Método de Ordenação — QuickSort

Código e Análise de Complexidade

Larissa; Mauricio Monteiro; Paulo Henrique

UFMG — ICEx
Departamento de Ciência da Computação — DCC
Ambientes de Computação

Visão geral da apresentação

Introdução

O trabalho tem como objetivo:

- traduzir o código escrito na *linguagem C* para a *linguagem Python*;
- compreender o funcionamento do algoritmo proposto;
- explicar a **Análise de Complexidade** do código.

Código em C


```c
#define MAXTAM 20

typedef long TipoChave;

typedef struct TipoItem
{
  TipoChave Chave;
} TipoItem;

typedef int TipoIndice;
typedef TipoItem TipoVetor[MAXTAM + 1];

void Particao(TipoIndice Esq, TipoIndice Dir, TipoIndice *i, TipoIndice *j, TipoItem *A)
{
  TipoItem 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);
}
```

Código em C


```
56
57  void Ordena(TipoIndice Esq, TipoIndice Dir, TipoItem *A)
58  {
59
60    TipoIndice i = 0, j = 0;
61
62    Particao(Esq, Dir, &i, &j, A);
63
64    if (Esq < j)
65    {
66      Ordena(Esq, j, A);
67    }
68
69    if (i < Dir)
70    {
71      Ordena(i, Dir, A);
72    }
73  }
74
75  void QuickSort(TipoItem *A, TipoIndice n)
76  {
77    Ordena(0, n - 1, A);
78  }
79
80  TipoVetor A;
81
```

Descobrimos que

Descobrimos que o algoritmo é um método de ordenação muito conhecida chamada **QuickSort**.

Quicksort é o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações, sendo provavelmente mais utilizado do que qualquer outro algoritmo.

O algoritmo foi inventado por **Charles Antony Richard Hoare** em 1960, quando visitava a Universidade de Moscou como estudante.

O algoritmo foi publicado mais tarde por Hoare (1962), após uma série de refinamentos.

Código em Python


```
24  def Ordena(l, r, A):
25
26      i, j = Particao(l, r, A)
27
28      if l < j:
29          Ordena(l, j, A)
30
31      if i < r:
32          Ordena(i, r, A)
33
34      pass
35
36  def QuickSort(A):
37      n = len(A) - 1
38      Ordena(0, n, A)
39      pass
40
```

Código em Python


```python
def Particao (i, j, A):

    x = A[(i + j) // 2] # Obtem o pivo x

    ok = True
    while ok:

        while x > A[i]:
            i += 1

        while x < A[j]:
            j -= 1

        if i <= j:
            A[i], A[j] = A[j], A[i] # swap(A[i], A[j])

            i += 1
            j -= 1

        ok = True if i <= j else False

    return i, j

```

Lógica do Algoritmo

- A idéia básica é a de partir o problema de ordenar um conjunto $A = \{a_0, a_1, \cdots, a_{n-1}, a_n\}$ com n itens em **dois problemas menores**.
- Os problemas menores são ordenados independentemente e depois os resultados são combinados para produzir a solução do problema maior.

Lógica do Algoritmo

- A parte mais delicada deste método é relativa ao procedimento partição, o qual tem que rearranjar o vetor $A = \{a_{\text{esq}}, \cdots, a_{\text{dir}}\}$ através da escolha arbitrária de um item x do vetor chamado **pivô**.

- Ao final o vetor A estará particionado em uma **parte esquerda** com chaves **menores ou iguais** a x e uma **parte direita** com chaves **maiores ou iguais** a x.

Lógica do Algoritmo

Este comportamento pode ser descrito pelo seguinte algoritmo:

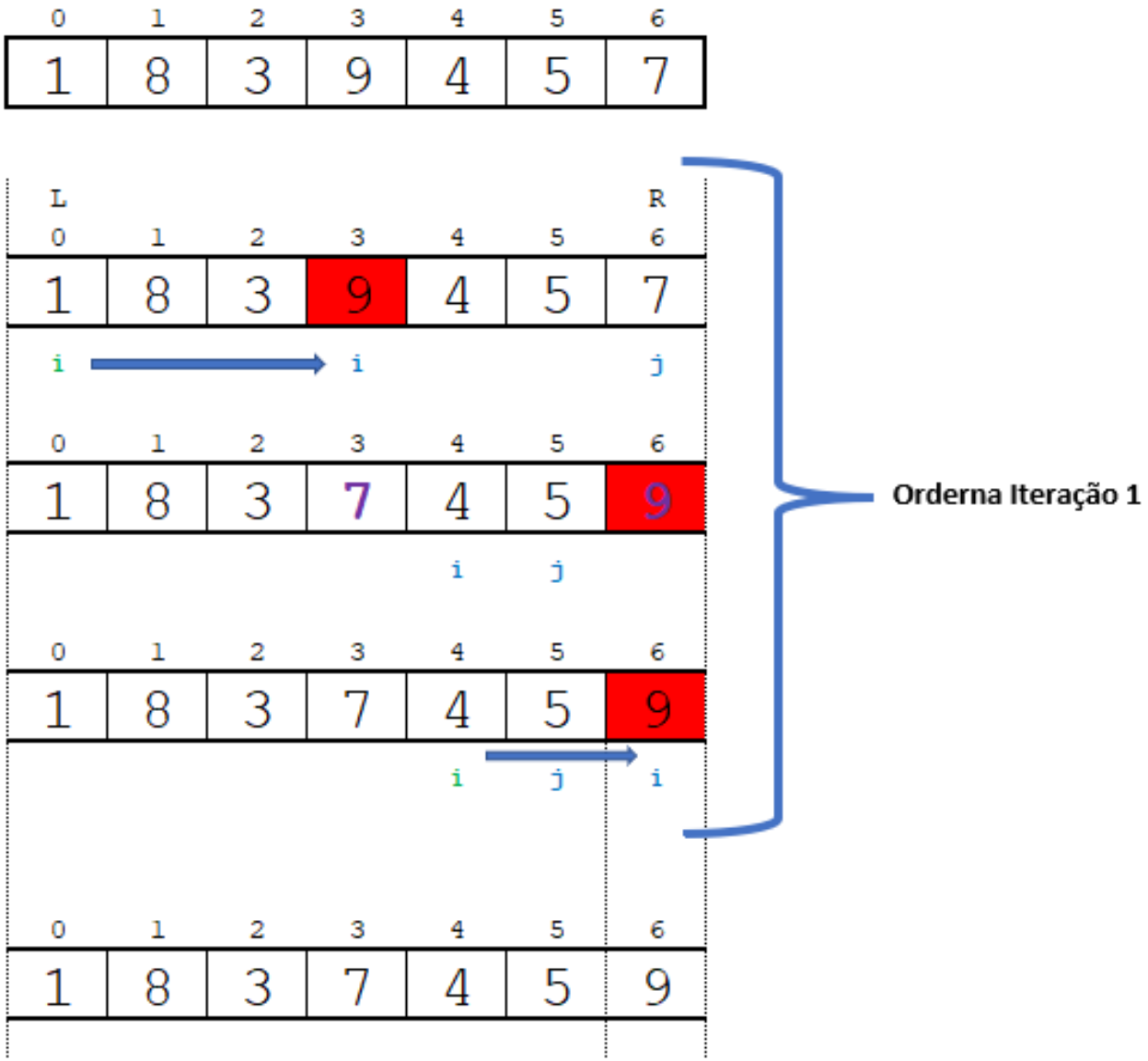
1. escolha arbitrariamente um item do vetor e o coloque em x;
2. percorra o vetor a partir da esquerda até que um item $a_i > x$ é encontrado.
3. da mesma forma percorra o vetor a partir da direita até que um item $a_j \leq x$ é encontrado;
4. como os dois itens A_i e A_j estão fora de lugar no vetor final então **troque-os de lugar**;
5. continue este processo até que os apontamentos i e j se **cruzem** em algum pronto do vetor.

Lógica do Algoritmo

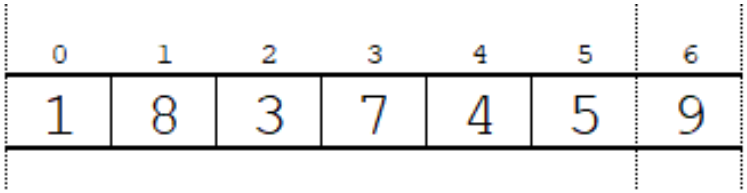
Ao final o vetor $A = \{a_{\text{esq}}, \cdots, a_{\text{dir}}\}$ está parcionado de tal forma que:

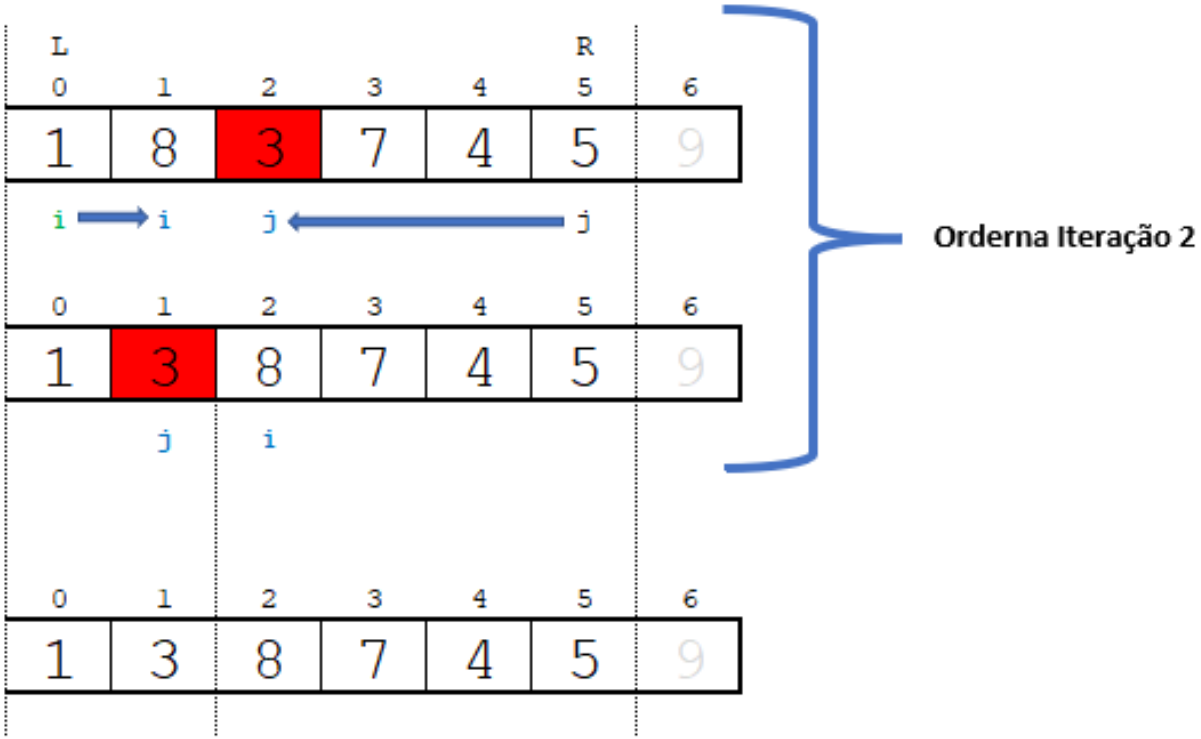
- os itens em $a_{\text{esq}}, a_{\text{esq}+1}, \cdots, a_j$ são **menores** ou iguais a x,
- os itens em $a_i, a_{i+1}, \cdots, a_{dir}$ são **maiores** ou iguais a x.

Passo a Passo

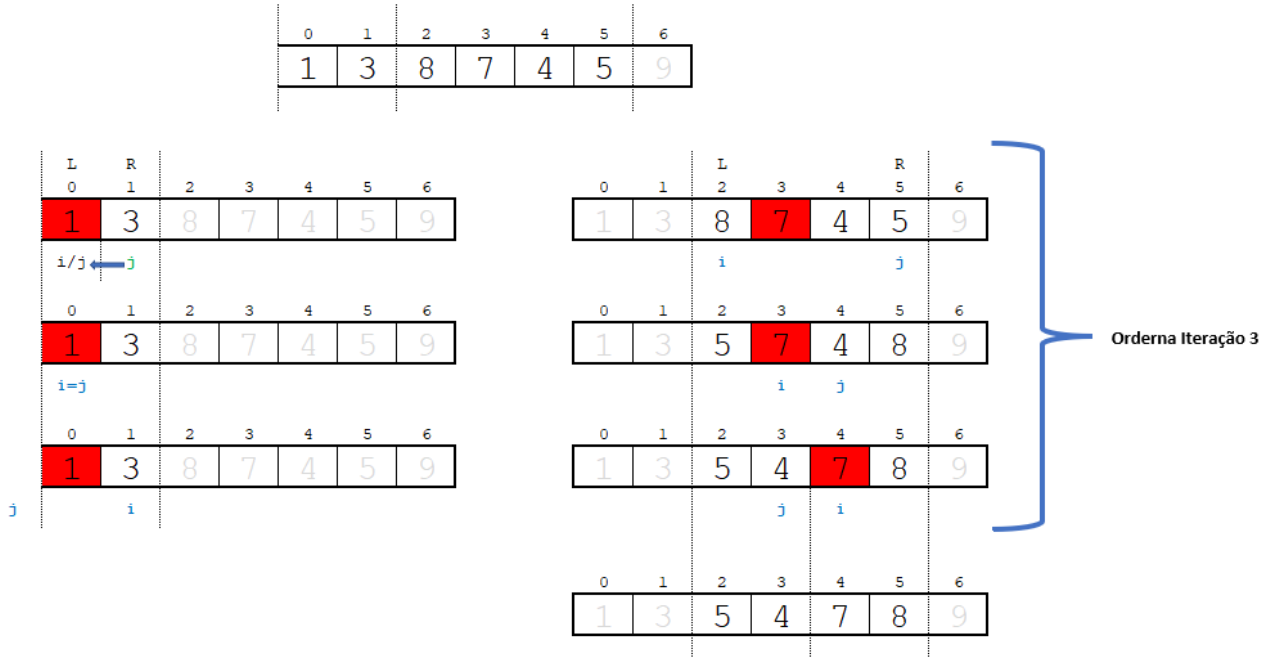


Passo a Passo

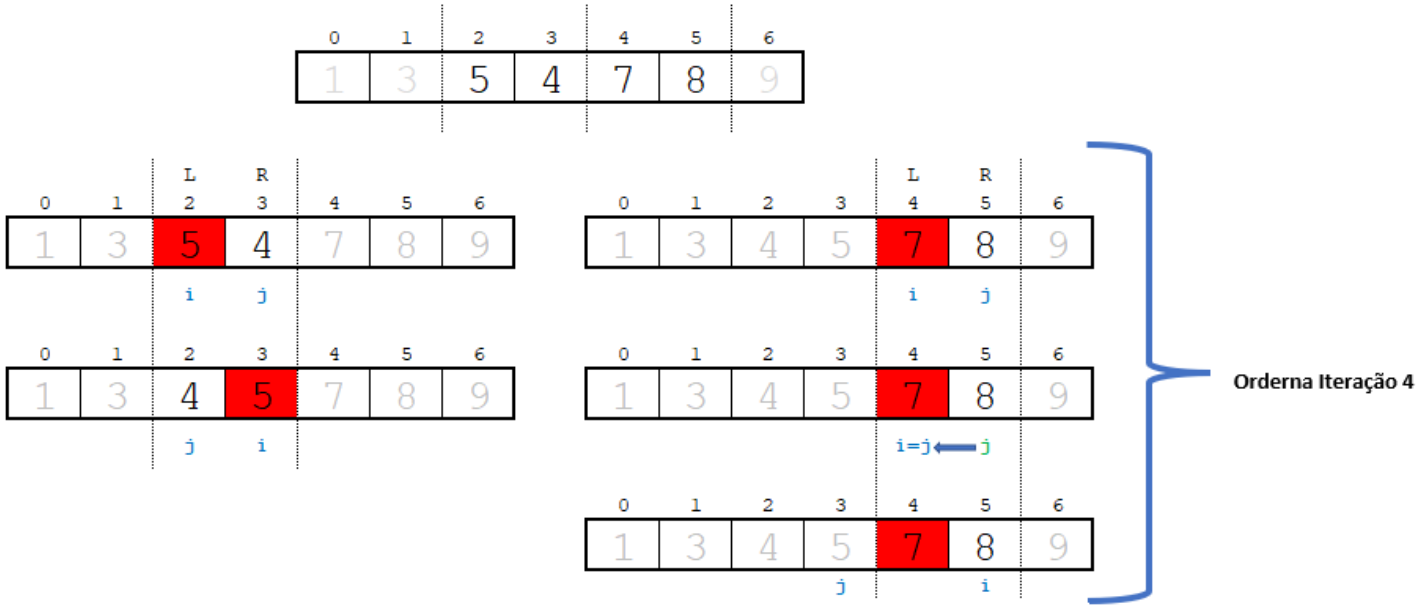




Passo a Passo



Passo a Passo

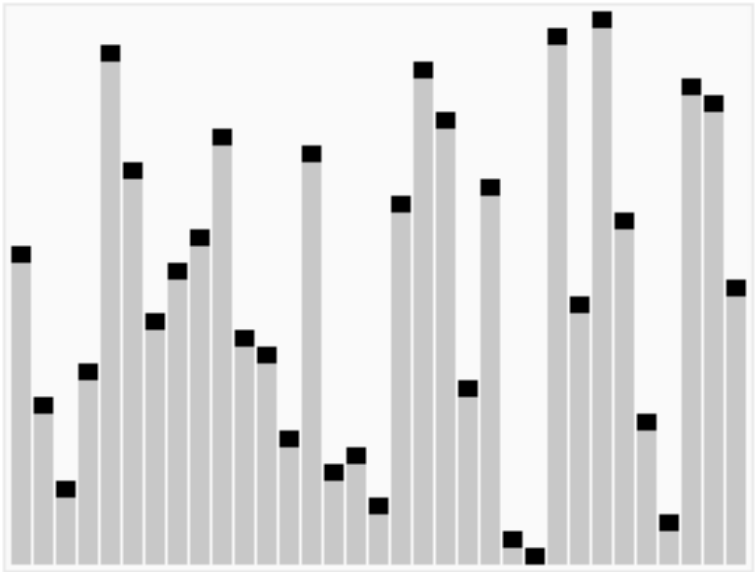


Passo a Passo

Animação do exemplo proposto:

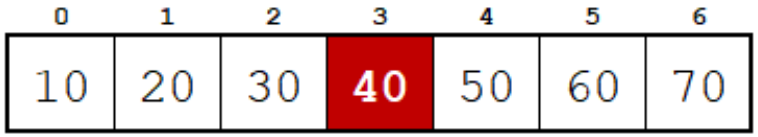
0	1	2	3	4	5	6
1	8	3	9	4	5	7

Outra animação de exemplo:



Análise de Complexidade

O Quicksort é ineficiênte para conjuntos de dados já ordenados, pois gera a escolha do pivô é inadequada.



$$x = A\left[\frac{(0+6)}{2}\right] = 40$$

- No exemplo acima escolha sistemática dos extremos de um conjuntos de dados já ordenado leva ao seu **pior caso**.
- As partições serão extremamente **desiguais**, e o procedimento *Ordena*(…) será chamado recursivamente n vezes, **eliminando apenas um item** em cada chamada.
- Esta situação é desastrosa pois o número de comparações chegar ser $\frac{n^2}{2}$, e o tamanho da pilha necessária para as chamadas recursivas chega ser de tamanho n.

Análise de Complexidade

Obrigado.

Perguntas?