INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Campos do Jordão

ESTRUTURA DE DADOS

Algoritmos de Ordenação por Troca, Seleção e Inserção

Bubble Sort, Selection Sort, Insertion Sort.

Professor Mestre Igor de Moraes Sampaio
igor.sampaio@ifsp.edu.br

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - **Campus Campos do Jordão**

Algoritmos de Ordenação

Definição Geral

- **Ordenação** é o processo de reorganizar uma coleção de elementos em uma determinada ordem - crescente, decrescente ou segundo algum outro critério.

Conceito Fundamental

- A ordenação é uma das operações mais básicas e importantes na ciência da computação, servindo como base para muitos outros algoritmos.

- Podemos ordenar praticamente qualquer tipo de dado que possa ser comparado: números, strings, datas, objetos complexos (usando uma chave de comparação).

Exemplo Intuitivo

Imagine organizar cartas de baralho na sua mão:

Características da Ordenação

- **Propriedades**
  - **Determinística**: Produz sempre o mesmo resultado para a mesma
  - entrada
  - **Comparativa**: Baseada em comparações entre elementos
  - **Transformativa**: Reorganiza os dados sem alterar seu conteúdo

- **Critérios de Ordenação**
  - **Crescente (Ascendente)**: Do menor para o maior valor
  - **Decrescente (Descendente)**: Do maior para o menor valor
  - **Lexicográfica**: Ordem alfabética para strings
  - **Personalizada**: Baseada em critérios específicos

- Embora pareça simples conceitualmente, a ordenação eficiente de grandes conjuntos de dados é um desafio computacional significativo.

Aplicações Práticas da Ordenação

- A ordenação é uma operação fundamental que melhora significativamente a eficiência de muitos processos computacionais:

Mais Aplicações Práticas da Ordenação

Detecção de Anomalias

Facilita a identificação de valores atípicos (outliers) e padrões incomuns nos dados.

Útil em sistemas de segurança e monitoramento de qualidade.

Compressão

Alguns algoritmos de compressão funcionam melhor com dados ordenados ou pré-processados.

Ordenar pode revelar padrões que aumentam a taxa de compressão.

Algoritmos Complexos

Muitos algoritmos avançados dependem de dados ordenados como pré-requisito.

Exemplos: algoritmos de grafos, geometria computacional, etc.

Algoritmos de Ordenação e Uso de Memória

- Uma forma importante de classificar algoritmos de ordenação é pelo seu uso de memória auxiliar durante o processo de ordenação:

- **In-place**

  - Algoritmos in-place realizam a ordenação diretamente no array original, usando apenas uma quantidade constante (O(1)) ou muito pequena de memória auxiliar.

- **Not-in-place**

  - Algoritmos not-in-place requerem espaço adicional proporcional ao tamanho da entrada (geralmente O(n)) para realizar a ordenação.

Algoritmos de Ordenação e Uso de Memória

Antes da Ordenação

7	2	5	1	8

Durante/Após a Ordenação

1	2	5	7	8

Exemplos:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Heap Sort

Array Original

7	2	5	1	8

Array Auxiliar

1	2	5	7	8

↓ Cópia de volta para o array original

Exemplos:

- Merge Sort
- Counting Sort
- Radix Sort

Algoritmos de Ordenação e Uso de Memória

Característica	In-place	Not-in-place
Uso de memória	O(1) ou muito pequeno	O(n) ou mais
Vantagem principal	Economia de memória	Geralmente mais rápidos
Desvantagem	Pode ser mais lento	Maior consumo de memória
Ideal para	Dispositivos com memória limitada	Quando a velocidade é prioritária

Estabilidade em Algoritmos de Ordenação

- A estabilidade é uma propriedade que determina se elementos com chaves iguais mantêm sua ordem relativa original após a ordenação.

Algoritmo Estável

Considere cartas com valores e naipes:

Mantêm a ordem relativa de registros com chaves iguais.

Exemplos:

- Bubble Sort
- Insertion Sort
- Merge Sort
- Counting Sort (se implementado corretamente)

Algoritmo Instável

Mesmo exemplo, mas com algoritmo instável:

5♥ vem antes de 5♠ na entrada, mas **5♠ vem antes de 5♥** na saída

Não garantem a preservação da ordem relativa de registros com chaves iguais.

Exemplos:

- Selection Sort
- Heap Sort
- Quick Sort (implementação padrão)

Quando a Estabilidade Importa?

- Ordenações múltiplas sequenciais
- Ordenação por múltiplos critérios
- Preservação da ordem natural dos dados

- **Exemplo Prático**
    - **Ordenar uma lista de alunos:**

      1. Primeiro por turma (A, B, C)
      2. Depois por nota (decrescente)

    - Se o algoritmo usado na segunda ordenação não for estável, a ordenação por turma será perdida!

A Importância da Eficiência

- Não basta apenas que um algoritmo resolva um problema; é crucial entender como ele o faz em termos de recursos computacionais.

- **Eficiência de Tempo**

  - Mede quanto tempo um algoritmo leva para executar em função do tamanho da entrada.

  - **Perguntas-chave:**
    - Quanto tempo leva para ordenar 1.000 elementos?
    - E se aumentarmos para 1.000.000?
    - Como o tempo cresce em relação ao tamanho da entrada?

A Importância da Eficiência

- Escala
  - Um algoritmo que funciona bem para 100 elementos pode ser impraticável para 1 milhão.

- Trade-offs
  - Frequentemente há um equilíbrio entre eficiência de tempo e espaço.

- Contexto
  - A escolha do algoritmo deve considerar o cenário específico e as restrições.

Algoritmos de Ordenação por Troca

Definição Geral

- Os algoritmos de ordenação por troca (ou exchange sorts) são métodos de ordenação baseados na **troca de posições entre pares de elementos** que estão fora de ordem.

- A ideia principal é simples:

    - Percorrer a lista comparando pares de elementos adjacentes (ou não) e trocá-los de lugar sempre que estiverem fora da ordem desejada.

- Esse processo se repete até que **nenhuma troca seja mais necessária**, o que significa que o vetor está ordenado.

Características Gerais

Característica	Descrição
Base da técnica	Comparações e trocas de pares de elementos
Estrutura usada	Geralmente um vetor ou lista
Complexidade média	O (n^2)
Tipo	Algoritmos simples e intuitivos
Uso prático	Mais comum em ensino e vetores pequenos, pois são lentos para grandes volumes de dados

Exemplos

Algoritmo	Tipo de Troca	Complexidade Média
Bubble Sort	Adjacentes	$O(n^2)$
Cocktail Sort	Adjacentes (duas direções)	$O(n^2)$
Odd-Even Sort	Adjacentes alternados	$O(n^2)$
Quick Sort	Arbitrárias (em torno de pivô)	O(n \log n)

Bubble Sort

A Ideia Principal

- Conceito Intuitivo
    - O Bubble Sort funciona como bolhas de ar na água: os elementos maiores "borbulham" para o topo (final do array) através de comparações e trocas sucessivas.
- Como Funciona
    1. Compara elementos adjacentes
    2. Troca se estiverem na ordem errada
    3. Repete até que não haja mais trocas

Como Funciona? (Passo a Passo)

Exemplo: Ordenando [5, 2, 8, 1, 9]

Estado Inicial:

Como Funciona? (Passo a Passo)

1ª Passada:

Compara 5 > 2? Sim → Troca

Compara 5 > 8? Não → Não troca

Como Funciona? (Passo a Passo)

Após 1ª Passada:

O maior elemento (9) chegou ao final!

- O processo continua até que nenhuma troca seja necessária

Desafio

Desafio

- Implemente o Bubble Sort.

Análise

Análise de Complexidade

Análise dos Casos

Pior Caso: $O(n^2)$

Array em ordem decrescente

```
[9, 8, 7, 6, 5] → Máximo de comparações e trocas
```

Comparações: $n(n-1)/2 \approx n^2/2$

Caso Médio: $O(n^2)$

Array em ordem aleatória

```
[5, 2, 8, 1, 9] → Algumas comparações e trocas
```

Em média: $n^2/4$ comparações

Melhor Caso: $O(n)$

Array já ordenado (versão otimizada)

```
[1, 2, 3, 4, 5] → Apenas comparações, sem trocas
```

Uma passada: $n-1$ comparações

Análise de Complexidade

Cálculo da Complexidade

Número de Comparações

1ª passada: (n-1) comparações

2ª passada: (n-2) comparações

3ª passada: (n-3) comparações

...

Última passada: 1 comparação

$$\text{Total} = (n-1) + (n-2) + \ldots + 1$$
$$= \mathbf{n(n-1)/2 = O(n^2)}$$

Análise de Complexidade

Complexidade de Espaço

- O(1) - Constante
    - Usa apenas uma variável temporária (temp) para as trocas, independentemente do tamanho do array.

Análise de Complexidade

Otimização Possível

Early Stop: Parar se nenhuma troca for feita em uma passada

```
bool swapped = false; // Flag para detectar trocas
```

Melhora o melhor caso para O(n)

Características, Vantagens e Desvantagens

- Simples de entender e implementar
- Não requer memória adicional (in-place)
- Estável (mantém ordem de elementos iguais)
- Detecta se o array já está ordenado
- Muito lento para arrays grandes ($O(n^2)$)
- Mais comparações que outros algoritmos $O(n^2)$
- Não é prático para uso em produção

Valor Educacional

O Bubble Sort é excelente para entender conceitos fundamentais: comparações, trocas, laços aninhados e análise de complexidade.

Algoritmos de Ordenação por Seleção: Selection Sort

Definição Geral

- Os algoritmos de ordenação por seleção (selection sorts) são baseados na ideia de **selecionar repetidamente o menor (ou maior) elemento** do conjunto de dados e **colocá-lo na posição correta** do vetor.
    - Em outras palavras: em cada passo, o algoritmo **procura o menor elemento** entre os não ordenados e o **coloca na próxima posição ordenada.**

Como Funciona

1. Divide o array em duas partes: ordenada (inicialmente vazia) e não ordenada.

2. Encontra o menor elemento na parte não ordenada.

3. Troca esse elemento com o primeiro elemento da parte não ordenada.

4. Expande a parte ordenada para incluir esse elemento.

5. Repete até que toda a lista esteja ordenada.

Características

- Simples de entender e implementar
- Número de trocas é minimizado (máximo n-1 trocas)
- Desempenho previsível (sempre $O(n^2)$)
- Não é adaptativo (não se beneficia de ordem parcial)
- Não é estável (pode alterar a ordem relativa de elementos iguais)

Como Funciona? (Passo a Passo)

Ordenando o array: [64, 25, 12, 22, 11]

Passo 1: Encontrar o menor elemento (11)

Trocar 11 com o primeiro elemento (64)

Passo 2: Encontrar o próximo menor (12)

Trocar 12 com o segundo elemento (25)

Como Funciona? (Passo a Passo)

Passo 3: Encontrar o próximo menor (22)

Trocar 22 com o terceiro elemento (25)

Resultado Final

O Selection Sort é eficiente em termos de número de trocas, realizando no máximo n-1 trocas (uma por iteração). Isso pode ser vantajoso quando o custo de troca é alto.

Desafio

Desafio

- Implemente o Selection Sort.

Análise

Análise de Complexidade

Complexidade de Tempo

Análise do Algoritmo:

- **Loop externo**: Executa (n-1) vezes
- **Loop interno**: Na primeira iteração executa (n-1) vezes, na segunda (n-2) vezes, e assim por diante
- **Total de comparações**: $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2 \approx n^2/2$

Complexidade:

- **Melhor caso**: $O(n^2)$ - Mesmo número de comparações
- **Caso médio**: $O(n^2)$
- **Pior caso**: $O(n^2)$

Análise de Complexidade

Complexidade de Espaço

- O(1) - Constante
    - Usa apenas algumas variáveis auxiliares (índices e temporária para troca)

Desafio

Desafio

- Quais são as características do Selection Sort:
  - In-place ou not-in-place?
  - Estável ou instável?
  - Vantagens e desvantagens.

Características, Vantagens e Desvantagens

- Ordenação in-place.
- Não é estável - Pode alterar a ordem relativa de elementos com valores iguais.
- Simples de entender e implementar.
- Número mínimo de trocas O(n).
- Funciona bem para arrays pequenos, mas ineficiente para grandes conjuntos de dados.
- Sempre $O(n^2)$, mesmo quando o array já está ordenado.
- Não é adaptativo (não se beneficia de ordem parcial).

Algoritmos de Ordenação por Inserção: Insertion Sort

Definição Geral

- Os **algoritmos de ordenação por inserção** (insertion sorts) constroem a sequência ordenada de forma incremental:
    - a cada passo, o algoritmo **insere o próximo elemento** na posição correta dentro da parte já ordenada do vetor.
- Em outras palavras:
    - Você vai **pegando um elemento por vez** e o **colocando na posição certa** em relação aos anteriores — como quando se organiza cartas na mão em um jogo.

Como Funciona

1. Divide o array em duas partes: ordenada (inicialmente apenas o primeiro elemento) e não ordenada.

2. Pega o primeiro elemento da parte não ordenada.

3. Insere este elemento na posição correta dentro da parte ordenada.

4. Expande a parte ordenada para incluir esse elemento.

5. Repete até que toda a lista esteja ordenada.

Características

- Simples de entender e implementar
- Eficiente para conjuntos pequenos ou quase ordenados
- Adaptativo (se beneficia de ordem parcial)
- Estável (mantém a ordem relativa de elementos iguais)
- In-place (usa memória constante)
- Ineficiente para grandes conjuntos desordenados

Como Funciona? (Passo a Passo)

Ordenando o array: [5, 2, 4, 6, 1, 3]

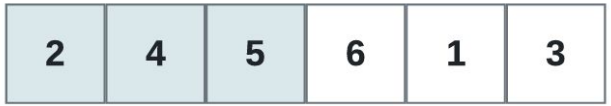
Estado inicial: Primeiro elemento já está "ordenado"

Parte ordenada: [5], Parte não ordenada: [2, 4, 6, 1, 3]

Passo 1: Inserir 2 na posição correta

Parte ordenada: [2, 5], Parte não ordenada: [4, 6, 1, 3]

Passo 2: Inserir 4 na posição correta



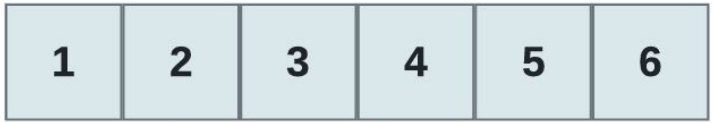
Parte ordenada: [2, 4, 5], Parte não ordenada: [6, 1, 3]

Como Funciona? (Passo a Passo)

Passo 3: Inserir 6 na posição correta

Parte ordenada: [2, 4, 5, 6], Parte não ordenada: [1, 3]

Resultado Final



Array completamente ordenado

O Insertion Sort é particularmente eficiente quando o array já está **parcialmente ordenado**. Nesse caso, muitos elementos já estarão em suas posições corretas ou próximas delas.

Desafios

Desafio

- Implemente o Insertion Sort.

Análise

Análise de Complexidade

Complexidade de Tempo

Análise do Algoritmo:

- **Loop externo**: Executa (n-1) vezes
- **Loop interno**: No pior caso, executa i vezes para cada i
- **Total de comparações (pior caso)**: $1 + 2 + ... + (n-1) = n(n-1)/2 \approx n^2/2$

Complexidade:

- **Melhor caso**: O(n) - Array já ordenado
- **Caso médio**: $O(n^2)$
- **Pior caso**: $O(n^2)$ - Array em ordem inversa

Análise de Complexidade

Complexidade de Espaço

- O(1) - Constante
    - Usa apenas algumas variáveis auxiliares (chave, índices)

Desafio

Desafio

- Quais são as características do Insertion Sort:
  - In-place ou not-in-place?
  - Estável ou instável?
  - Vantagens e desvantagens.

Características, Vantagens e Desvantagens

- Ordenação in-place.
- Estável - Mantém a ordem relativa de elementos com valores iguais.
- Adaptativo - Quanto mais ordenado o array, menos comparações e movimentações são necessárias
- Online - pode ordenar os dados à medida que chegam.
- Funciona bem para arrays pequenos, mas ineficiente para grandes conjuntos de dados.
- $O(n^2)$ no pior caso e caso médio.
- Muitas operações de deslocamento para arrays grandes.