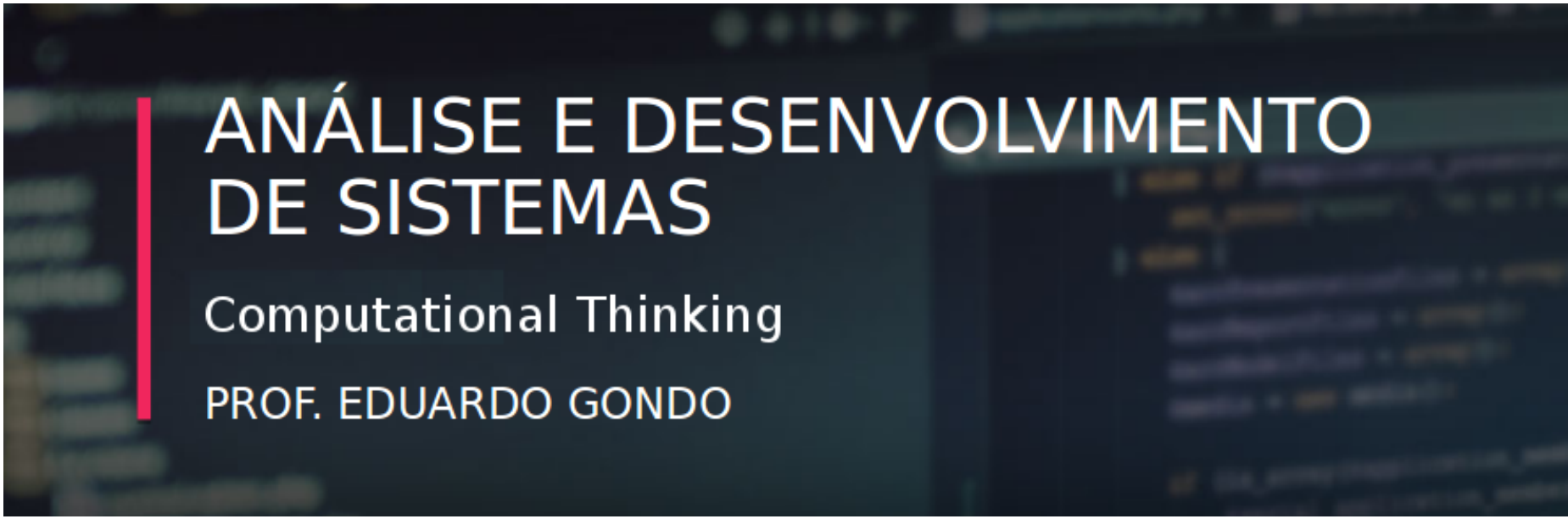

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

Intercala

Considere o seguinte problema: dados um conjunto/lista v de inteiros e três inteiros p, q e r tal que $v[p..q]$ está ordenado e $v[q+1..r]$ está ordenado. Escreva um método que retorna o vetor $v[p..r]$ com todos elementos ordenados.

Entrada

-2	5	8	14	-7	3	7	10	20

Saída

-7	-2	3	5	7	8	10	14	20

Qual é o valor de q que deverá ser passado como parâmetro?

Intercala — Algoritmo 1

- uma solução seria executar o algoritmo Selection Sort no vetor $v[p..r]$
- este algoritmo ignora totalmente o parâmetro q
- só que ele levará mais tempo para ordenar o vetor
- podemos fazer uso do fato das partes do vetor estarem ordenadas

Intercala — Algoritmo 2

- vamos tentar encontrar um algoritmo que faz uso do fato do vetor ser composto por dois subvetores ordenados
- imagine uma situação com duas pilhas de provas, onde cada uma delas está ordenada
- como fazer uma única pilha ordenada com todas as provas
- note que precisamos de um vetor extra para armazenar o vetor ordenado
- o seguinte invariante nos ajuda a elaborar um algoritmo para juntar as duas pilhas
- a cada iteração um elemento é selecionado de um subvetor ordenado para o vetor auxiliar

Intercala — Algoritmo 2

```
def intercala(vetor, p, q, r):

    //Exercício
```

Usaremos o `intercala` para ordenar um vetor rapidamente, mas antes vamos conhecer mais uma técnica de desenvolvimento de algoritmos que será aplicada para a ordenação

Divisão e Conquista

Divisão Divida o problema em duas ou mais partes, criando subproblemas menores.

Conquista Os subproblemas são resolvidos recursivamente usando a mesma técnica. Caso os subproblemas tenham instâncias pequenas vamos resolvê-los diretamente.

Combina Junte cada uma das soluções dos subproblemas para obter a solução do problema original.

Ordenacão — MergeSort

- o algoritmo mergesort é um algoritmo de ordenação baseado na técnica de divisão e conquista
- usamos o seguinte esquema para aplicar a técnica de divisão e conquista:
  - **dividir** dividir o vetor em duas partes
  - **consquistar** ordene cada uma das partes
  - **combinar** intercale os subvetores ordenados obtendo um vetor resultante da união dos subvetores ordenado

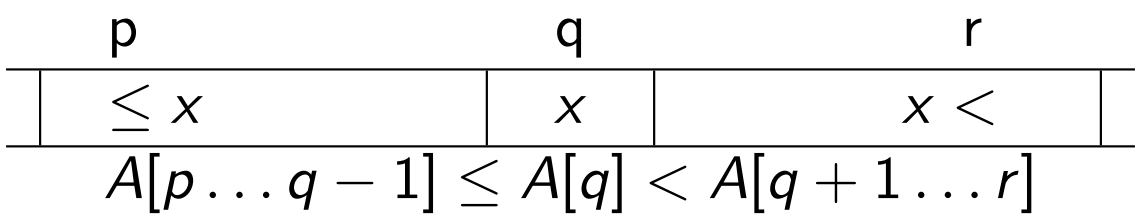
```
def mergeSort(vetor, p, r):
    if p < r:
        q = (p + r)// 2
        mergeSort(vetor, p, q)
        mergeSort(vetor, q+1, r)
        intercala(vetor, p, q, r)
```

QuickSort

- inventado por C.A.R. Hoare em 1960
- não necessita de espaço extra como o MergeSort
- o algoritmo de ordenação (baseado em comparações) mais rápido conhecido — na prática
- porém, no pior caso ele pode executar no mesmo tempo dos algoritmos mais lentos de ordenação
- algoritmo baseado em divisão e conquista

QuickSort — Algoritmo

o algoritmo quicksort é um algoritmo de ordenação baseado na técnica de divisão e conquista
usamos o seguinte esquema para aplicar a técnica de divisão e conquista:
dividir divida o vetor A em dois subvetores $A[p \ldots q-1]$ e $A[q+1 \ldots r]$ tais que:



conquistar ordene cada um dos subvetores usando o quicksort
combinar nada a fazer, pois o vetor está ordenado

QuickSort — Separação

Problema da Separação dados um vetor A de inteiros e dois inteiro p e r indicando o início e fim do vetor. Escreva um método que seleciona um elemento de A elegendo como pivô e movimenta os elementos de A de tal forma que no início (lado esquerdo) ficam todos os elementos menores ou iguais ao pivô e no fim (lado direito) somente os elementos maiores do que o pivô. Seu algoritmo deverá retornar a posiçao do pivô no vetor após a separação.

Entrada: o vetor abaixo e pivô valendo 7

3	5	10	2	-7	9	15	-4	8	1	7

Saída

3	5	2	-7	-4	1	7	10	9	15	8

Separação — Discussão

- o algoritmo que descreveremos pode não reproduzir exatamente a saída do eslaide anterior
- porém, note que a saída atende aos requisitos do problema
- podemos garantir que o pivô está examente na posição se o vetor estivesse ordenado
- pois se a posição do pivô é q, temos que $A[i] \leq A[q]$ para todo $i \leq q$ e $A[j] > A[q]$ para todo $j > q$
- para ordenarmos o vetor precisamos ordenar o vetor à esquerda e à direita da posição q
- note que no exemplo, houve um bom balanceamento dos elementos do vetor, ou seja, 5 elementos ficaram à esquerda e 4 à direita
- sempre gostaríamos de encontrar pivôs que fizessem uma boa separação dos elementos, mais ou menos no meio porém isso nem sempre é garantido

Separação — Algoritmo 1

Segue abaixo uma implementação do algoritmo de separação.

```
def separacao(A, p, r):
    pivo = A[r]
    i = p - 1
    for j in range(p, r):
         if A[j] <= pivo:
             i = i + 1
             aux = A[i]
             A[i] = A[j]
             A[j] = aux

    aux = A[r]
    A[r] = A[i+1]
    A[i+1] = aux
    return i+1
```

Separação — Algoritmo 2

Podemos criar um outro algoritmo para a separação. Para começar podemos selecionar o pivô fazendo uma média aritmética entre p e r ao invés de pegar sempre a posição r do vetor.

A separação do vetor pode ser feita através de dois ponteiros percorrendo os índices do vetor. Um começando com o valor de p e outro com o valor de r. Eles vão se movimentando até que encontrem valores que estejam no local errado. Quando encontrados eles devem ser trocados. O algoritmo continua até que os dois ponteiros tenham verificado todo o vetor.

Implemente o algoritmo descrito.

QuickSort — Algoritmo

Agora que temos o método de separação podemos mostrar a implementação do QuickSort.

```
def quickSort(A, p, r):
    if p < r:
        q = separacao(A, p, r)
        quickSort(A, p, q-1)
        quickSort(A, q+1, r)
```

Exercícios

1) Escreva o método intercala
2) Implemente o MergeSort e faça um exemplo de uso do MergeSort
3) Simule a execução do MergeSort para um vetor de 8 elementos
4) Simule a execução do QuickSort para um vetor de 8 elementos
5) Tente fazer um programa que compara o tempo de execução dos algoritmos MergeSort e QuickSort com o InsertionSort e BubbleSort. Faça testes com vários tipos de listas dos mais variados tamanhos pegando o tempo de execução deles.

Bibliografia

- Algoritmos - Cormen, Leiserson, Rivest e Stein - Editora Campus
- Projeto de Algoritmos - Nivio Ziviani - Editora Thomson
- Algoritmos em Linguagem C - Paulo Feofiloff - Editora Campus/Elsevier
- Algoritmos e Estrutura de Dados - Niklaus Wirth - Editora LTC

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados
- Documentação do Python - https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle

Copyleft

Copyleft © 2021 Prof. Eduardo Gondo Todos direitos liberados.
Reprodução ou divulgação total ou parcial deste documento é liberada.