Estrutura de Dados

Aula 09

Prof. Luiz Antonio Schalata Pacheco, Dr. Eng.

Instituto Federal de Santa Catarina
Câmpus Garopaba
Curso Superior de Tecnologia em Sistemas para Internet

schalata@ifsc.edu.br

18/05/2023

Algoritmos de Ordenação

Motivação

- Inserir um dado em um vetor ordenado gera um Big(O) elevado, pois tem de fazer o remanejamento dos itens
- Ter os dados ordenados é um passo preliminar para pesquisá-los:
  - pesquisa linear (lenta)
  - pesquisa binária
- Logo, muitas vezes é necessário ordenar uma base de dados já construída:
  - Nome em ordem alfabética
  - Alunos por nota
  - Vendas por preço

Vários métodos de ordenação

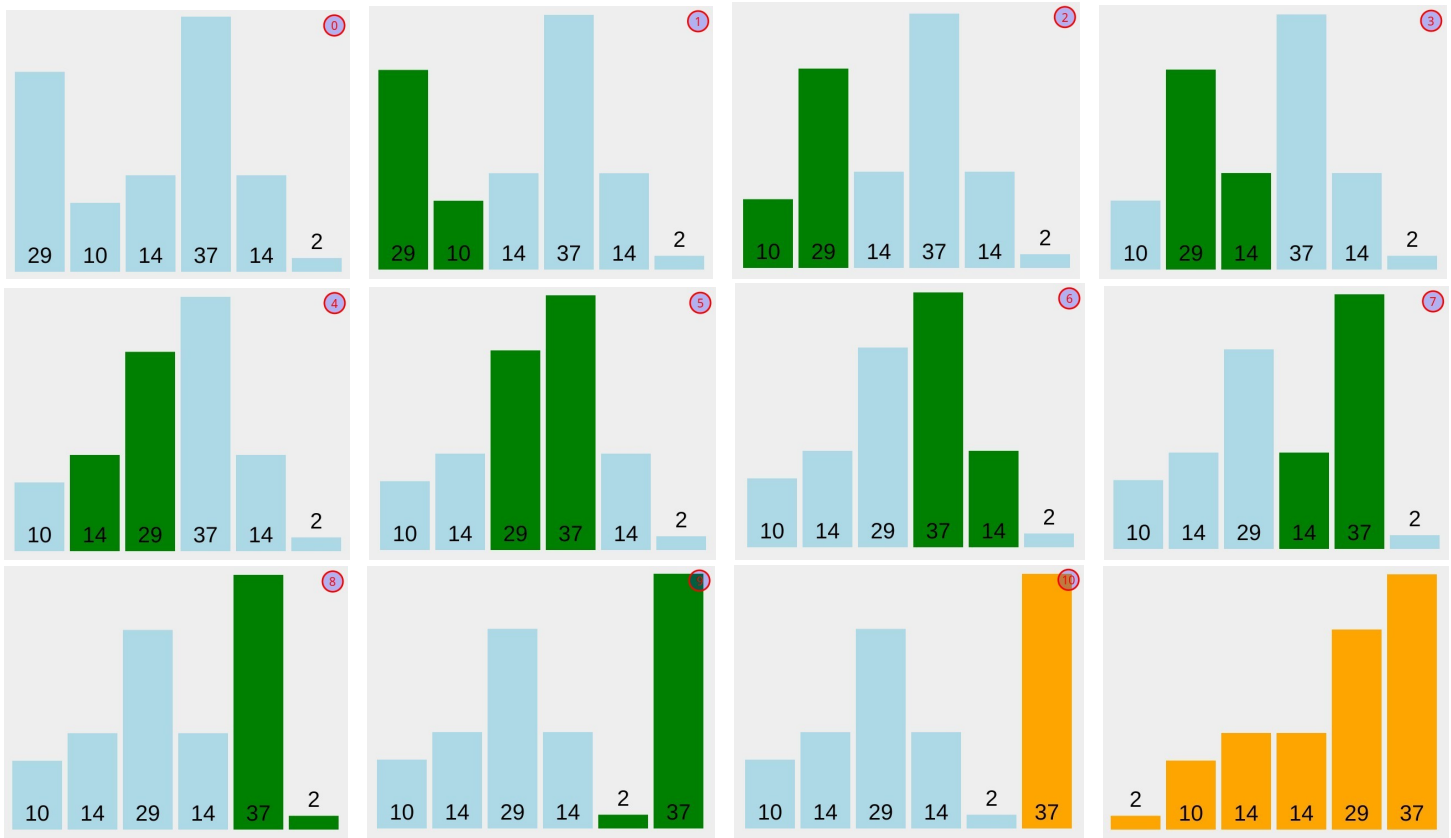
- Bubble sort
- Selection sort
- Insertion sort
- Shell sort
- Merge sort
- Quick sort

Bubble Sort

Bubble Sort - Conceito

- É o algoritmo de ordenação mais lento e mais simples
- Funcionamento:
  - Compara dois números
  - Se o da esquerda for maior, os elementos devem ser trocados
  - Desloca-se uma posição à direita
  - A medida que algoritmo avança, os itens maiores "surgem como uma bolha" na extremidade superior do vetor
  - Exige várias rodadas

Bubble Sort em funcionamento



- https://visualgo.net/en/sorting

Análise da complexidade

- O algoritmo para ordenação de 6 números faz:
  - 5 comparações na primeira passagem
  - 4 comparações na segunda passagem
  - 3 comparações na terceira passagem
  - 2 comparações na quarta passagem
  - 1 comparações na quinta passagem
- Para 6 itens: 15 comparações, ou seja, $(n-1)+(n-2)+(n-3)+(n-4)+(n-5)$
- Generalizando: $(n-1)+(n-2)+...+(n-(n-1))$, onde n é o número de itens a ordenar
- O número de trocas depende de como os dados estão organizados
- Cada troca leva 3 passos

Análise da complexidade

- Big-O: $O(n^2)$
- São realizadas cerca de $n^2/2$ comparações
- Ocorrem menos trocas que comparações, pois os elementos só serão trocados se não estiverem em ordem
- A quantidade de trocas necessária é estimada em $n^2/4$ considerando dados aleatórios
- Pior caso: dados ordenados de modo inverso

Implementação

- Implementar e testar o algoritmo de ordenação Bubble Sort

Implementação: Bubble Sort

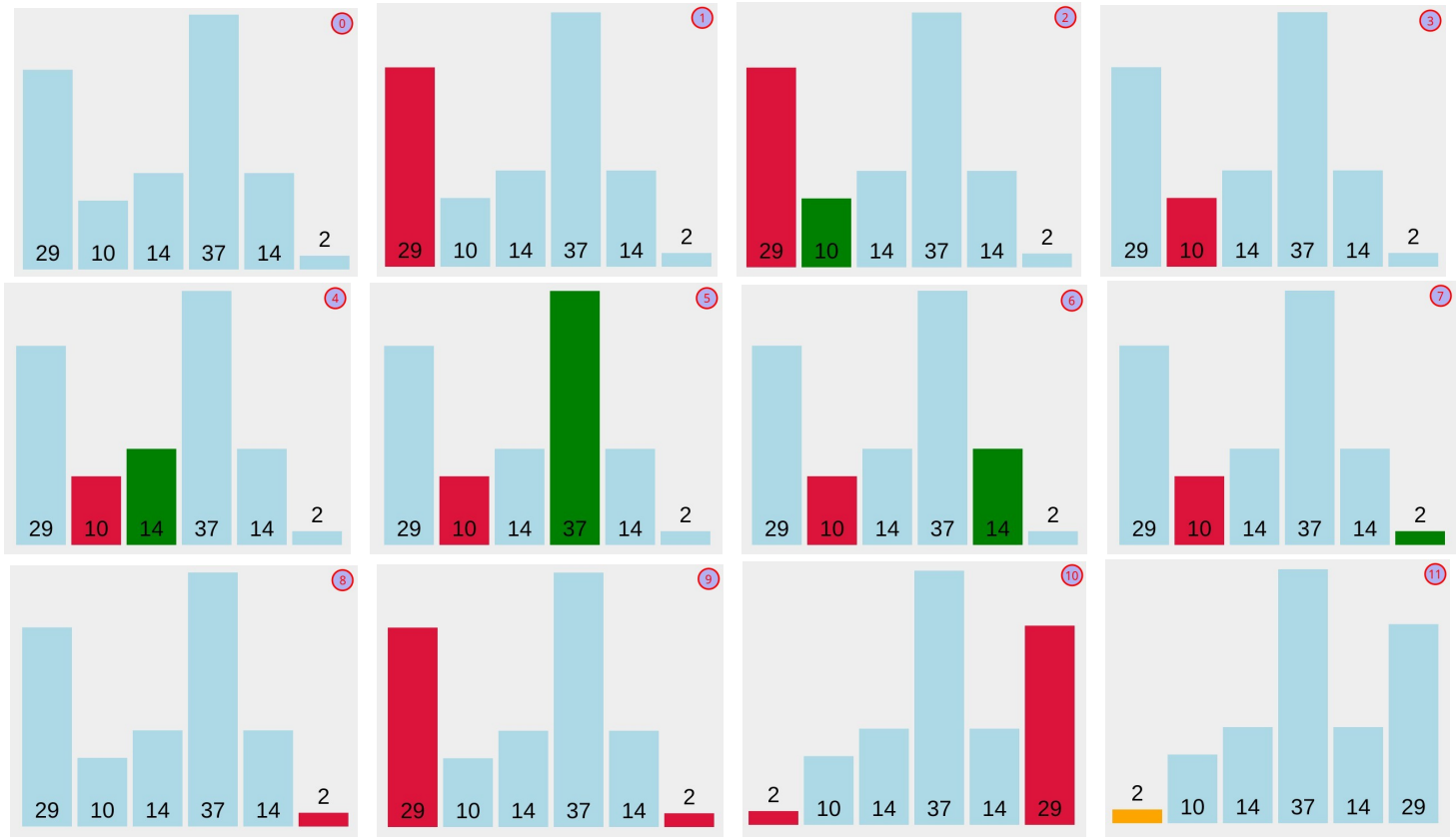
```
1  import numpy as np
2
3  def bubble_sort(vetor): # Recebe um vetor nao ordenado
4    n = len(vetor) # Tamanho do vetor
5
6    for i in range(n):
7      # Executa a ordenacao ate o valor que ja esta
       ordenado, por isso tem que usar (n - i) - 1 no range
8      for j in range(0, (n - i) - 1):
9        # Se o elemento da esquerda for maior que o da
       direita, entao faz a troca dos valores
10       if vetor[j] > vetor[j + 1]:
11         temp = vetor[j]
12         vetor[j] = vetor[j + 1]
13         vetor[j + 1] = temp
14   return vetor
15
16   print(bubble_sort(np.array([29, 10, 14, 37, 14, 2])))
```

Selection Sort

Selection Sort - Conceito

- Reduz o número de trocas de N^2 para N
- O número de comparações permanace $N^2/2$
- Funcionamento:
  - Percorre todos os elementos e seleciona o menor
  - O menor elemento é trocado com o elemento da extremidade esquerda do vetor
  - Os elementos ordenados acumulam-se na esquerda

Selection Sort em funcionamento



- https://visualgo.net/en/sorting

Análises

- Mesmo número de comparações do Bubble Sort
- Para 6 itens: 15 comparações
- Big-O: $O(n^2)$
- Geralmente uma troca é feita a cada passagem. Então para 6 elementos são requeridas menos de 6 trocas.
- Se tiver 100 itens, são requiridas 4950 comparações, mas menos de 100 trocas

Implementação

- Implementar e testar o algoritmo de ordenação Selection Sort

Implementação: Selection Sort

```
1  def selection_sort(vetor):
2    n = len(vetor)
3
4    for i in range(n): # Cada iteracao indica uma rodada
5      id_minimo = i # Posicao onde esta o menor valor
6      for j in range(i + 1, n): # Parte de i + 1 porque os
       elementos ordenados nao precisam ser percorridos
       novamente e eles estao na posicao inicial do vetor
7        if vetor[id_minimo] > vetor[j]:
8          id_minimo = j # Se o valor comparado for menor,
       sua posicao passa a ser a nova posicao de id_minimo
9      # Ao final faz a troca do valores
10     temp = vetor[i]
11     vetor[i] = vetor[id_minimo]
12     vetor[id_minimo] = temp
13
14   return vetor
```

Insertion Sort

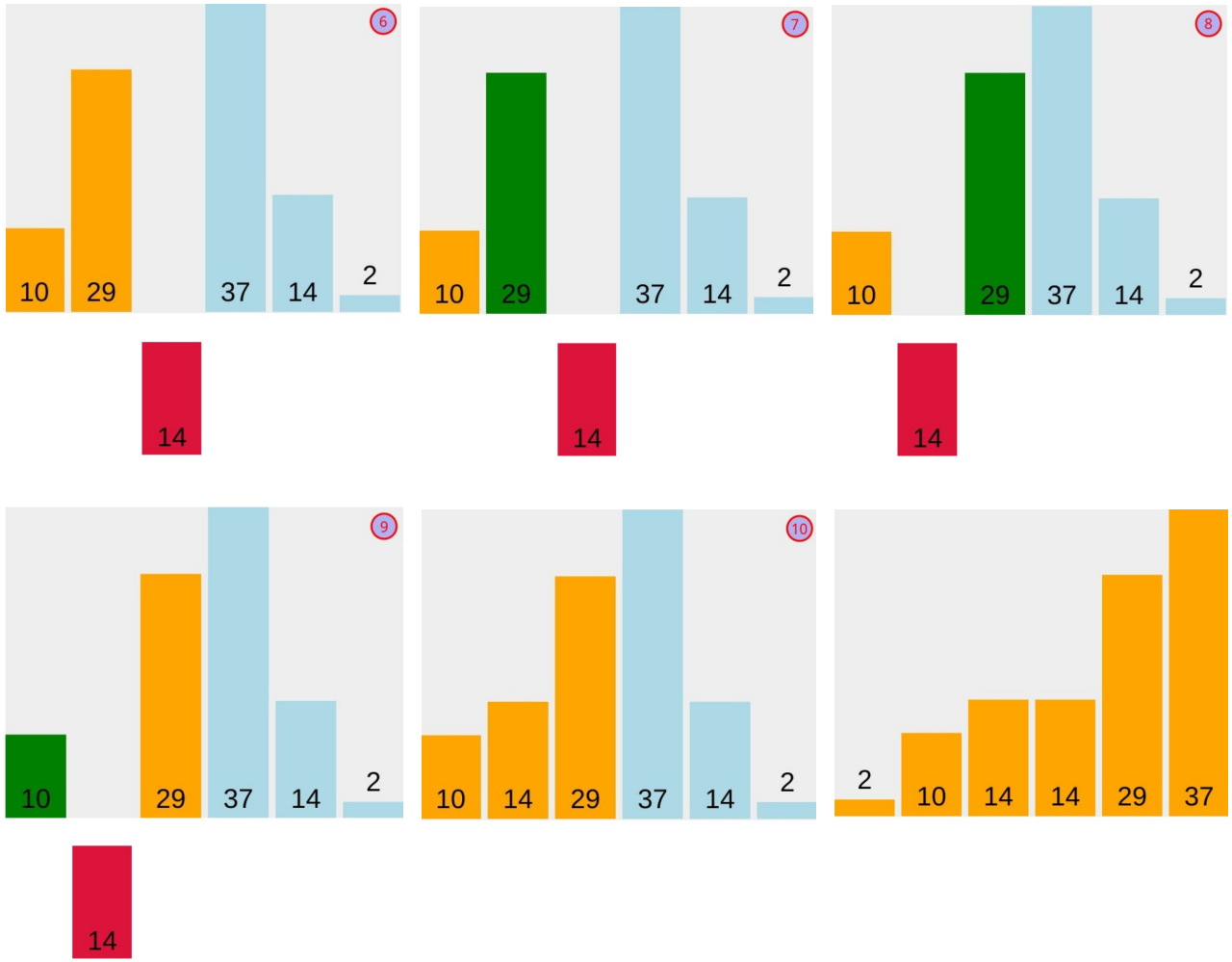
Insertion Sort - Conceito

- É cerca de duas mais rápido que o Bubble Sort e um pouco mais rápido que o Selection Sort (com dados aleatórios!)
- Funcionamento:
  - Há um marcador em algum lugar no meio do vetor
  - Os elementos a esquerda do marcados estão parcialmente ordenados (estão ordenados entre eles, mas não estão em suas posições finais)
  - Os elementos a direita do vetor não estão ordenados

Insertion Sort em funcionamento

- https://visualgo.net/en/sorting

Insertion Sort em funcionamento



Análises

- Na primeira passagem, é comparado no **máximo** um item. Na segunda passagem, no máximo de 2 itens e assim por diante
- Para 6 itens: $1 + 2 + 3 + 4 + 5 = 15$ comparações
- Generalizando: N*(N - 1)/2 comparações
- Como, na média, apenas metade do número máximo de itens é comparado, temos: N*(N - 1)/4 comparações
- O número de cópias (não são trocas!) é aproximadamente o mesmo número de comparações
- É importante conhecer previamente a base de dados a ser ordenada:
  - Para dados pre ordenados esse algoritmo é ainda mais eficiente
  - Para dados em ordem inversa, torna-se mais lento que o Bubble Sort, pois são executadas todas as comparações e deslocamentos

Implementação

- Implementar e testar o algoritmo de ordenação Insertion Sort

Implementação: Insertion Sort

```
1  def insertion_sort(vetor):
2    n = len(vetor)
3
4    for i in range(1, n): # Inicia na 2a posicao do vetor
5      marcado = vetor[i]
6
7      j = i - 1
8      # Faz comparacoes ate o inicio do vetor (j >= 0) e
       somente enquanto o valor marcado for menor que a
       posicao do vetor que esta sendo comparada
9      while j >= 0 and marcado < vetor[j]:
10       vetor[j + 1] = vetor[j] # Copia o elemento uma
         posicao para frente
11       j -= 1
12     vetor[j + 1] = marcado # Copia o elemento marcado na
       posicao correta
13
14   return vetor
```