DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estrutura de Dados

Ordenação: Quicksort

Professores: Luiz Chaimowicz e Raquel Prates

Introdução

- Proposto por C. A. R. Hoare em 1960
  - Publicado em 1962 (refinamentos)

- Algoritmo de ordenação interna mais rápido que se conhece para diversas situações
  - Provavelmente é o mais utilizado

Introdução

- Técnica de divisão e conquista
  - Subdividir um problema em subproblemas
  - Solução direta
- Passos
  - Divide
  - Conquista
  - Combina

Introdução

- A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores.
- Os problemas menores são ordenados independentemente.
- Os resultados são combinados para produzir a solução final.

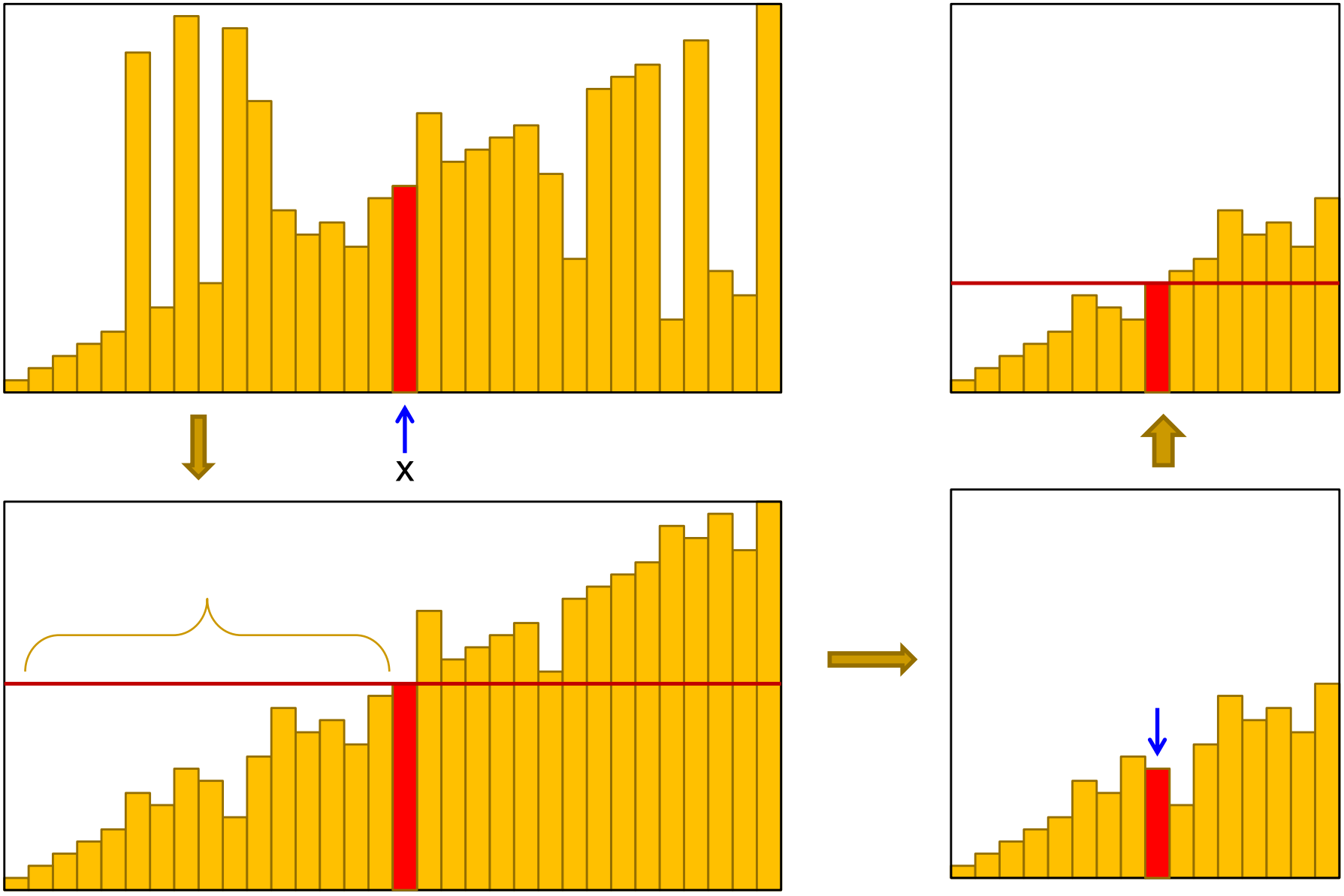
Introdução

- A parte mais delicada do método é o processo de partição.

- O vetor A [Esq ... Dir] é rearranjado por meio da escolha arbitrária de um **pivô** x.

- O vetor A é particionado em duas partes:
  - Parte esquerda: chaves ≤ x.
  - Parte direita: chaves ≥ x.

Exemplo



Quicksort - Partição

- Algoritmo para o particionamento:
  1. Escolha arbitrariamente um **pivô** x.
  2. Percorra o vetor com um índice **i** a partir da esquerda até que A[i] ≥ x.
  3. Percorra o vetor com um índice **j** a partir da direita até que A[j] ≤ x.
  4. Troque A[i] com A[j].
  5. Continue este processo (de 2 a 4) até os apontadores i e j se cruzarem.

Quicksort – Após a Partição

- Ao final, do algoritmo de partição:
  - o vetor A[Esq..Dir] está particionado de tal forma que:
    - Os itens em A[Esq], A[Esq + 1], ..., A[j] são menores ou iguais a *x*;
    - Os itens em A[i], A[i +1], ..., A[Dir] são maiores ou iguais a *x*.

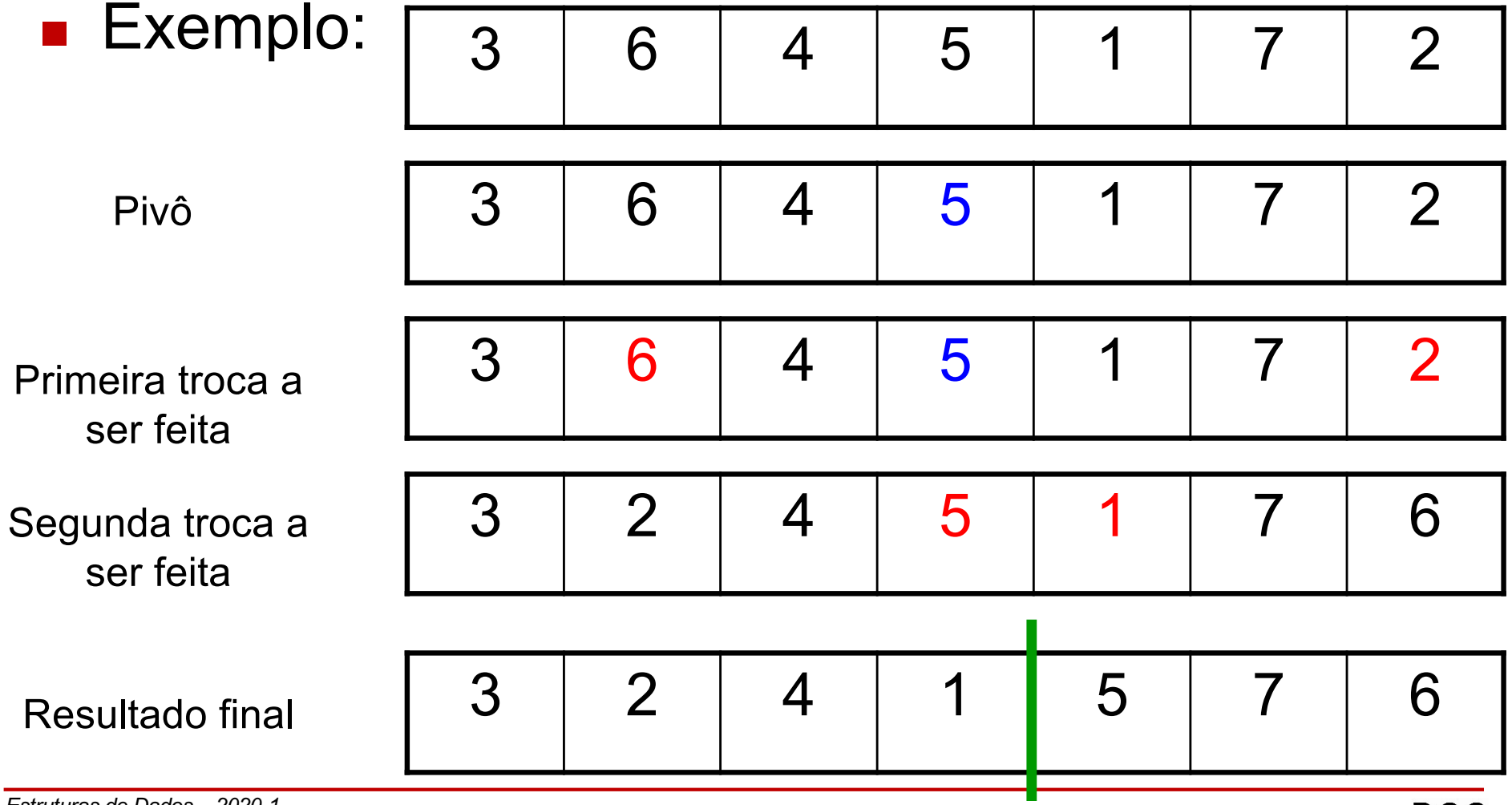
Partição - Exemplo

- O pivô x é escolhido como sendo A[(i + j) / 2].
- Exemplo:

3	6	4	5	1	7	2

Partição - Exemplo

- O pivô x é escolhido como sendo A[(i + j) / 2].
- Exemplo:



Quicksort - Partição

```
void Particao(int Esq, int Dir,
              int *i, int *j, Item *A)
{
  Item x, w;
  *i = Esq; *j = Dir;
  x = A[(*i + *j)/2]; /* obtem o pivo x */
  do
  {
    while (x.Chave > A[*i].Chave) (*i)++;
    while (x.Chave < A[*j].Chave) (*j)--;
    if (*i <= *j)
    {
      w = A[*i]; A[*i] = A[*j]; A[*j] = w;
      (*i)++; (*j)--;
    }
  } while (*i <= *j);
}
```

Índices inicial e final do vetor

Vetor

Índices que vão percorrer o vetor (passados por referência)

Quicksort - Partição

```
void Particao(int Esq, int Dir,
              int *i, int *j, Item *A)
{
  Item x, w;
  *i = Esq; *j = Dir;
  x = A[(*i + *j)/2]; /* obtem o pivo x */
  do
  {
    while (x.Chave > A[*i].Chave) (*i)++;
    while (x.Chave < A[*j].Chave) (*j)--;
    if (*i <= *j)
    {
      w = A[*i]; A[*i] = A[*j]; A[*j] = w;
      (*i)++; (*j)--;
    }
  } while (*i <= *j);
}
```

Inicializa índices i e j, que vão percorrer o vetor

Inicializa pivô com elemento central

Até que os índices se cruzem

Quicksort - Partição

```
void Particao(int Esq, int Dir,
              int *i, int *j, Item *A)
{
   Item x, w;
   *i = Esq; *j = Dir;
   x = A[(*i + *j)/2]; /* obtem o pivo x */
   do
   {
      while (x.Chave > A[*i].Chave) (*i)++;
      while (x.Chave < A[*j].Chave) (*j)--;
      if (*i <= *j)
      {
         w = A[*i]; A[*i] = A[*j]; A[*j] = w;
         (*i)++; (*j)--;
      }
   } while (*i <= *j);
}
```

i procura elemento maior que pivô

j procura elemento menor que pivô

Quicksort - Partição

```
void Particao(int Esq, int Dir,
              int *i, int *j, Item *A)
{
   Item x, w;
   *i = Esq; *j = Dir;
   x = A[(*i + *j)/2]; /* obtem o pivo x */
   do
   {
      while (x.Chave > A[*i].Chave) (*i)++;
      while (x.Chave < A[*j].Chave) (*j)--;
      if (*i <= *j)
      {
         w = A[*i]; A[*i] = A[*j]; A[*j] = w;
         (*i)++; (*j)--;
      }
   } while (*i <= *j);
}
```

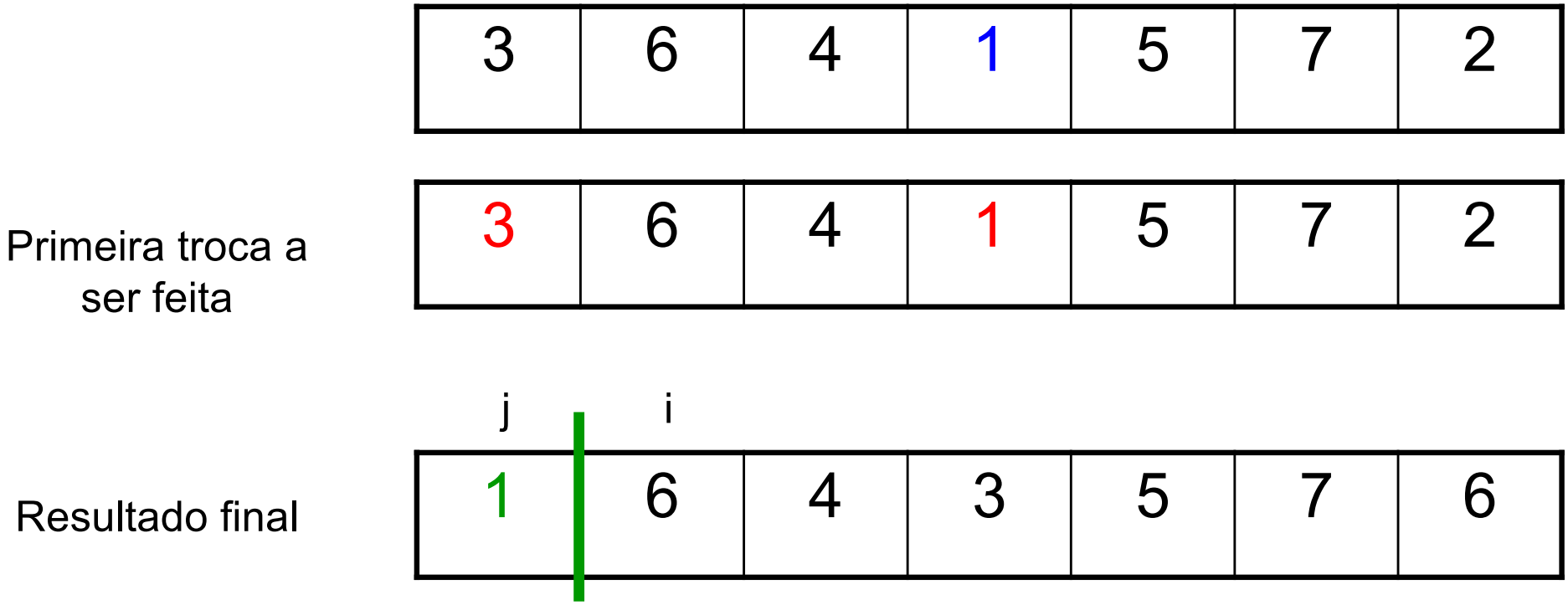
Se i e j ainda não se cruzaram, então:

Troca os elementos de índice i e j de lugar

i e j passam para próxima posição

Partição – "Casos Especiais"

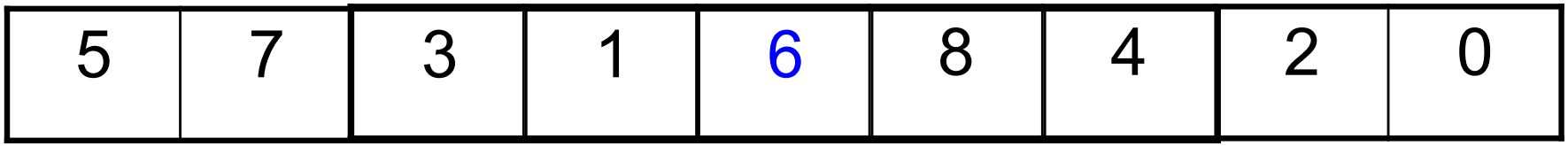
- Pivô é o menor ou maior de todos



Depois da primeira troca, o índice i fica parado no elemento 6 (6 >= pivô) enquanto J é decrementado até parar no elemento 1, que é o próprio pivô. Como os índices se cruzam o procedimento termina

Partição – "Casos Especiais"

- Pivô não fica em uma das "bordas" após a partição



Partição – "Casos Especiais"

- Pivô não fica em uma das "bordas" após a partição

5	7	3	1	6	8	4	2	0

5	7	3	1	6	8	4	2	0

5	0	3	1	6	8	4	2	7

5	0	3	1	2	8	4	6	7

A partição continua ate os índices se cruzarem, mesmo após o pivô ter movido

5	0	3	1	2	4	8	6	7

Partição – "Casos Especiais"

- Pivô na posição correta

3	6	1	4	5	7	2

Partição – "Casos Especiais"

- Pivô na posição correta

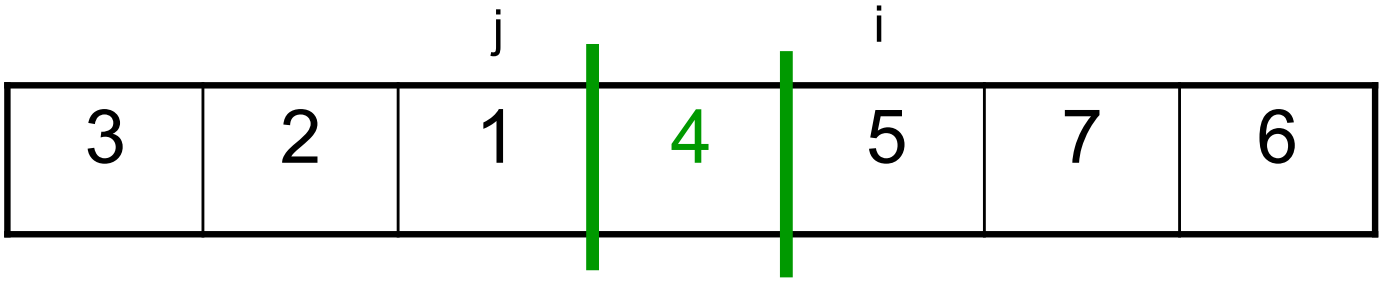
3	6	1	4	5	7	2

Primeira troca a ser feita

3	6	1	4	5	7	2

Após a primeira troca, os índices i e j continuam e param sobre o pivô. O pivô é trocado com ele mesmo e a partição termina com duas partições e mais um elemento (pivô) já na posição correta.

Resultado final



Quicksort

- O anel interno da função de Partição é extremamente simples.

- Razão pela qual o algoritmo Quicksort é tão rápido.

Quicksort - Função

```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```

Vetor

Intervalo do vetor a ser ordenado

Quicksort - Função

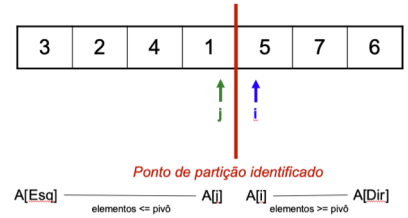
```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```

Índices i e j, que vão percorrer o vetor

Chama partição, com índices i e j passados por referência

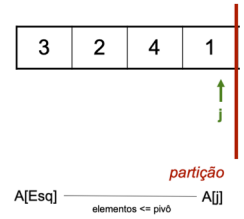
Quicksort - Função

```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```



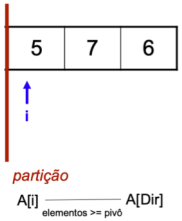
Quicksort - Função

```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```



Quicksort - Função

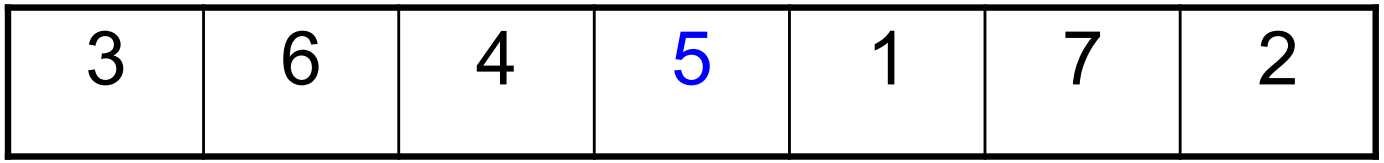
```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```



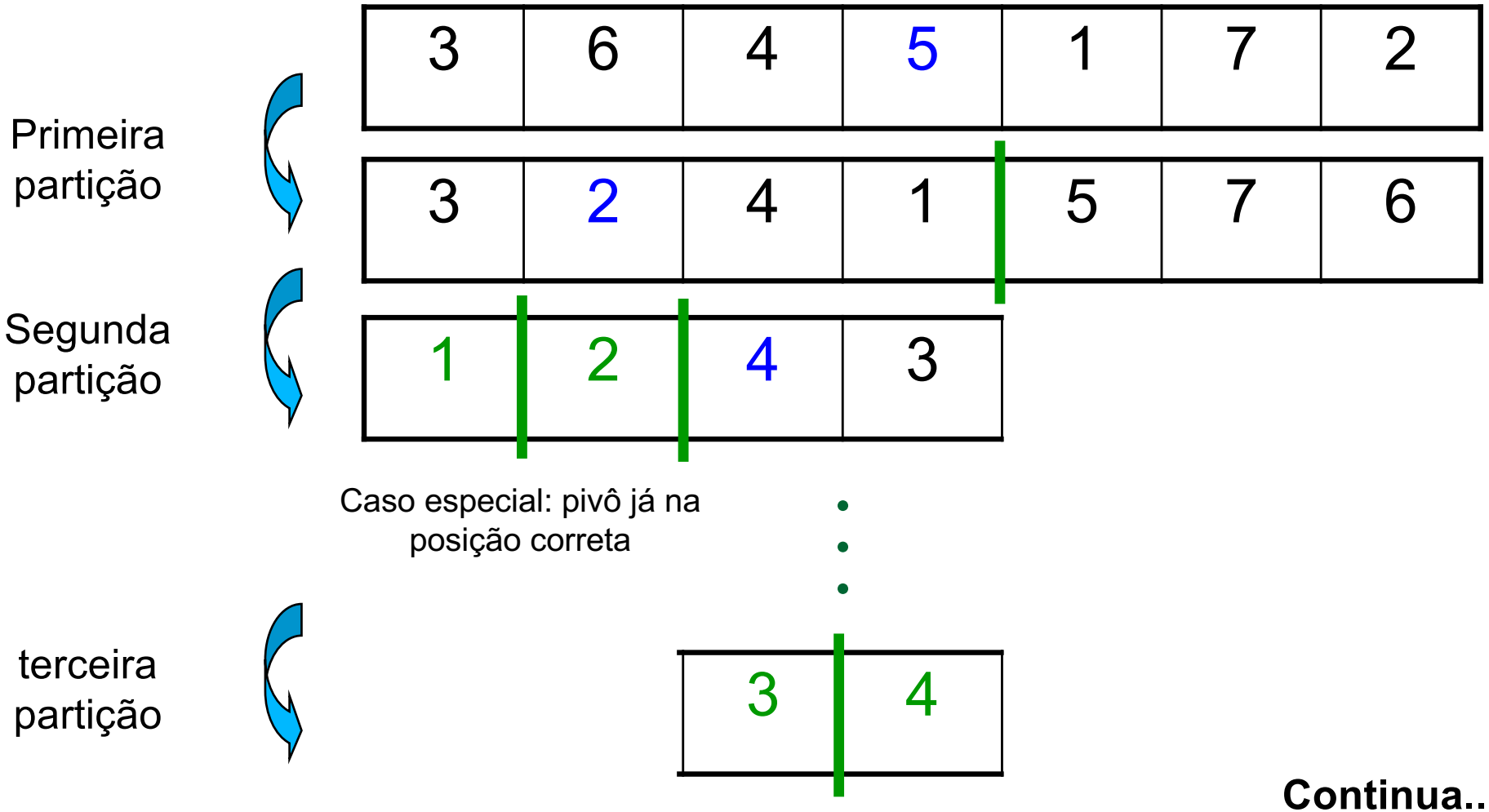
Quicksort - Função

```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, int j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```

Quicksort - Exemplo

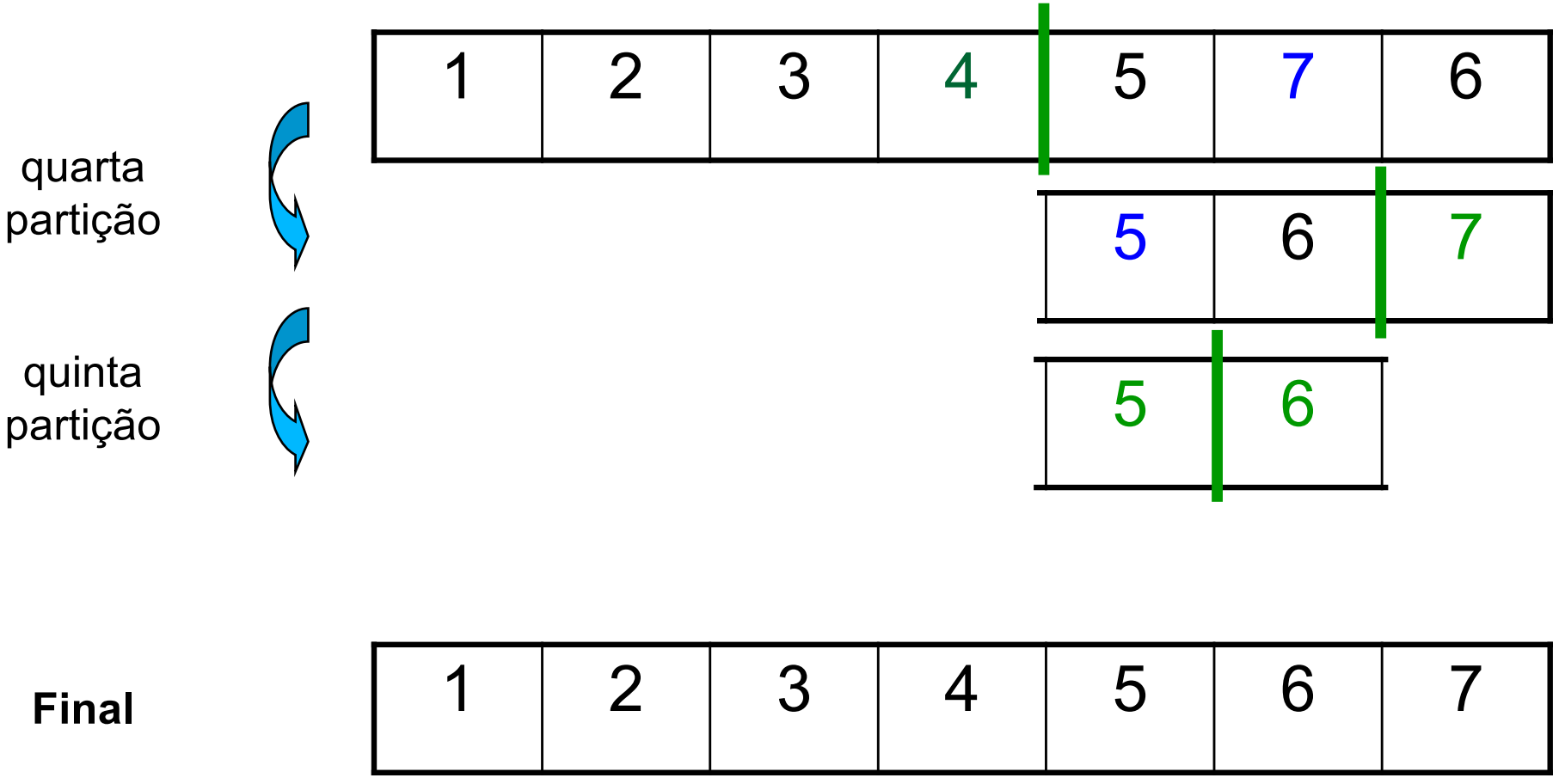


Quicksort - Exemplo



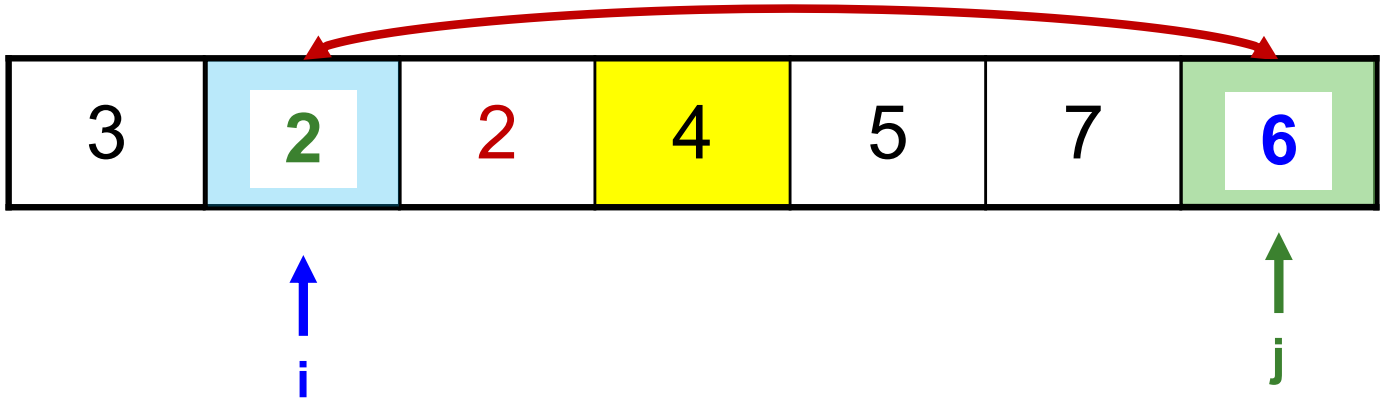
Continua...

Quicksort - Exemplo

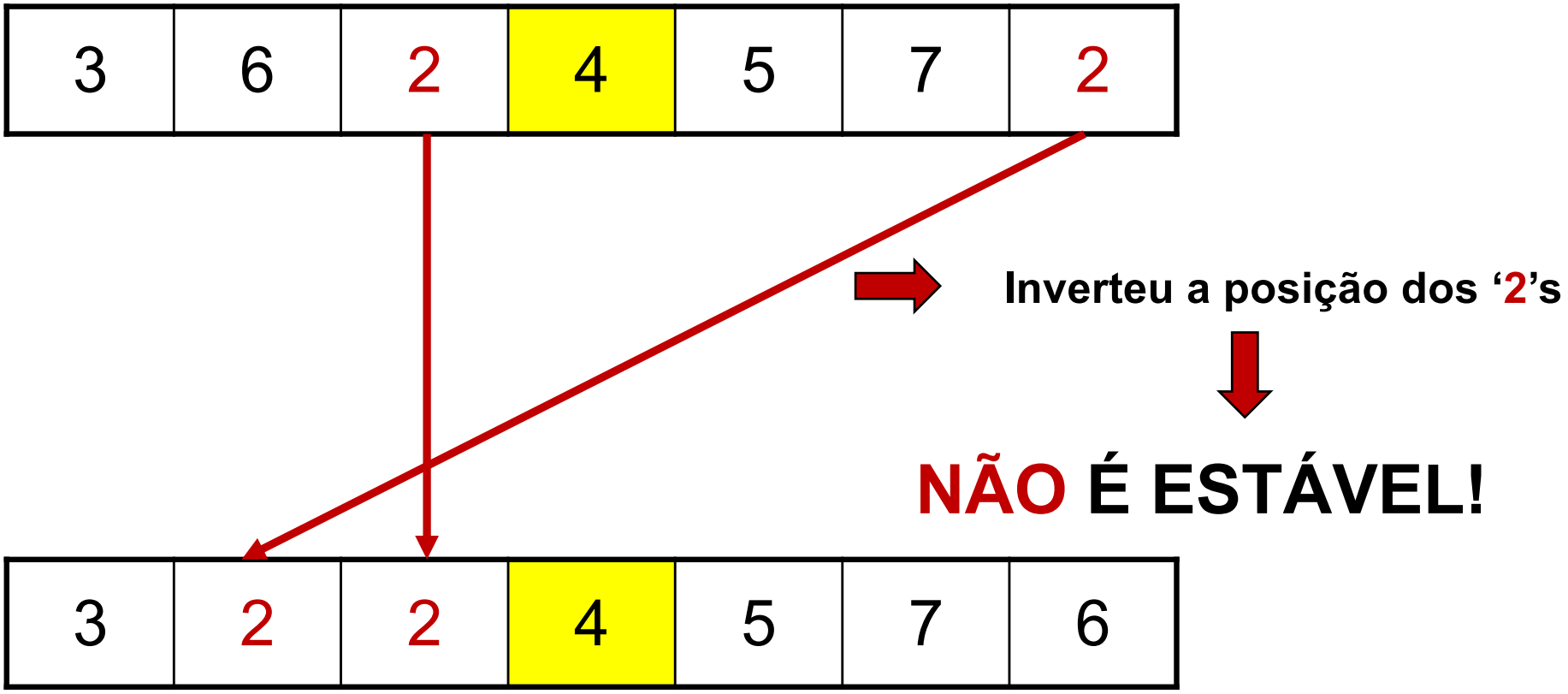


QUICKSORT - ANÁLISE

O Quicksort é estável?

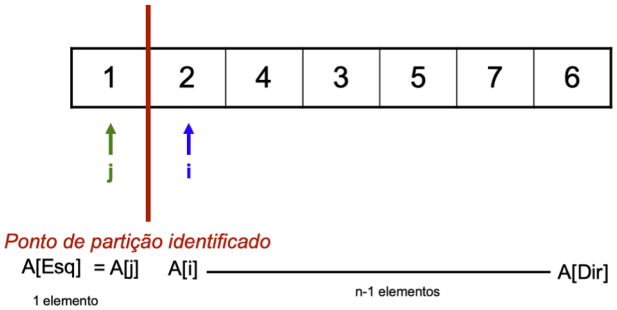


O Quicksort é estável?

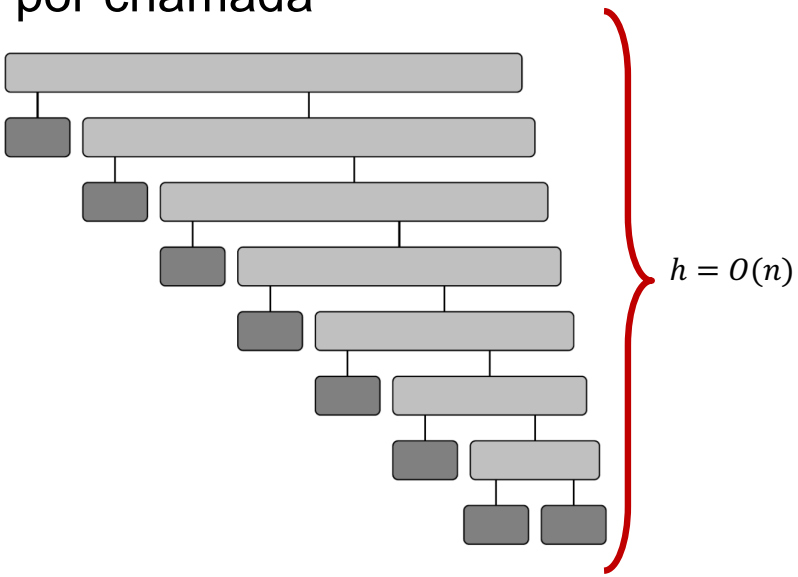


Qual o pior caso?

- Vimos que se o pivô escolhido for o maior ou menor elemento do vetor, a partição será 1 e n-1.
- Se para todas as partições da chamada, acontecer esta situação, será ordenado um elemento por vez.



$O(n)$ operações por chamada



Análise – Pior caso

- O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo.
  - Por exemplo, se escolhe o pivô como 1º ou último elemento do vetor, e este está ordenado
- Isto faz com que o procedimento Ordena seja chamado recursivamente n vezes, eliminando apenas um item em cada chamada.

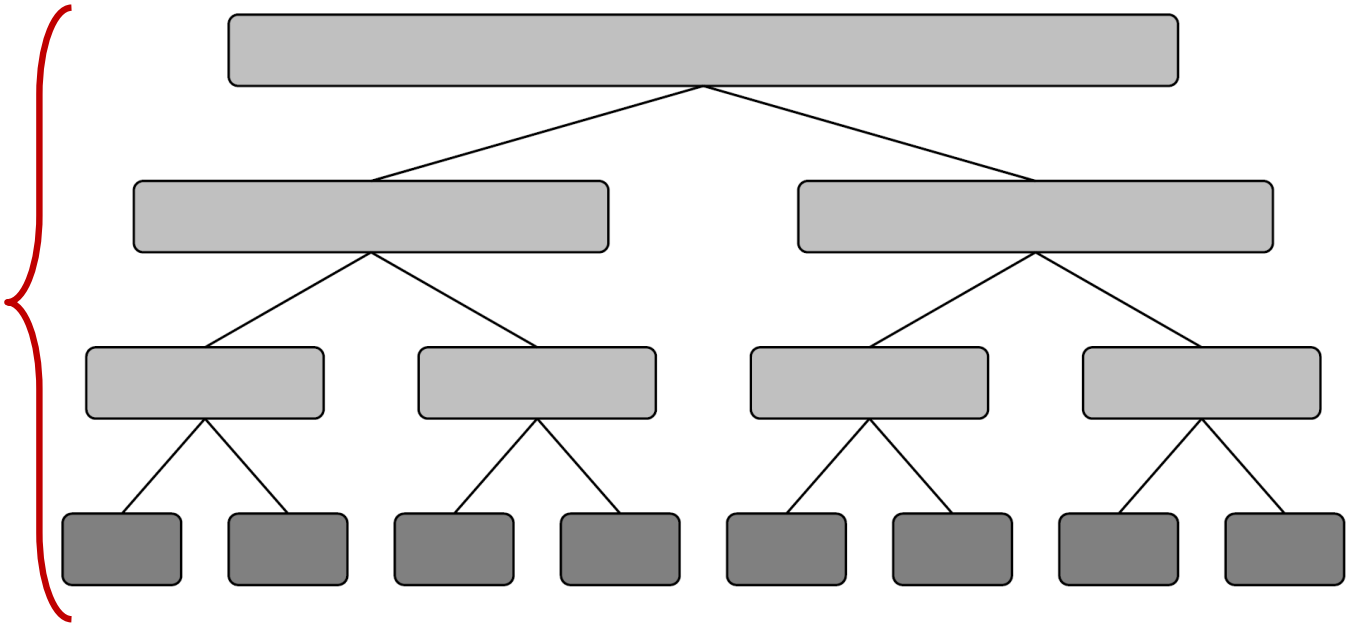
$$T(n) = n + T(n-1) \Rightarrow C(n) = O(n^2)$$

Qual o Melhor caso?

- Quando o vetor é dividido em 2 partes iguais

$O(n)$ operações por nível da árvore

$h = O(\log n)$



Análise – Melhor caso

- O melhor caso ocorre quando cada partição divide o conjunto em duas partes iguais.

$$T(n) = 2T(n/2) + n \Rightarrow C(n) = O(n \log n)$$

Análise – Caso médio

- **Caso médio** de acordo com Sedgewick e Flajolet (1996, p. 17):

  $C(n) \approx 1{,}386n \log n - 0{,}846n,$

- Isso significa que em média o tempo de execução do Quicksort é ***O(n log n)***.

Quicksort

- Vantagens:
  - É extremamente eficiente para ordenar arquivos de dados.
  - Necessita de apenas uma pequena pilha como memória auxiliar.
  - Requer cerca de *n log n* comparações em média para ordenar n itens.

- Desvantagens:
  - Tem um pior caso $O(n^2)$ comparações.
  - Sua implementação é muito delicada e difícil:
    - Um pequeno engano pode levar a efeitos inesperados para algumas entradas de dados.
  - O método não é **estável**.

Melhorias no Quicksort

- Escolha do pivô: mediana de três
  - Evita o pior caso
- Utilizar um algoritmo simples (seleção, inserção) para partições de tamanho pequeno
- Quicksort não recursivo
  - Evita o custo de várias chamadas recursivas

QUICKSORT NÃO RECURSIVO

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, int n)
{
     TipoPilha pilha;
     TipoItem item;  // campos esq e dir
     int esq, dir, i, j;

     FPVazia(&pilha);
     esq = 0;
     dir = n-1;
     item.dir = dir;
     item.esq = esq;
     Empilha(item, &pilha);
```

Declara a pilha a ser usada

Guarda o intervalo do vetor sendo considerado

```
     do
        if (dir > esq)  {
           Particao(A,esq,dir,&i, &j);
           if ((j-esq)>(dir-i))  {
              item.dir = j;
              item.esq = esq;
              Empilha(item, &pilha);
              esq = i;
           }
           else {
               item.esq = i;
               item.dir = dir;
               Empilha(item, &pilha);
               dir = j;
           }
        }
        else {
            Desempilha(&pilha,&item);
            dir = item.dir;
            esq = item.esq;
     } while (!Vazia(pilha));
}
```

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, int n)
{
     TipoPilha pilha;
     TipoItem item;  // campos esq e dir
     int esq, dir, i, j;

     FPVazia(&pilha);
     esq = 0;
     dir = n-1;
     item.dir = dir;
     item.esq = esq;
     Empilha(item, &pilha);
```

Declara as variáveis que vão armazenar o intervalo sendo considerado a cada iteração (esq e dir) e que vão percorrer o vetor (i, j).

```
     do
         if (dir > esq)  {
            Particao(A,esq,dir,&i, &j);
            if ((j-esq)>(dir-i))  {
               item.dir = j;
               item.esq = esq;
               Empilha(item, &pilha);
               esq = i;
            }
            else {
                item.esq = i;
                item.dir = dir;
                Empilha(item, &pilha);
                dir = j;
            }
         }
         else {
             Desempilha(&pilha,&item);
             dir = item.dir;
             esq = item.esq;
     } while (!Vazia(pilha));
}
```

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, int n)
{
     TipoPilha pilha;
     TipoItem item;  // campos esq e dir
     int esq, dir, i, j;

     FPVazia(&pilha);
     esq = 0;
     dir = n-1;
     item.dir = dir;
     item.esq = esq;
     Empilha(item, &pilha);

     do
        if (dir > esq)  {
           Particao(A,esq,dir,&i, &j);
           if ((j-esq)>(dir-i))  {
              item.dir = j;
              item.esq = esq;
              Empilha(item, &pilha);
              esq = i;
           }
           else {
               item.esq = i;
               item.dir = dir;
               Empilha(item, &pilha);
               dir = j;
           }
        }
        else {
            Desempilha(&pilha,&item);
            dir = item.dir;
            esq = item.esq;
     } while (!Vazia(pilha));
}
```

Inicializa a pilha como vazia

Inicializa o intervalo como sendo todo o vetor

Empilha o intervalo de todo o vetor

Enquanto a pilha não estiver vazia

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, int n)
{
      TipoPilha pilha;
      TipoItem item;  // campos esq e dir
      int esq, dir, i, j;

      FPVazia(&pilha);
      esq = 0;
      dir = n-1;
      item.dir = dir;
      item.esq = esq;
      Empilha(item, &pilha);
      do
        if (dir > esq)   {
          Particao(A,esq,dir,&i, &j);
          if ((j-esq)>(dir-i))   {
            item.dir = j;
            item.esq = esq;
            Empilha(item, &pilha);
            esq = i;
          }
          else {
            item.esq = i;
            item.dir = dir;
            Empilha(item, &pilha);
            dir = j;
          }
        }
        else {
            Desempilha(&pilha,&item);
            dir = item.dir;
            esq = item.esq;
        } while (!Vazia(pilha));
}
```

Se o intervalo ainda tem elementos a serem considerados

Verifica o maior lado

Empilha o lado esquerdo

Atualiza o índice da esquerda com sendo o 1º do lado direito.

Empilha o lado direito

Atualiza o índice da direita com sendo o último do lado esquerdo.

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, int n)
{
     TipoPilha pilha;
     TipoItem item;  // campos esq e dir
     int esq, dir, i, j;

     FPVazia(&pilha);
     esq = 0;
     dir = n-1;
     item.dir = dir;
     item.esq = esq;
     Empilha(item, &pilha);
     do
         if (dir > esq)  {
            Particao(A,esq,dir,&i, &j);
            if ((j-esq)>(dir-i))  {
               item.dir = j;
               item.esq = esq;
               Empilha(item, &pilha);
               esq = i;
            }
            else {
                item.esq = i;
                item.dir = dir;
                Empilha(item, &pilha);
                dir = j;
            }
         }
         else {
             Desempilha(&pilha,&item);
             dir = item.dir;
             esq = item.esq;
         }
     } while (!Vazia(pilha));
}
```

Desempilha o próximo intervalo a ser considerado

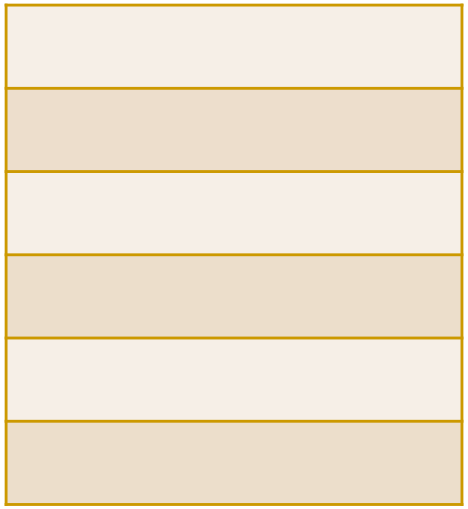
Atualiza o os índices para considerar o intervalo desempilhado

Quicksort – Não Recursivo

Trecho do código

0	1	2	3	4	5	6
3	6	4	5	1	7	2

Pilha



RAM

```
esq =
dir =
item.dir =
item.esq =
i =
j =
```

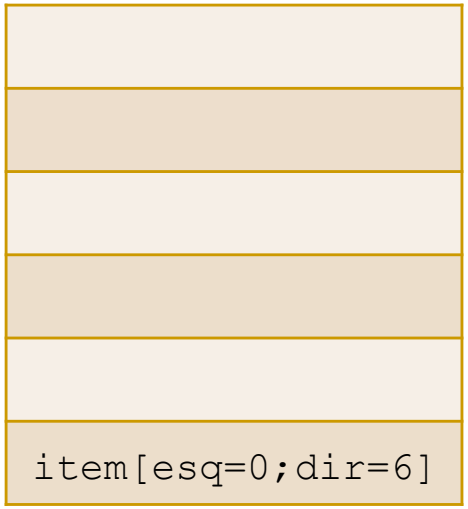
Quicksort – Não Recursivo

Trecho do código

```
FPVazia(&pilha);
esq = 0;
dir = n-1;
item.dir = dir;
item.esq = esq;
Empilha(item, &pilha);
```

0	1	2	3	4	5	6
3	6	4	5	1	7	2

Pilha



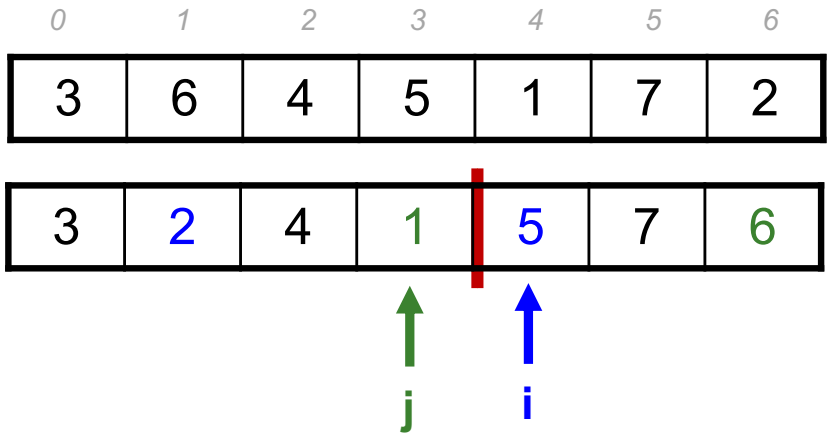
RAM

```
n= 7
esq = 0
dir = 6
item.dir = 6
item.esq = 0
i =
j =
```

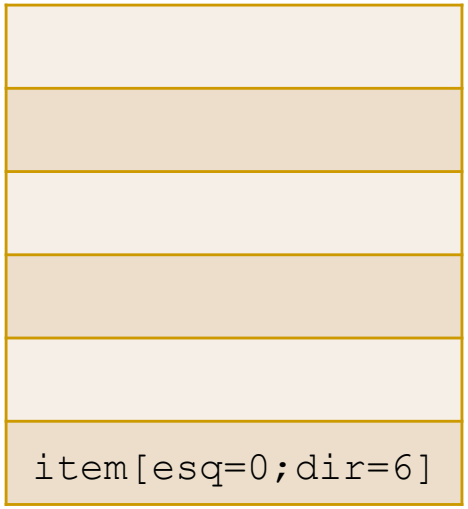
Quicksort – Não Recursivo

Trecho do código

```
do
  if (dir > esq)  {
    Particao(A,esq,dir,&i, &j);
    if ((j-esq)>(dir-i))  {
      item.dir = j;
      item.esq = esq;
      Empilha(item, &pilha);
      esq = i;
    }
    else {
      item.esq = i;
      item.dir = dir;
      Empilha(item, &pilha);
      dir = j;
  }
  […]
} while (!Vazia(pilha));
```



Pilha



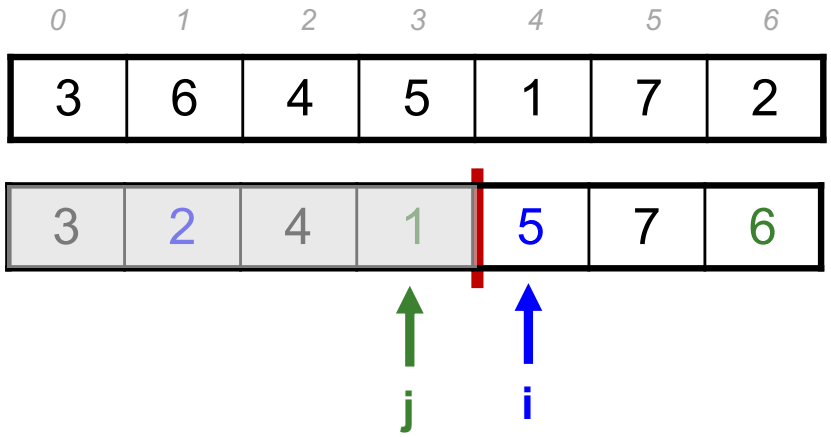
RAM

```
n= 7
esq = 0
dir = 6
item.dir = 6
item.esq = 0
i = 4
j = 3
```

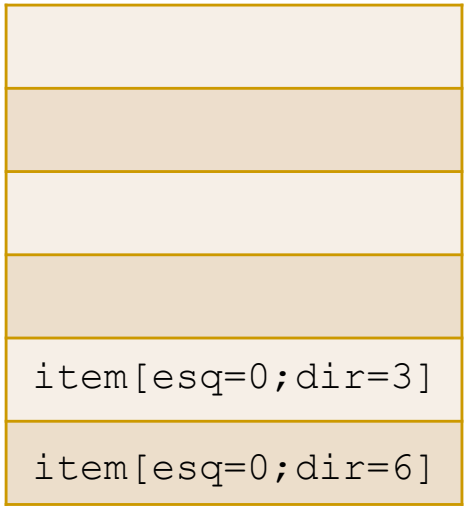
Quicksort – Não Recursivo

Trecho do código

```
do
 if (dir > esq)  {
   Particao(A,esq,dir,&i, &j);
   if ((j-esq)>(dir-i))  {
     item.dir = j;
     item.esq = esq;
     Empilha(item, &pilha);
     esq = i;
   }
   else {
     item.esq = i;
     item.dir = dir;
     Empilha(item, &pilha);
     dir = j;
 }
 […]
} while (!Vazia(pilha));
```



Pilha



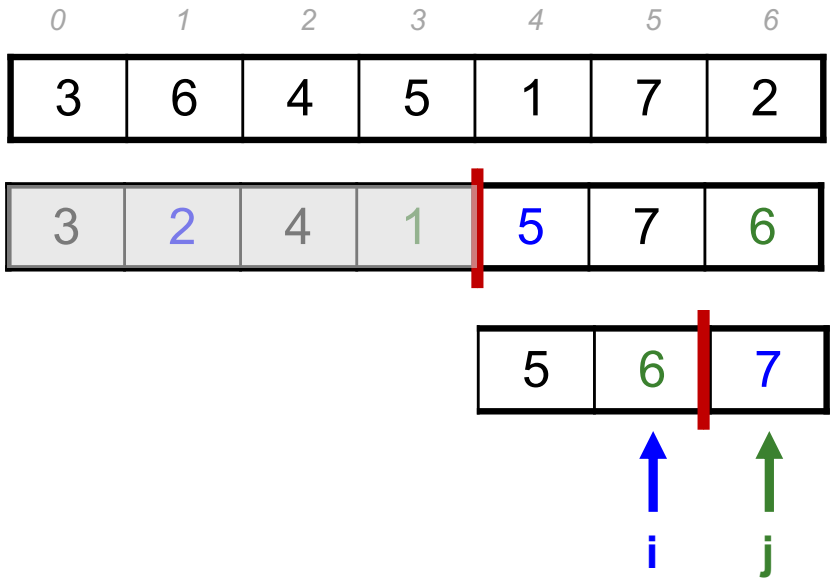
RAM

```
n= 7
esq = 0
dir = 6
item.dir = 3
item.esq = 0
i = 4
j = 3
```

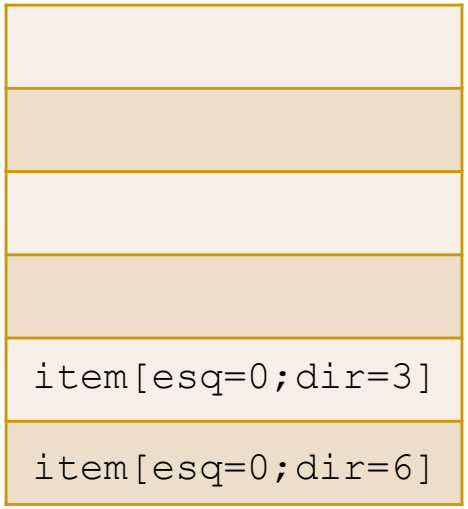
Quicksort – Não Recursivo

Trecho do código

```
do
  if (dir > esq)  {
    Particao(A,esq,dir,&i, &j);
    if ((j-esq)>(dir-i))  {
      item.dir = j;
      item.esq = esq;
      Empilha(item, &pilha);
      esq = i;
    }
    else {
      item.esq = i;
      item.dir = dir;
      Empilha(item, &pilha);
      dir = j;
  }
  […]
} while (!Vazia(pilha));
```



Pilha



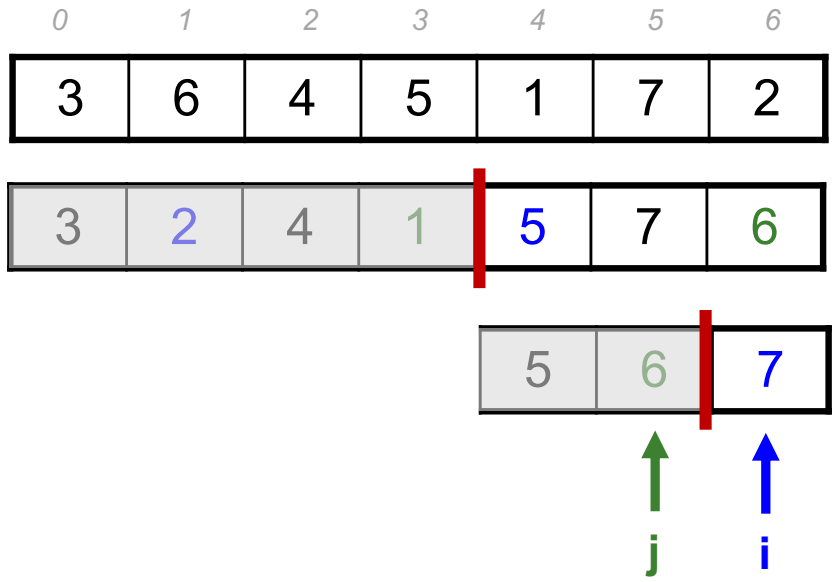
RAM

```
n= 7
esq = 4
dir = 6
item.dir = 3
item.esq = 0
i = 4 5
j = 3 6
```

Quicksort – Não Recursivo

Trecho do código

```
do
 if (dir > esq)  {
   Particao(A,esq,dir,&i, &j);
   if ((j-esq)>(dir-i))  {
     item.dir = j;
     item.esq = esq;
     Empilha(item, &pilha);
     esq = i;
   }
   else {
     item.esq = i;
     item.dir = dir;
     Empilha(item, &pilha);
     dir = j;
 }
 […]
} while (!Vazia(pilha));
```



Pilha

item[esq=4;dir=5]
item[esq=0;dir=3]
item[esq=0;dir=6]

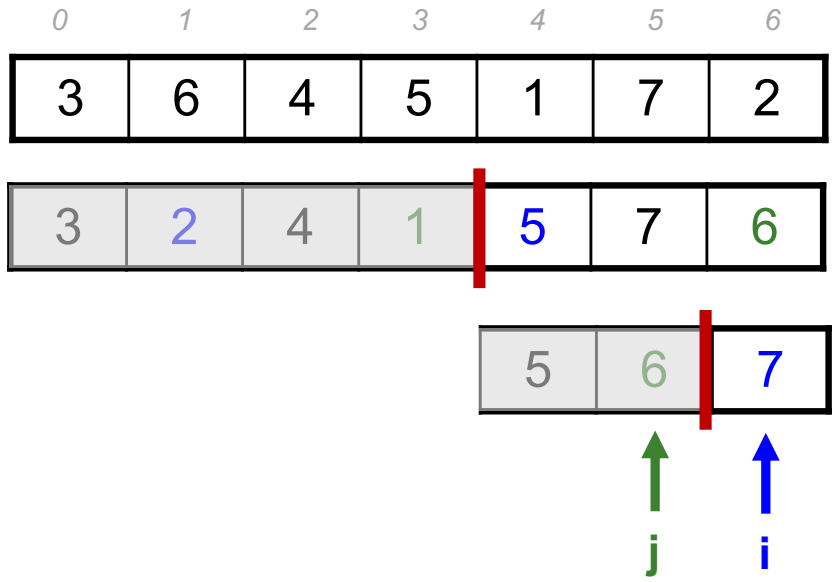
RAM

```
n= 7
esq = 46
dir = 6
item.dir = 5
item.esq = 4
i = 6
j = 5
```

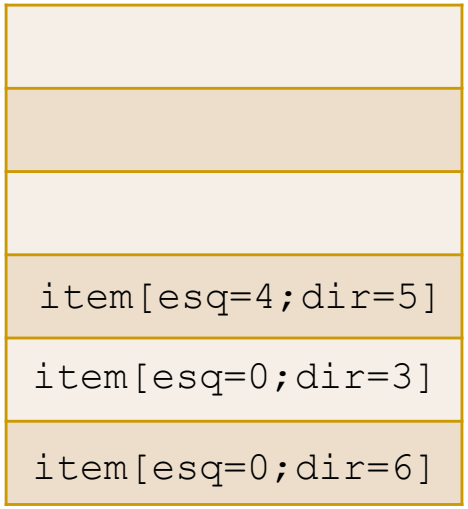
Quicksort – Não Recursivo

Trecho do código

```
do
 if (dir > esq)  {
    Particao(A,esq,dir,&i, &j);
    if ((j-esq)>(dir-i))  {
      item.dir = j;
      item.esq = esq;
      Empilha(item, &pilha);
      esq = i;
    }
    else {
      item.esq = i;
      item.dir = dir;
      Empilha(item, &pilha);
      dir = j;
  }
  […]
} while (!Vazia(pilha));
```



Pilha



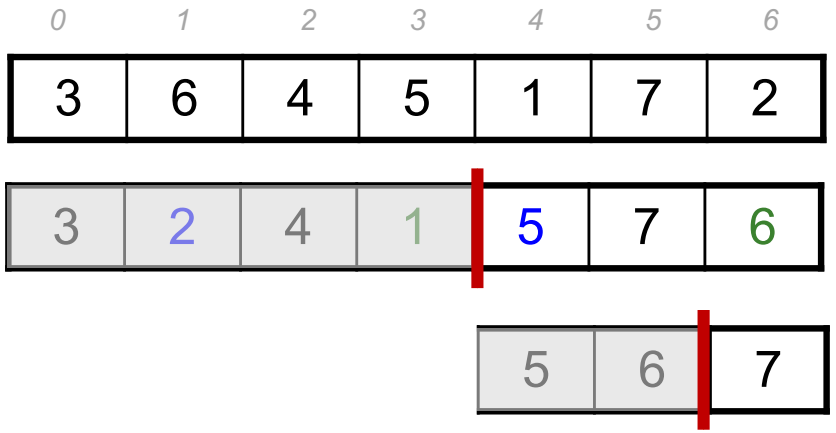
RAM

```
n= 7
esq = 6
dir = 6
item.dir = 5
item.esq = 4
i = 6
j = 5
```

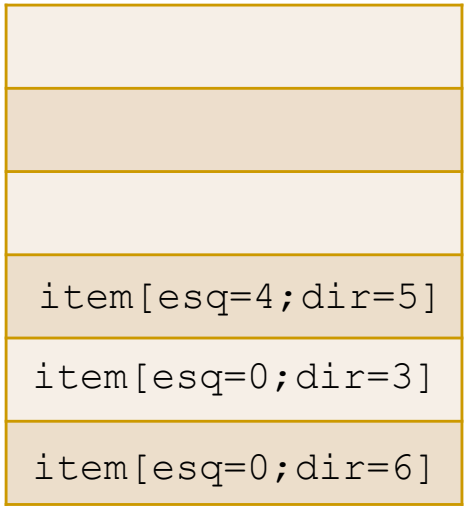
Quicksort – Não Recursivo

Trecho do código

```
do
 if (dir > esq)  {
  […]
 }

 else {
  Desempilha(&pilha,&item);
  dir = item.dir;
  esq = item.esq;

} while (!Vazia(pilha));
```



Pilha



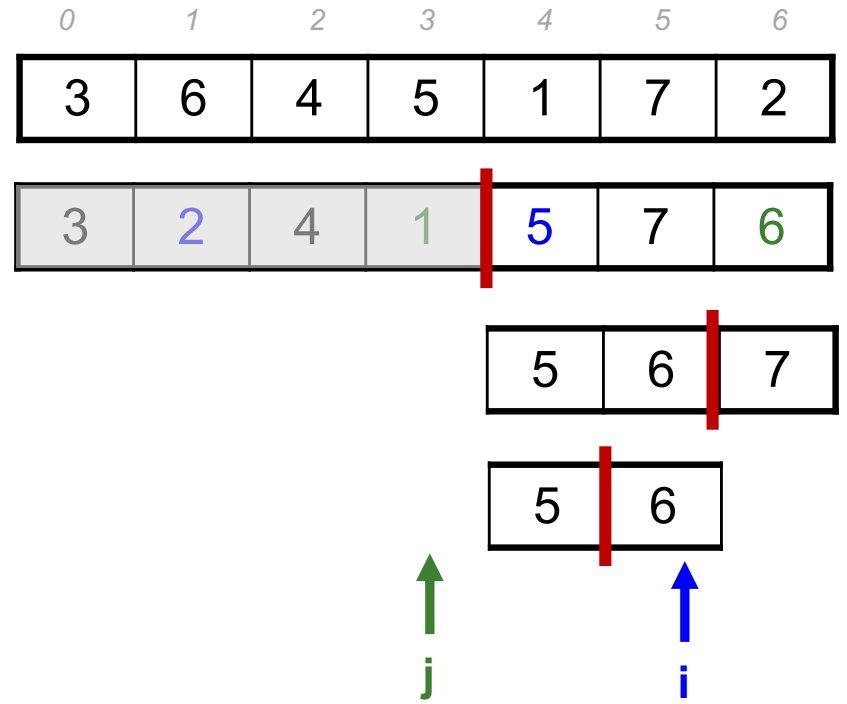
RAM

```
n= 7
esq = 6 4
dir = 6 5
item.dir = 5
item.esq = 4
i = 6
j = 5
```

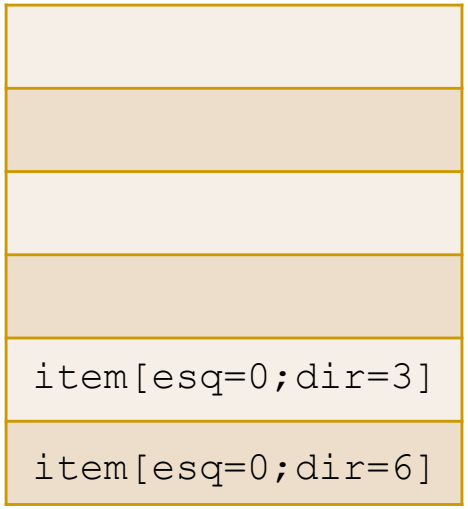
Quicksort – Não Recursivo

Trecho do código

```
do
  if (dir > esq)  {
    Particao(A,esq,dir,&i, &j);
    if ((j-esq)>(dir-i))  {
      item.dir = j;
      item.esq = esq;
      Empilha(item, &pilha);
      esq = i;
    }
    else {
      item.esq = i;
      item.dir = dir;
      Empilha(item, &pilha);
      dir = j;
  }
  […]
} while (!Vazia(pilha));
```



Pilha



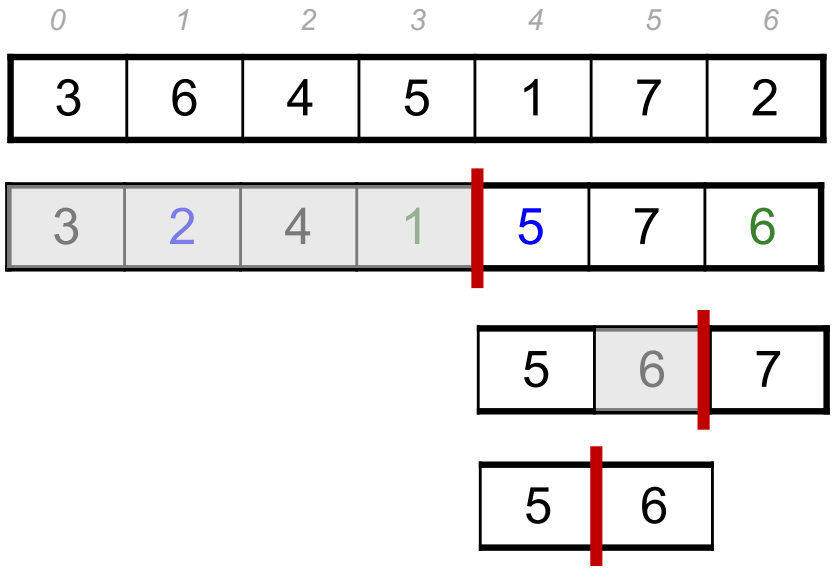
RAM

```
n= 7
esq = 4
dir = 5
item.dir = 5
item.esq = 4
i = 6 5
j = 5 3
```

Quicksort – Não Recursivo

Trecho do código

```
do
  if (dir > esq)  {
     Particao(A,esq,dir,&i, &j);
     if ((j-esq)>(dir-i))  {
       item.dir = j;
       item.esq = esq;
       Empilha(item, &pilha);
       esq = i;
     }
     else {
       item.esq = i;
       item.dir = dir;
       Empilha(item, &pilha);
       dir = j;
  }
  […]
} while (!Vazia(pilha));
```



Pilha

item[esq=5;dir=5]
item[esq=0;dir=3]
item[esq=0;dir=6]

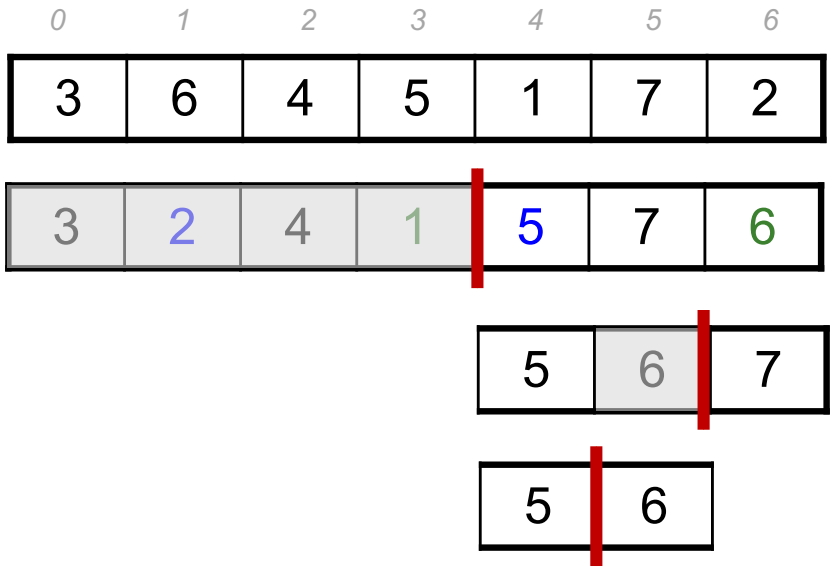
RAM

```
n= 7
esq = 4
dir = 5 3
item.dir = 5
item.esq = 45
i = 5
j = 3
```

Quicksort – Não Recursivo

Trecho do código

```
do
 if (dir > esq)  {
   […]
 }

 else {
   Desempilha(&pilha,&item);
   dir = item.dir;
   esq = item.esq;

} while (!Vazia(pilha));
```



Pilha

item[esq=5;dir=5]
item[esq=0;dir=3]
item[esq=0;dir=6]

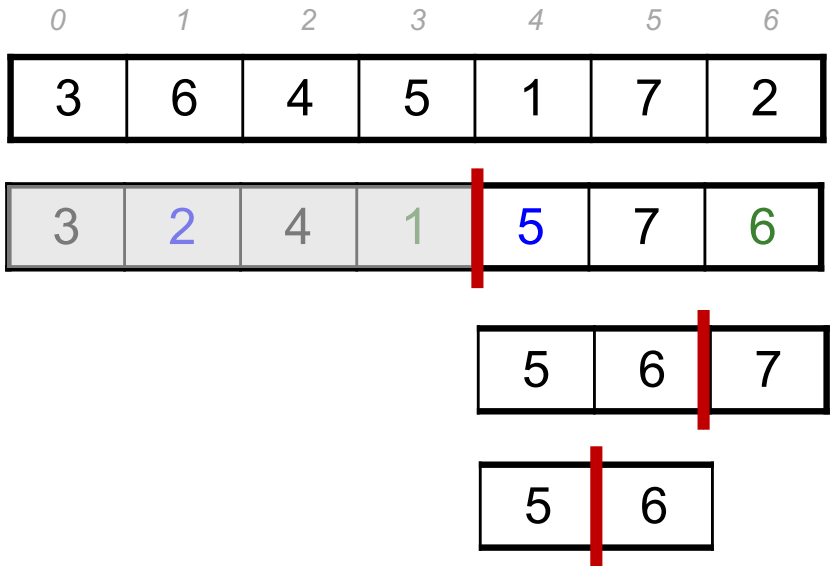
RAM

```
n= 7
esq = 4 5
dir = 3 5
item.dir = 5
item.esq = 5
i = 5
j = 3
```

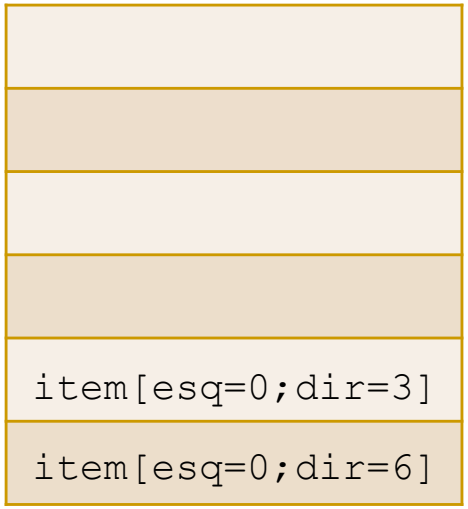
Quicksort – Não Recursivo

Trecho do código

```
do
  if (dir > esq)  {
    […]
  }

  else {
    Desempilha(&pilha,&item);
    dir = item.dir;
    esq = item.esq;

} while (!Vazia(pilha));
```



Pilha



RAM

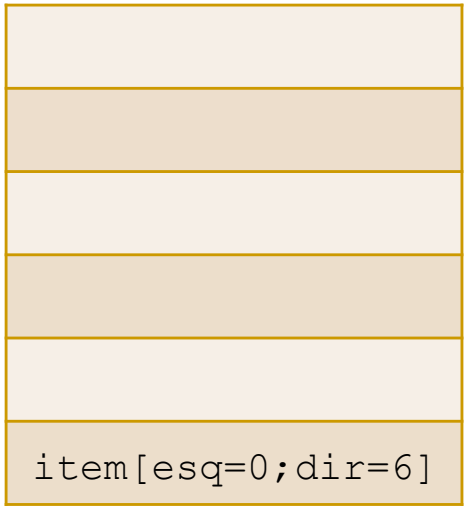
```
n= 7
esq = 5 0
dir = 5 3
item.dir = 35
item.esq = 05
i = 5
j = 3
```

Quicksort – Não Recursivo

Trecho do código

```
do
 if (dir > esq)  {
   Particao(A,esq,dir,&i, &j);
   if ((j-esq)>(dir-i))  {
     item.dir = j;
     item.esq = esq;
     Empilha(item, &pilha);
     esq = i;
   }
   else {
     item.esq = i;
     item.dir = dir;
     Empilha(item, &pilha);
     dir = j;
 }
 […]
} while (!Vazia(pilha));
```

Pilha



RAM

```
n= 7
esq =  0
dir =  3
item.dir =3
item.esq = 5
i = 52
j = 30
```