ALGORITMOS SIMPLES DE ORDENAÇÃO

DCE792 - AEDs II (Prática)

Atualizado em: 22 de outubro de 2025

Iago Carvalho

Departamento de Ciência da Computação

ORDENAÇÃO

Ordenação é uma das tarefas mais básicas existentes tem computação

- Colocar uma série de itens em ordem
  - Crescente
  - Decrescente
- Podemos trabalhar com qualquer tipo de item
  - Inteiros
  - Palavras
  - Datas
  - ...

Um algoritmo de ordenação é aquele que atua sobre um conjunto de elementos e os coloca em ordem

- Na maioria das vezes, este algoritmo atua sobre um vetor

ORDENAÇÃO

A ordenação é baseada em uma chave

- A chave de ordenação é o **campo** utilizado para comparação
  - Valor de uma variável
  - Campo *nome* em uma struct
  - Determinada coluna em uma tabela
  - ...

Comparando uma chave com outra é que sabemos se um determinado elemento está ou não a frente de outros no conjunto

CHAVE DE ORDENAÇÃO

Após a escolha da chave, devemos também definir a regra de ordenação

- Como dizer se uma chave é menor, igual ou maior que outra

Podemos utilizar ordem numérica, alfabética, alfa-numérica

A ordenação pode ser crescente ou decrescente de acordo com a chave e regra escolhida

Também podem existir regras mais complexas de ordenação

- Regras multi-chaves lexicográficas

ORDENAÇÃO INTERNA OU EXTERNA

Temos duas grandes classes de algoritmos de ordenação

Ordenação interna: assunto desta disciplina

- Conjunto de dados cabe na RAM
- Qualquer elemento pode ser imediatamente acessado

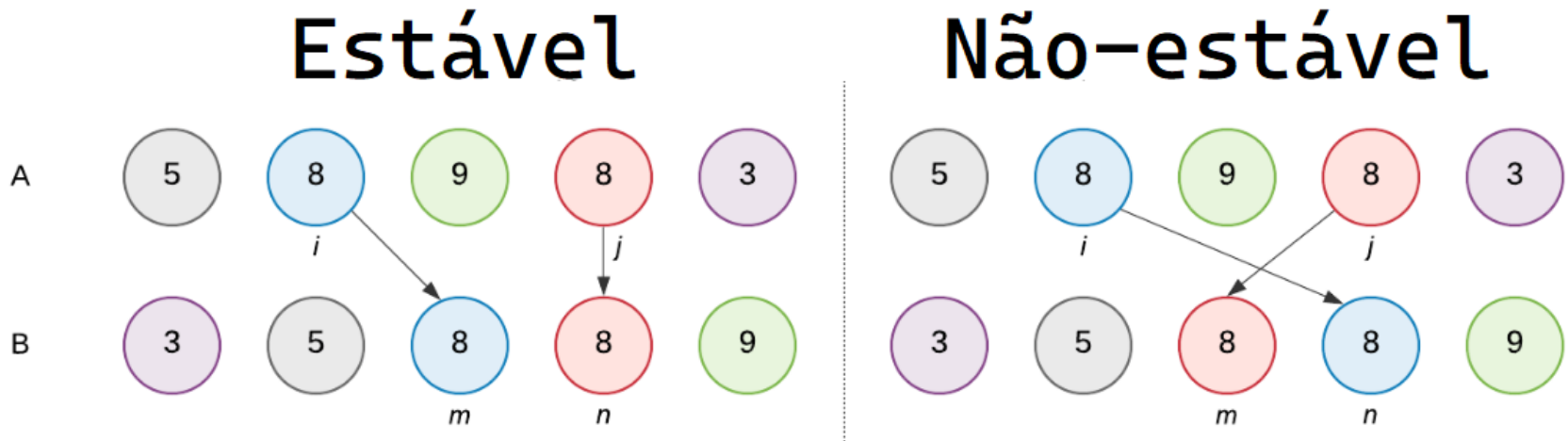
Ordenação externa

- Conjunto de dados não cabe na RAM
  - Devem ser armazenados em memória secundária
- Dados podem ser acessados em ordem sequencial ou em grandes blocos

ORDENAÇÃO ESTÁVEL OU NÃO ESTÁVEL

Um algoritmo é dito ser **estável** caso a ordem dos elementos com chaves iguais não seja mudada durante a ordenação

- Caso a ordem possa ser mudada, o algoritmo é considerado **não-estável**



MÉTODOS DE ORDENAÇÃO

Nesta aula, estudaremos métodos de ordenação simples

- Tempo quadrático
- Fácil implementação
- Auxiliam o entendimento de algoritmos mais complexos

Vamos estudar 3 algoritmos

- Bubble sort
- Selection sort
- Insertion sort

BUBBLE SORT

BUBBLE SORT

Método extremamente simples (e pouco eficiente) de ordenação

Compara pares de valores adjacentes e os troca de lugar caso necessário

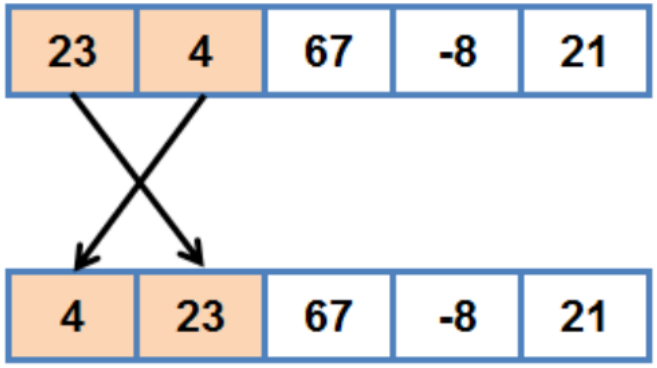
- Trabalha fazendo comparações simples
- Compara todos os pares de elementos

O processo é realizado de forma iterativa até que o vetor inteiro esteja ordenado

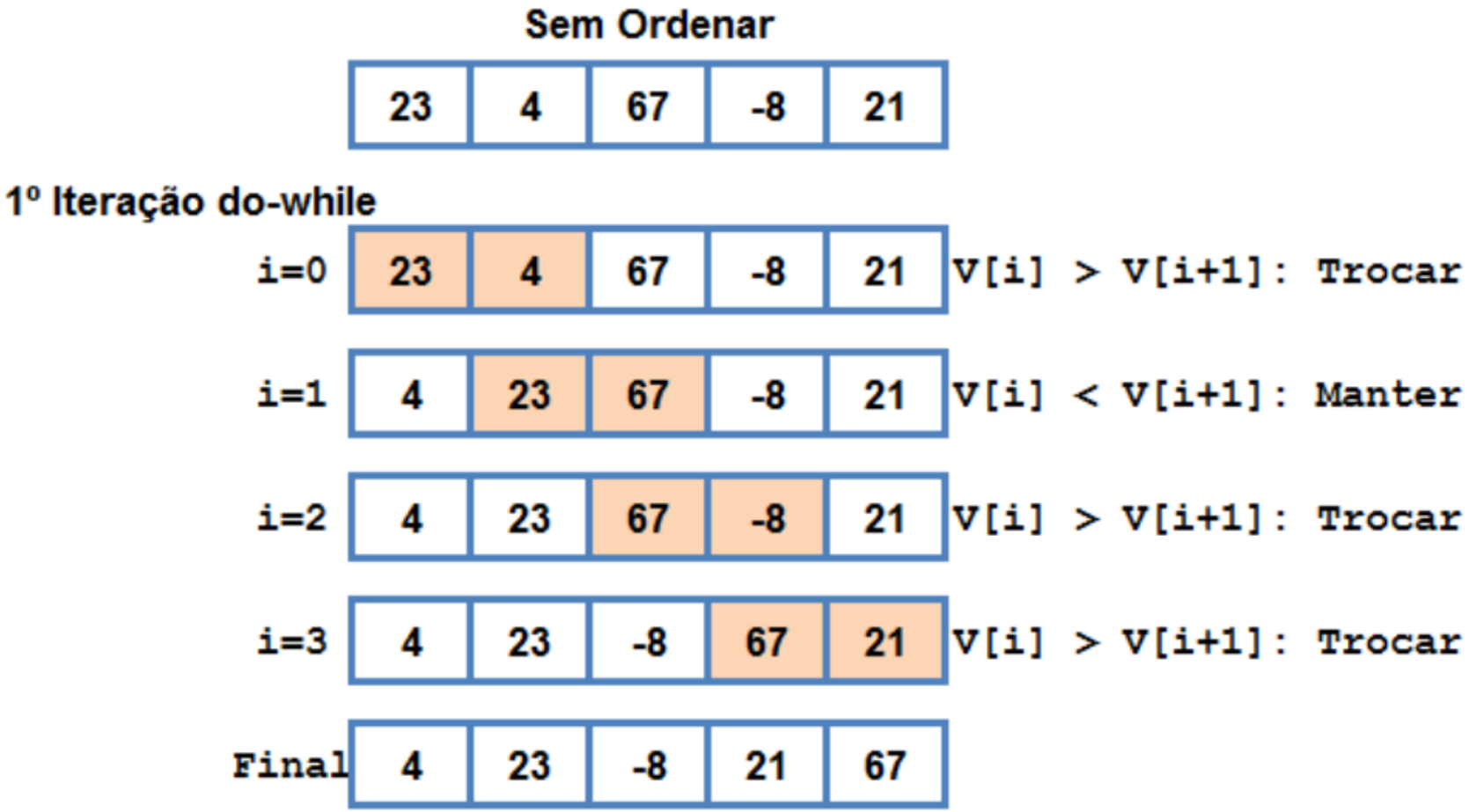
BUBBLE SORT

```
def bubbleSort(V):
    N = len(V)
    continua = True
    while(continua):
        continua = False
        for i in range(N-1):
            if(V[i] > V[i+1]):
                aux = V[i]
                V[i] = V[i+1]
                V[i+1] = aux
                continua = True
        N = N - 1
```

Troca dois valores consecutivos no vetor

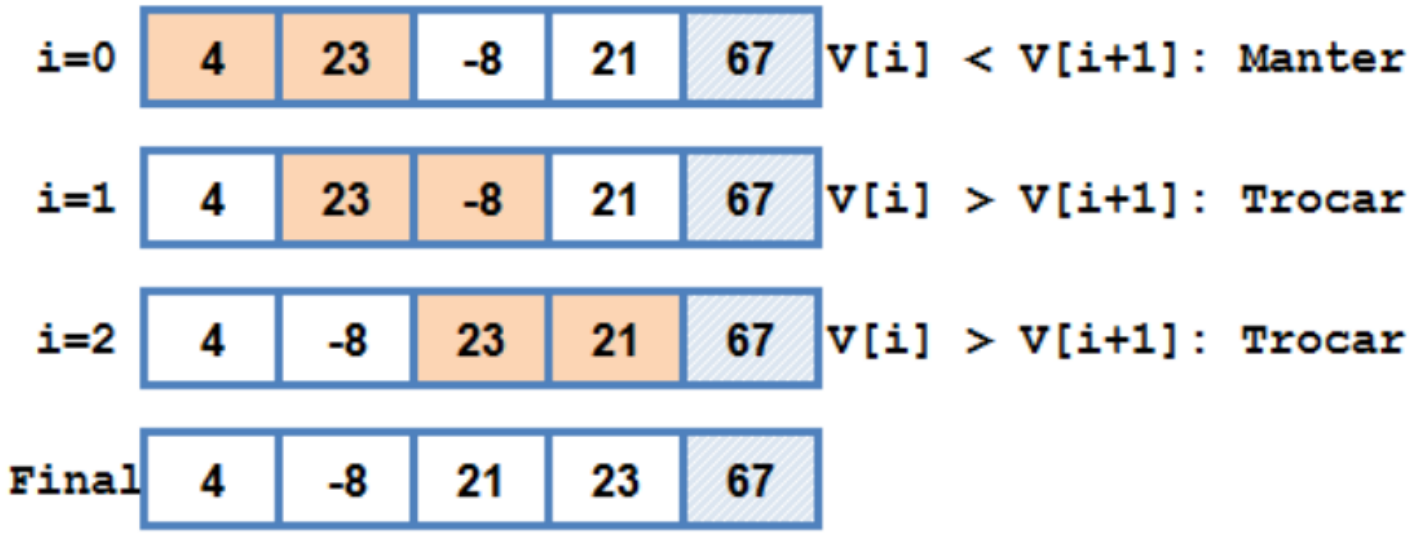


BUBBLE SORT


Sem Ordenar
23 4 67 -8 21
1º Iteração do-while
i=0 23 4 67 -8 21 V[i] > V[i+1]: Trocar
i=1 4 23 67 -8 21 V[i] < V[i+1]: Manter
i=2 4 23 67 -8 21 V[i] > V[i+1]: Trocar
i=3 4 23 -8 67 21 V[i] > V[i+1]: Trocar
Final 4 23 -8 21 67

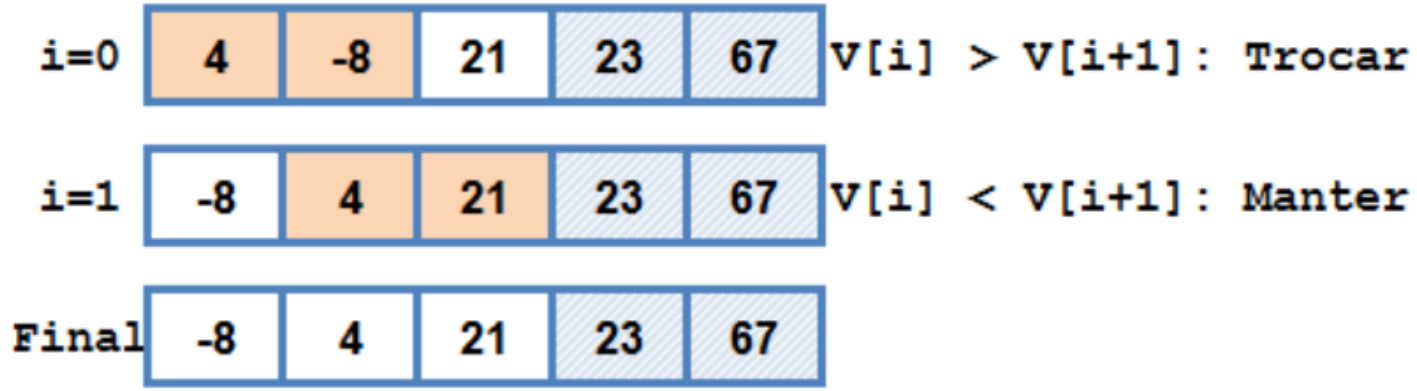
BUBBLE SORT

2º Iteração do-while



BUBBLE SORT

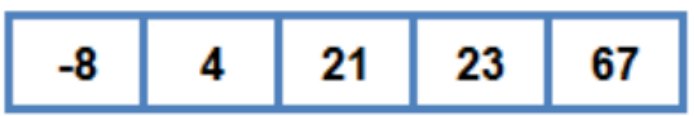
3º Iteração do-while



4º Iteração do-while

Não houve mudanças: ordenação concluída

Ordenado



SELECTION SORT

SELECTION SORT

Outro método simples e pouco eficiente de ordenação

- Também conhecido como ordenação por seleção

Método iterativo

- A cada iteração, seleciona o item com melhor ordem do vetor
  - De acordo com a regra de ordenação
- Insere o item selecionado na primeira posição
- Destaca-se a primeira posição do vetor e repete o processo
  - Processo é repetido até que todas as posições sejam destacadas

Na prática, é um pouco superior ao Bubble Sort

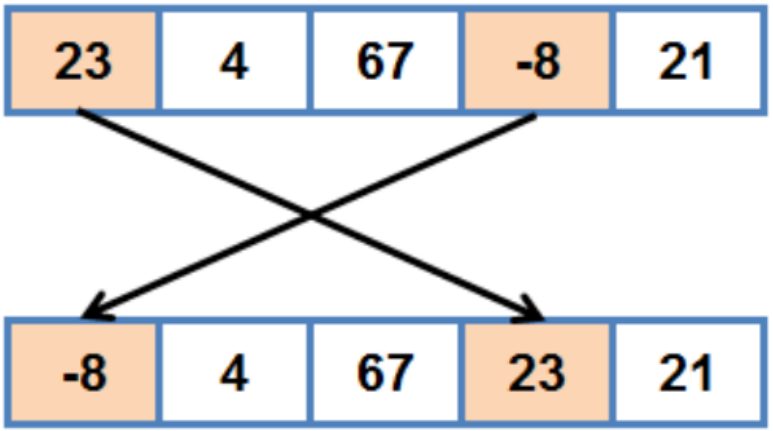
- Faz um número menor de comparações

SELECTION SORT

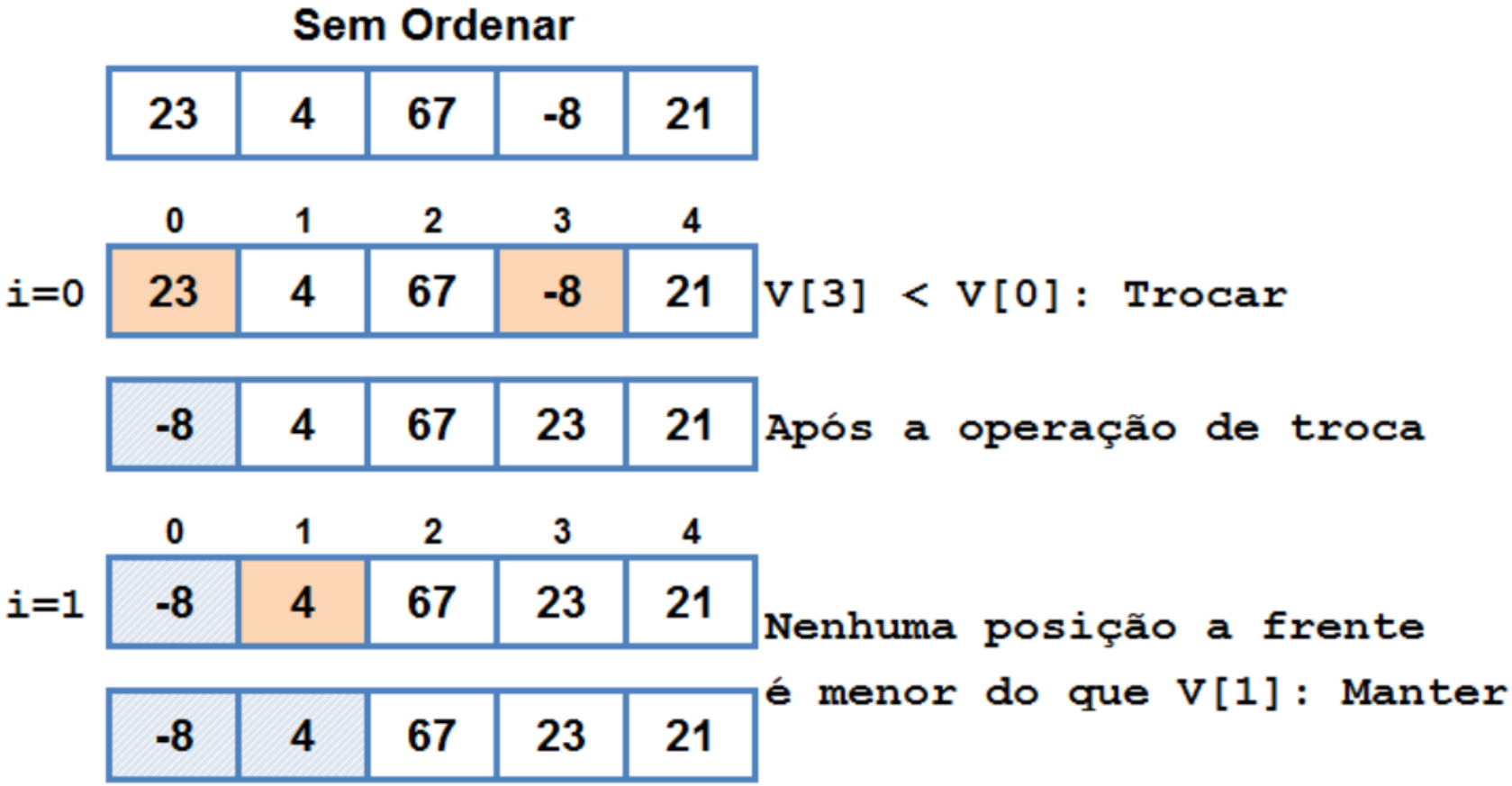
```
def selectionSort(V):
    N = len(V)
    for i in range(N-1):
        menor = i
        for j in range(i+1,N):
            if(V[j] < V[menor]):
                menor = j

        if(i != menor):
            troca = V[i]
            V[i] = V[menor]
            V[menor] = troca
```

Procura o menor elemento em relação a “i”

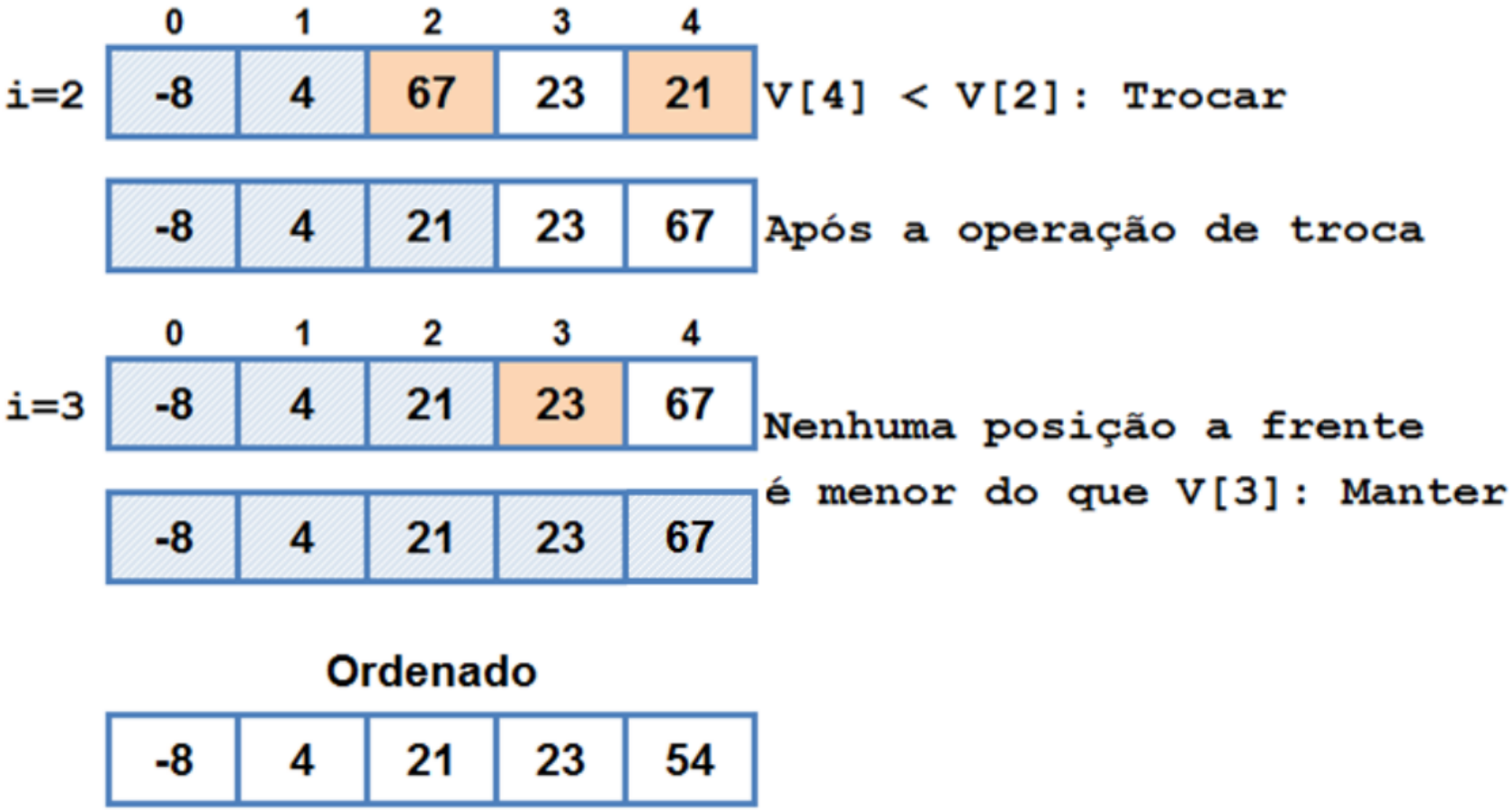
Troca os valores da posição atual com a “menor”



SELECTION SORT



SELECTION SORT



INSERTION SORT

INSERTION SORT

Um terceiro método simples e pouco eficiente de ordenação

- Também conhecido como ordenação por inserção

Similar ao processo que você realiza ao ordenar as cartas de baralho em sua mão

- Pegue uma carta por vez e a insira na posição correta
- Processo realizado de forma iterativa
  - Até que toda sua mão esteja ordenada

INSERTION SORT - FUNCIONAMENTO

O algoritmo percorre o vetor inteiro

- Para cada posição, verifica se a chave está corretamente posicionada
  - Isto é realizado percorrendo o vetor na ordem inversa
  - A partir da posição analisada até o início do vetor
- Caso a chave esteja corretamente posicionada, não realiza nenhuma ação
- Caso contrário, a insere na posição correta
  - Necessário deslocar todas as outras chaves

Esse processo de deslocamento é barato caso seja utilizado uma lista encadeada

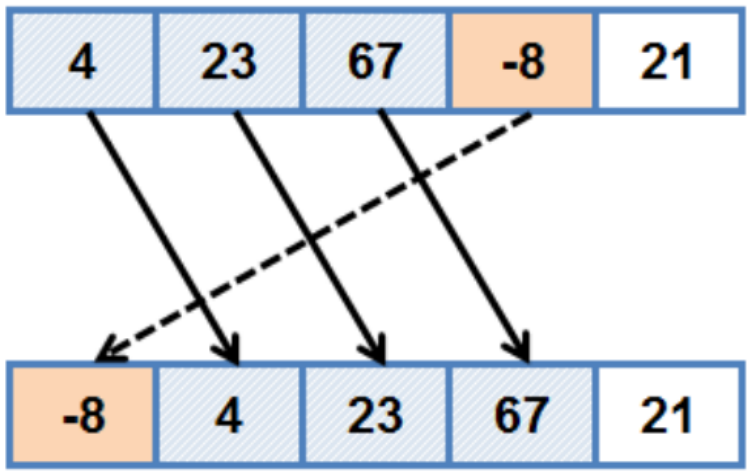
- Necessário somente uma operação com ponteiros

Caso seja um vetor estático, é necessário literalmente trocar todas as posições de memória

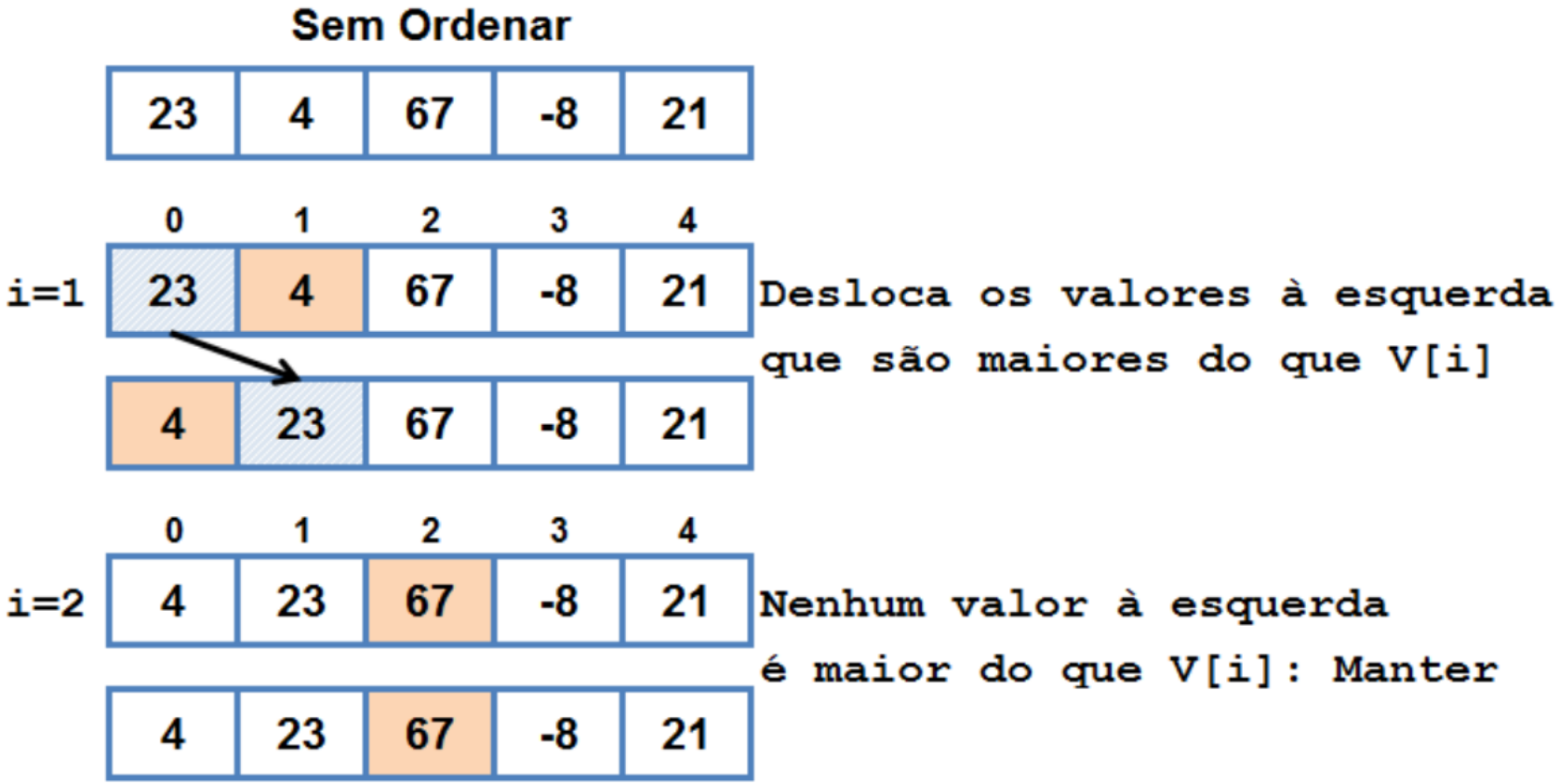
INSERTION SORT

```
def insertionSort(V):
    N = len(V)
    for i in range(1,N):
        aux = V[i]
        j = i
        while(j > 0 and aux < V[j - 1]):
            V[j] = V[j - 1]
            j = j - 1
        V[j] = aux
```

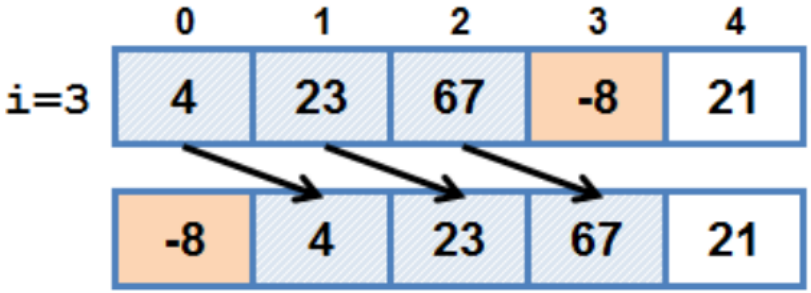
Move as cartas maiores para frente e insere na posição vaga



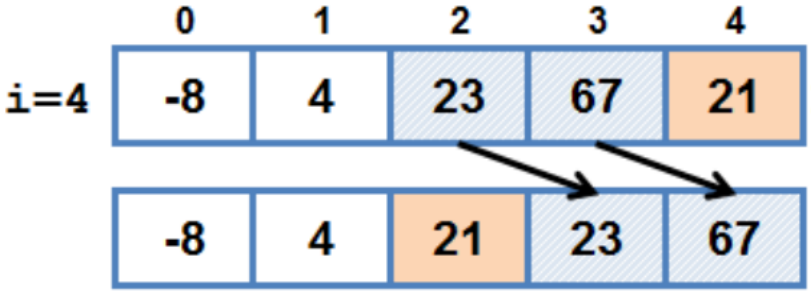
INSERTION SORT



INSERTION SORT



Desloca os valores à esquerda que são maiores do que V[i]



Desloca os valores à esquerda que são maiores do que V[i]

Ordenado

-8	4	21	23	67

ATIVIDADE PRÁTICA

ATIVIDADE

Deve-se implementar os três algoritmos de ordenação

Código-base está disponível no Github ▸ Link

Os algoritmo pode ser visualizados em ▸ Link

PRÓXIMA AULA:

ALGORITMOS DE ORDENAÇÃO ÓTIMOS