



Algoritmos de ordenação

– por intercalação e seleção –

Prof. Dieisson Martinelli
dieisson.martinelli@udesc.br


Programa

- Introdução
- Ordenação por intercalação
 - Merge Sort
- Ordenação por seleção
 - Selection Sort
 - Heap Sort

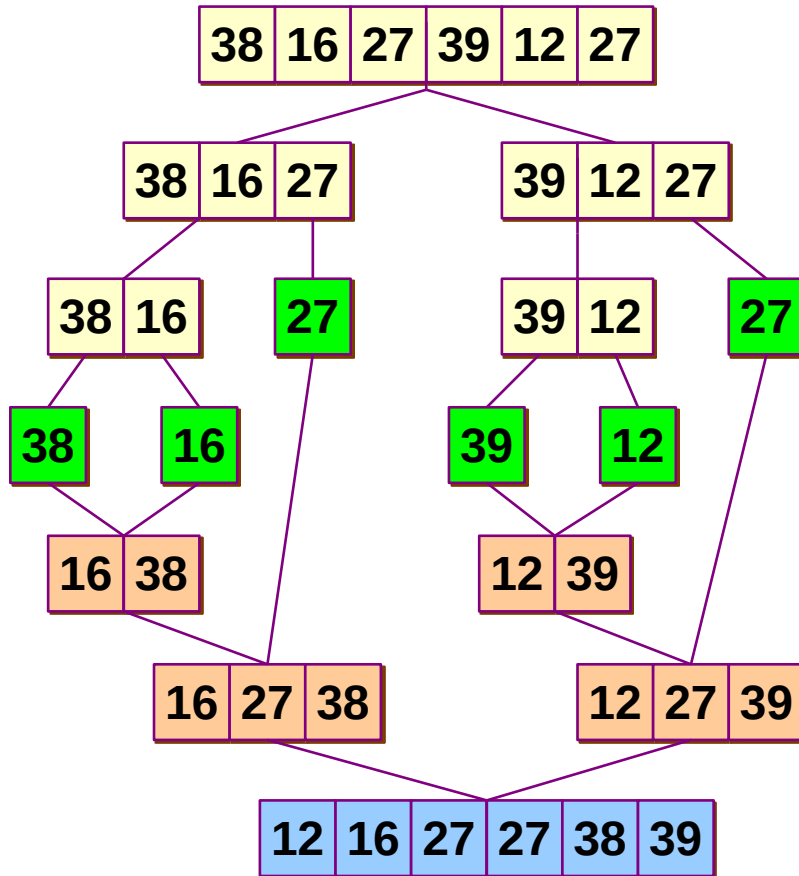
Introdução

- Alguns **métodos de ordenação** procuram **dividir** um **conjunto desordenado de dados** em **pequenas frações**, **ordenando** e **intercalando** os resultados
- Outros métodos procuram **selecionar o maior** (ou **o menor**) **elemento**, para colocá-lo em sua **posição correta** dentro da futura **lista ordenada**

Merge Sort

-  O **Merge Sort**, ou ordenação por **mistura** (ou **intercalação**), é um exemplo de algoritmo de ordenação do tipo “**dividir para conquistar**”
- **Características**: pode ser implementado de **duas formas** diferentes
 - **1ª forma**: forma **dividir** o **vetor inicial** em **dois** de **tamanho igual** à **metade** do **vetor original**, ou dividir o vetor em dois escolhendo os elementos **aleatoriamente**
 - **2ª forma**: forma consiste em fazer um **vetor com os elementos de posição ímpar** e outro com os **elementos pares**

Exemplo de Merge Sort



← vetor inicial (desordenado)

← 1ª subdivisão recursiva

← 2ª subdivisão recursiva

← 3ª subdivisão recursiva

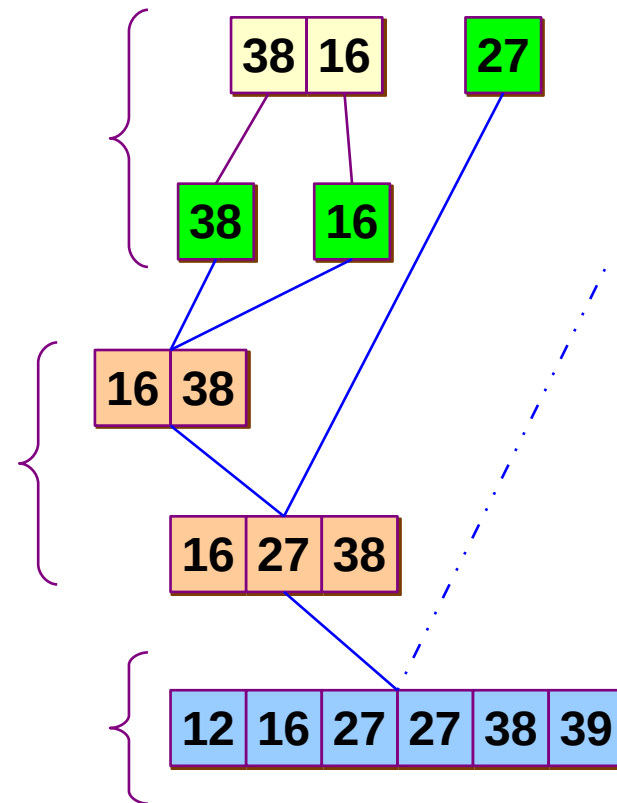
← 1ª fusão ordenada

← 2ª fusão ordenada


← vetor final (ordenado)

Funcionamento do Merge Sort

- A ideia básica do **Merge Sort** é a seguinte:
 - **Dividir** os dados em **subseqüências menores**, em pares, **recursivamente** até ficarem apenas em **duas partes**
 - **Ordenar** todas as duas **metades recursivamente**
 - Juntar as duas metades em um único conjunto já classificado (ordenado)



Selection Sort

- 
- Selection Sort** ou ordenação por **seleção** é um algoritmo que opera passando sempre o **menor valor do vetor** para a **primeira posição** (ou o **maior valor** dependendo da ordem requerida)
- Depois o de **segundo menor valor** para a **segunda posição**, e assim é feito **sucessivamente** para os **demaís valores**
 - Ao final de todas as iterações o vetor se encontrará em ordem **crescente** ou **decrescente**

Exemplo de Selection Sort

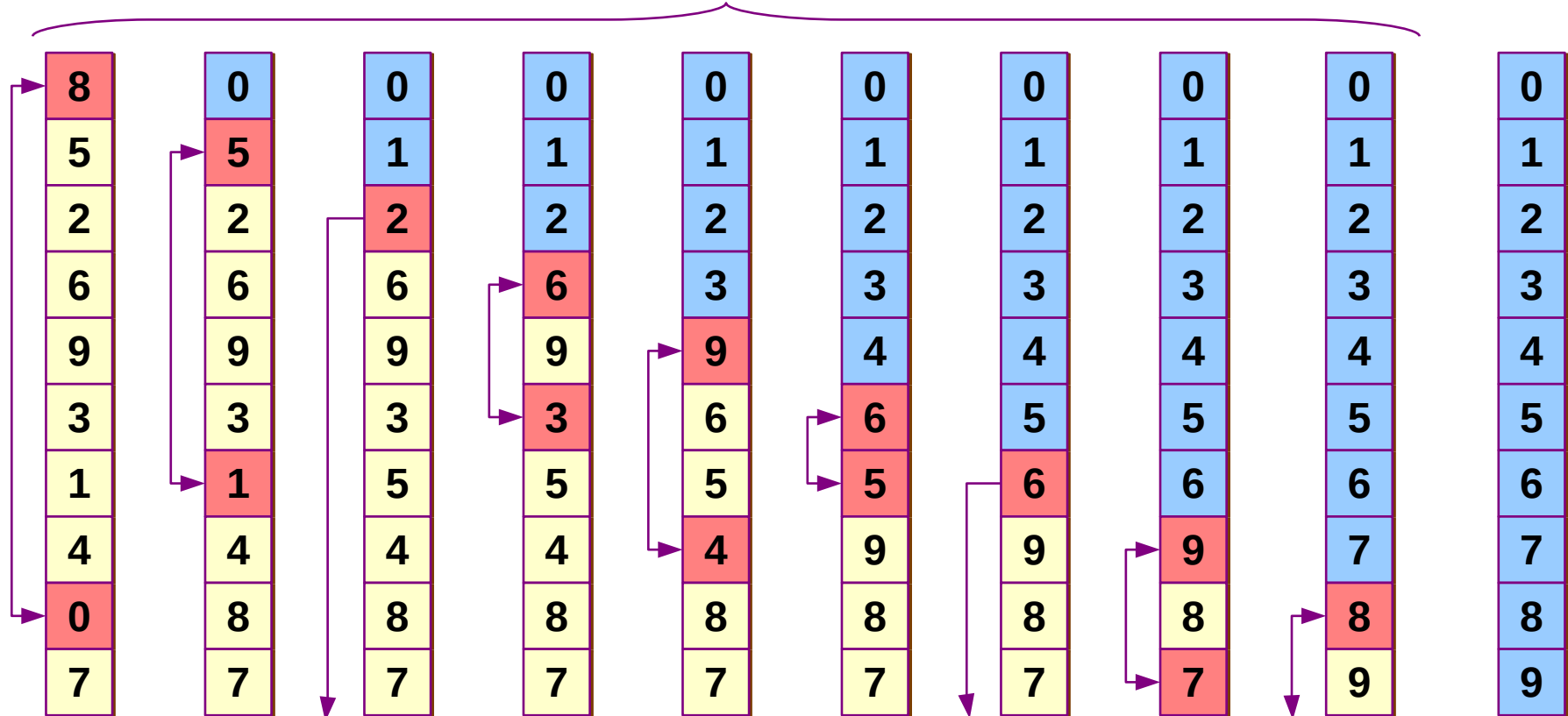
desordenado



iterações



ordenado



Funcionamento do Selection Sort

- Iniciar achando o **menor elemento** e **trocar** com o **primeiro elemento** do vetor

- Assim parte do vetor está agora ordenado

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

- Achar o **menor elemento** da parte **não ordenada** e **trocar** com o **primeiro elemento** da parte não ordenada

- Assim a parte ordenada do vetor **vai aumentando**

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

- O algoritmo para quando a **parte não ordenada** tem **apenas um número**, sendo este o **maior elemento**

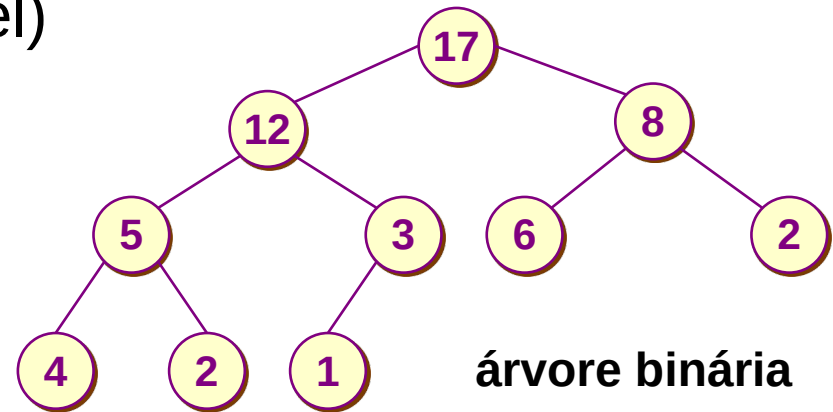
Heap Sort

► Foi **desenvolvido** em **1964** por **Robert Floyd** e faz parte da família dos algoritmos de **ordenação por seleção**

- **Características:** utiliza uma **estrutura de dados** chamada **heap**, para **ordenar** os elementos **a medida que os insere**
- A **heap** pode ser **representada** como uma **árvore binária** (completa, exceto no último nível) ou como um **vetor**



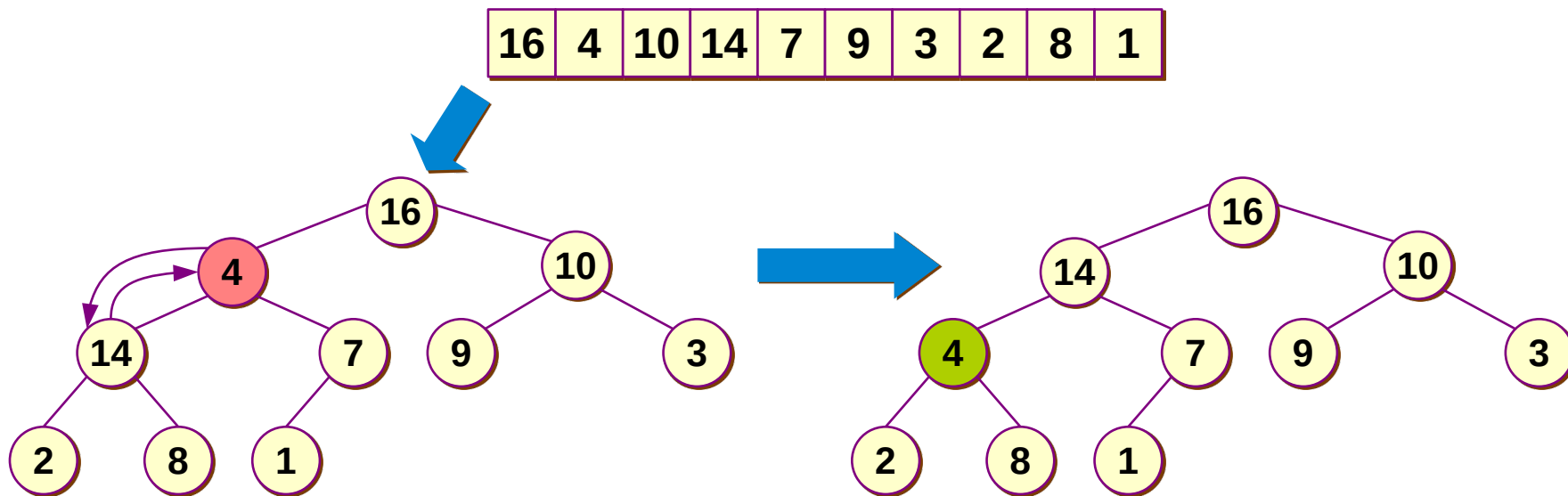
vetor



árvore binária

Funcionamento do Heap Sort

- O algoritmo de ordenação utiliza três funções: **Heapify**, **Buildheap** e **Heapsort**
 - **Heapify**: verifica se existem **elementos filhos** maiores que o **elemento pai** e reorganiza

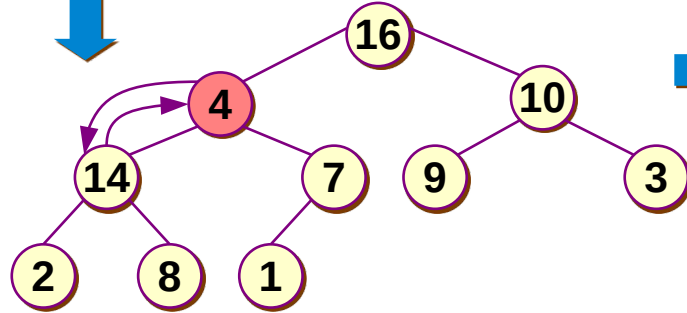


Funcionamento do Heap Sort

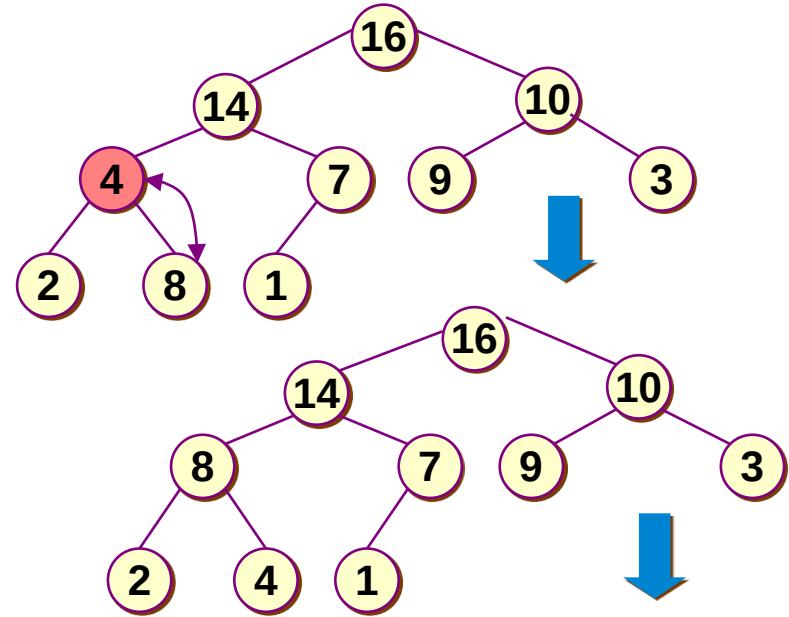
- **Buildheap**: utiliza a função **heapify** de forma **bottom-up** (de baixo para cima) para transformar um **vetor** $V[a_1, a_2, a_3, \dots, a_n]$ em uma **heap** com n elementos

vetor
inicial

16	4	10	14	7	9	3	2	8	1
----	---	----	----	---	---	---	---	---	---



Todo nó deve ter valor **maior** ou **igual** com relação aos seus filhos

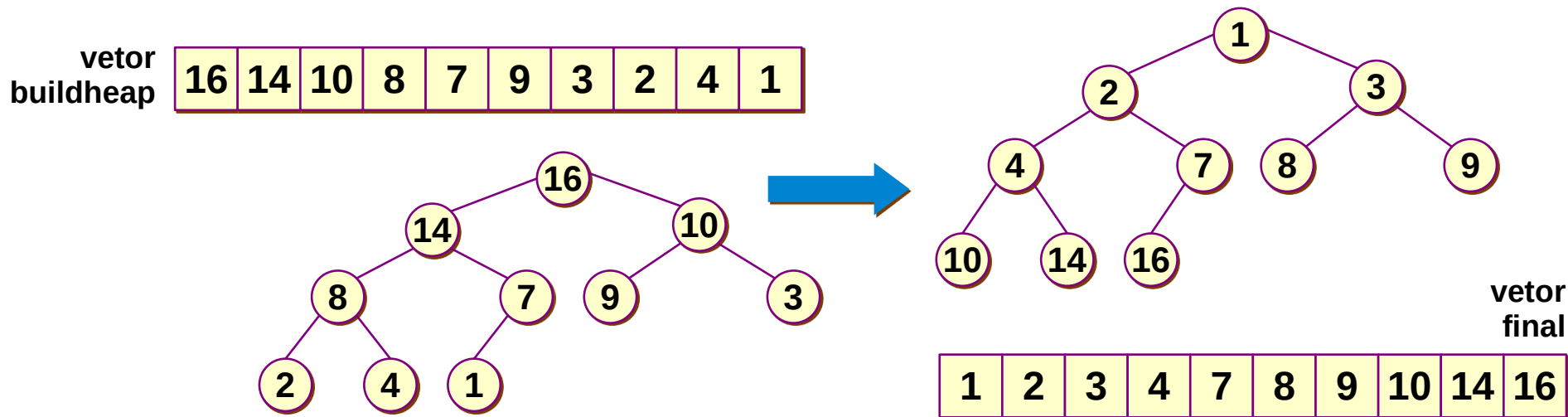


vetor
buildheap

16	14	10	8	7	9	3	2	4	1
----	----	----	---	---	---	---	---	---	---

Funcionamento do Heap Sort

- **Heapsort**: constrói um **heap** a partir de um **vetor de entrada** (**buildheap**), fazendo a **ordenação**



- Para **ordenação crescente**, deve ser construído o **heap máximo** (o maior elemento fica na raiz), e na **decrescente**, deve ser construído um **heap mínimo** (o menor elemento fica na raiz)