ESTRUTURA DE DADOS I

Aula 03

Prof. Sérgio Luis Antonello

FHO - Fundação Hermínio Ometto

10/03/2025

Plano de Ensino

1. Unidade I – Métodos de ordenação em memória principal (objetivos d, e, f)
   1.1. Revisão de tipos de dados básicos em C, variáveis indexadas e recursividade
   1.2. Noções de complexidade computacional
   1.3. Conceitos e métodos de ordenação de dados
   1.4. Bubble sort, Insert sort e Select sort
   1.5. Quick sort e Merge sort
   1.6. Shell sort e Radix sort
2. Unidade II – Métodos de pesquisa em memória principal (objetivos e, f)
   2.1. Pesquisa sequencial
   2.2. Pesquisa binária
   2.3. Hashing
3. Unidade III – Tipo abstrato de dados (TAD) (objetivo a)
   3.1. Revisão de registros, ponteiros e alocação dinâmica de memória
   3.2. Tipo abstrato de dados (TAD): conceitos e aplicações
4. Unidade IV – Estrutura de dados lineares (objetivos a, b, c)
   4.1. Lista Encadeada: conceitos e aplicações
   4.2. Pilha: conceitos e aplicações
   4.3. Fila: conceitos e aplicações

Cronograma do Plano de Ensino

- 17/02 - Recursividade; Complexidade de tempo; Notação Big-Oh.
- 24/02 - Métodos de ordenação: Bubble sort; Insert sort; Select sort.
- 10/03 - Métodos de ordenação: Quick sort; Merge sort.
- 17/03 - Métodos de ordenação: Shell sort; Radix sort.
- 24/03 - Métodos de pesquisa: Sequencial; Binária.
- 31/03 - Métodos de pesquisa: Hashing.
- 07/04 – Desenvolvimento do trabalho A1.
- **14/04 - Prova 1**

Sumário

- Primeiro momento (revisão)
  - Bubble sort
  - Insert sort
  - Select sort
- Segundo momento
  - Métodos de ordenação em memória primária
    - Quick sort
    - Merge sort
- Terceiro momento (síntese)
  - Retome pontos importantes da aula

1. Primeiro momento: revisão

- Bubble sort

  https://www.youtube.com/watch?v=lyZQPjUT5B4

  http://math.hws.edu/eck/js/sorting/xSortLab.html

- Insert sort

  https://www.youtube.com/watch?v=ROalU379l3U

  https://visualgo.net/en/sorting

- Select sort

  https://www.youtube.com/watch?v=Ns4TPTC8whw

  https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

1. Primeiro momento: revisão

- Sistemas de busca na internet:

  Quando realizamos uma pesquisa no Google, por exemplo, os resultados são ordenados por relevância. Os algoritmos de ordenação permitem que os resultados mais relevantes sejam exibidos primeiro, proporcionando uma experiência de busca mais eficiente.

- E-commerce:

  As redes sociais utilizam algoritmos de ordenação para exibir as publicações mais relevantes para cada usuário. Esses algoritmos consideram fatores como o número de curtidas, comentários e compartilhamentos, garantindo que o conteúdo mais interessante seja exibido primeiro.

1. Primeiro momento: revisão

- Sistemas de recomendação:

  Empresas que lidam com grandes volumes de dados, como bancos e empresas de telecomunicações, utilizam algoritmos de ordenação para organizar e analisar informações. Esses algoritmos permitem identificar padrões, tendências e anomalias nos dados, auxiliando na tomada de decisões estratégicas.

- Jogos:

  Estes exemplos demonstram a importância dos métodos de ordenação de dados em diversas áreas do conhecimento e do mercado de trabalho. Ao dominar esses métodos, vocês estarão preparados para enfrentar os desafios da área de Sistemas de Informação e contribuir para o desenvolvimento de soluções inovadoras.

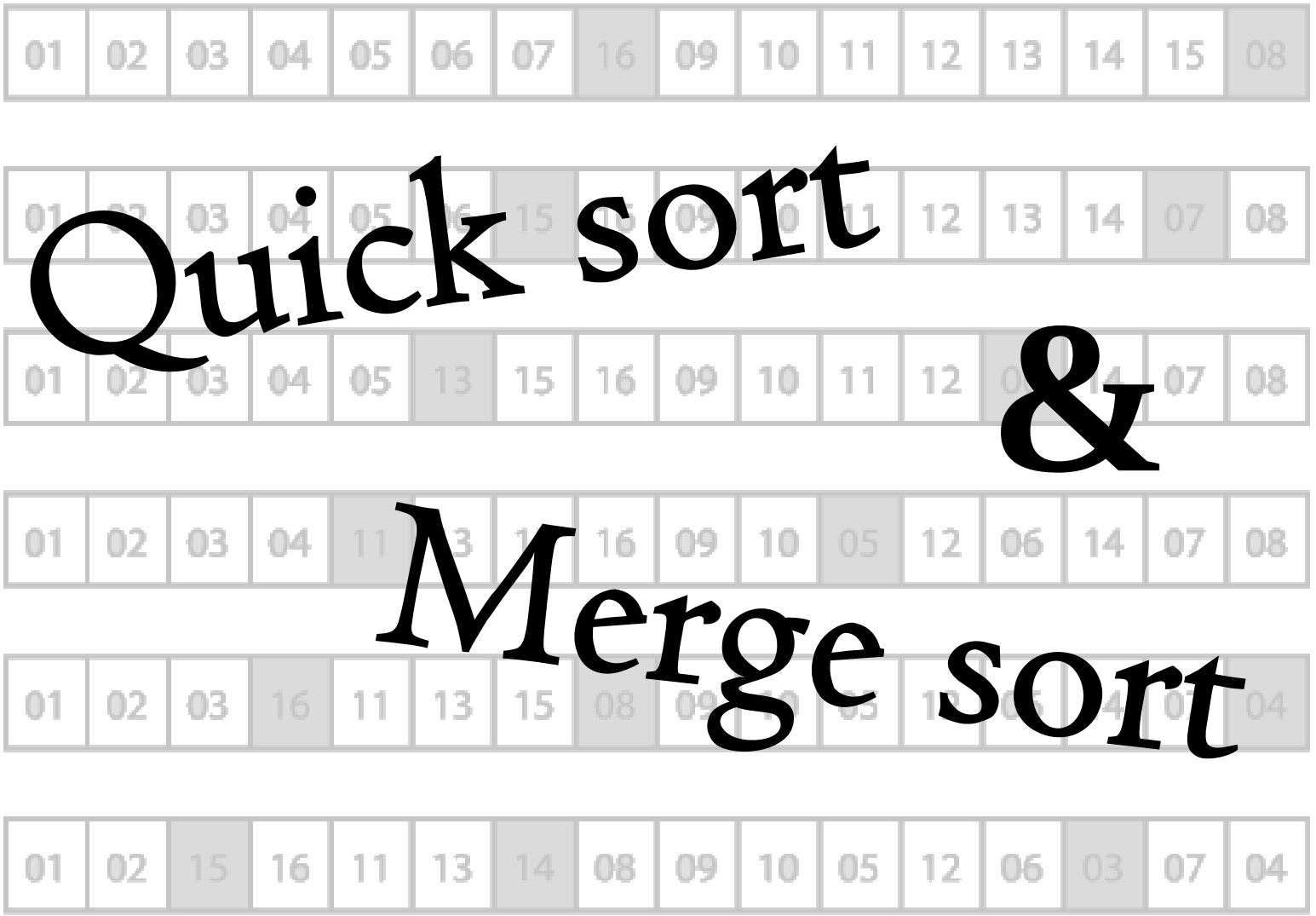
1. Primeiro momento: revisão

- O método de ordenação Bubble sort percorre a estrutura várias vezes até que não ocorram mais trocas de elementos. Em cada percurso, compara-se dois-a-dois os elementos a serem trabalhados.
- O método de ordenação Insert sort separa logicamente a estrutura em duas partes. A esquerda ficam os elemento já ordenados e a direita ficam os elementos ainda não ordenados. A parte a esquerda inicia com apenas um elemento e vai crescendo a medida que o processo avança. Na parte da direita ocorre o inverso.
- O método de ordenação Select sort é composto pela repetição do seguinte processo: Na parte da estrutura a ser trabalhada, busca-se o menor elemento e troca de lugar com o primeiro elemento da estrutura. Repete o processo selecionando o segundo menor elemento e trocando de lugar com o segundo elemento e assim por diante.

1. Primeiro momento: revisão

Correção de exercícios

2. Segundo momento



2. Motivação

Um arquivo histórico municipal guarda informações públicas ou privadas de domínio público. Como exemplo estão: os dados da folha de pagamento da prefeitura e da câmara municipal; os dados dos impostos dos contribuintes; os dados de serviços públicos prestados a todas a propriedades do município, como água e esgoto; os dados dos imigrantes e seus antepassados; etc.

Acontece que a demanda por uso dos dados armazenados é enorme e muitas vezes, para uma única pesquisa, pode-se levar dias até a obtenção de uma resposta.

O Departamento de água de um determinado município precisa provar que efetuou a compra de uma grande área dentro da cidade, na qual está construída uma de suas estações de tratamento de água para distribuição por meio de suas adutoras.

A compra foi efetuada na década de 50 do século passado, e existem dezenas de milhares de caixas no arquivo municipal para pesquisa aos documentos comprobatórios.

Como proceder?

2. Ordenação de dados

- Os métodos de ordenação **interna** são subdivididos em dois grupos.

Métodos simples

- Bubble sort
- Insert sort
- Select sort

Métodos eficientes

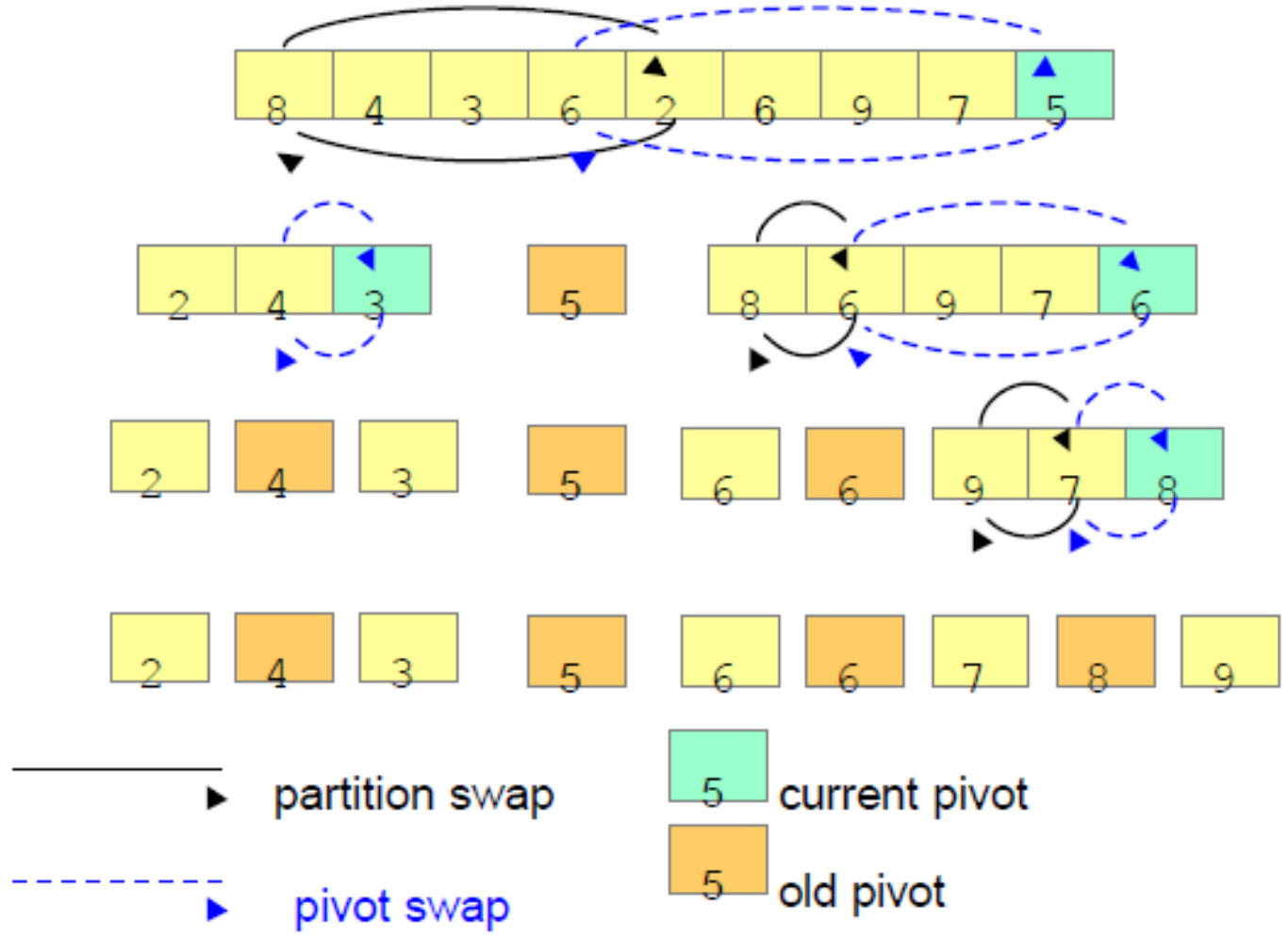
- **Quick sort**
- **Merge sort**
- Shell sort
- Radix sort

- Animação de métodos de ordenação interna
  https://www.toptal.com/developers/sorting-algorithms

2.1. Quick sort

- Proposto por Hoare em 1960 e publicado em 1962.
- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.
- Baseia-se na solução de problemas conhecida como **Divisão e Conquista** (Divide-and-Conquer).
- Os problemas menores são ordenados independentemente.
- Os resultados são combinados para produzir a solução final.

Recursive Partitioning

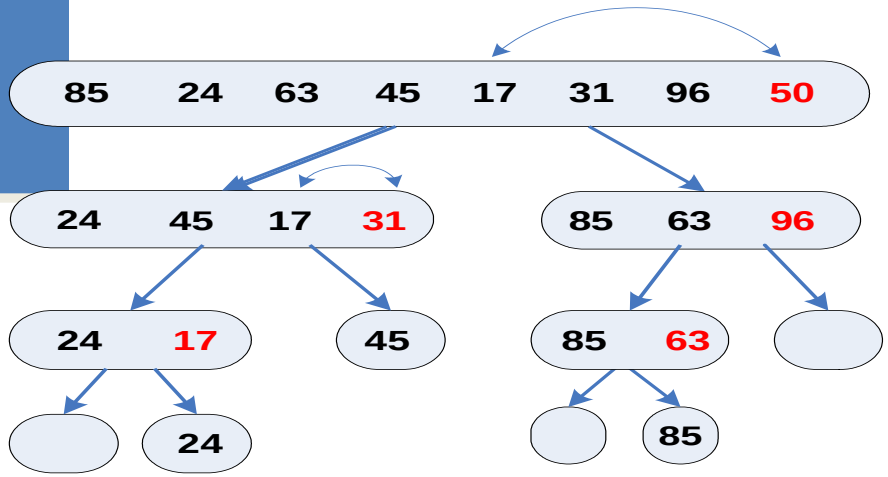


2.1. Quick sort

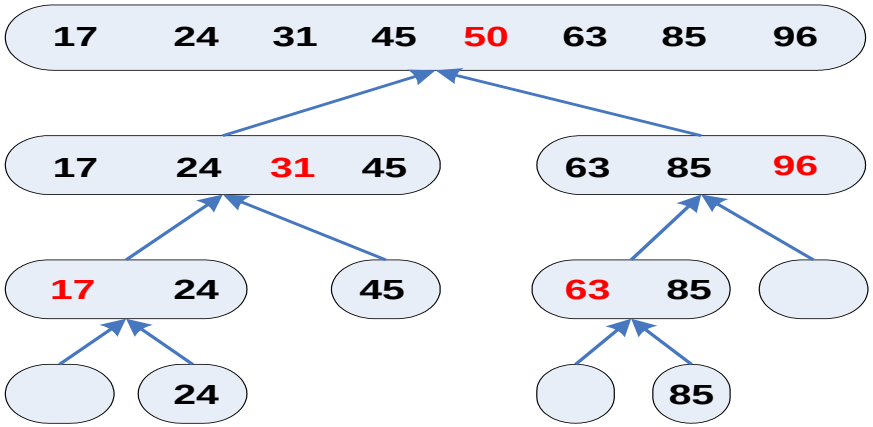
- Pode ser descrito, de maneira geral, como sendo composto de 3 fases:
  - **Divisão**: divide-se os dados de entrada em dois ou mais conjuntos separados;
  - **Recursão**: soluciona-se os problemas associados aos subconjuntos recursivamente;
  - **Conquista**: obtém-se as soluções dos subproblemas e junta-se as mesmas em uma única solução.

2.1. Quick sort

- Escolha arbitrária de um pivô x.
- Consequentemente, o vetor v é particionado em duas partes, a parte esquerda com chaves ≤ x e a parte direita com chaves > x.



Fase de Divisão



Fase de Conquista

2.1. Quick sort

- **Em resumo:**
  - Inicialmente, o vetor de chaves C é **particionado** em três segmentos S1, S2 e S3.
  - S2 deverá conter apenas UMA chave denominada pivô.
  - S1 deverá conter todas as chaves cujos valores são MENORES ou IGUAIS ao pivô. Esse segmento está posicionado à esquerda de S2.
  - S3 deverá conter todas as chaves cujos valores são MAIORES do que o pivô. Esse segmento está posicionado à direita de S2.
  - Por meio de **recursividade**, o particionamento é reaplicado aos segmentos S1 e S3 e a todos os segmentos daí resultantes com quantidade de chaves maior que 1.

2.1. Quick sort

```
função quickSort (vetor, esquerda, direita):
    escolha um elemento para ser o pivô
    distribua os elementos ao redor do pivô:
     (elementos menores que o pivô ficam à esquerda; maiores à direita)
        faça i = esquerda E j = direita
        enquanto i <= j:
            enquanto vetor[i] < pivô E i < direita: incremente i
            enquanto vetor[j] > pivô E j > esquerda: decremente j
            se i <= j:
                troque vetor[i] com vetor[j]
                incremente i E decremente j
    chame o procedimento recursivo:
        se (j > esquerda): quickSort(vetor, esquerda, j)
        se (i < direita): quickSort(vetor, i, direita)
```

2.1. Quick sort

- **Observação:** no pseudo-código anterior:
  - *"esquerda"* é a posição (índice) inicial do vetor.
  - *"direita"* é a posição (índice) final do vetor.

- Assim, a primeira chamada à função *"quickSort"* será:

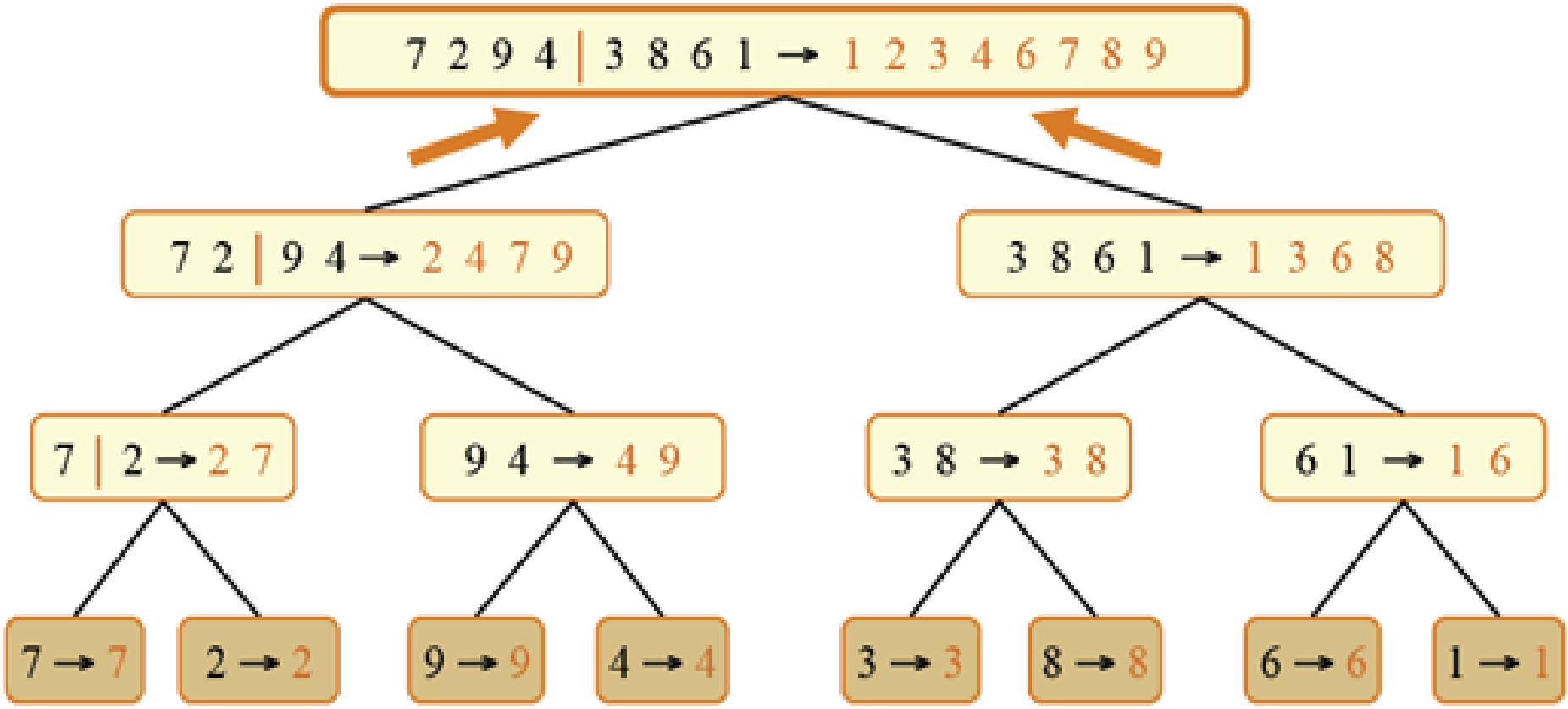
```
quickSort(vetor, 0, N-1)
```

sendo "N" o tamanho do vetor.

2.2. Merge sort

- Também baseia-se na solução de problemas **Divisão e Conquista** (Divide-and-Conquer).
- Consiste em:
  - Iniciar o processo dividindo recursivamente o vetor a ser ordenado em dois vetores até obter n vetores de um único elemento.
  - Em seguida aplicar o algoritmo de merge tendo como entrada 2 vetores de um elemento e formando um vetor ordenado de dois elementos.
  - Repete-se este processo formando vetores ordenados cada vez maiores até que todo o vetor esteja ordenado.

2.2. Merge sort



2.2. Merge sort

```
função mergeSort (vetor, N):

    se N > 1:

        meio = N/2

        chama o procedimento recursivo:

            mergeSort(vetor, meio)

            mergeSort(vetor com ponteiro no meio, (N – meio))

        combina ambas as partições do vetor:

            merge(vetor, N)
```

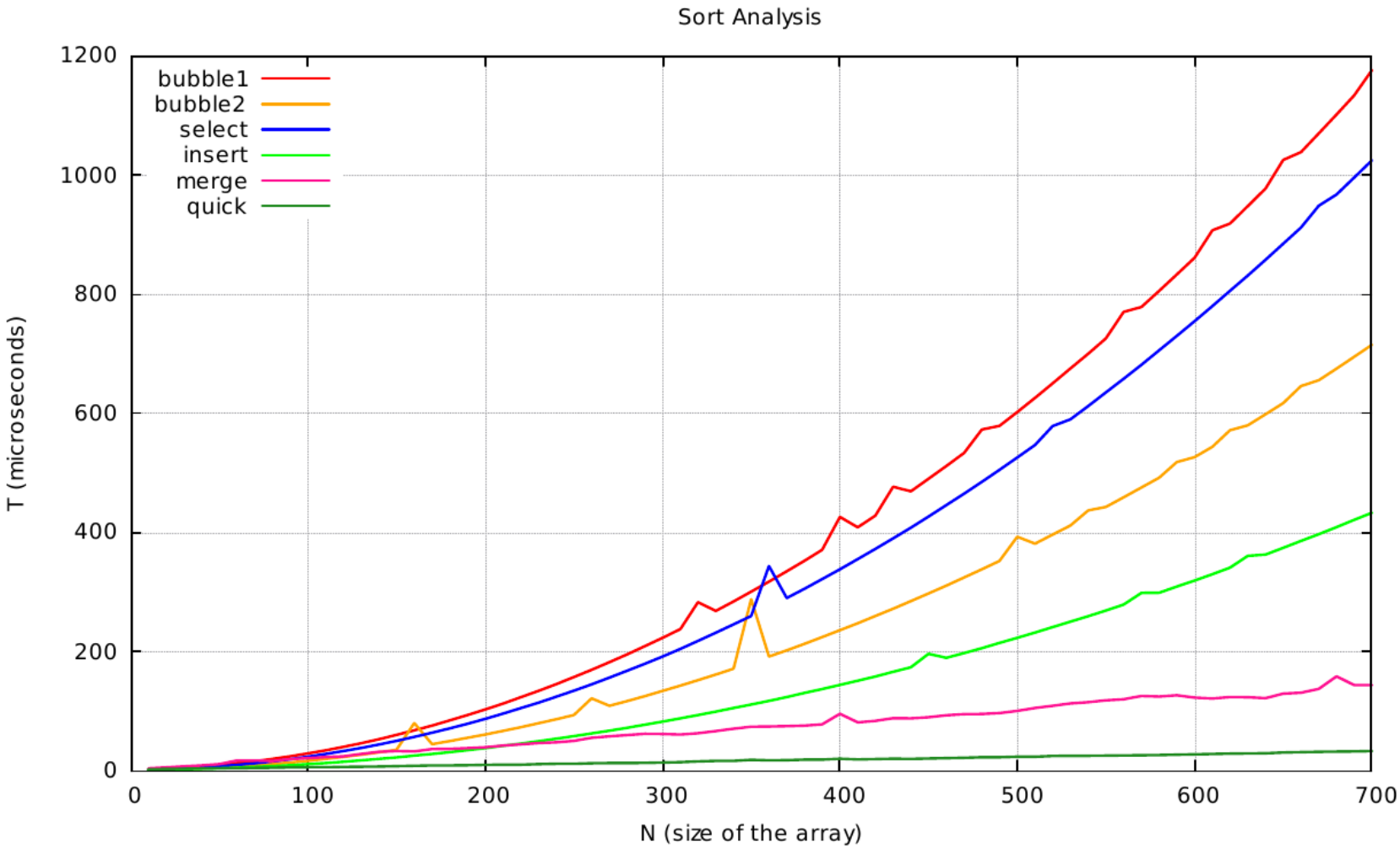
2.2. Merge sort

```
função merge (vetor, N):
    meio = N/2
    declare e aloque vetorAux com tamanho N
    considere que "partição 1" são os elementos de 0 a meio-1
    considere que "partição 2" são os elementos do meio até N-1
    compare cada par de elementos em ambas as partições:
        faça i=0, j=meio e k=0
        enquanto i < meio e j < N, faça:
            se vetor[i] <= vetor[j]: vetorAux[k++] = vetor[i++]
            senão: vetorAux[k++] = vetor[j++]
        se i == meio: vetorAux[k++] = vetor[j++], enquanto j < N
        senão: vetorAux[k++] = vetor[i++], enquanto i < meio
    copie vetorAux para vetor
```

3. Complexidade de Tempo

Algoritmo	Melhor Caso	Caso Médio	Pior Caso
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$
Select sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insert sort	O(n)	$O(n^2)$	$O(n^2)$
Quick sort	O(n log n)	O(n log n)	$O(n^2)$
Merge sort	O(n log n)	O(n log n)	O(n log n)

3. Complexidade de Tempo



4. Exercícios

Vamos Praticar!

4. Exercícios

1) Desenvolva código em C, que dado um Vetor com 10 elementos carregados com valores inteiros, classifique-o usando o Merge sort.

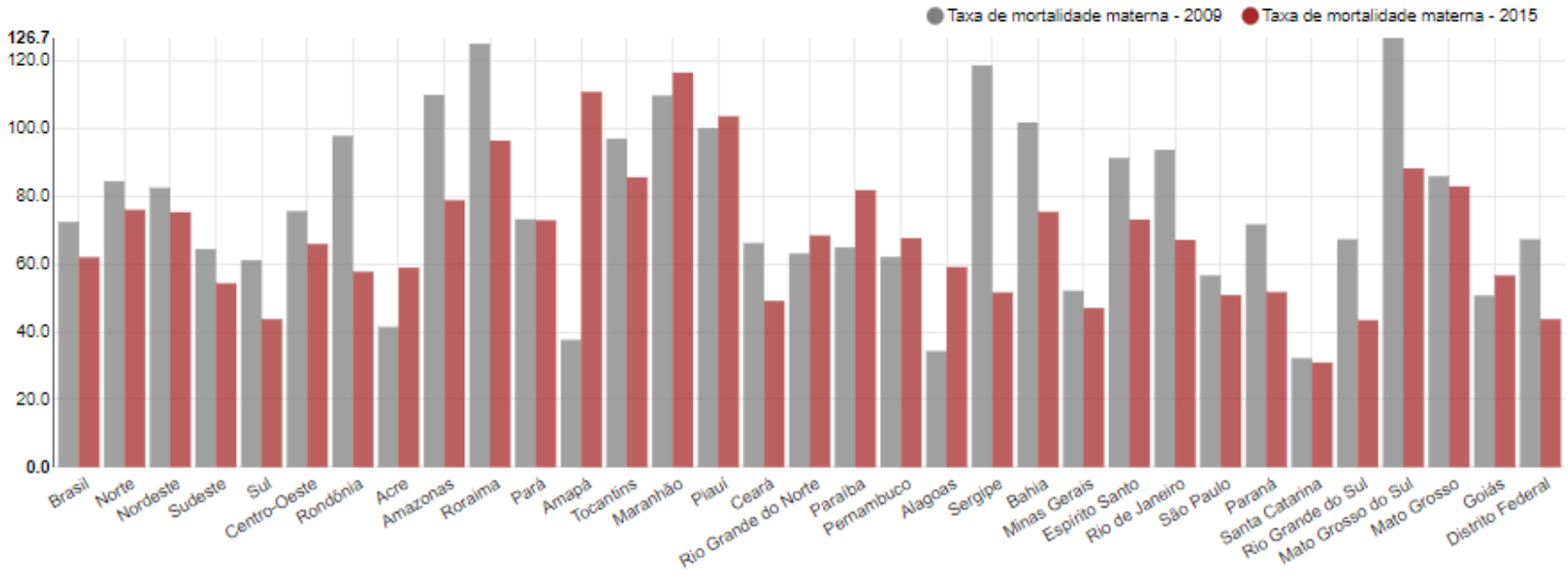
4. Exercícios

4. Exercícios

OBJETIVOS DE DESENVOLVIMENTO SUSTENTÁVEL

Objetivo 3 - Saúde e Bem-Estar

Indicador 3.1.1 - Razão de mortalidade materna



4. Exercícios

OBJETIVOS DE DESENVOLVIMENTO SUSTENTÁVEL

Indicador 3.1.1 - Razão de mortalidade materna

Taxa de mortalidade materna (Óbitos por 100.000 nascidos vivos)							
Brasil, Grande Região Unidades de Federação	2009	2010	2011	2012	2013	2014	2015
Brasil	72,4	68,9	61,8	59,3	62,1	63,8	62,0
Norte	84,4	78,4	74,7	73,2	83,8	93,6	76,0
Rondônia	97,8	83,2	58,3	64,1	94,2	87,7	57,7
Acre	41,4	48,8	33,7	53,9	58,6	46,7	58,9
Amazonas	109,9	119,5	84,6	85,0	81,6	120,8	78,8
Roraima	125,0	13,8	67,5	38,0	74,5	60,4	96,4
Pará	73,2	69,7	73,8	78,9	86,7	96,4	72,9
Amapá	37,6	26,8	53,3	54,1	119,7	66,0	110,8
Tocantins	97,0	76,8	112,6	53,3	61,2	60,1	85,6
Nordeste	82,5	83,3	77,9	72,7	82,4	77,9	75,3
Maranhão	109,7	133,7	109,8	96,1	123,8	100,9	116,5
Piauí	100,1	125,0	107,1	126,0	130,2	86,8	103,6
Ceará	66,2	70,6	68,4	69,4	74,5	65,3	49,1
Rio Grande do Norte	63,1	36,6	68,6	66,0	54,5	74,8	68,4
Paraíba	64,9	61,8	61,7	61,5	70,9	63,0	81,8
Pernambuco	62,1	61,9	57,5	56,0	64,3	63,4	67,6

4. Exercícios

2) Baseado nos dados do indicador ODS Brasil 3.1.1, defina uma struct que contenha o nome do estado, taxa de mortalidade materna de 2009 , taxa de mortalidade materna de 2015 e um índice que mostre o crescimento ou decréscimo entre essas duas taxas.

   Desenvolva um código em C, na qual seja declarado um vetor do tipo definido com a struct, seja preenchido com dados do indicador acima estabelecido e por último tenha seus dados ordenados e exibidos de forma crescente pelo índice de mortalidade materna de 2015.

   No processo de ordenação, use o método **Quick sort**.

4. Exercícios (opcionais)

3) Organizador de vagões
https://judge.beecrowd.com/pt/problems/view/1162

4) Spurs Rocks
https://judge.beecrowd.com/pt/problems/view/1303

5. Terceiro momento: síntese

- O Quick sort é o algoritmo de ordenação interna mais rápido que se conhece para uma variedade de situações. Baseia-se na solução de divisão e conquista.
- O Merge sort é um algoritmos que baseia-se na solução de divisão e conquista.