PROGRAMAÇÃO I

Curso Técnico Subsequente em Informática

Lucas Sampaio Leite

O problema da ordenação

- Consiste em organizar um conjunto de elementos de acordo com um critério específico, geralmente em uma sequência crescente ou decrescente.

O problema da ordenação

INSTITUTO FEDERAL
Baiano

- Consiste em organizar um conjunto de elementos de acordo com um critério específico, geralmente em uma sequência crescente ou decrescente.
  - Nos primórdios da computação era de conhecimento comum que até 30% de todos os ciclos de processamento era gasto com ordenação.[1]
  - Hoje, essa proporção é menor, devido à eficiência dos algoritmos de ordenação.[1]

[1]Robert Sedgewick and Kevin Wayne. 2011. *Algorithms* (4th ed.). Addison-Wesley Professional

Bubble sort

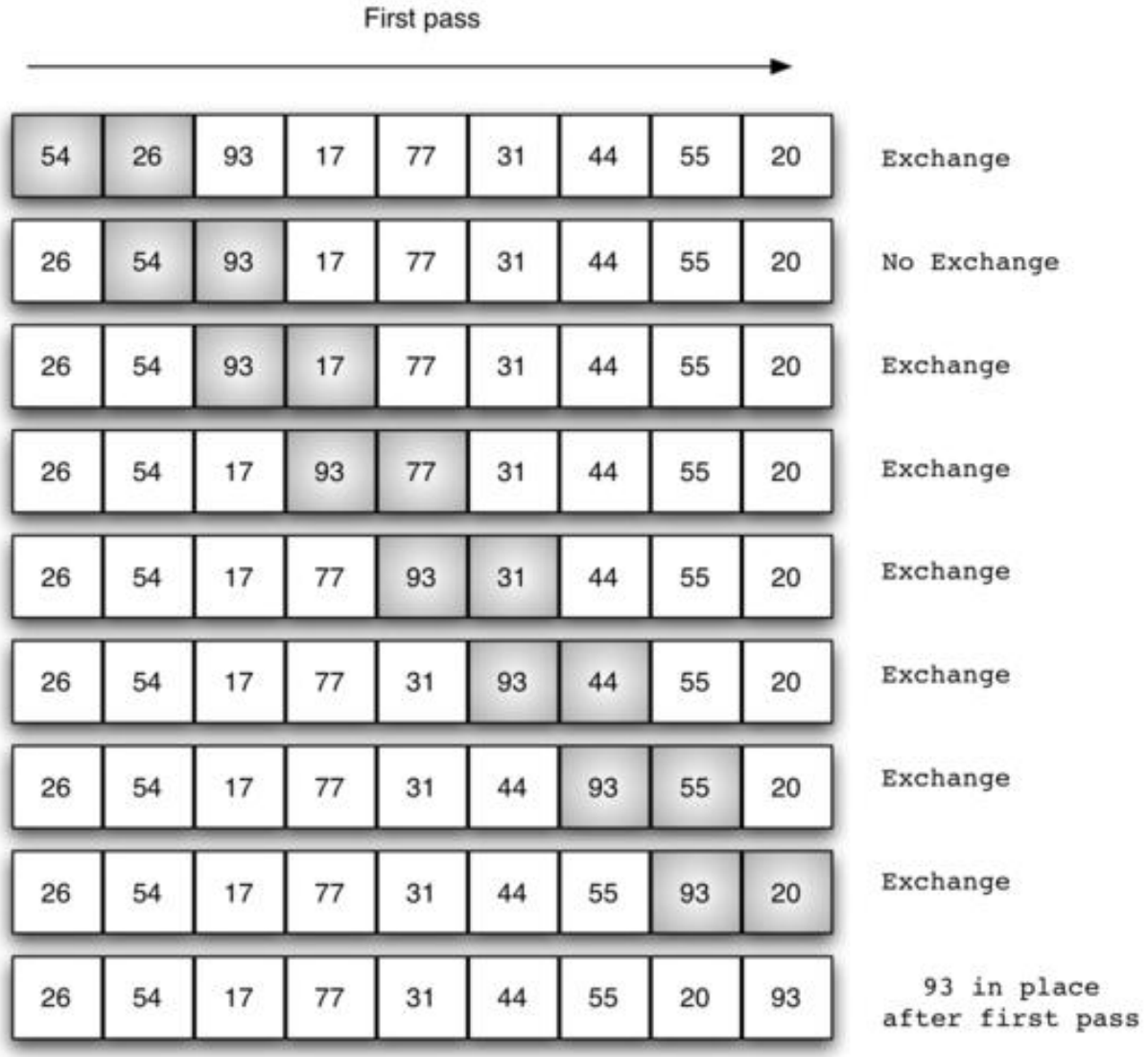
INSTITUTO FEDERAL
Baiano

- O Bubble Sort baseia-se na ideia de comparar repetidamente pares de elementos adjacentes e, em seguida, trocar as suas posições se estiverem na ordem errada.
- Etapas:

1. Compara dois elementos;
2. Se o da esquerda for maior que o da direita, troca-se os elementos de lugar;
3. O processo é repetido até que se percorra a lista inteira sem nenhuma troca ter sido feita.

Bubble sort

- Exemplo (primeira passagem):



Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OBubbleSort.html#lst-bubble

Bubble sort

INSTITUTO FEDERAL
Baiano

```
def bubble_sort(lista):
    for i in range(len(lista) - 1):
        for j in range(0, len(lista) - i - 1):
            if lista[j] > lista[j + 1]:
                aux = lista[j]
                lista[j] = lista[j + 1]
                lista[j + 1] = aux
```

Bubble sort

INSTITUTO FEDERAL
Baiano

```python
def modified_bubble_sort(lista):
    for i in range(len(lista) - 1):
        swapped = False
        for j in range(0, len(lista) - i - 1):
            if lista[j] > lista[j + 1]:
                aux = lista[j]
                lista[j] = lista[j + 1]
                lista[j + 1] = aux
                swapped = True
        if not swapped:
            break
```

Bubble sort

INSTITUTO FEDERAL
Baiano

- Vantagens:
  - Muito intuitivo: fácil de entender porque só compara elementos vizinhos.
  - Fácil de implementar: poucas linhas de código; ótimo para quem está começando.
  - Utilizado em contextos educacionais: ajuda a introduzir o conceito de ordenação e troca de posições.
- Desvantagens:
  - Extremamente lento para listas grandes: faz trabalho repetitivo demais, mesmo quando a maior parte já está ordenada.
  - Por isso, na prática, quase nunca é usado em sistemas reais, apenas como ferramenta de aprendizado.

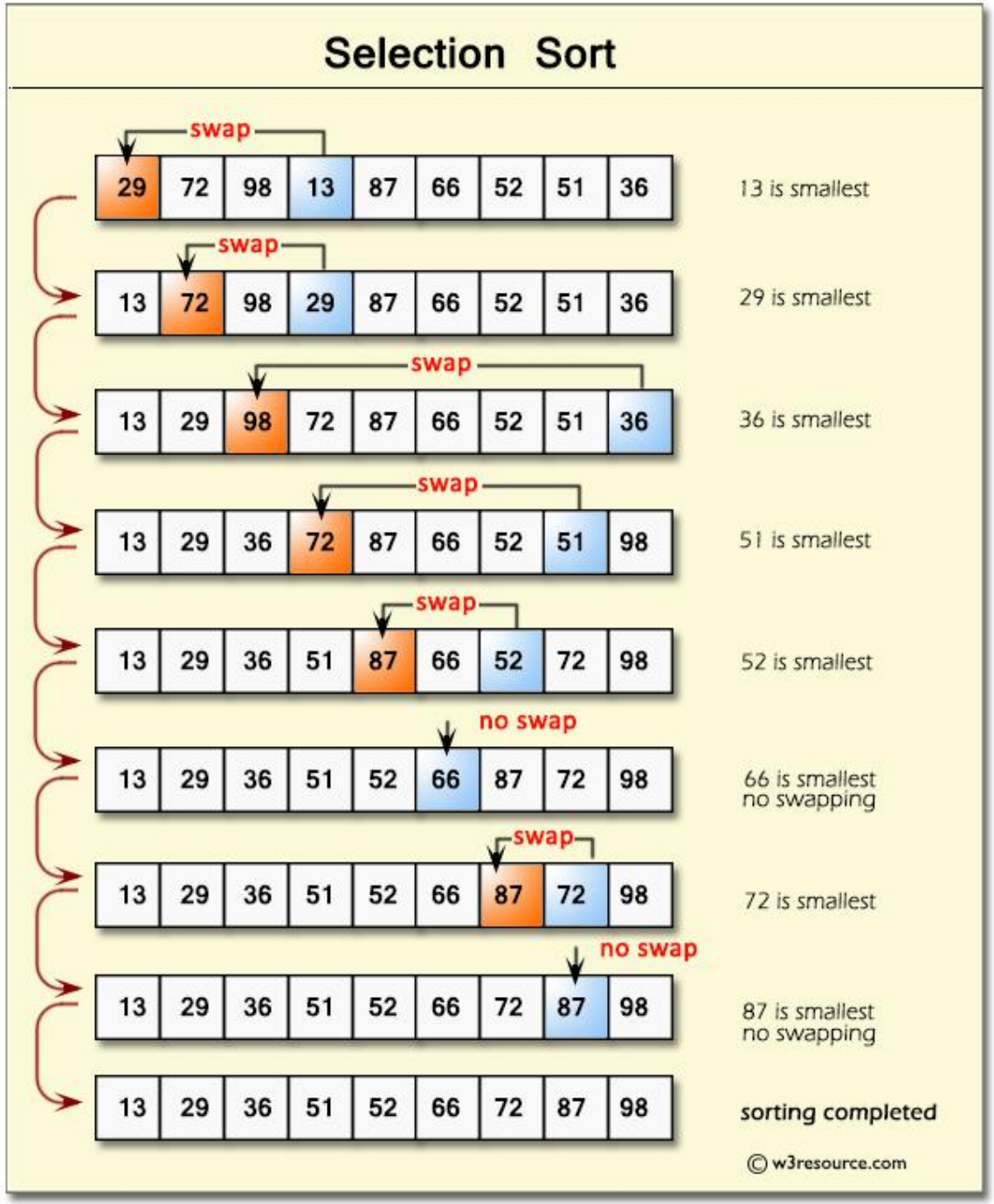
Selection sort

INSTITUTO FEDERAL
Baiano

- A ideia é similar à do Bubble Sort, mas as trocas não são feitas imediatamente;
- Para cada posição da lista, seleciona-se o menor elemento (ou o maior) da vez e o coloca nesta posição;
- As próximas comparações e trocas são feitas com o último menor (ou maior) valor encontrado;
- Uma vez percorrido o vetor inteiro, incrementa-se a posição para comparação.

Bubble sort

- Exemplo:



Fonte da imagem: https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-11.php

Selection sort

INSTITUTO FEDERAL
Baiano

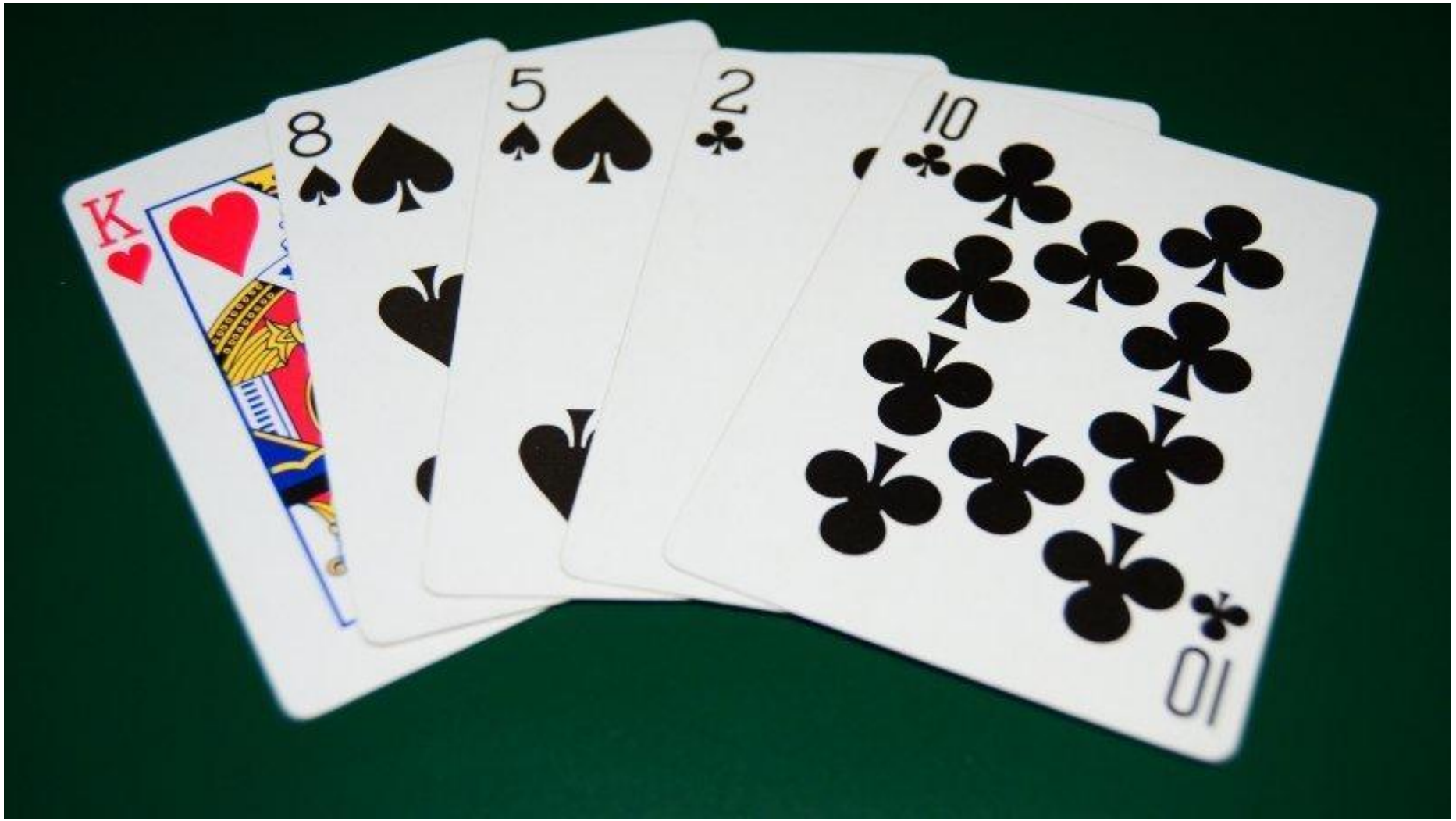
```
def selection_sort(lista):
    for i in range(len(lista) - 1):
        min_index = i
        for j in range(i + 1, len(lista)):
            if lista[j] < lista[min_index]:
                min_index = j

        aux = lista[i]
        lista[i] = lista[min_index]
        lista[min_index] = aux
```

Selection sort

INSTITUTO FEDERAL
Baiano

- Vantagens:
  - Poucas trocas: troca apenas quando encontra o menor valor da etapa, ao contrário do Bubble Sort, que troca muitas vezes.
  - Fácil de entender e implementar: segue uma lógica simples de “encontrar o menor”.
- Desvantagem:
  - Número de comparações muito alto: apesar de trocar pouco, ele ainda precisa verificar praticamente todos os elementos várias vezes.
  - Custo computacional elevado: para listas grandes, ele se torna lento porque exige muito trabalho repetitivo.

Insertion sort



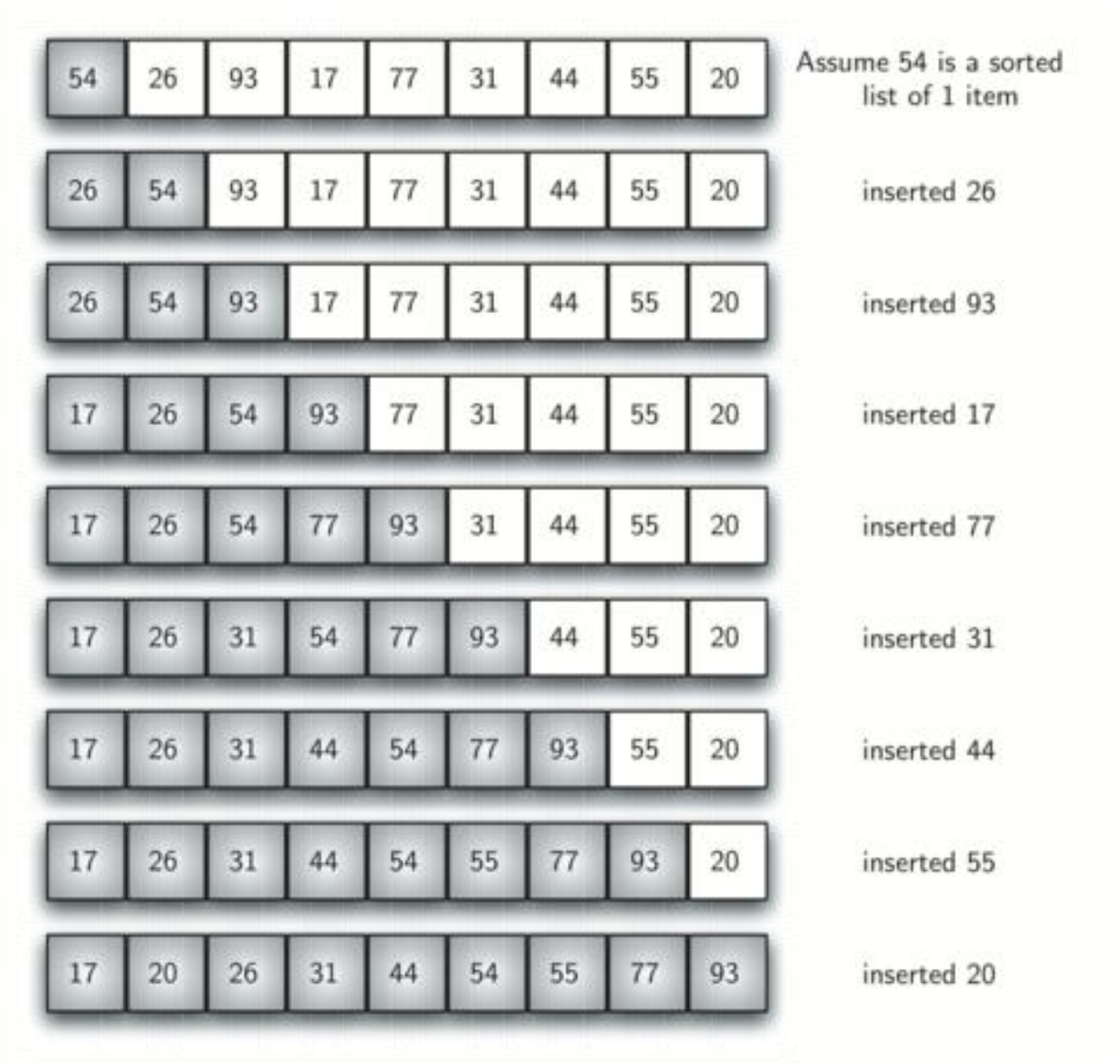
INSTITUTO FEDERAL
Baiano

Insertion sort

INSTITUTO FEDERAL
Baiano

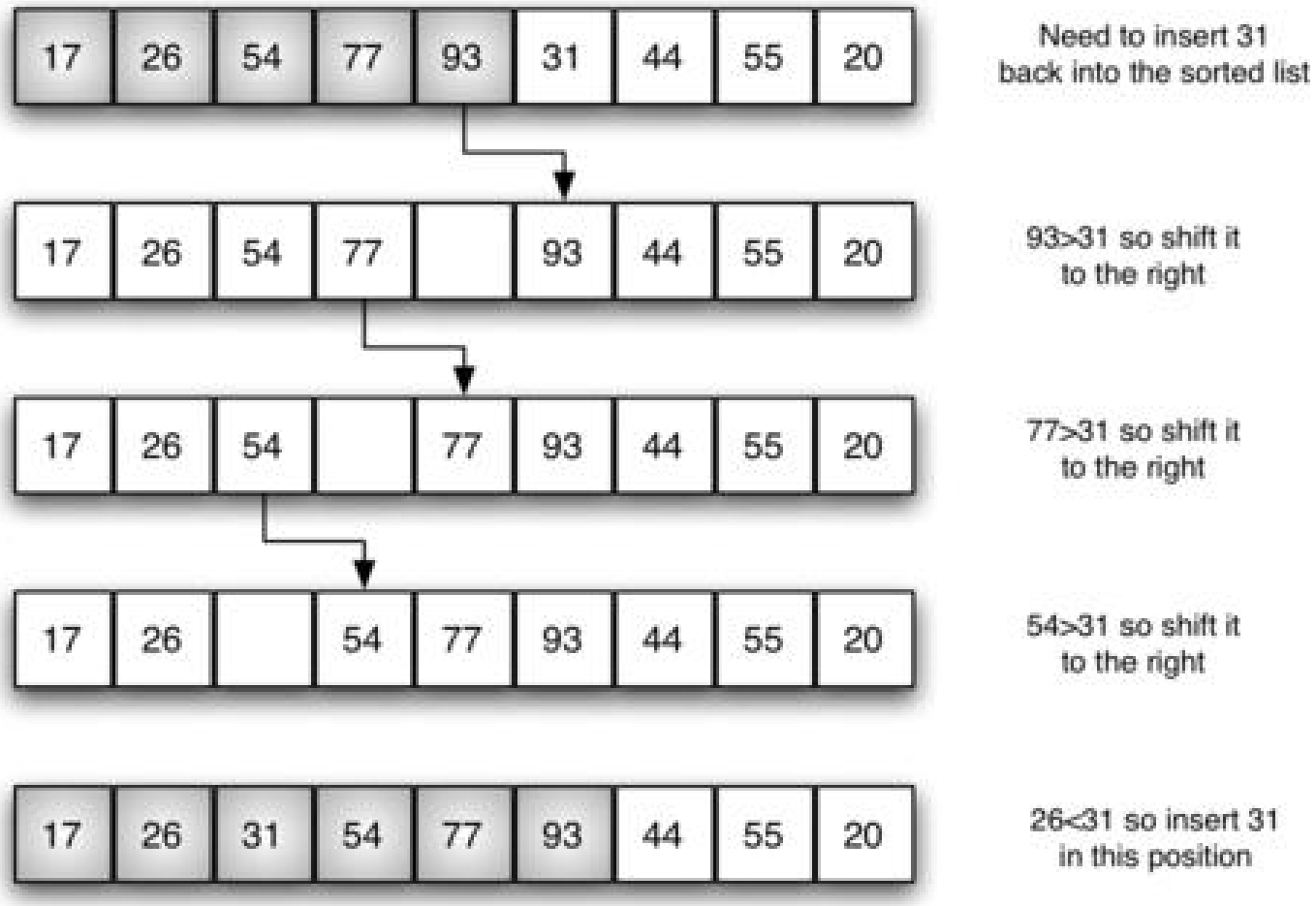
- Defina dois subconjuntos da lista primária: sorted e unsorted
- Inicialmente, o subconjunto sorted contém apenas o primeiro elemento da lista inteira.
- O algoritmo então percorre o subconjunto unsorted, um elemento por vez, e insere cada elemento na posição correta do subconjunto sorted, movendo os elementos maiores para a direita para abrir espaço para o novo elemento.
- Este processo é repetido até que todos os elementos tenham sido inseridos no conjunto sorted.

Insertion sort



Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/AOrdenacaoPorInsercao.html

Insertion sort



Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/AOrdenacaoPorInsercao.html

Insertion sort

INSTITUTO FEDERAL
Baiano

```python
def insertion_sort(lista):
    for i in range(1, len(lista)):
        key = lista[i]
        j = i - 1

        while j >= 0 and lista[j] > key:
            lista[j + 1] = lista[j]
            j = j - 1

        lista[j + 1] = key
```

Insertion sort

INSTITUTO FEDERAL
Baiano

- Vantagens:
  - Mais eficiente na prática: funciona muito bem quando os elementos já estão parcialmente ordenados.
  - Menos trocas: ele só desloca elementos maiores para a direita, evitando trocas desnecessárias.
  - Lógica intuitiva: funciona como organizar cartas, simples e fácil de visualizar.
- Desvantagens:
  - Custo computacional elevado em listas grandes: quando a lista está desordenada, ele precisa mover muitos elementos várias vezes, tornando o processo demorado.
  - Passa repetidamente pelos elementos: ordenar n itens exige várias passagens, o que não escala bem para tamanhos maiores.

Visualgo

INSTITUTO FEDERAL
Baiano

https://visualgo.net/en/sorting

Algoritmos eficientes

INSTITUTO FEDERAL
Baiano

- Exemplos de algoritmos eficientes incluem Merge Sort, Heap Sort, Quick Sort e Radix Sort.
- Eles utilizam estratégias mais avançadas, como dividir o problema em partes menores, usar estruturas de dados específicas ou distribuir os valores, o que reduz o número de comparações e movimentações.
- Enquanto algoritmos simples ficam muito lentos em listas grandes, os eficientes mantêm um desempenho muito melhor e conseguem lidar com grandes volumes de dados.
- Por isso, são os preferidos em aplicações reais, ao contrário dos métodos quadráticos, usados quase sempre para fins didáticos.

Dúvidas

INSTITUTO FEDERAL
Baiano

PROGRAMAÇÃO I

Curso Técnico Subsequente em Informática

Lucas Sampaio Leite