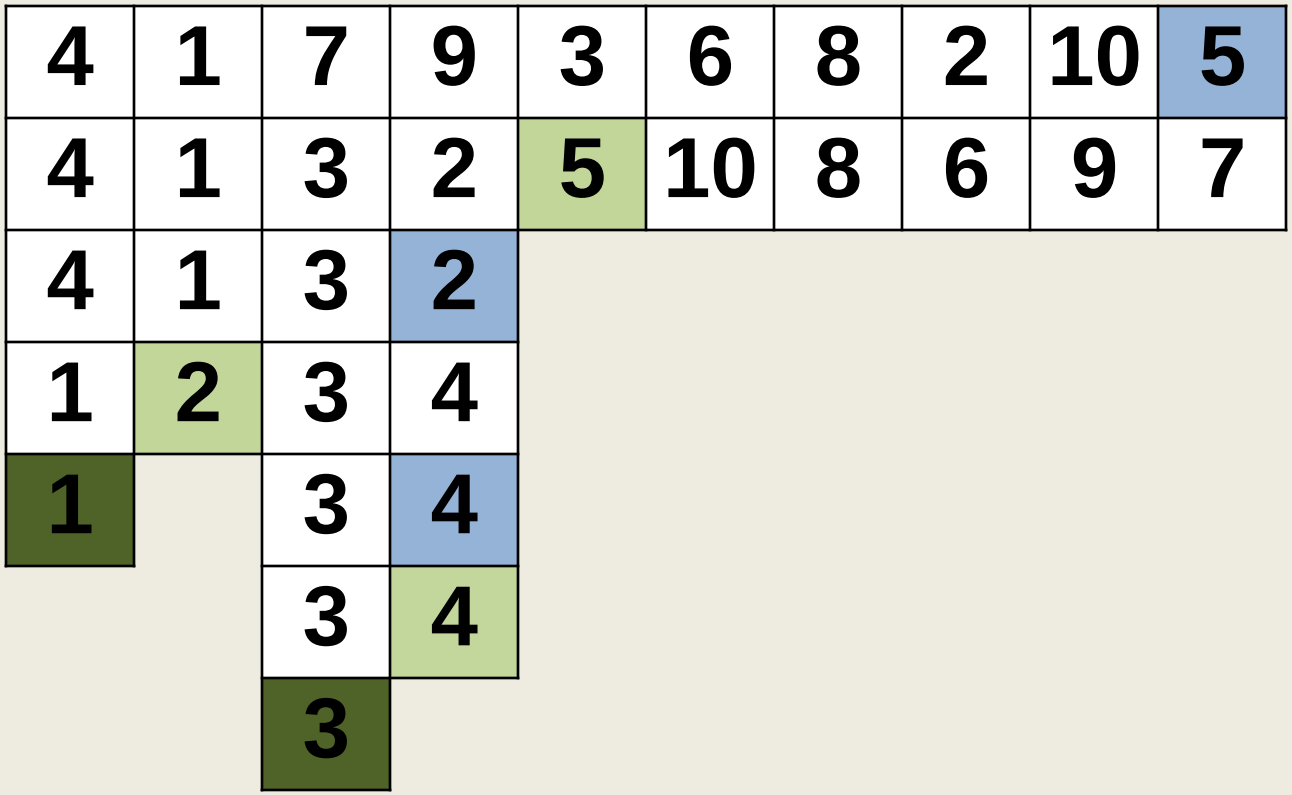
Ordenação por partição – Ex.



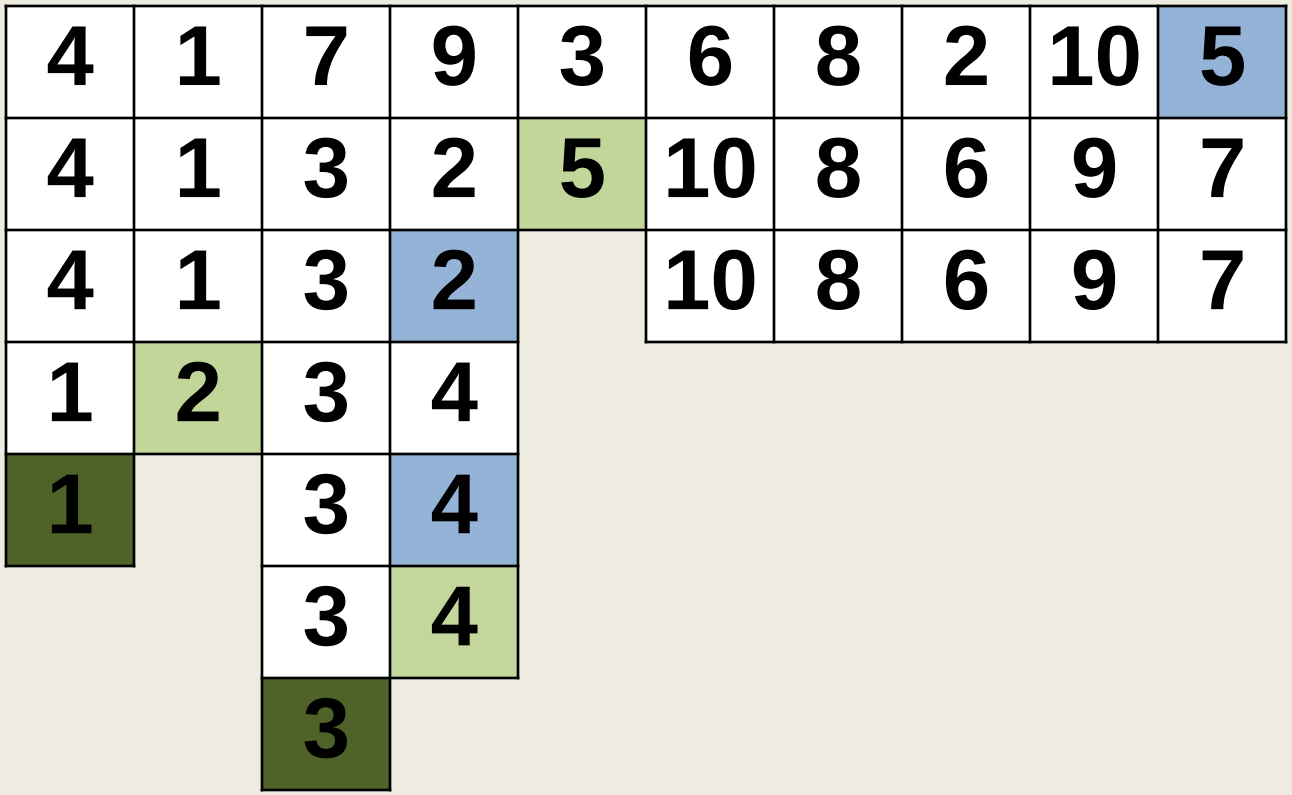
Ordenação por partição – Ex.



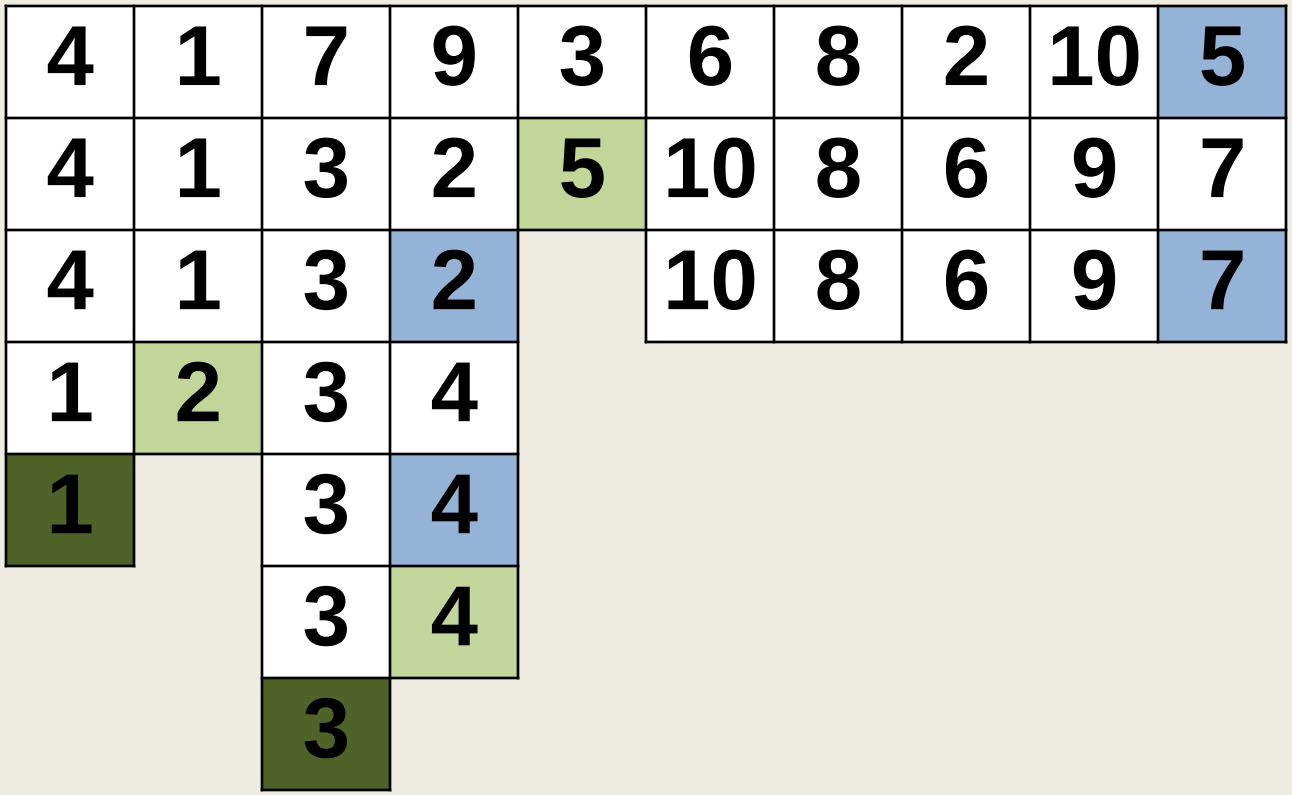
Ordenação por partição – Ex.



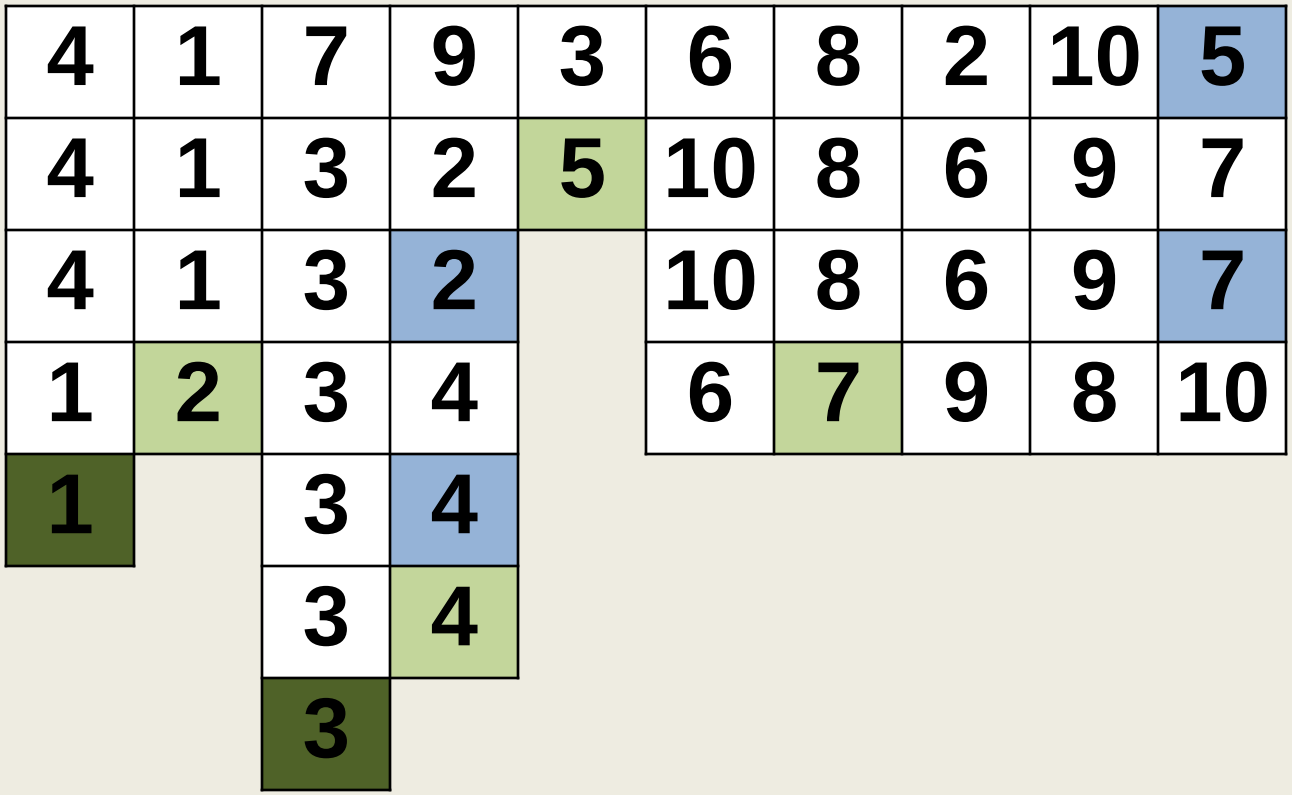
Ordenação por partição – Ex.



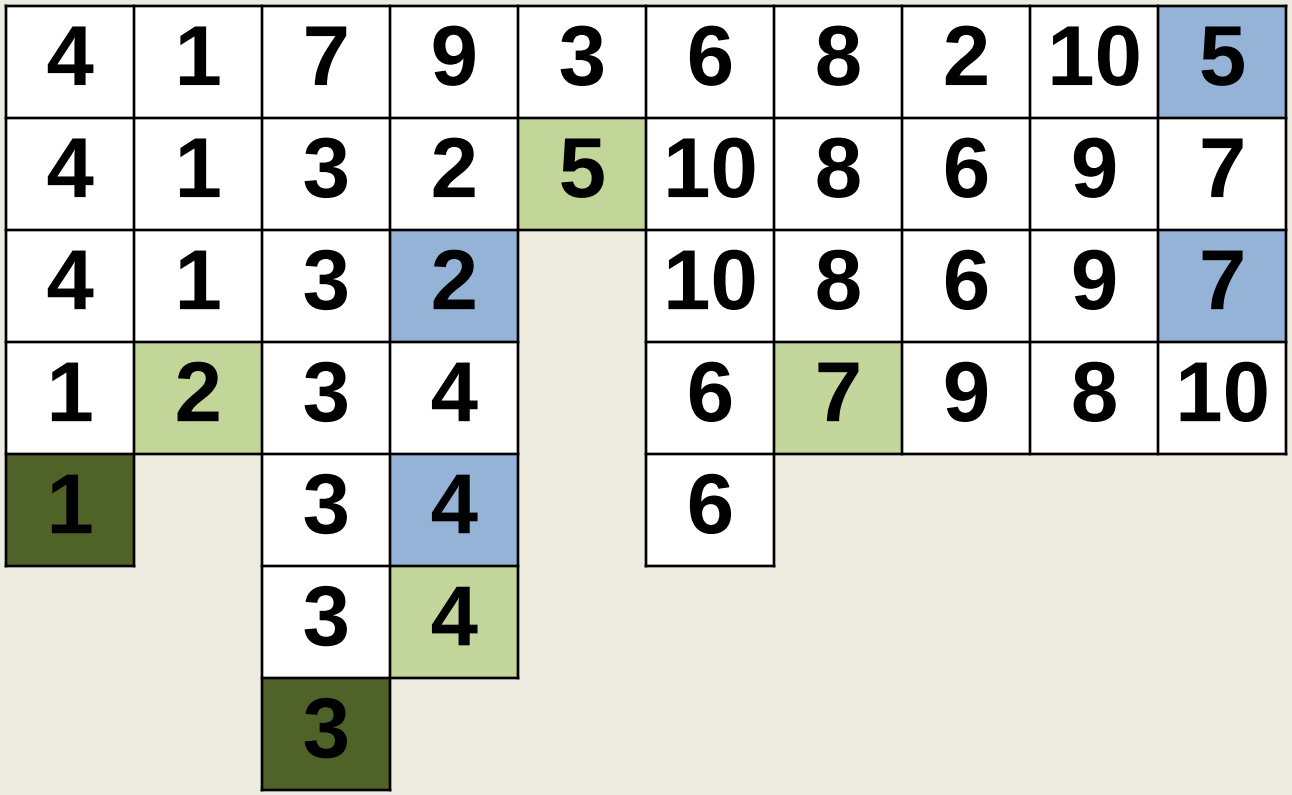
Ordenação por partição – Ex.



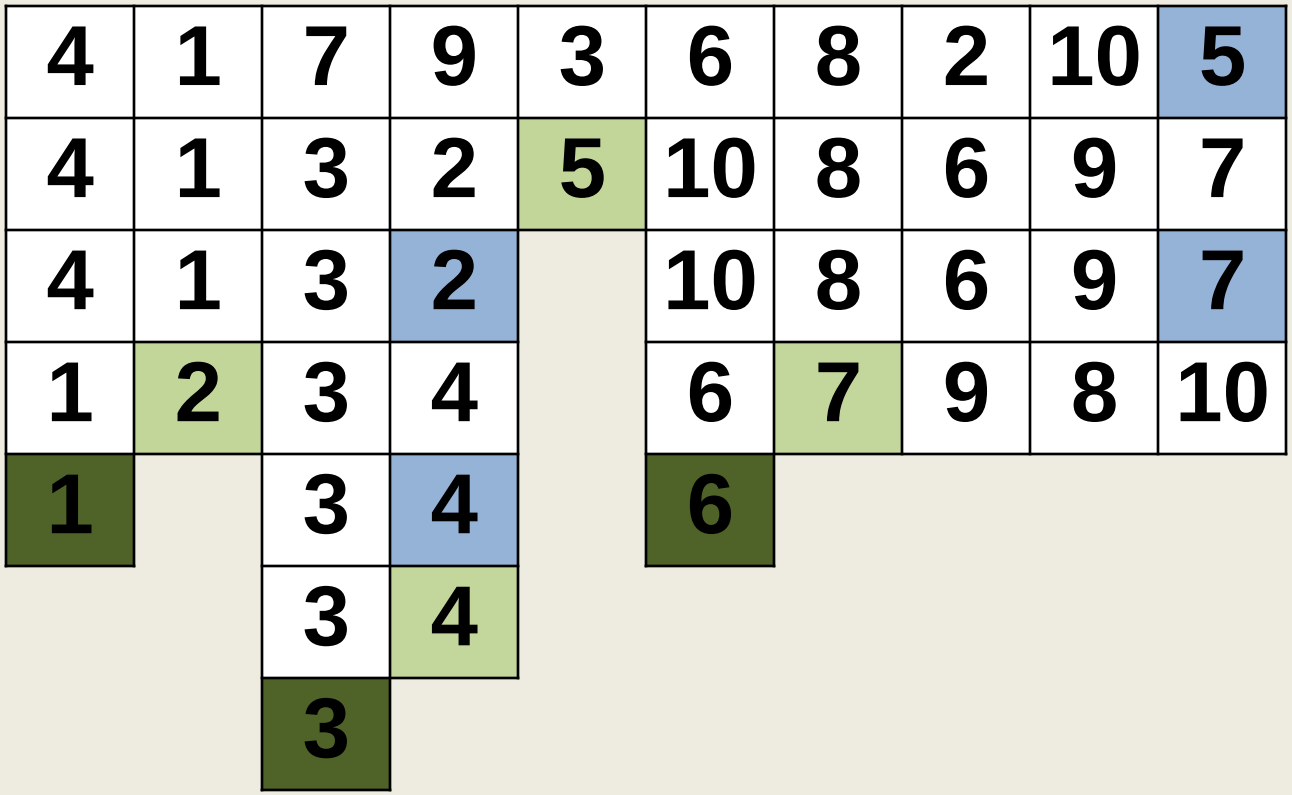
Ordenação por partição – Ex.



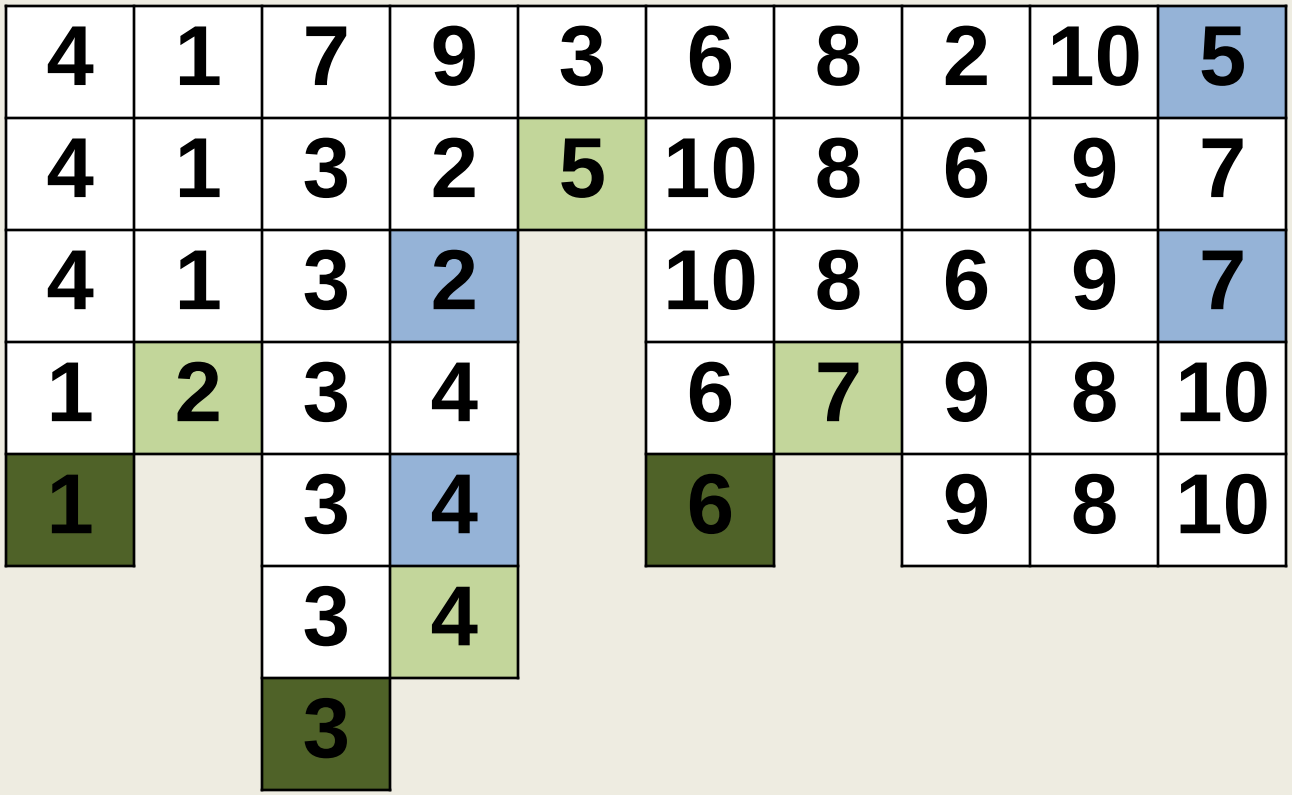
Ordenação por partição – Ex.



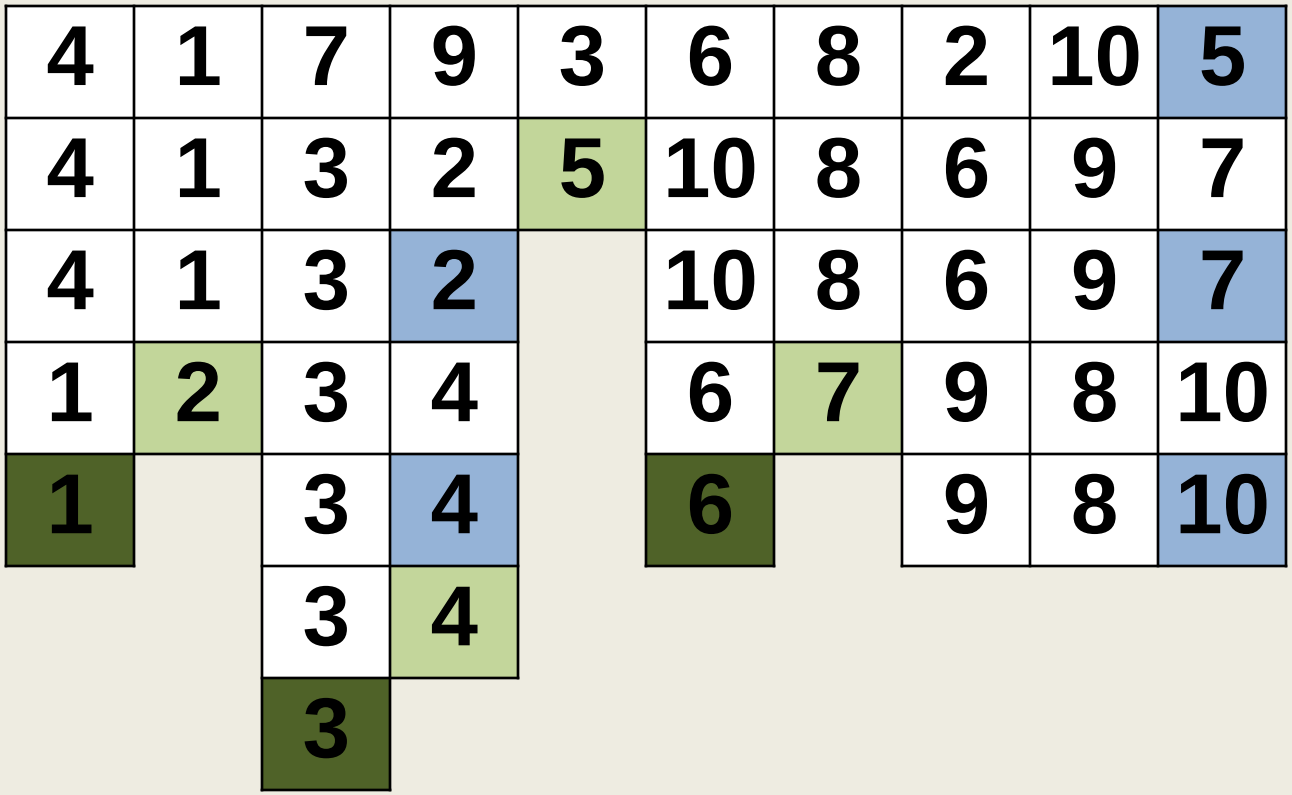
Ordenação por partição – Ex.



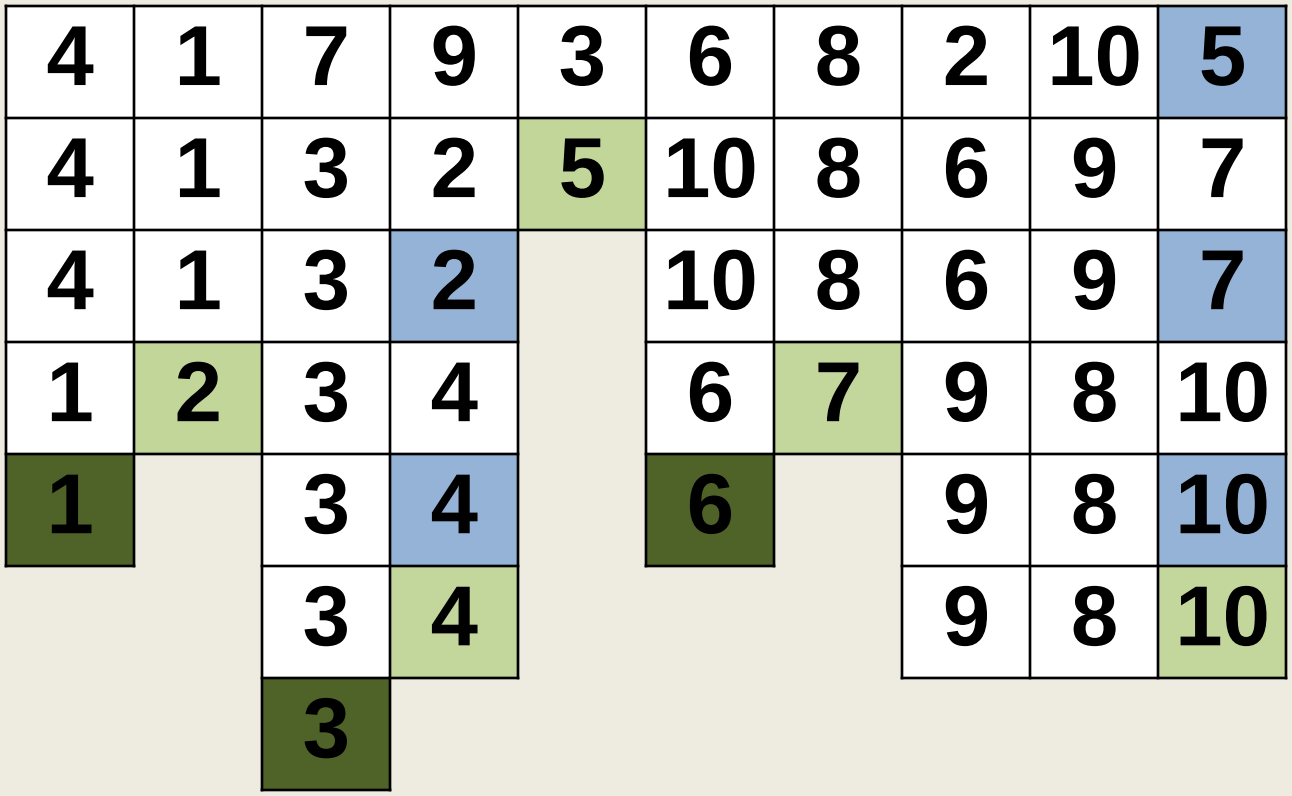
Ordenação por partição – Ex.



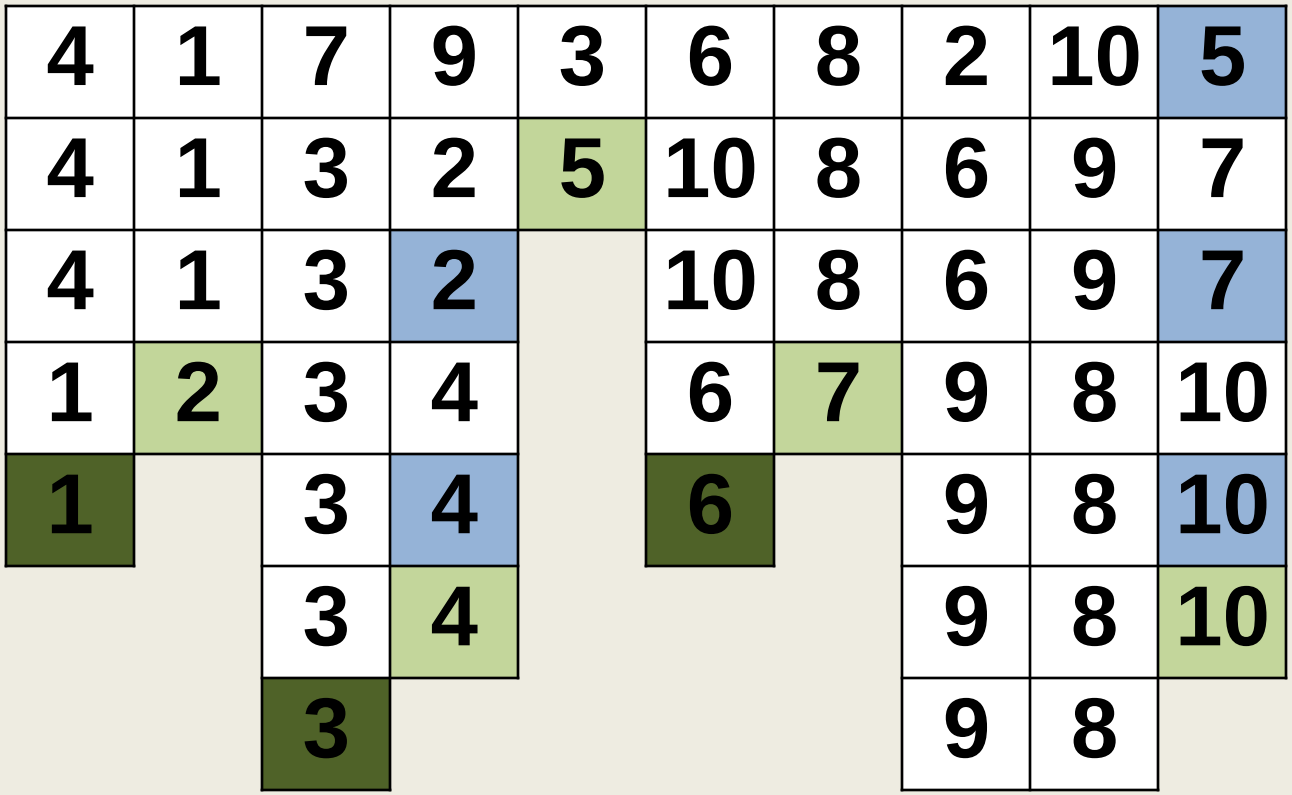
Ordenação por partição – Ex.



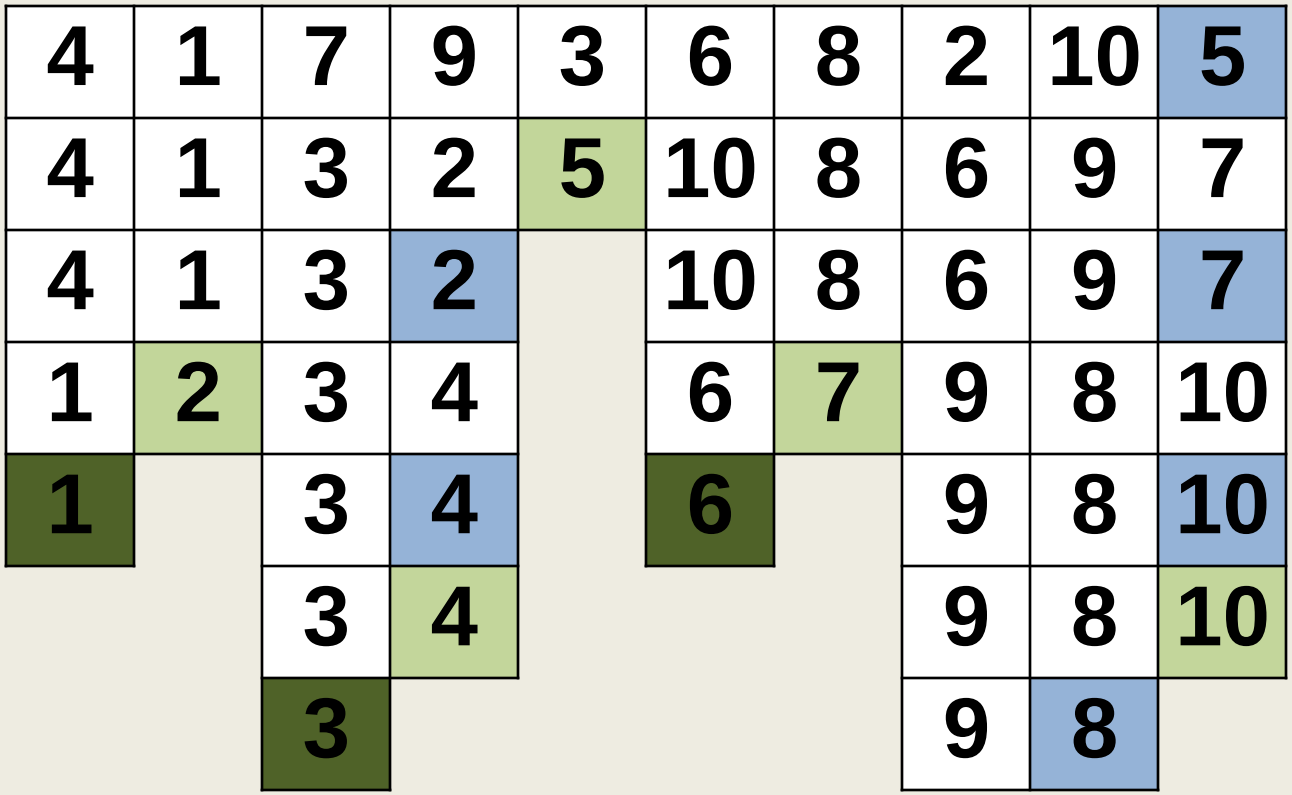
Ordenação por partição – Ex.



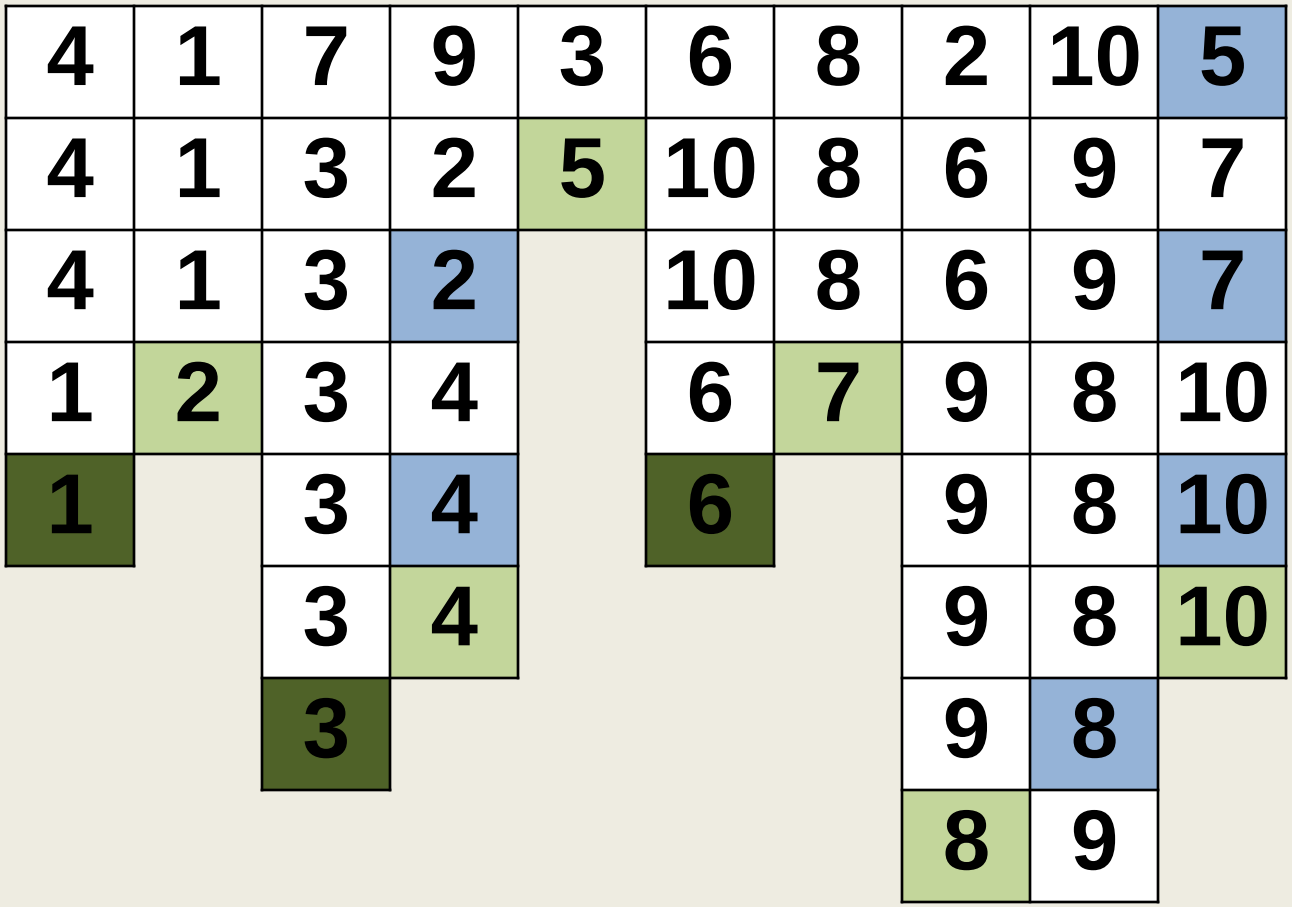
Ordenação por partição – Ex.



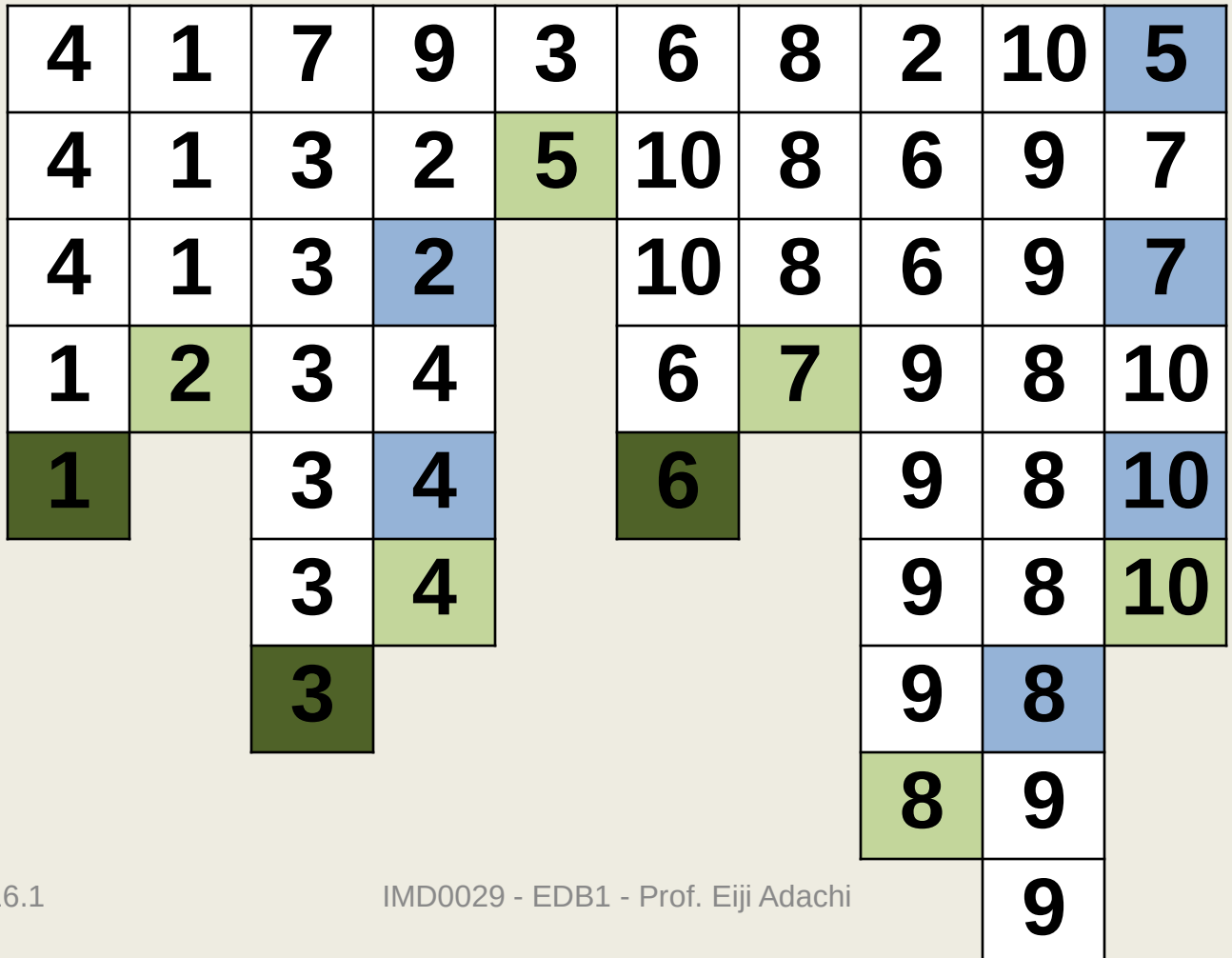
Ordenação por partição – Ex.



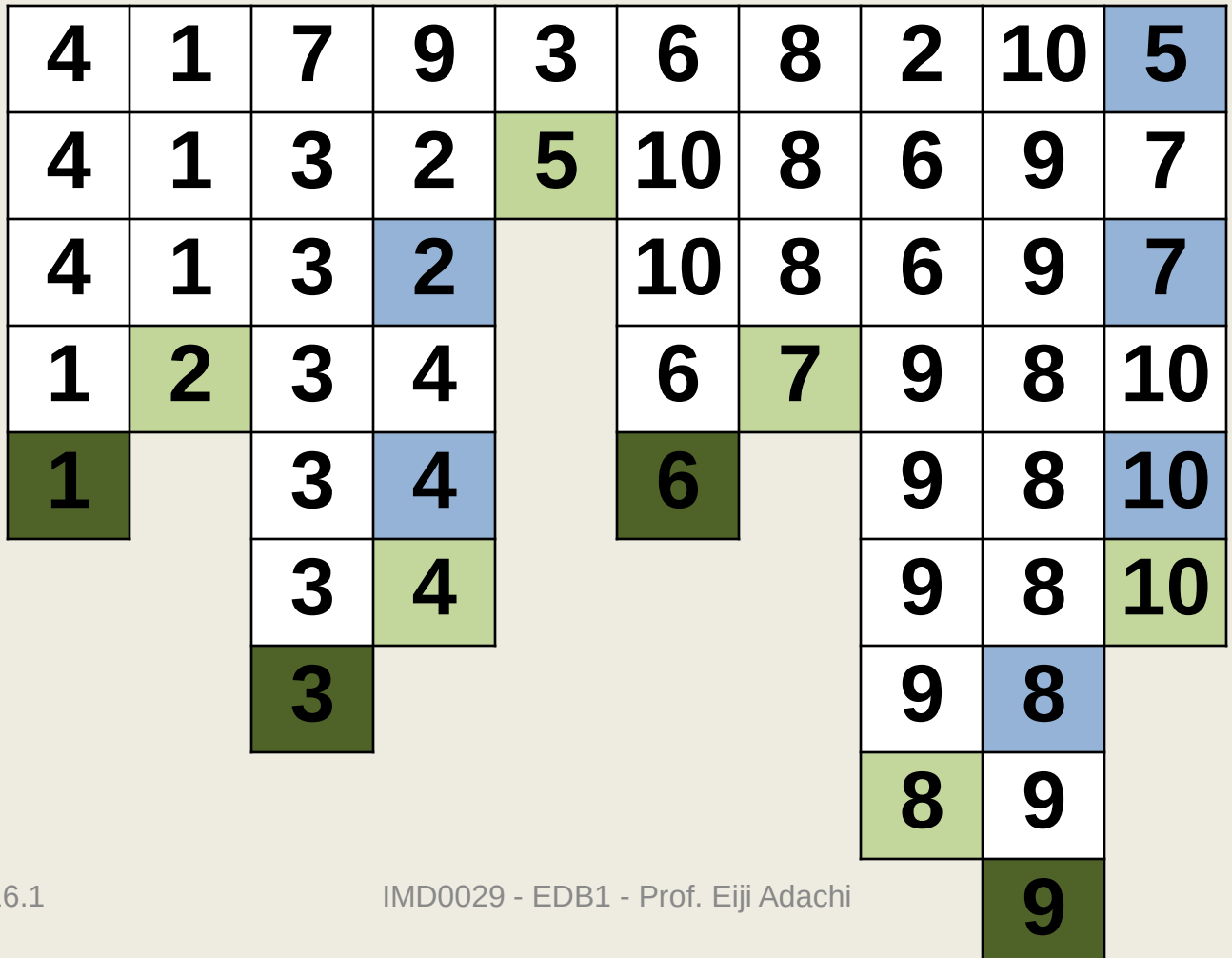
Ordenação por partição – Ex.



Ordenação por partição – Ex.



Ordenação por partição – Ex.

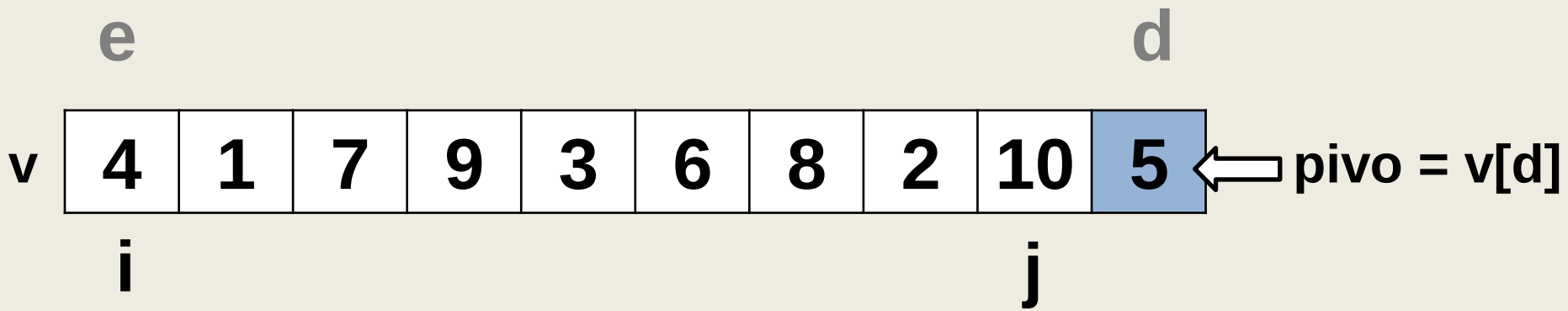


Ordenação por partição

```
OrdenarParticao( v[n], esquerda, direita ) :
   SE esquerda < direita ENTÃO :
       i_pivo = Particionar(v, esquerda, direita )
       OrdenarParticao( v, esquerda, i_pivo-1 )
       OrdenarParticao( v, i_pivo+1, direita )
   FIM_SE
FIM
```

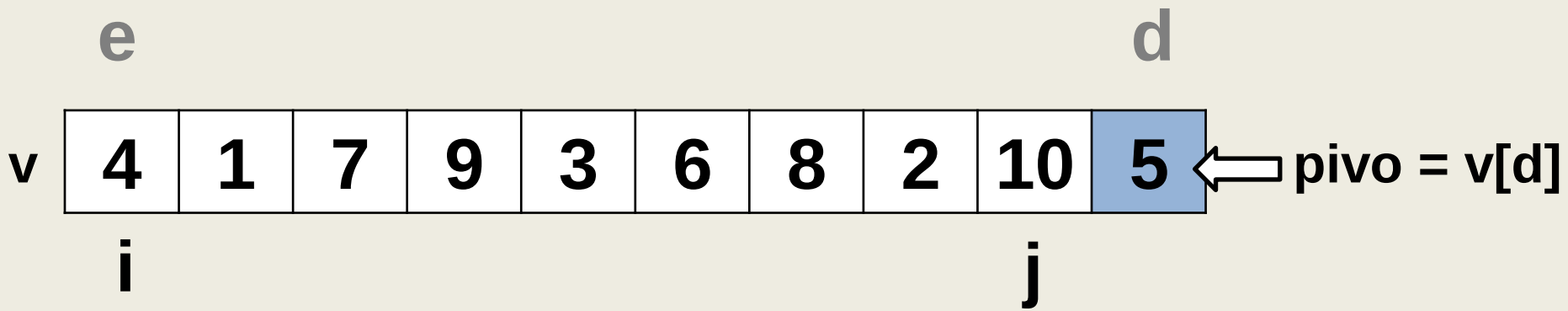
Como particionar sem usar vetor auxiliar?

Partição – Versão otimizada



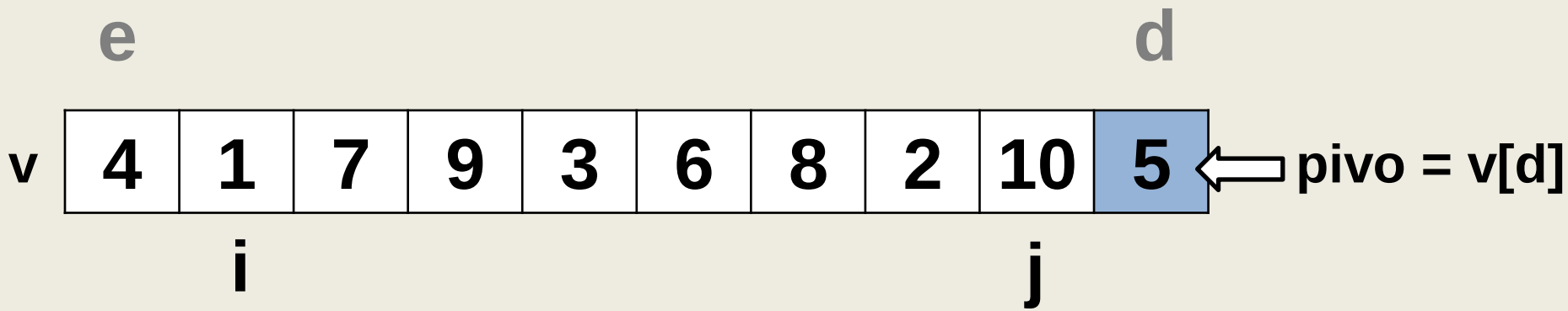

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



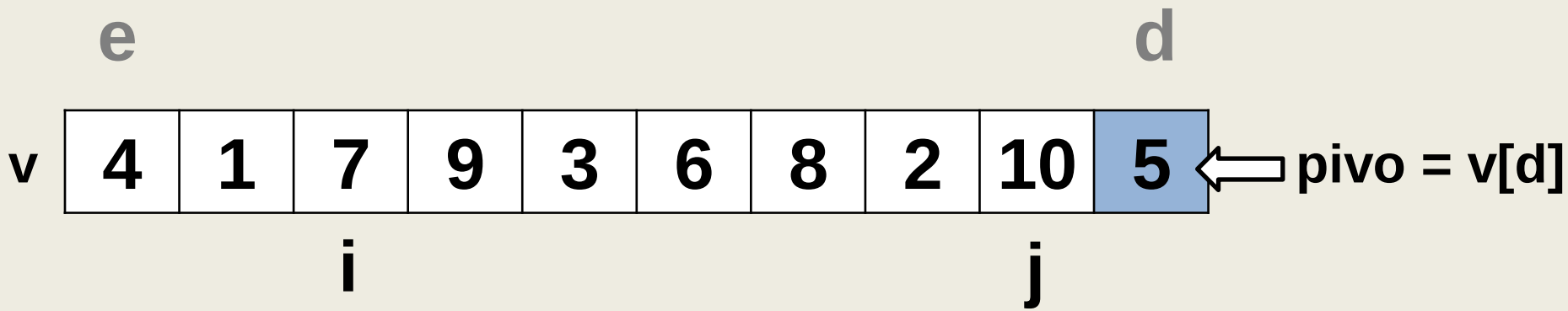

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



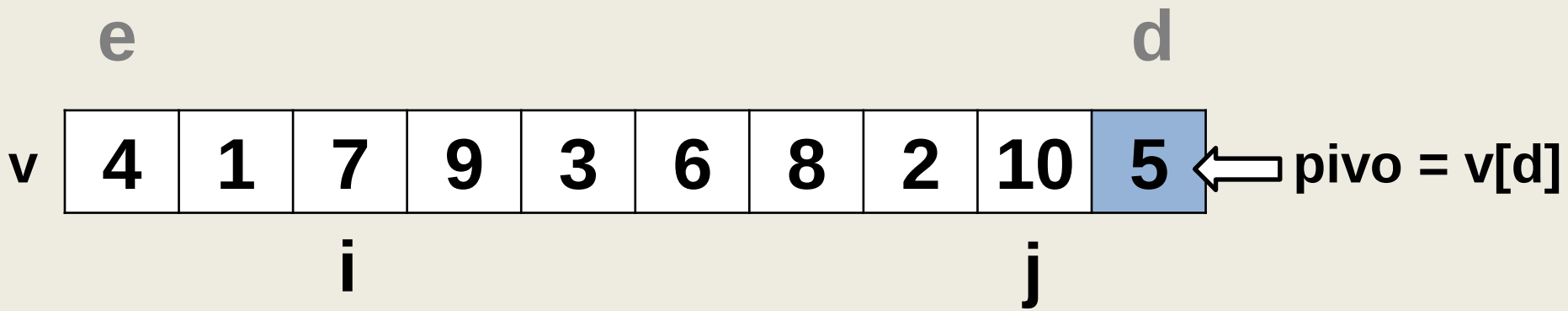

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



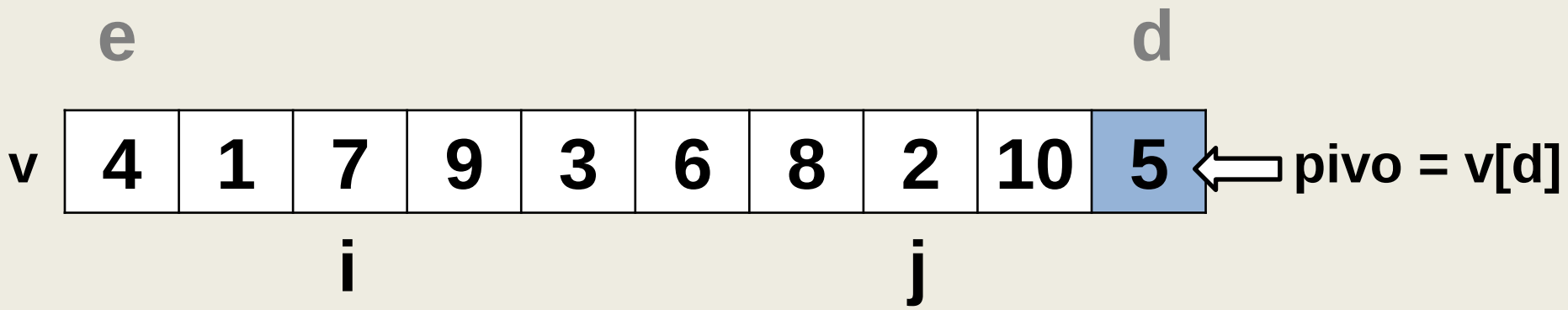

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



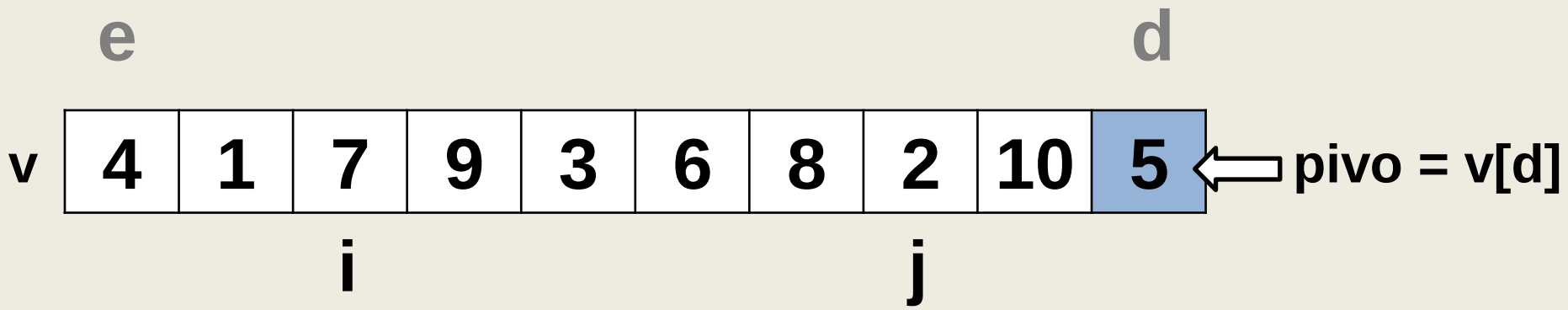

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



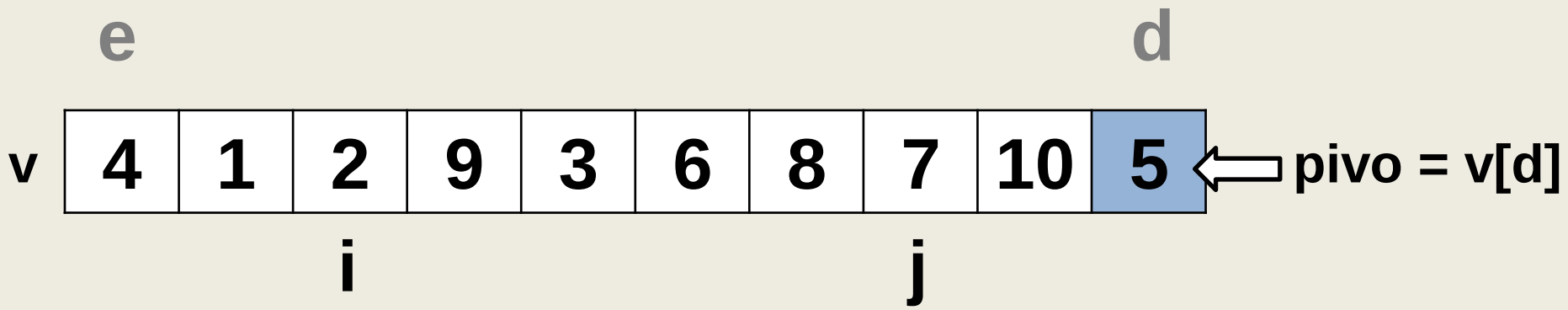

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



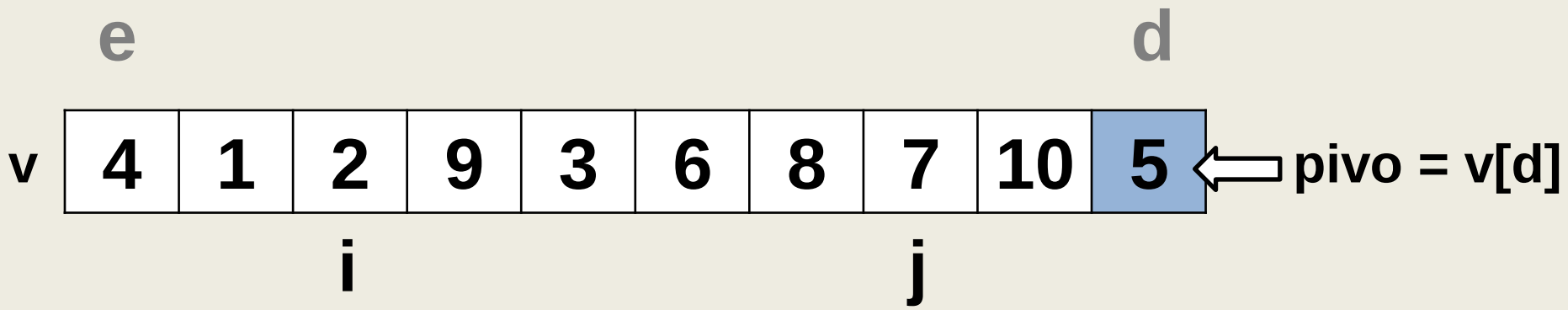

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



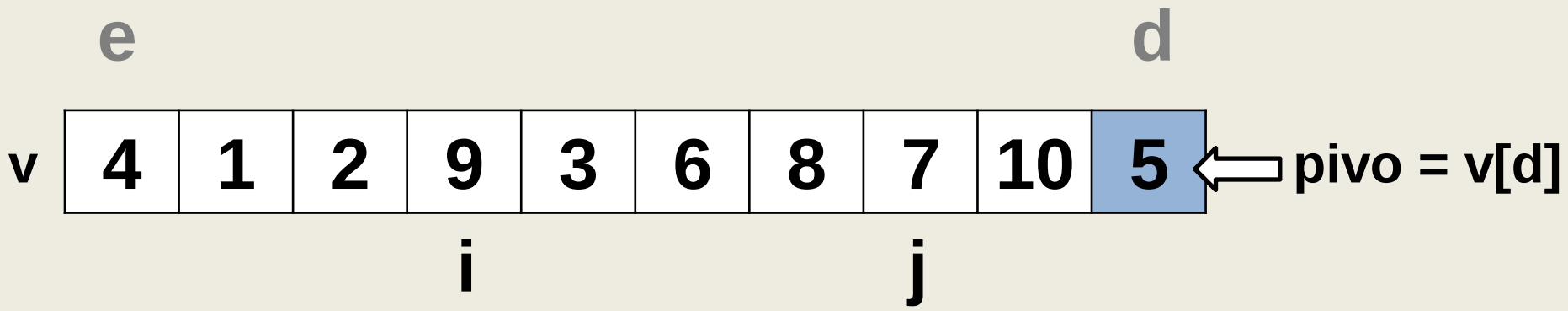

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



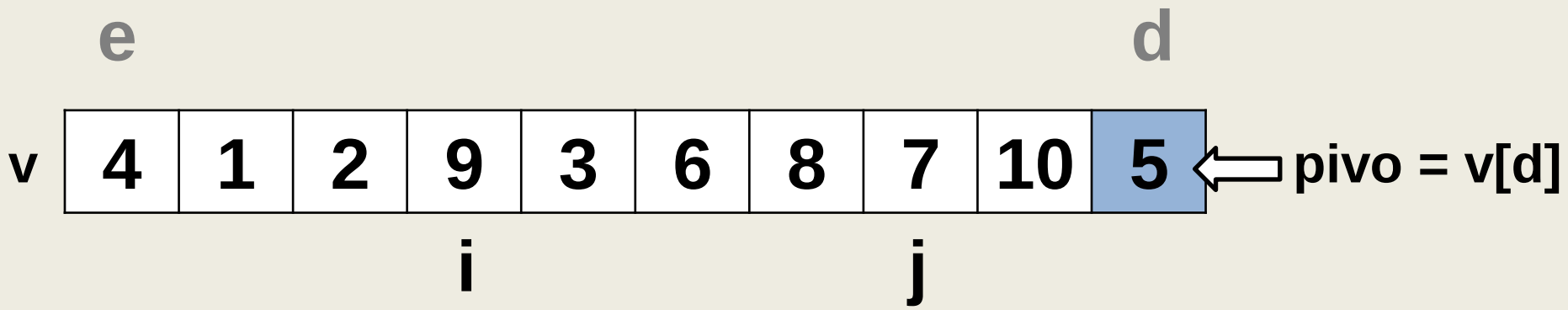

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



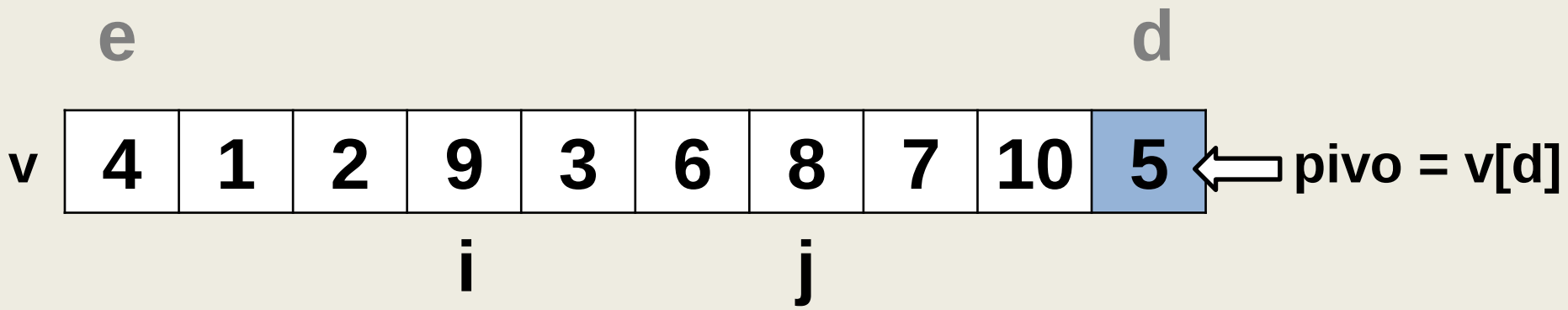

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



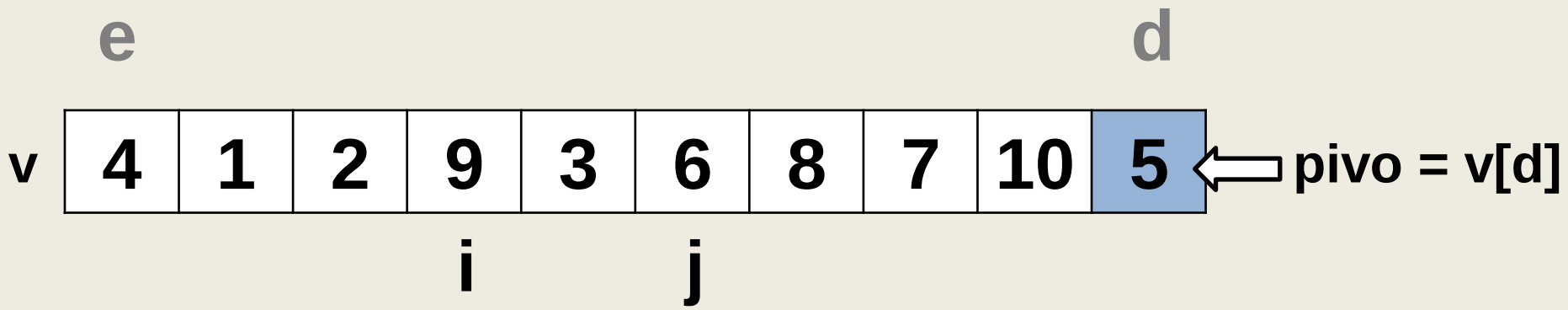

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



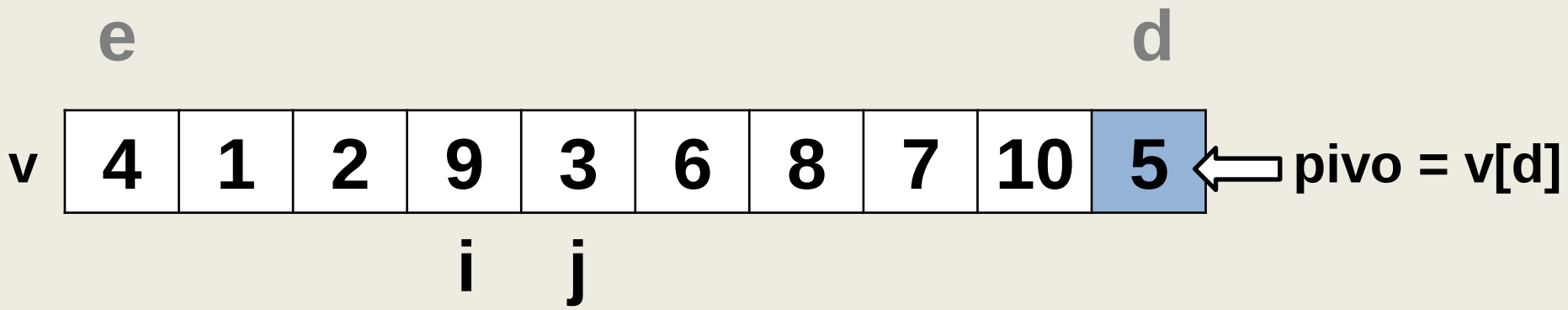

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



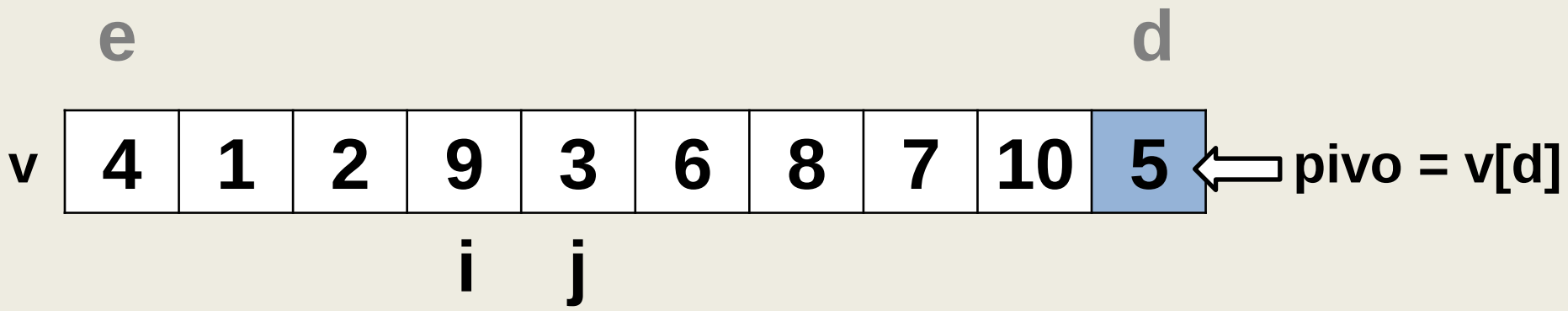

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



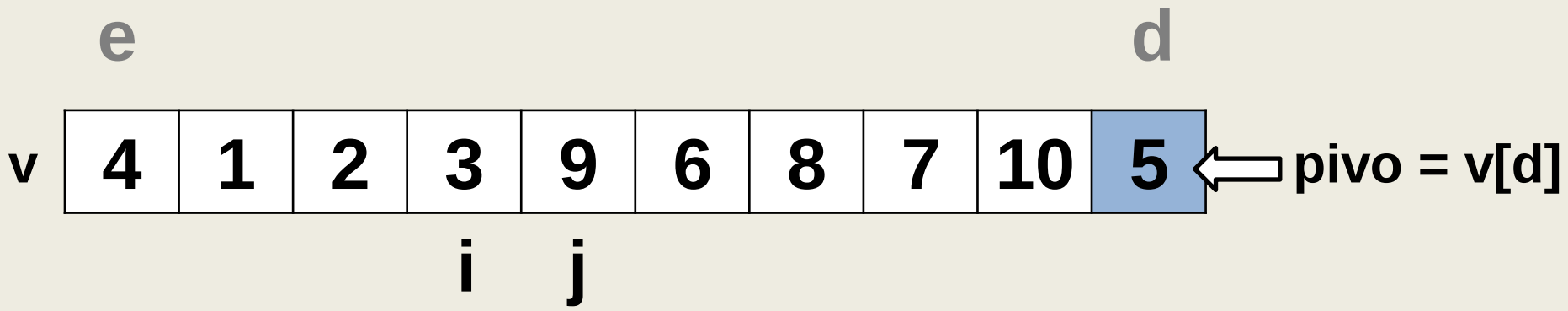

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



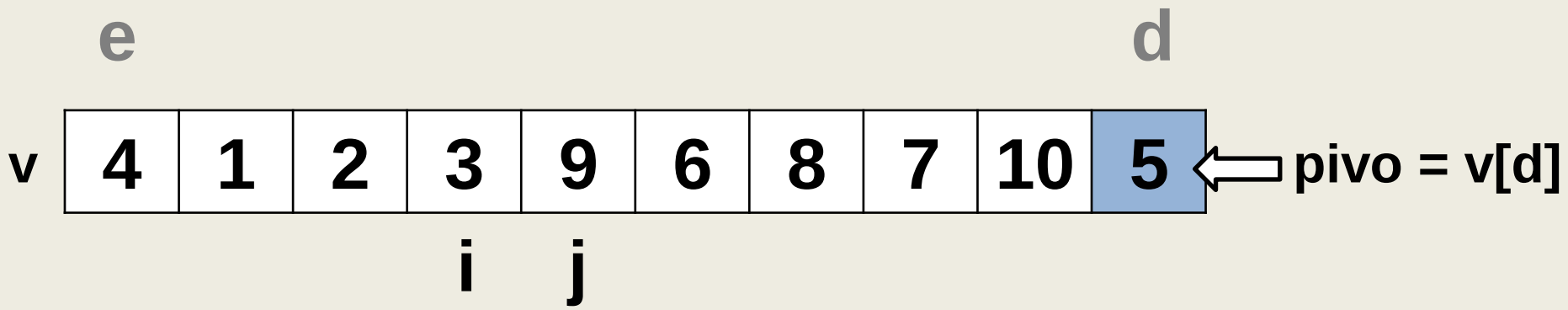

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



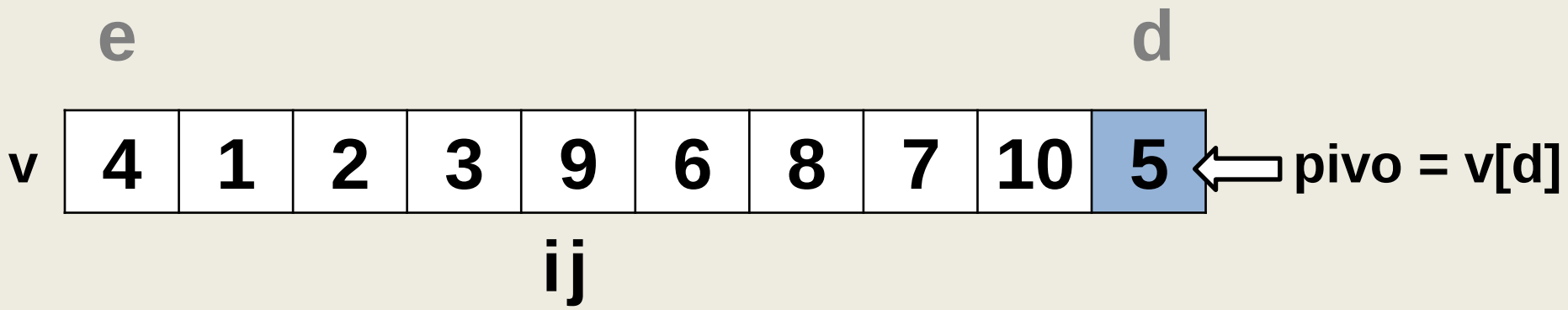

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



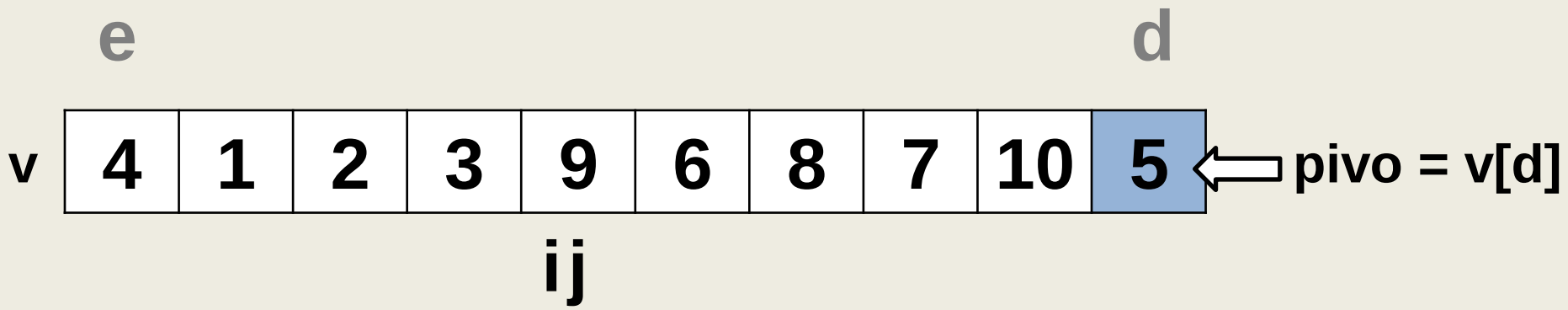

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



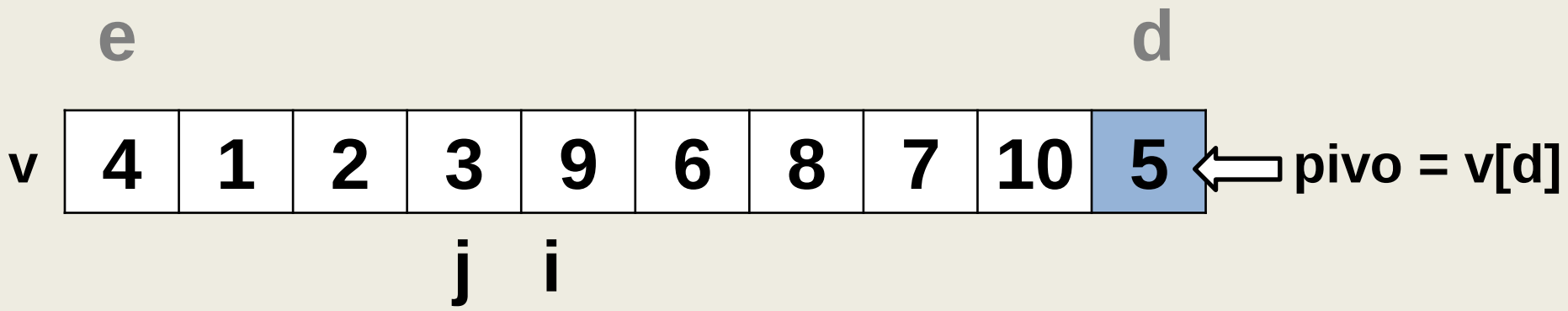

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



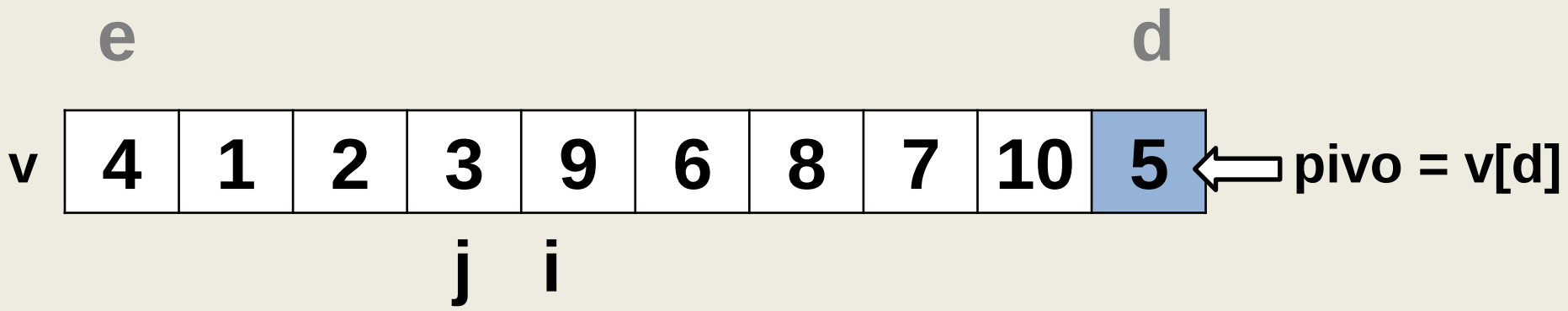

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada



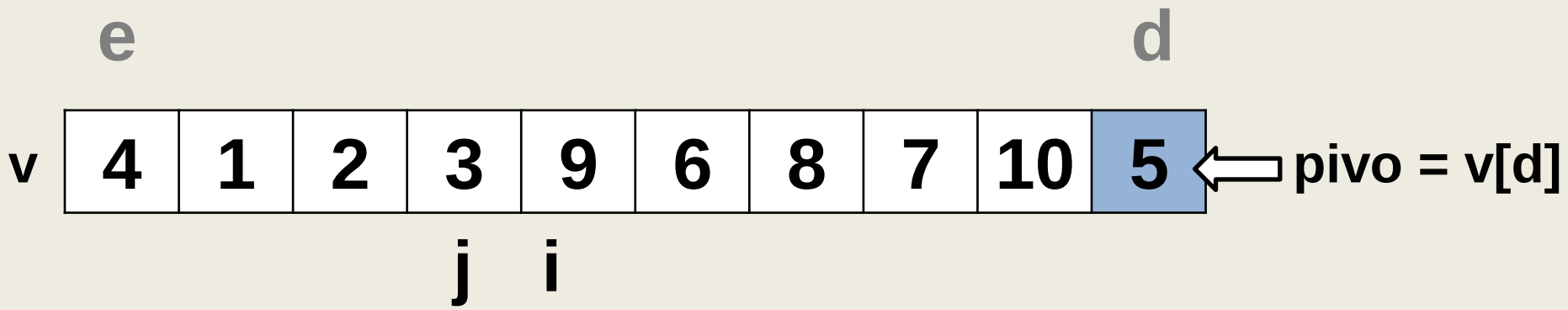

```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Partição – Versão otimizada




```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

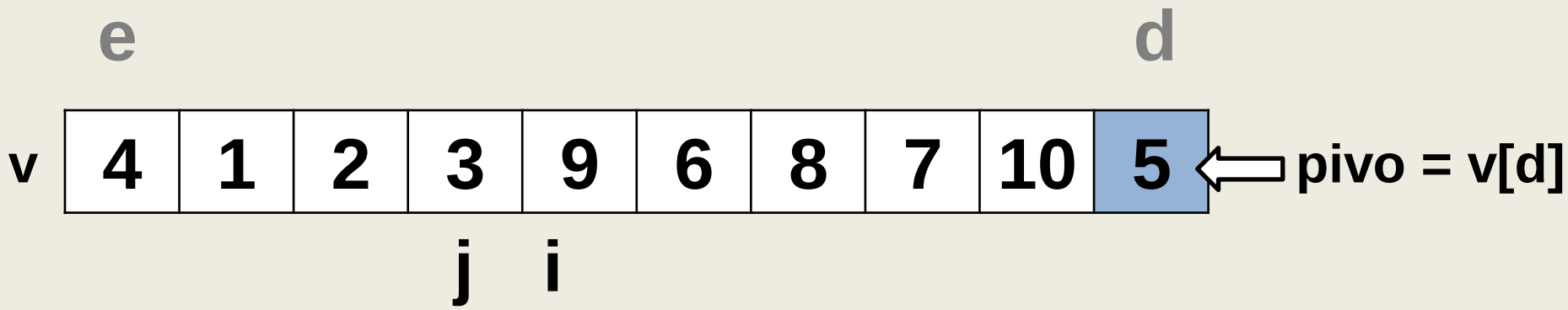
Partição – Versão otimizada




```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Fim deste laço

Partição – Versão otimizada

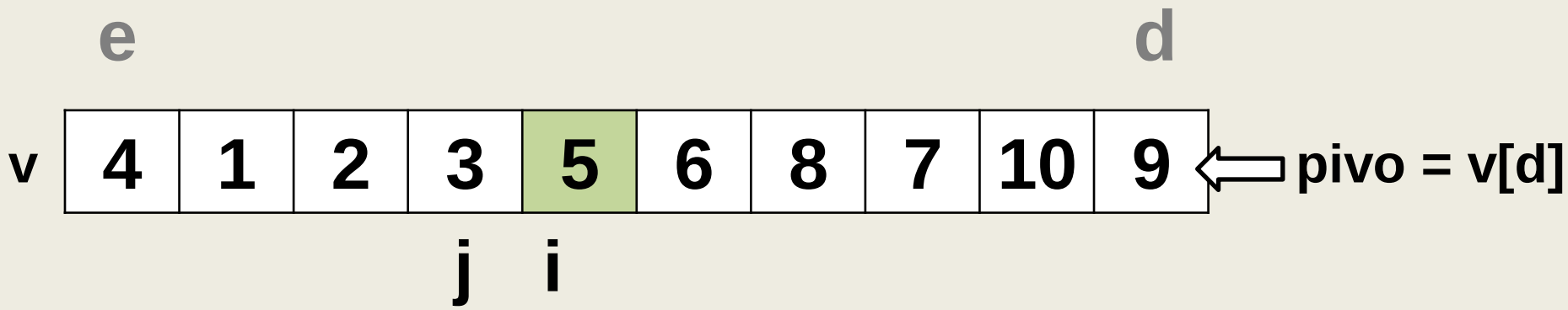



```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Fim deste laço

```
TROQUE v[i], v[direita]
```

Partição – Versão otimizada




```
ENQUANTO v[i] < pivo ENTÃO
    i=i+1
FIM_ENQUANTO
ENQUANTO v[j] > pivo ENTÃO
    j = j-1
FIM_ENQUANTO
SE j ≥ i ENTÃO
    TROQUE v[i], v[j]
FIM_SE
```

Fim deste laço

```
TROQUE v[i], v[direita]
```

Partição – Versão otimizada

```
Particionar( v[1...n], esquerda, direita ) :
   pivo = v[direita]
   i = esquerda, j = direita-1
   WHILE j ≥ i :
        WHILE v[i] < pivo && j ≥ i :
                i=i+1
        END_WHILE
        WHILE v[j] > pivo && j ≥ i :
                j = j-1
        END_WHILE
        IF j ≥ i :
                Swap v[i], v[j]
        END_IF
   END_WHILE
   Swap v[i], v[direita]; Return i;
END
```

Qual a complexidade (pior e melhor caso)?

ANÁLISE DE COMPLEXIDADE

Ordenação por partição

```
OrdenarParticao( v[1...n], esquerda, direita ) :
   SE esquerda < direita ENTÃO :
      i_pivo = Particionar(v, esquerda, direita )
      OrdenarParticao( v, esquerda, i_pivo-1 )
      OrdenarParticao( v, i_pivo+1, direita )
   FIM_SE
FIM
```

Análise de complexidade depende da posição do pivô

Ordenação por partição

- Custo total:
  - T(n) = T(p) + T(n-p-1) + n
    - T(p) é o custo de resolver o problema para o sub-vetor à esquerda do pivô
    - T(n-p-1) é o custo de resolver o problema para o sub-vetor à direita do pivô
    - Custo do algoritmo de partição é proporcional a n
- Melhor caso:
  - Algoritmo de partição sempre divide em dois sub-vetores de mesmo tamanho:
    - T(n) = T(n/2) + T(n/2) + n = 2*T(n/2) + n

Ordenação por partição

- Pior caso:
  - Algoritmo de partição divide em um sub-vetor vazio e outro sub-vetor com n-1 elementos: T(n) = T(0) + T(n-1) + n
    - T(n) = T(n-1) + n
    - T(n-1) = T(n-2) + (n-1)
    - T(n-2) = T(n-3) + (n-2)
    - ...
    - T(1) = T(0) + 1
    - T(n) = ?

$$T(n) = \sum_{i=0}^{n} n - i$$

$$T(n) = \Theta(n^2)$$

Ordenação por partição

- Melhor caso: $\Theta(n^*\log_2 n)$

- Pior caso: $\Theta(n^2)$

INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Estrutura de Dados Básicas I

Algoritmos de ordenação V

Prof. Eiji Adachi M. Barbosa