

ESTRUTURA DE DADOS I

Aula 04

Prof. Sérgio Luis Antonello


FHO - Fundação Hermínio Ometto

17/03/2025

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
 - 1.2. Recursividade
 - 1.3. Noções de complexidade computacional
 - 1.4. Conceitos e métodos de ordenação de dados
 - 1.5. Bubble sort, Insert sort e Select sort
 - 1.6. Quick sort e Merge sort
 - 1.7. 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)

- Quick sort
- Merge sort

■ Segundo momento

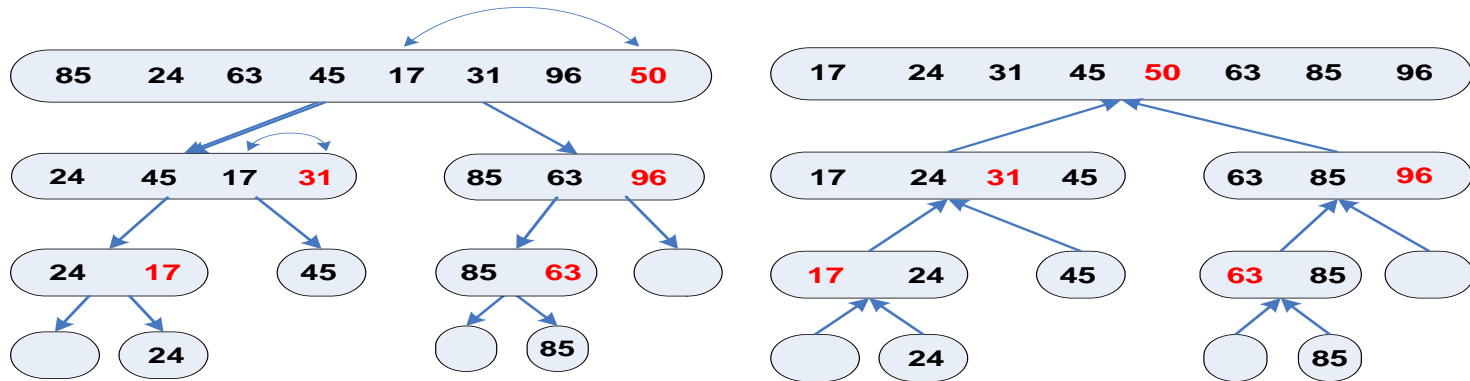
- Métodos de ordenação em memória primária
 - Shell sort
 - Radix sort

■ Terceiro momento (síntese)

- Retome pontos importantes da aula

1. Primeiro Momento: Revisão

■ QuickSort (divisões baseadas em pivô)



1. Primeiro Momento: Revisão

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 $i = \text{esquerda}$ E $j = \text{direita}$

enquanto $i \leq j$:

enquanto $\text{vetor}[i] < \text{pivô}$ E $i < \text{direita}$: incremente i

enquanto $\text{vetor}[j] > \text{pivô}$ E $j > \text{esquerda}$: decrémente j

se $i \leq j$:

troque $\text{vetor}[i]$ com $\text{vetor}[j]$

incremente i E decrémente j

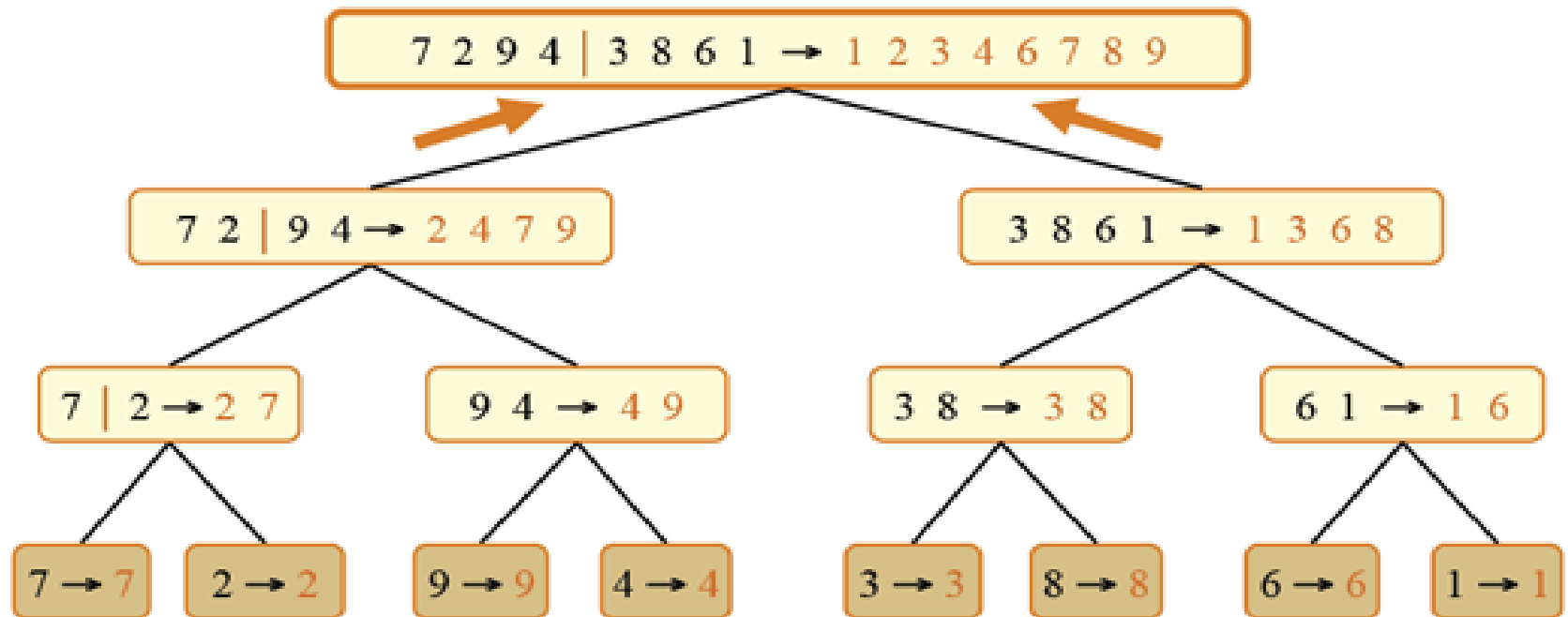
chame o procedimento recursivo:

se $(j > \text{esquerda})$: quickSort(vetor, esquerda, j)

se $(i < \text{direita})$: quickSort(vetor, i , direita)

1. Primeiro Momento: Revisão

- MergeSort (divisões ao meio - [animação](#))



1. Primeiro Momento: Revisão

`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)`

1. Primeiro Momento: Revisão

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 $i=0$, $j=\text{meio}$ e $k=0$

enquanto $i < \text{meio}$ e $j < N$, faça:

se $\text{vetor}[i] \leq \text{vetor}[j]$: $\text{vetorAux}[k++] = \text{vetor}[i++]$

senão: $\text{vetorAux}[k++] = \text{vetor}[j++]$

se $i == \text{meio}$: $\text{vetorAux}[k++] = \text{vetor}[j++]$, enquanto $j < N$

senão: $\text{vetorAux}[k++] = \text{vetor}[i++]$, enquanto $i < \text{meio}$

copie vetorAux para vetor

1. Primeiro momento: revisão

**Corrigir os códigos das
tarefas 02 e 03.**

```

3
1 3 2
4
4 3 2 1

```

Optimal train swapping takes 6 swaps.

■ Bubble sort

1. Primeiro Momento: Revisão

■ BEE1162 – Organizador de vagões

3

1 3 (não troca)

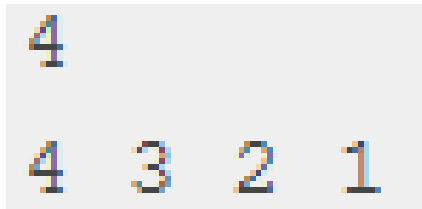
1 3 2

3 2 (troca)

1 2 3 – após primeira passagem pelo trem

1. Primeiro Momento: Revisão

■ BEE1162 – Organizador de vagões



A diagram of a train with four cars. The cars are numbered 4, 3, 2, and 1 from left to right. The number 4 is blue, 3 is green, 2 is yellow, and 1 is red. The train is shown on a light gray background.

4 3 (troca)

4 2 (troca)

4 1 (troca)

3 2 1 4 – após primeira passagem pelo trem

3 2 (troca)

3 1 (troca)

2 1 3 4 – após segunda passagem pelo trem

2 1 (troca)

1 2 3 4 – após última passagem pelo trem

1. Primeiro momento: revisão

**Opcional
BEE1162 – organizador de
vagões.**

2. Segundo momento

01	02	03	04	05	06	07	16	09	10	11	12	13	14	15	08
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

01	02	03	04	05	06	15	16	09	10	11	12	13	14	07	08
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

01	02	03	04	05	13	15	16	09	10	11	12	06	14	07	08
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

01	02	03	04	11	13	15	16	09	10	05	12	06	14	07	08
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

01	02	03	16	11	13	15	08	09	10	05	12	06	14	07	04
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

01	02	15	16	11	13	14	08	09	10	05	12	06	03	07	04
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Shell Sort
&
Radix Sort

2. Segundo momento: **motivação**

- A Maratona FHO de Programação é uma competição entre times.
- Cada equipe deve submeter um a um os programas que consideram corretos.
- O critério primordial de classificação é o número de problemas resolvidos com status “Yes” de aceito.
- Em caso de empate por número de problemas resolvidos, será mais bem classificada aquela com menor Tempo Acumulado (TA).
- O TA é calculado da seguinte forma: cada vez que a equipe apresentar um programa com status Y será somado ao seu TA o intervalo de tempo transcorrido desde o início da competição até o momento em que o programa foi submetido, mais a punição por julgamentos incorretos (20 minutos por cada submissão incorreta para aquele problema, ou seja, problemas resolvidos com status “N” de não aceito).
- Na última competição o professor Antero estava com problema no que diz respeito a montagem da classificação final da competição. Como poderíamos tê-lo ajudado?

3. Métodos de ordenação interna

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

Métodos simples

- Bubble sort
- Insert sort
- Select sort

Métodos eficientes

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

- Animação dos métodos simples de ordenação:

<http://visualgo.net/sorting>

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

4. Shell sort

- Proposto por Shell em 1959.
- É usado juntamente com outros tipos de ordenação, preferencialmente o Insert sort.
- Permite trocas de registros distantes um do outro, dentro da estrutura.
- É uma ótima opção para estruturas de tamanho moderado, sua implementação é simples e requer uma quantidade de código pequena.
- O tempo de execução do algoritmo é sensível à ordem inicial dos dados.

4. Shell sort

- Classifica subestruturas separadas da estrutura original.
- Essas subestruturas contêm todo k-ésimo elemento do arquivo original. O valor de k é chamado incremento.
- Exemplo: Se $k=5$, cinco subestruturas lógicas, cada um contendo um quinto dos elementos da estrutura original, são classificadas.
 - subestrutura 1 -> $x[0]$ $x[5]$ $x[10]$...
 - subestrutura 2 -> $x[1]$ $x[6]$ $x[11]$...
 - subestrutura 3 -> $x[2]$ $x[7]$ $x[12]$...
 - subestrutura 4 -> $x[3]$ $x[8]$ $x[13]$...
 - subestrutura 5 -> $x[4]$ $x[9]$ $x[14]$...

4. Shell sort

- O processo anterior se repete em:
 - Depois que as primeiras k estruturas estiverem classificadas (geralmente por inserção simples), será escolhido um NOVO VALOR MENOR de k e o arquivo será novamente particionado em novos conjuntos de subestruturas.
 - Cada um dessas subestruturas maiores será classificada e o processo se repetirá novamente com um número ainda menor de k .
 - Em algum momento, o valor de k será definido com 1, de modo que a subestrutura consistindo na estrutura inteira será classificada.
 - Uma sequência decrescente de incrementos é determinada no início do processo inteiro.

4. Shell sort

Início	25	57	48	37	12	92	86	33
K = 5	25	57	48	37	12	92	86	33
Resultado	25	57	33	37	12	92	86	48
K = 3	25	57	33	37	12	92	86	48
Resultado	25	12	33	37	48	92	86	57
K = 1	25	12	33	37	48	92	86	57
Resultado	12	25	33	37	48	57	86	92

4. Shell sort

■ Animação

<https://www.youtube.com/watch?v=CmPA7zE8mx0>

<https://www.youtube.com/watch?v=M9YCh-ZeC7Y>

<https://www.youtube.com/watch?v=qzXAVXddcPU>

4. Shell sort

função shell Sort (vetor, N):

define o vetor de incrementos

para cada incremento k do vetor de incrementos, faça:

para i = k; enquanto i < N; i++:

aux = vetor[i]

para j=i-k; enquanto j>=0 E vetor[j]>aux; j-=k:

vetor[j+k] = vetor[j]

vetor[j+k] = aux;

fim do para

fim do para

5. Radix sort

- Proposto por Herman Hollerith em 1887, quando construía uma *máquina* para processar os resultados do censo americano de 1890.



5. Radix sort

- Ordena o vetor/lista de maneira lexicográfica (ou seja, baseia-se nos valores dos dígitos de cada elemento);
- Duas possíveis versões: LSD e MSD
 - LSD (*Least Significant Digit*): inicia a ordenação pelo **dígito menos significativo**, avançando até o dígito mais significativo;
 - MSD (*Most Significant Digit*): inicia a ordenação pelo **dígito mais significativo**, recuando até o dígito menos significativo.

mais significativo → 4 1 3 2 ← menos significativo
(3) (2) (1) (0)

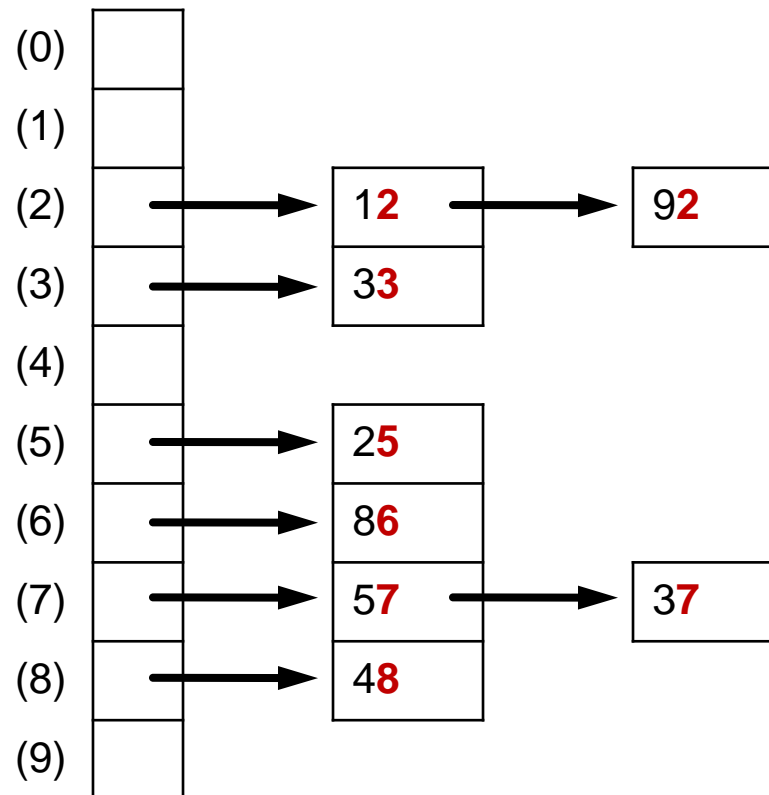
5. Radix sort

- Versão LSD (inicia pelo dígito menos significativo)

Vetor original: [25, 57, 48, 37, 12, 92, 86, 33]

PASSO 1

dígito d na
posição 0 dos
números



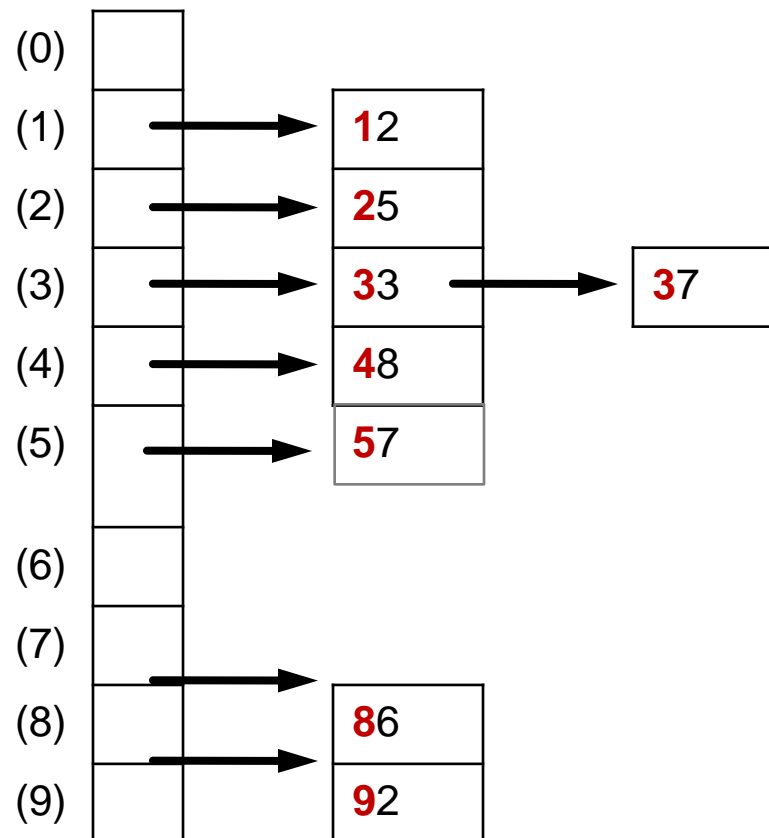
5. Radix sort

- Versão LSD (inicia pelo dígito menos significativo)

Vetor após Passo 1: [12, 92, 33, 25, 86, 57, 37, 48]

PASSO 2

dígito d na
posição 1 dos
números



5. Radix sort

- Versão LSD (inicia pelo dígito menos significativo)

Vetor após Passo 2: [12, 25, 33, 37, 48, 57, 86, 92]

► VETOR ORDENADO!

5. Radix sort

■ Animação

https://www.youtube.com/watch?time_continue=18&v=ibtN8rY7V5k

https://www.youtube.com/watch?v=xuU-DS_5Z4g

5. Radix sort

função radixSort (vetor, N):

cria 10 filas vazias (indexadas de 0 a 9)

para d = posição do dígito menos significativo, até d =
posição do dígito mais significativo, faça:

para cada elemento i do vetor, faça:

aux = elemento i;

j = d-ésimo dígito do elemento i

insere aux na fila j

fim do para

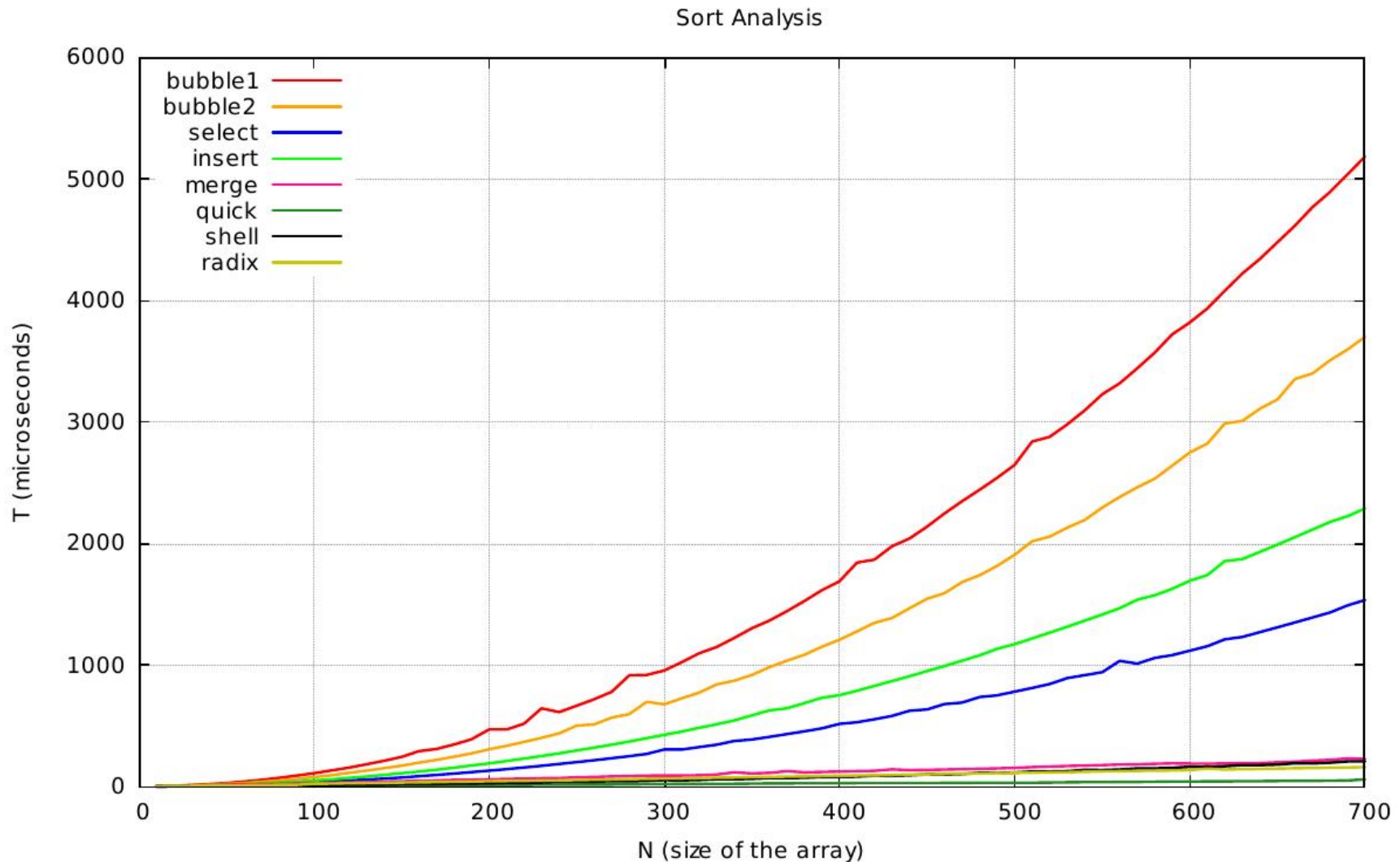
remonta o vetor retirando os elementos das filas
(na ordem de 0 a 9)

fim do para

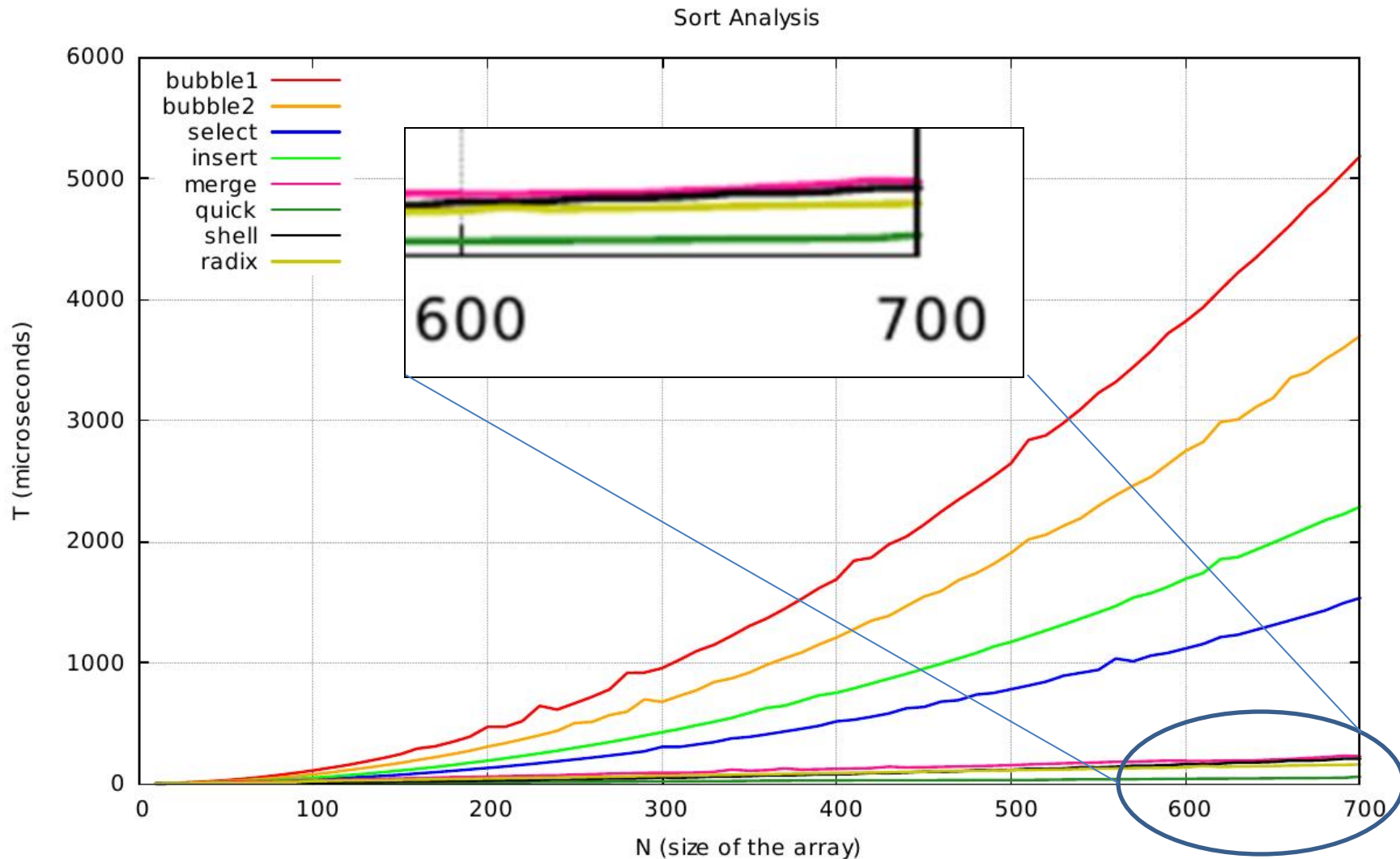
6. Complexidade dos algoritmos de ordenação estudados

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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Shell sort	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$
Radix sort	$O(n)$	$O(n \cdot m)$	$O(n \log n)$

6. Complexidade dos algoritmos de ordenação estudados



6. Complexidade dos algoritmos de ordenação estudados



7. Exercícios

**Vamos
Praticar!**



7. Exercícios

1) Pares e ímpares

Usando o método de ordenação que desejar, desenvolver a solução para o problema BEE1259. Incluir como comentários no início do programa o seu código da plataforma beecrowd, seu nome e a linha de status accepted.

<https://judge.beecrowd.com/pt/problems/view/1259>

7. Exercícios

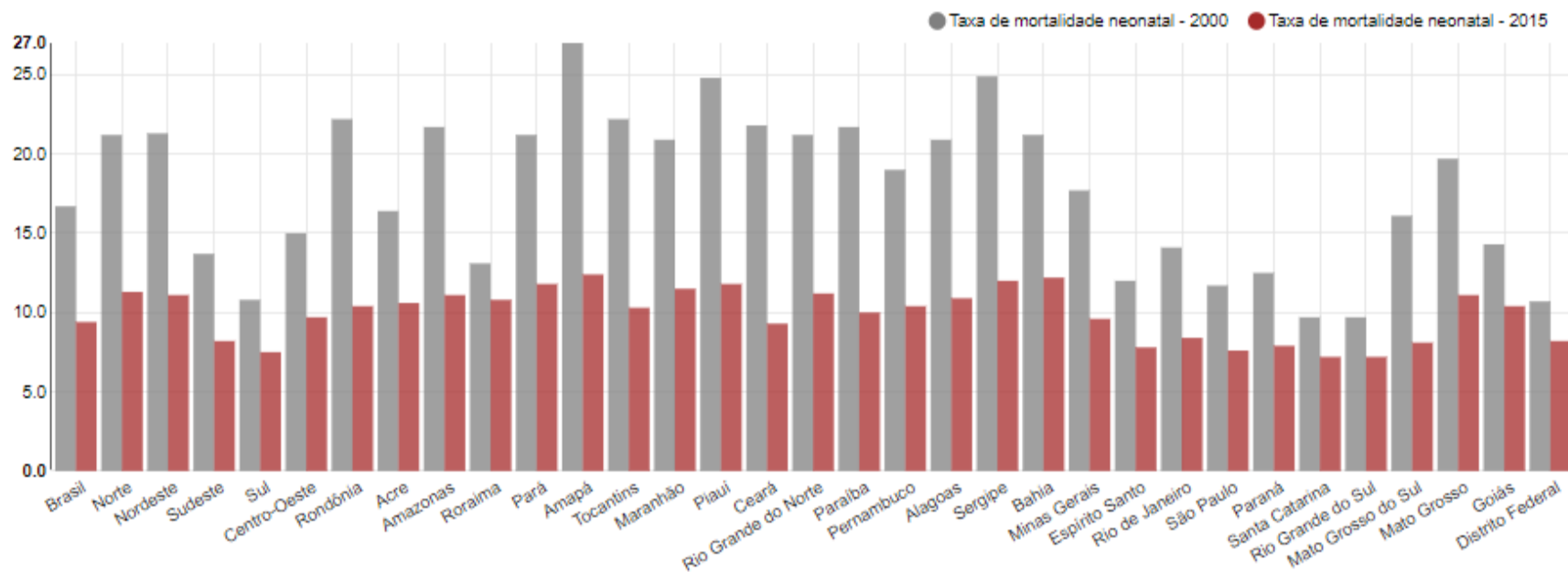
OBJETIVOS DE DESENVOLVIMENTO SUSTENTÁVEL



7. Exercícios



Indicador 3.2.2 - Taxa de mortalidade neonatal



7. Exercícios

Para explorar os dados disponíveis [clique aqui](#).

Indicador 3.2.2 - Taxa de mortalidade neonatal



Taxa de mortalidade neonatal (Óbitos por 1000 nascidos vivos)

Brasil, Grande Região Unidades de Federação	2009	2010	2011	2012	2013	2014	2015
Brasil	11,5	11,1	10,6	10,3	9,9	9,5	9,4
Norte	14,9	14,0	13,4	12,6	11,8	11,8	11,3
Rondônia	13,7	13,2	10,9	10,1	9,2	9,5	10,4
Acre	12,6	12,9	10,8	9,7	10,9	10,3	10,6
Amazonas	13,8	12,4	13,1	12,3	11,3	10,7	11,1
Roraima	10,4	10,4	9,9	10,3	11,4	11,3	10,8
Pará	16,1	15,2	14,8	13,6	12,5	12,2	11,8
Amapá	21,3	17,9	17,0	16,8	16,2	16,2	12,4
Tocantins	12,7	12,2	10,7	10,7	10,7	10,2	10,3
Nordeste	14,0	13,6	12,7	12,4	11,7	11,3	11,1
Maranhão	15,8	15,1	13,9	13,4	11,9	11,7	11,5
Piauí	16,1	15,1	15,4	14,4	13,3	13,1	11,8
Ceará	12,3	11,6	10,4	10,5	10,1	9,6	9,3

7. Exercícios

- 2) Baseado nos dados do indicador [ODS Brasil 3.2.2 - Taxa de mortalidade neonatal](#), defina uma struct que contenha o nome do estado, taxa de mortalidade neonatal de 2009, taxa de mortalidade neonatal 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 **decrecente** pelo índice de mortalidade materna de 2015.

No processo de ordenação, use o método **Shell Sort**.

8. Terceiro momento: síntese

- Dentre as características do Shell sort ressalta-se a que o tempo de execução do algoritmo é sensível à ordem inicial dos dados.
- O Radix sort usa rotinas que se baseiam nos valores dos dígitos de cada elemento armazenado na estrutura.