Atividade Prática: Análise de Algoritmos de Busca em Grandes Volumes de Dados

Objetivos

- Implementar e analisar o desempenho de algoritmos de busca (sequencial e binária) em C.
- Compreender a diferença de eficiência entre as buscas sequencial e binária, especialmente com grandes volumes de dados.
- Comparar o tempo de execução e o número de comparações de cada algoritmo.
- Avaliar a importância da ordenação dos dados para a eficiência da busca.

Cenário

Você é um desenvolvedor e precisa criar um sistema eficiente para buscar informações em uma base de dados que pode conter milhões de registros. A tarefa é implementar e comparar dois algoritmos de busca para determinar qual é o mais adequado para o cenário.

Descrição da Tarefa

1. **Geração de Dados:**
    - Crie uma função em C que gere um vetor de inteiros com um tamanho definido pelo usuário (por exemplo, 100.000, 1.000.000, 10.000.000).
    - Preencha este vetor com números aleatórios.
    - Após o preenchimento, a mesma função deve **ordenar** o vetor usando um algoritmo de ordenação eficiente (como o **Quicksort** ou **Mergesort**). O uso de qsort() da biblioteca padrão de C é permitido e recomendado para simplificar esta etapa.

2. **Implementação dos Algoritmos de Busca:**
    - **Busca Sequencial (buscaSequencial)**: Implemente uma função que receba o vetor e o valor a ser buscado. A função deve retornar o índice da primeira ocorrência do valor, ou -1 caso não o encontre. A função também deve contar e retornar o número de comparações realizadas.
    - **Busca Binária (buscaBinaria)**: Implemente uma função que receba o vetor **ordenado** e o valor a ser buscado. A função deve retornar o índice do valor ou -1 caso não o encontre. Assim como a busca sequencial, a função deve contar e retornar o número de comparações realizadas.

3. **Análise e Comparação:**
    - No programa principal (main), você deve testar os dois algoritmos.
    - Crie um vetor com **10.000.000** de elementos.

- Escolha um valor para buscar. O valor pode ser um que você sabe que está no vetor, ou um que você sabe que não está. Para uma análise mais completa, realize a busca para ambos os casos.
- Para cada algoritmo de busca, meça o tempo de execução e o número de comparações. Para medir o tempo, use a função clock() da biblioteca <time.h>.
    - **Exemplo de medição de tempo:**

```c
clock_t inicio, fim;
double tempo_execucao;

inicio = clock();
// Chamada da sua função de busca
fim = clock();

tempo_execucao = (double)(fim - inicio) / CLOCKS_PER_SEC;
```

- Imprima na tela os resultados da seguinte forma:
    - **Tamanho do vetor:** [valor]
    - **Valor buscado:** [valor]
    - ---
    - **Busca Sequencial:**
    - Status: Encontrado no índice [índice] ou Não encontrado.
    - Comparações: [número de comparações]
    - Tempo de Execução: [tempo] segundos
    - ---
    - **Busca Binária:**
    - Status: Encontrado no índice [índice] ou Não encontrado.
    - Comparações: [número de comparações]
    - Tempo de Execução: [tempo] segundos

Entrega

O estudante deve entregar o código-fonte (.c) devidamente comentado e um pequeno relatório (.pdf ou .docx) respondendo às seguintes perguntas:

1. Qual a principal diferença entre os dois algoritmos de busca em termos de complexidade de tempo? Explique por que a **Busca Binária** é tão mais eficiente para grandes volumes de dados.
2. Descreva os resultados obtidos (tempo de execução e número de comparações) para o vetor de 10.000.000 de elementos. Os resultados confirmam a teoria da complexidade de tempo?
3. Qual a desvantagem da Busca Binária? Qual a importância da ordenação prévia dos dados para este algoritmo?

4. Se a base de dados fosse desordenada, qual seria a única opção de busca viável? E se fosse preciso buscar em uma base de dados desordenada, qual seria a estratégia para usar a Busca Binária?