

# Prof. esp. Thalles Canela

- **Graduado:** Sistemas de Informação - Wyden Facimp
- **Pós-graduado:** Segurança em redes de computadores - Wyden Facimp
- **Professor:** Todo núcleo de T.I. (Graduação e Pós) - Wyden Facimp
- **Diretor:** SCS
- **Gerente de Projetos:** Motoca Systems

## Redes sociais:

- **Linkedin:** <https://www.linkedin.com/in/thalles-canela/>
- **YouTube:** <https://www.youtube.com/aXR6CyberSecurity>
- **Facebook:** <https://www.facebook.com/axr6PenTest>
- **Instagram:** [https://www.instagram.com/thalles\\_canela](https://www.instagram.com/thalles_canela)
- **Github:** <https://github.com/ThallesCanela>
- **Github:** <https://github.com/aXR6>
- **Twitter:** <https://twitter.com/Axr6S>



# Por que ordenar?

- Ordenar dados é fundamental para muitas aplicações: de bases de dados a algoritmos de busca, de gráficos a análises estatísticas.
-



# Eficácia e Eficiência

- Software bem projetado precisa ser eficaz (fazer o que é esperado) e eficiente (fazer isso rapidamente). Ordenar dados adequadamente pode melhorar o desempenho, reduzir o tempo de busca e tornar o software mais responsivo.
-



# Bases de Dados

- Bancos de dados frequentemente necessitam ordenar registros para operações como busca, fusão e relatórios. Um algoritmo de ordenação eficiente pode significar a diferença entre um sistema que responde em milissegundos e um que leva vários segundos.
-



# Algoritmos de Busca

- Muitos algoritmos de busca, como a busca binária, exigem que os dados estejam ordenados. Ordenar dados pode melhorar a eficiência da busca.
-



# Usabilidade

- Do ponto de vista do usuário, listas ordenadas (seja em uma interface de usuário ou em relatórios) melhoram a usabilidade e a experiência do usuário.
-



# Manutenção e Refatoração

- Na Engenharia de Software, a manutenção é uma fase crucial. Dados bem organizados e ordenados tornam o código mais legível e, consequentemente, mais fácil de manter e refatorar.
-



# Conceitos fundamentais

- Existem diversas técnicas de ordenação, e a escolha do método depende dos dados e do problema. Algumas métricas a serem consideradas incluem eficiência, memória e estabilidade.
-





# Necessidade de Eficiência

- Na Engenharia de Software, a eficiência é crítica.
  - Os usuários esperam tempos de resposta rápidos e sistemas confiáveis.
  - A ordenação adequada dos dados pode ser a diferença entre um sistema que é percebido como 'rápido' ou 'lento'.
-



# Estabilidade na Ordenação

- A estabilidade é um conceito onde, quando dois elementos têm valores iguais, a ordem original é preservada na ordenação. Em muitos sistemas, essa estabilidade é crucial, especialmente quando lidamos com múltiplos atributos.
-

# O que é Estabilidade em Ordenação?

- **Múltiplos Critérios de Ordenação:** Em muitas aplicações, os dados podem precisar ser ordenados por mais de um critério. Por exemplo, considere uma lista de estudantes onde você primeiro ordena por nota e, em seguida, por nome. Se o algoritmo de ordenação for estável, os estudantes com a mesma nota ainda serão ordenados por nome na lista final.
  - **Preservação da Ordem Original:** Em algumas aplicações, a ordem original dos dados tem um significado ou valor intrínseco. Usando um algoritmo de ordenação estável, essa ordem original é preservada, o que pode ser crucial para a análise subsequente ou para a apresentação dos dados.
-



# Uso de Memória e Escalabilidade

- Softwares de grande escala, como sistemas de gerenciamento de banco de dados ou aplicações de nuvem, lidam com uma enorme quantidade de dados. A escolha do algoritmo de ordenação pode afetar drasticamente o uso da memória e a escalabilidade do software.
-



# Complexidade e Manutenção de Código

- Um código mais complexo pode ser mais difícil de manter. Na Engenharia de Software, é vital equilibrar a necessidade de eficiência com a legibilidade e manutenção do código.
-



# Testabilidade e Qualidade

- Cada algoritmo de ordenação deve ser rigorosamente testado para garantir a qualidade. Erros em algoritmos de ordenação podem ter consequências graves, levando a decisões de negócios erradas ou até mesmo falhas de segurança.
-



# Adequação ao Problema

- Em Engenharia de Software, é fundamental escolher a ferramenta certa para o trabalho. Nem todos os algoritmos de ordenação são adequados para todos os problemas. O entendimento profundo do problema é essencial para selecionar a técnica de ordenação adequada.
-



# Métricas para escolha de algoritmo

- **Não existe um 'tamanho único' em algoritmos de ordenação. Devemos considerar:** Tempo de execução, Uso de memória, Estabilidade, Complexidade no pior/best caso.
-





# Tempo de execução

- A rapidez com que um algoritmo pode processar dados. No entanto, a otimização prematura é a raiz de todo o mal. Nem sempre o mais rápido é o melhor no contexto do projeto.
  - **Nota da Engenharia de Software:** Um algoritmo mais simples e ligeiramente mais lento pode ser mais fácil de manter e entender por outros desenvolvedores.
-



# Uso de memória

- Quanta memória um algoritmo usa durante sua execução. Algoritmos que usam menos memória são geralmente preferidos em sistemas com restrições de memória.
  - **Nota da Engenharia de Software:** Em aplicações de tempo real ou sistemas embarcados, a memória é um recurso valioso. A escolha de um algoritmo que consome menos memória pode ser crucial.
-



# Estabilidade

- Um algoritmo de ordenação é estável se a ordem relativa dos registros com chaves iguais permanece inalterada.
  - **Nota da Engenharia de Software:** Em sistemas que dependem da consistência e previsibilidade dos dados, a estabilidade pode ser um fator determinante.
-



# Complexidade (Pior e Melhor Caso)

- Refere-se ao comportamento do algoritmo nos cenários mais desfavoráveis e favoráveis. É crucial saber como o algoritmo se comporta em grandes volumes de dados.
  - **Nota da Engenharia de Software:** Ao dimensionar sistemas para lidar com grandes conjuntos de dados ou altas demandas de usuário, entender a complexidade do algoritmo é fundamental.
-



# Manutenibilidade e Adaptabilidade

- Como é fácil para outros desenvolvedores entenderem, alterarem ou adaptarem o código no futuro.
  - **Nota da Engenharia de Software:** O código é frequentemente mantido por equipes que não o escreveram originalmente. A legibilidade e a modularidade do código são essenciais para a sustentabilidade a longo prazo do software.
-



# Bubble Sort

- Um dos algoritmos mais simples, mas não necessariamente o mais eficiente. Funciona por repetidas trocas de elementos adjacentes se estiverem na ordem errada.
-



# Visão Geral do Bubble Sort

- Funciona por repetidas trocas de elementos adjacentes se estiverem na ordem errada.
  - Apesar de ser simples, é ineficiente para grandes conjuntos de dados.
-



# Considerações da Engenharia de Software:

- **Eficiência:**
  - O Bubble Sort tem complexidade temporal de  $O(n^2)$  no pior caso, o que o torna impraticável para grandes conjuntos de dados.
-





# Considerações da Engenharia de Software:

- **Manutenção:**
  - É um algoritmo simples e, portanto, fácil de entender e manter. Mas sua ineficiência muitas vezes supera esse benefício.
-



# Considerações da Engenharia de Software:

- **Uso prático:**
  - Raramente usado em aplicações do mundo real devido à sua ineficiência. No entanto, é útil como uma introdução didática aos conceitos de ordenação.
-




# Considerações da Engenharia de Software:

- **Testabilidade:**
  - Sendo um algoritmo simples, é fácil de ser testado. Contudo, sua lenta performance pode ser um obstáculo em testes automatizados que dependem de ordenação rápida.
-

- Exemplo em C:

c

 Copy code

```
for (int i = 0; i < n-1; i++) {  
    for (int j = 0; j < n-i-1; j++) {  
        if (arr[j] > arr[j+1]) {  
            // Troca os elementos  
            int temp = arr[j];  
            arr[j] = arr[j+1];  
            arr[j+1] = temp;  
        }  
    }  
}
```

Comentário: Este código realiza o Bubble Sort em um array.



# Quick Sort

- Técnica de 'dividir para conquistar'. É uma das técnicas de ordenação mais rápidas e é amplamente utilizada.
-

- Exemplo em C (comentado):

```
c Copy code

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high-1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

Comentário: Este código realiza a ordenação Quick Sort em um array. Ele usa a técnica de dividir o array e ordenar as duas metades separadamente.



# Eficiência e Performance:

- Na engenharia de software, a performance é crítica. Escolher o Quick Sort em situações onde a divisão eficiente é possível pode otimizar a performance de uma aplicação.
-



# Manutenibilidade:

- A clareza do código é crucial. Embora o Quick Sort possa ser eficiente, a sua compreensão e manutenção podem ser desafiadoras para equipes grandes ou para engenheiros menos experientes.
-





# Escalabilidade:

- À medida que os dados crescem, a eficiência dos algoritmos de ordenação torna-se ainda mais crítica. O Quick Sort, em seu melhor cenário, tem uma complexidade  $O(n \log n)$ , tornando-o uma boa escolha para grandes conjuntos de dados.
-



# Contexto de Aplicação:

- Como engenheiros, devemos ponderar onde e quando usar o Quick Sort. Se a previsibilidade for mais valiosa que a velocidade máxima (por exemplo, em sistemas de tempo real), outros algoritmos, como o Merge Sort, podem ser mais apropriados.
-