



Algoritmos de ordenação

– por substituição e inserção –

Prof. Dieisson Martinelli
dieisson.martinelli@udesc.br

Programa

- Introdução
- Ordenação por substituição
 - Bubble Sort
 - Quick Sort
- Ordenação por inserção
 - Insertion Sort
 - Shell Sort

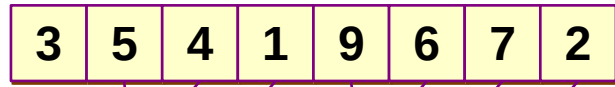
Introdução

- Ordenar é o **processo de reorganizar** um **conjunto de elementos** em uma **ordem ascendente** ou **descendente**
- A ordenação **visa facilitar** a posterior **recuperação de itens** do **conjunto ordenado**. Por exemplo, p/ um algoritmo de busca binária
- Considerando a **capacidade de memória** do computador, os algoritmos de ordenação podem ser classificados em:
 - **Ordenação interna**: quando os dados a serem ordenados estão na memória principal (memória RAM)
 - **Ordenação externa**: quando os dados não cabem na memória e necessitam de armazenamento em memória auxiliar (por exemplo, um disco rígido)
- Nesta aula serão apresentados alguns métodos de ordenação interna

Bubble Sort

- ▶ O **Bubble Sort**, ou método bolha, ordena comparando cada par adjacente de itens em uma lista e, se necessário, faz a **substituição** (troca) dos itens, repetindo o repasse da lista até que não hajam mais substituições
- Cada vez que o algoritmo **percorre os N itens** de uma lista, **todos eles são comparados** com o seu próximo para verificar se estão na ordem desejada
- Os pares não ordenados **são trocados de posição**

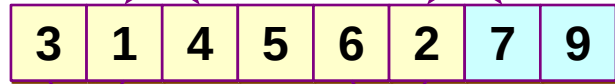
Exemplo de Bubble Sort



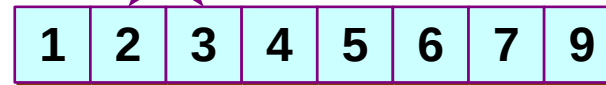
vetor inicial (desordenado)



1ª iteração



última iteração



vetor final (ordenado)

Funcionamento do Bubble Sort

- Na **primeira iteração**, é encontrado o **maior elemento** e o mesmo é **deslocado** até a **última posição** (esse exemplo de *bubble sort* também desloca elementos intermediários)




- Na **segunda iteração**, é encontrado o **segundo maior elemento** e o mesmo é **deslocado** até a **penúltima posição**



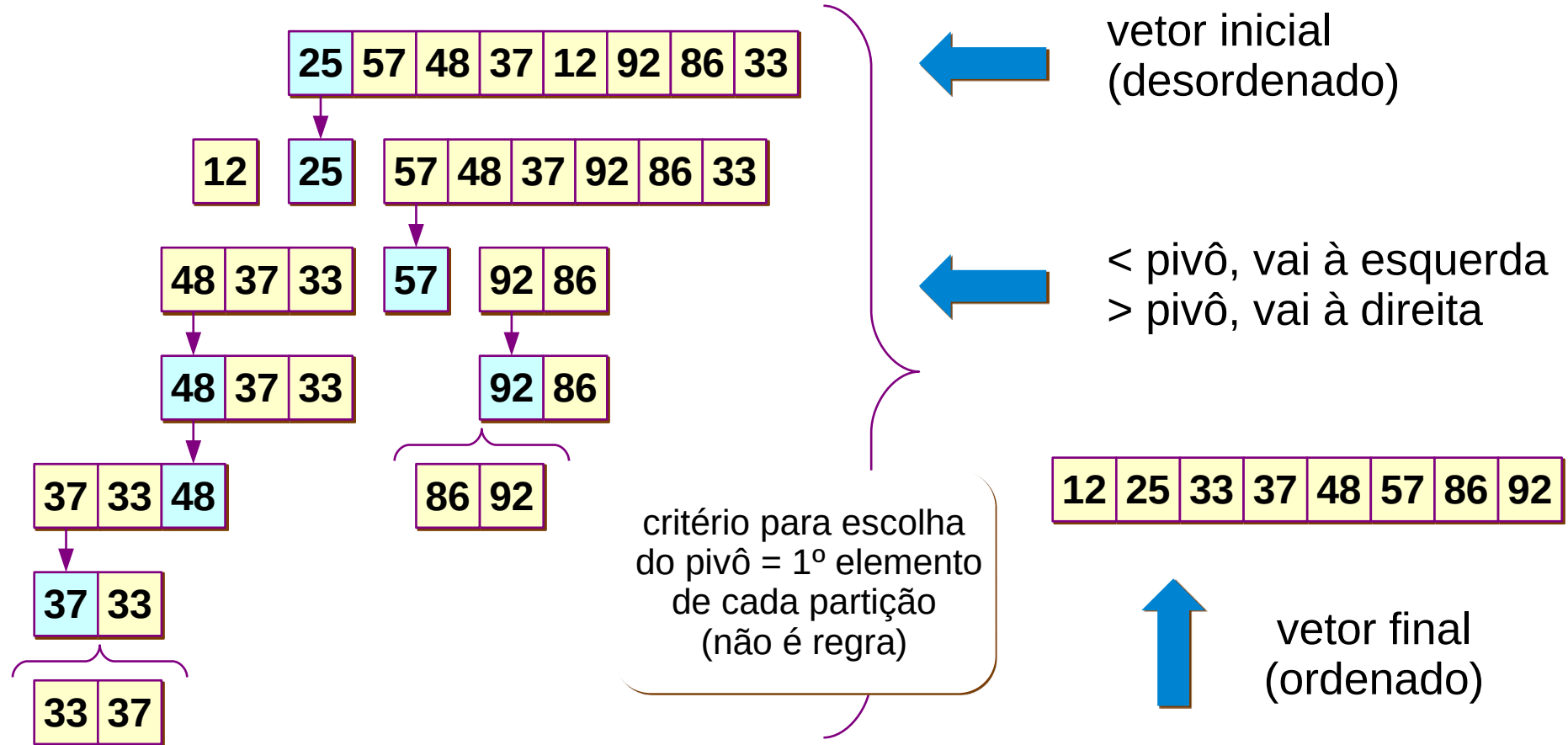
- E continua até que todos os elementos sejam ordenados



Quick Sort

- 
- O **Quick Sort** utiliza a estratégia de “**dividir para conquistar**”, e é um dos **algoritmos mais populares** baseado em **ordenação por substituição** (ou troca)
- **Características**: **escolher um dos elementos** do vetor como **pivô**, em torno do qual os outros elementos serão **reorganizados em partições** (e cada partição também é subdividida)
 - Tudo o que for **menor que o pivô** é **movido à esquerda** (partição à esquerda), e tudo o que for **maior do que o pivô** é **movido à direita** (partição à direita)
 - Depois, **cada partição é recursivamente montada**

Exemplo de Quick Sort



Funcionamento do Quick Sort

- Na **1ª iteração**, é escolhido o **1º elemento como pivô**

25	57	48	37	12	92	86	33
----	----	----	----	----	----	----	----

- Todos os **menores** que o pivô vão para a **partição da esquerda**, todos os **maiores** para a **partição da direita**

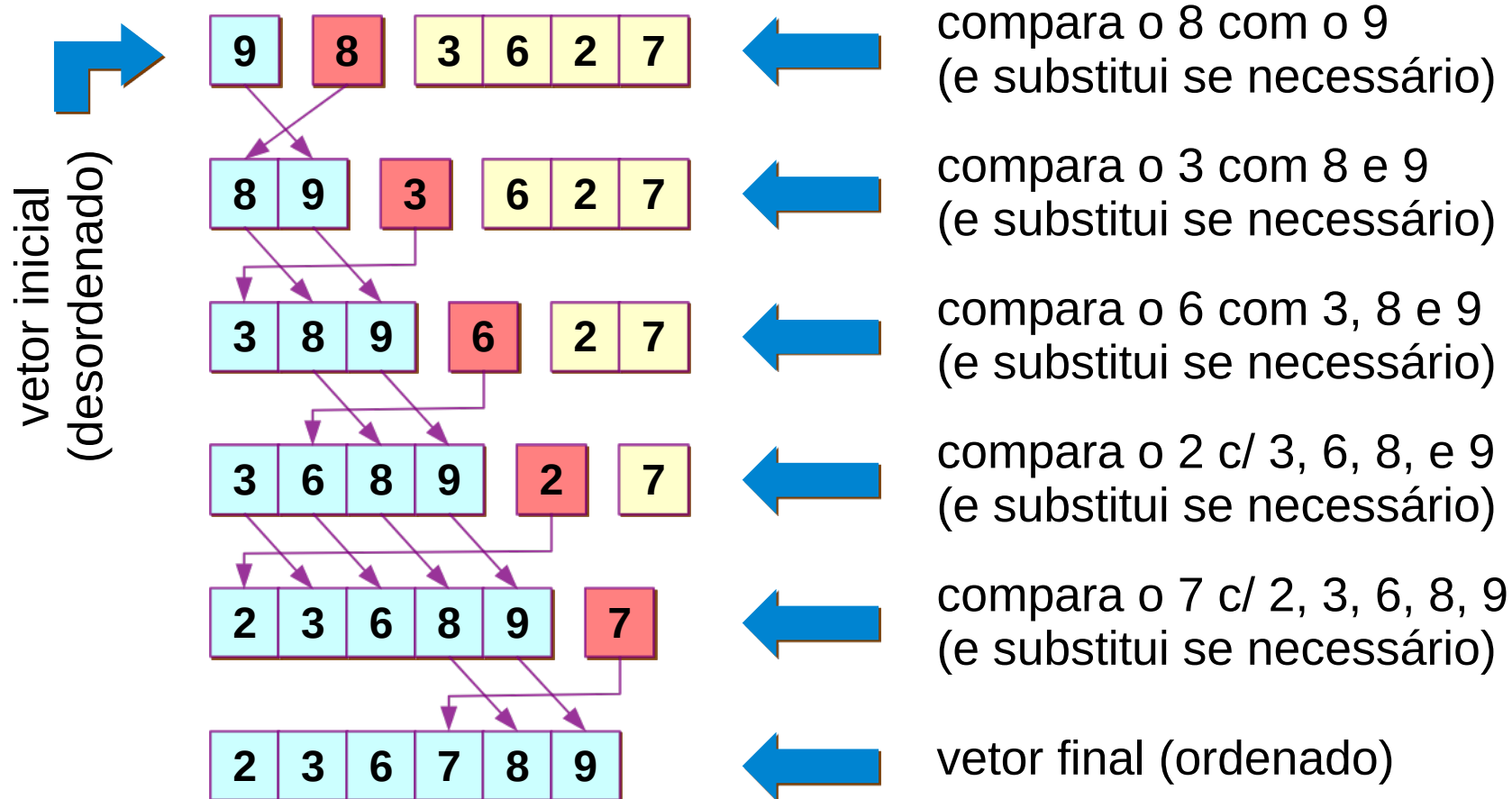
12	25	57	48	37	92	86	33
----	----	----	----	----	----	----	----

- De cada partição da iteração anterior**, também é escolhido o **1º elemento como pivô**, subdividindo em novas partições
- E continua até que todos os elementos sejam ordenados **montando recursivamente todas as partições ordenadas**

Insertion Sort

- ▶ O **Insertion Sort** é um simples algoritmo de **ordenação por inserção**, eficiente quando aplicado a um pequeno número de elementos
- **Características**: percorrer um vetor de elementos **da esquerda para a direita** e, à medida que avança, **deixar os elementos mais à esquerda ordenados**
- Pode ser cerca de duas vezes **mais rápido** que o **Bubble Sort** e, geralmente, é utilizado como estágio final de **métodos mais complexos**, como o **Quick Sort**

Exemplo de Insertion Sort



Funcionamento do Insertion Sort

- A **partir do 2º elemento** do conjunto de dados, buscar onde ele **deve ficar no subvetor à esquerda**, de modo que fique **ordenado** (mas não é a posição definitiva)



- Após ordenar o subvetor à esquerda, **avançar 1 posição no subvetor à direita** (não ordenado) e repetir passo anterior
- O processo de ordenação termina quando **todos os elementos a partir do 2º elemento** forem visitados e **inseridos ordenadamente no subvetor à esquerda**

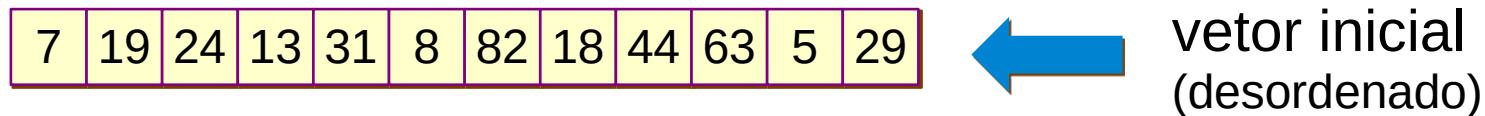
Shell Sort

- ▶ O **Shell Sort** é um método baseado na **ordenação por inserção**, criado em 1959 por Donald L. Shell
 - É **eficiente para *arrays* de tamanho médio**, porém não é tão rápido quanto o Quick Sort na ordenação de grandes volumes de dados
 - **Características:** ordenar uma sequência $[a_1, a_2, \dots, a_n]$ de **n elementos** pela ordenação de sucessivas subsequências cujos **valores** estão **misturados na sequência original**
 - As **subsequências a serem ordenadas** são determinadas por um conjunto de parâmetros denominados **incrementos**

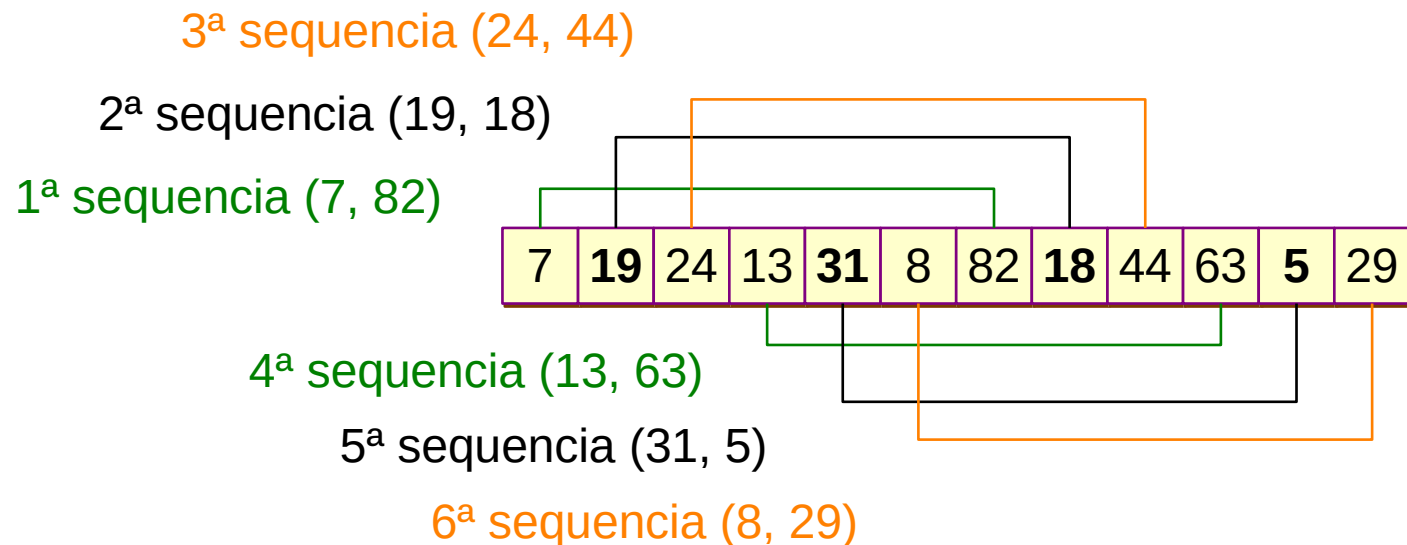
Shell Sort

- Assim, dado um **conjunto de incrementos** $[h_t, h_{t-1}, \dots, h_1]$, é obrigatório que $h_1 = 1$. Por exemplo: $[6, 4, 3, 2, 1]$
- ▶ Principais passos:
 - 1º passo: começando por h_t , **divide-se a sequencia original** em h_t **subsequencias** com n/h_t **elementos**
 - 2º passo: **ordenar as subsequencias** com qualquer outro algoritmo de ordenação (normalmente o Insertion Sort)
 - 3º passo: **repete-se os passos 1 e 2** para $h_{t-1}, h_{t-2}, \dots, h_1$ quando então a sequencia ficará ordenada

Funcionamento do Shell Sort



- 1ª iteração: **separar** o vetor em **6 subsequências de tamanho 2** **cada**, depois ordená-las com **inserção direta** (Insertion Sort)



Incremento = 6,
 $12 / 6 = \text{tamanho } 2$

Funcionamento do Shell Sort

7	18	24	13	5	8	82	19	44	63	31	29
---	----	----	----	---	---	----	----	----	----	----	----

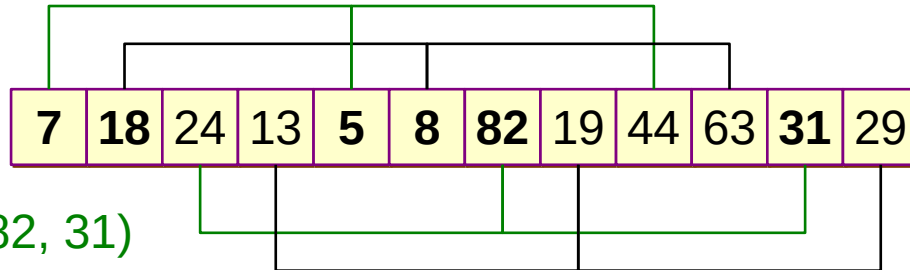


vetor após
a 1ª. iteração

- 2ª iteração: com as **subsequencias** ordenadas do **passo anterior**, **separar** o vetor em **4 subsequencias** de **tamanho 3** cada

2ª sequencia (18, 8, 63)

1ª sequencia (7, 5, 44)



3ª sequencia (24, 82, 31)

4ª sequencia (13, 19, 29)

Incremento = 4,
 $12 / 4 = \text{tamanho } 3$

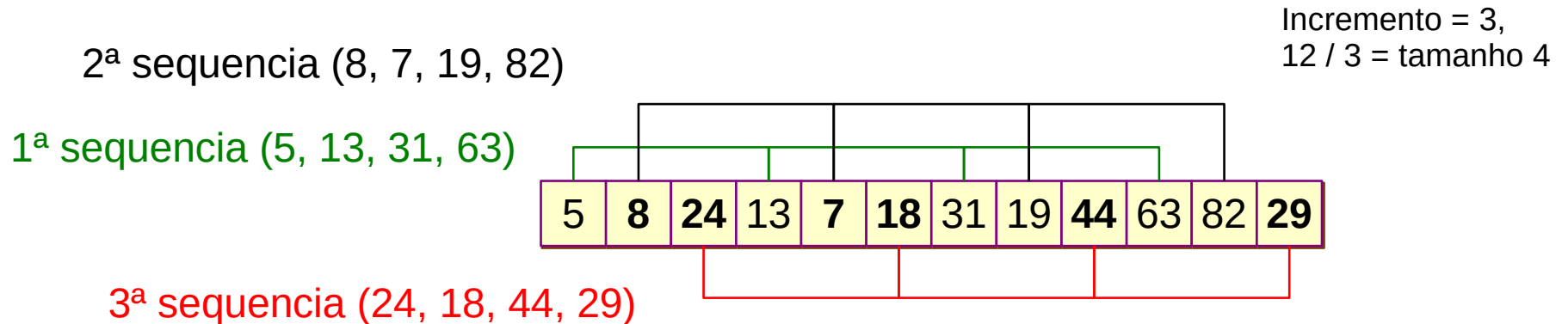
Funcionamento do Shell Sort

5	8	24	13	7	18	31	19	44	63	82	29
---	---	----	----	---	----	----	----	----	----	----	----



vetor após
a 2ª. iteração

- 3ª iteração: com as **subsequencias** ordenadas do **passo anterior**, **separar** o vetor em **3 subsequencias** de **tamanho 4** cada



Funcionamento do Shell Sort

5	7	18	13	8	24	31	19	29	63	82	44
---	---	----	----	---	----	----	----	----	----	----	----



vetor após
a 3ª. iteração

- 4ª iteração: com as **subsequencias** ordenadas do **passo anterior**, **separar** o vetor em **2 subsequencias** de **tamanho 6** cada

1ª sequencia (5, 18, 8, 31, 29, 82)

5	7	18	13	8	24	31	19	29	63	82	44

2ª sequencia (7, 13, 24, 19, 63, 44)

--	--	--	--	--	--	--	--	--	--	--	--

Incremento = 2,
 $12 / 2 = \text{tamanho } 6$

Funcionamento do Shell Sort

5	7	8	13	18	19	29	24	31	44	82	63
---	---	---	----	----	----	----	----	----	----	----	----



vetor após
a 4ª. iteração

- 5ª iteração: com as **subsequencias** ordenadas do **passo anterior**, **separar** o vetor em **1 subsequencia** de **tamanho 12**

sequencia (5, 7, 8, 13, 18, 19, 29, 24, 31, 44, 82, 63)

5	7	8	13	18	19	29	24	31	44	82	63

Incremento = 1,
 $12 / 1 = \text{tamanho } 12$

5	7	8	13	18	19	24	29	31	44	63	82
---	---	---	----	----	----	----	----	----	----	----	----



vetor final
(ordenado)