Quicksort

Livro “Projeto de Algoritmos” – Nívio Ziviani

Capítulo 4 – Seção 4.1.4

http://www2.dcc.ufmg.br/livros/algoritmos/

Quicksort

- Proposto por C. A. R. Hoare em 1960 e publicado em 1962.
- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.
- Provavelmente é o mais utilizado.
- A idéia 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.

Quicksort

- O ponto principal 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.**

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 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:

	3	6	4	5	1	7	2
Pivô	3	6	4	5	1	7	2
Primeira troca a ser feita	3	6	4	5	1	7	2
Segunda troca a ser feita	3	2	4	5	1	7	6
Resultado final	3	2	4	1	5	7	6

Quicksort - Partição

```
void Particao(Indice Esq, Indice Dir,
              Indice *i, Indice *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);
}
```

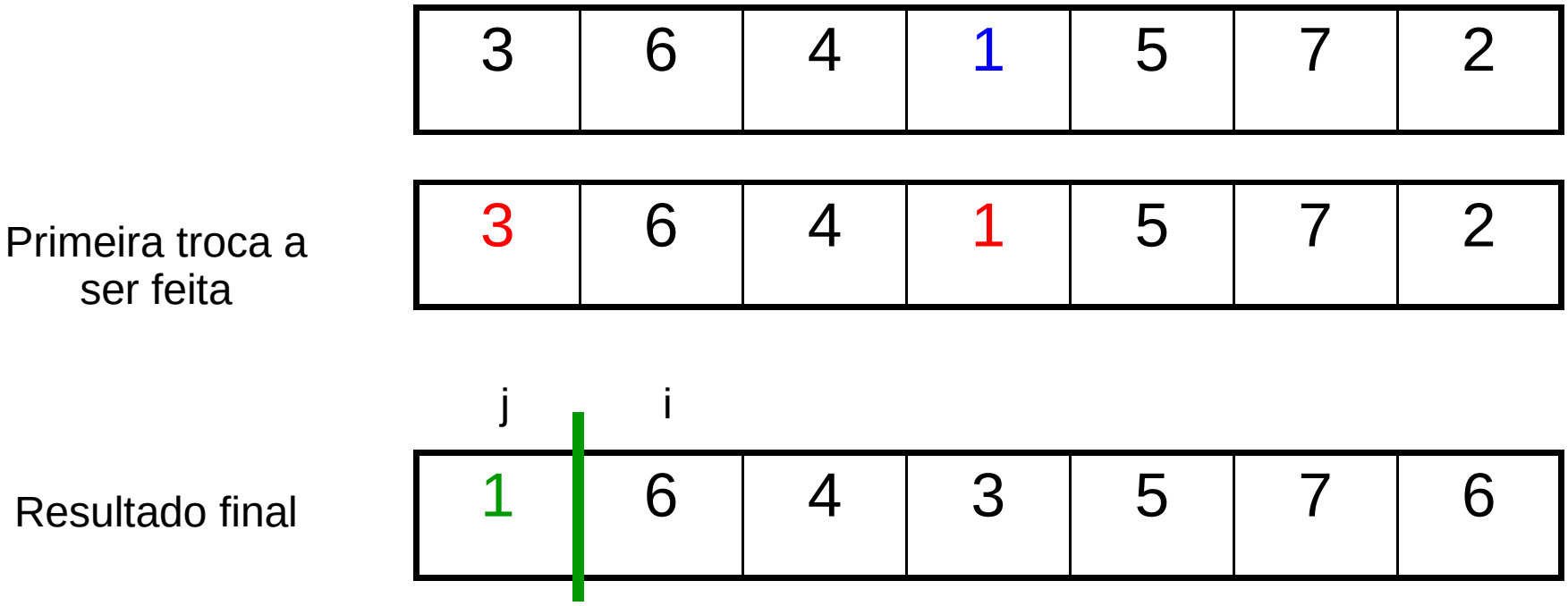
Partição – “Casos Especiais”

- Pivô é o menor ou maior de todos

3	6	4	1	5	7	2

Partição – “Casos Especiais”

- Pivô é o menor ou maior de todos



Depois da primeira troca, o ídice 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

5	7	3	1	6	8	4	2	0

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

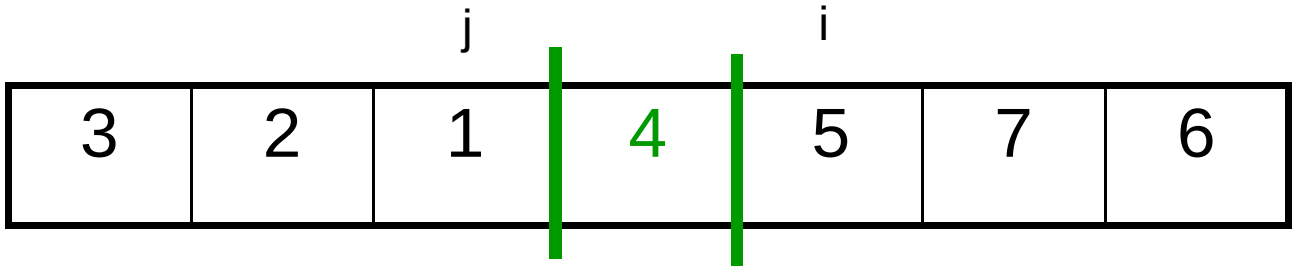
3	6	1	4	5	7	2

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 particões e mais um elemento (pivô) já na posição correta.

Resultado final



Quicksort

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

- Razão pela qual o algoritmo Quicksort é tão rápido.

Quicksort - Função

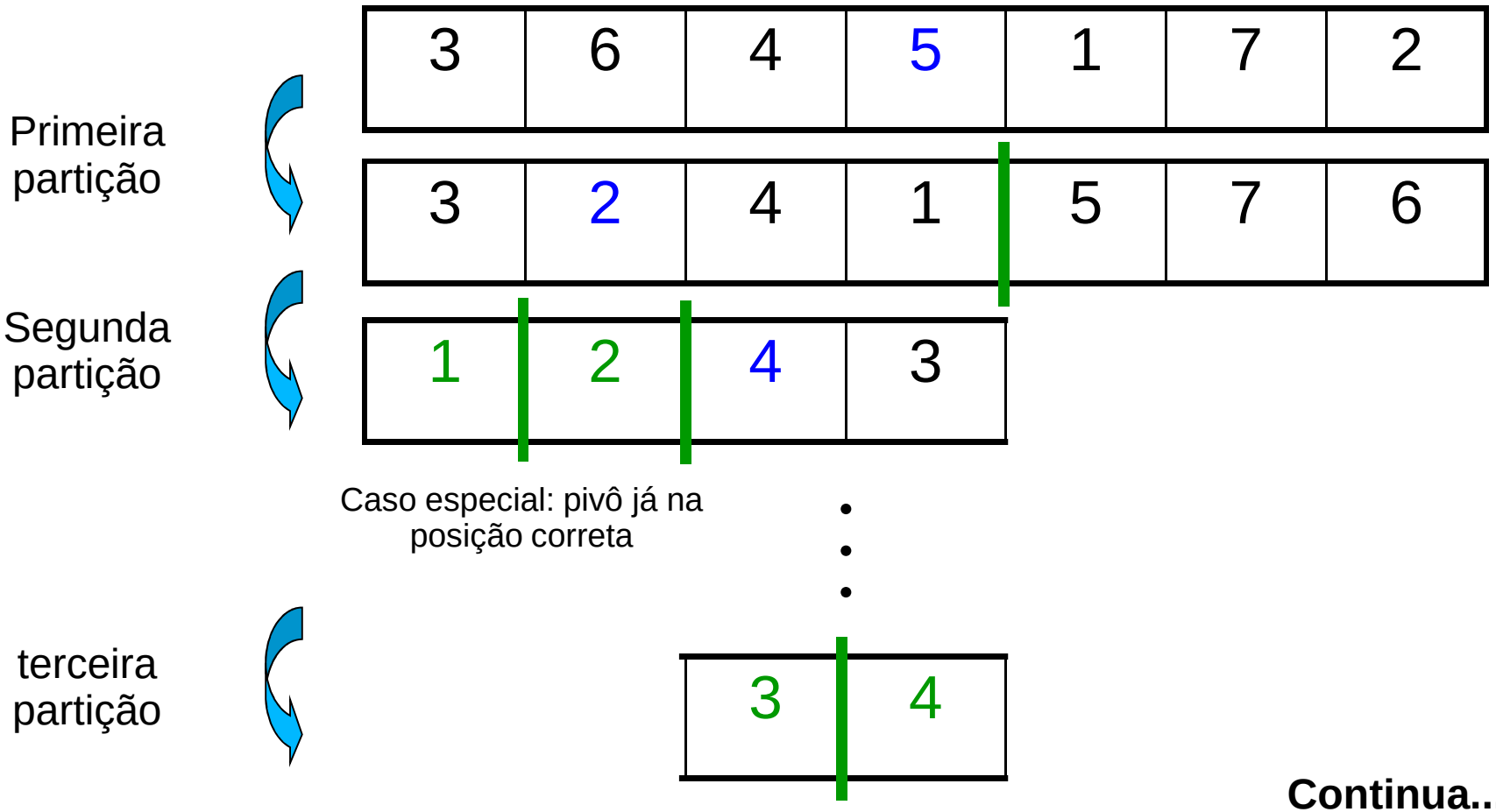
```
/* Entra aqui o procedimento Particao */
void Ordena(Indice Esq, Indice Dir, Item *A)
{ int i, j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, Indice *n)
{
  Ordena(1, *n, A);
}
```

Quicksort - Exemplo

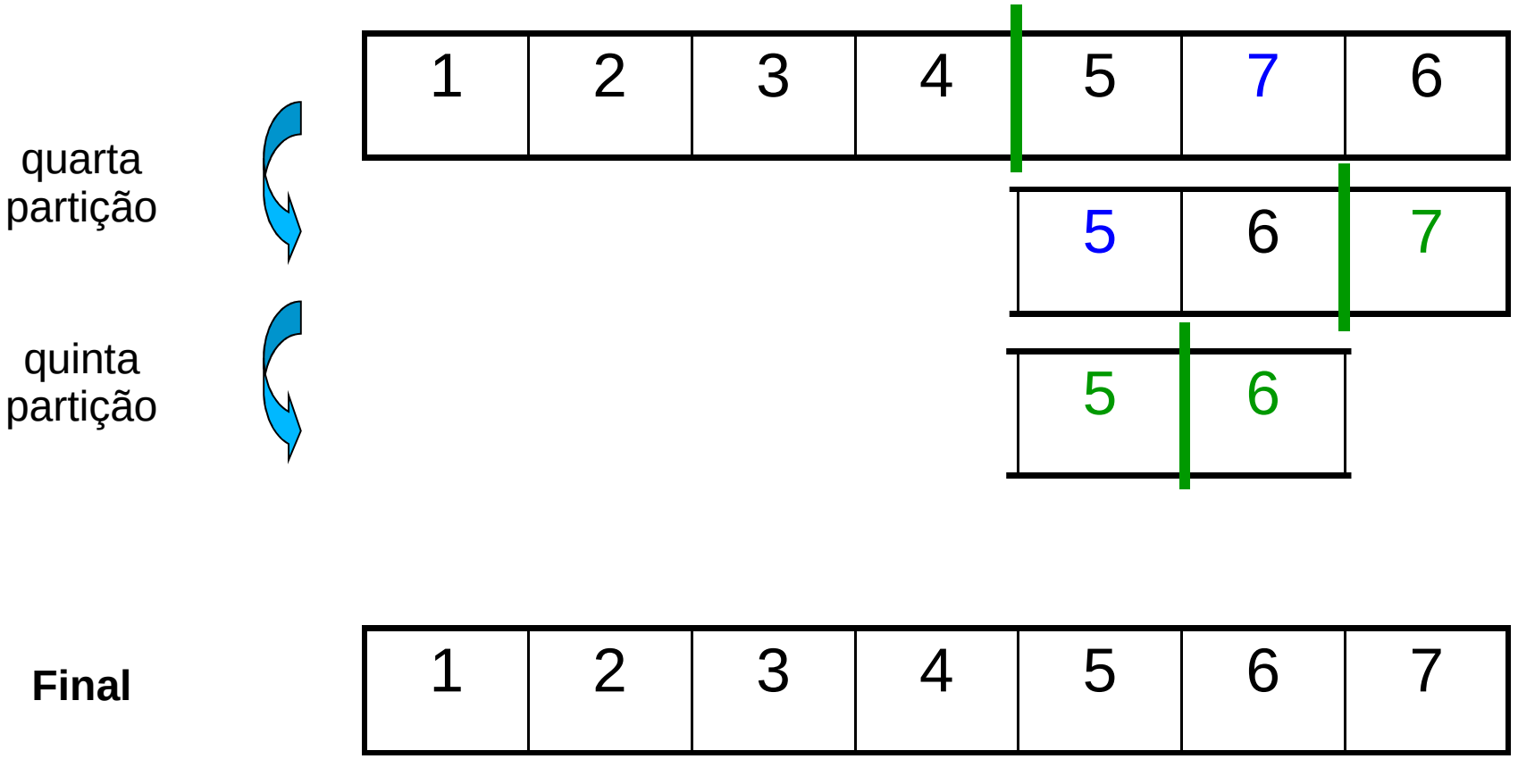
3	6	4	5	1	7	2

Quicksort - Exemplo



Continua...

Quicksort - Exemplo



Quicksort

- **Características**
  - Qual o pior caso para o Quicksort? Por que? Qual sua ordem de complexidade?
  - Qual o melhor caso? Por que? Qual sua ordem de complexidade?
  - O algoritmo é estável?

Quicksort

- **Análise**
  - Seja C(n) a função que conta o número de comparações.
  - Pior caso: $C(n) = O(n^2)$
  - O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.
  - Isto faz com que o procedimento Ordena seja chamado recursivamente n vezes, eliminando apenas um item em cada chamada.
  - O pior caso pode ser evitado empregando pequenas modificações no algoritmo.
  - Para isso basta escolher três itens quaisquer do vetor e usar a **mediana dos três** como pivô.

Quicksort

- **Análise**
  - Melhor caso:

    $C(n) = 2C(n/2) + n = n \log n – n + 1$
  - Esta situação ocorre quando cada partição divide o arquivo em duas partes iguais.
  - 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
- Depois da partição, trabalhar primeiro no subvetor de menor tamanho
  - Diminui o crescimento da pilha
- Utilizar um algoritmo simples (seleção, inserção) para partições de tamanho pequeno
- Quicksort não recursivo
  - Evita o custo de várias chamadas recursivas

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, Indice 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));
}
```

Para estudar o quicksort: “Professort”

- Ferramenta de auxílio (vai ser disponibilizada no learnloop)

- Algoritmos:
  - Quicksort não recursivo
  - Quicksort recursivo
  - Heapsort

Exercício

- Execute o quicksort no vetor abaixo indicando qual é o pivô e os subvetores resultantes de cada partição.

65	77	51	25	03	84	48	21	05