INSTITUTO FEDERAL
Sul de Minas Gerais
Campus Passos

# TRABALHO PRÁTICO I (TPI)

| | | | |
|---|---|---|---|
| **Curso:** | Bacharelado em Ciência da Computação | **Ano:** | 2024 |
| **Disciplina:** | Estrutura de Dados II | **Período:** | 4º |
| **Professor:** | Hiran Nonato M. Ferreira | **Data:** | 30/10/2024 |
| Trabalho deve ser realizado em dupla ou em trio. | | | |
| **OBSERVAÇÕES IMPORTANTES!** | | **Valor: 2,0 pontos** | |

* Cópia de trabalhos = 0 (zero);
* Os trabalhos deverão ser enviados para *hiran.ferreira@ifsuldeminas.edu.br* até às **23h59 do dia 24/11/2024** com o assunto "[ED II 2024] Trabalho Prático I";
* Apenas um aluno deverá enviar o trabalho. Colocar os nomes dos membros no conteúdo do e-mail;
* Mesmo que não consiga elaborar todo o trabalho, não deixe de enviar e apresentar o que desenvolveu;
* As apresentações dos trabalhos **serão no dia 27/11/2024 (quarta-feira) e 29/11/2024 (sexta-feira)** segundo ordem a ser definida;
* Trabalhos enviados após a data estipulada serão penalizados em 25% por dia de atraso.

## ANÁLISE DE DESEMPENHO DE DIFERENTES ALGORITMOS DE ORDENAÇÃO EM DIFERENTES CENÁRIOS

Este Trabalho Prático consiste em analisar o desempenho de diferentes algoritmos de ordenção em diferentes cenários, que serão descritos a seguir. A análise consistirá em comparar os algoritmos considerando três métricas de desempenho:

- Número de comparações de chaves;
- Número de cópias de registros;
- O tempo total gasto para ordenação (tempo de processamento e não o tempo de relógio).

As entradas deverão obedecer a um conjunto de elementos com chaves aleatoriamente geradas (uma função específica do seu algoritmo gerará as chaves aleatórias). Para obter o tempo de processamento na linguagem C, você pode utilizar o comando getrusage, lembrando de somar os tempos gastos em modo de usuário (utime ou user time) e em modo de sistema (stime ou system time). Recomendo fortemente que o seu algoritmo seja implementado em um sistema com kernel Unix.

Para o desenvolvimento deste trabalho você(s) deverá(ão) considerar os três cenários apresentados a seguir (é obrigatório a apresentação dos três cenários):

1. Algoritmos de ordenação e Estruturas de Dados
2. Variações do MergeSort
3. HeapSort e AlunoSort

### Cenário 1: Algoritmos de ordenação e Estruturas de Dados

Você deverá avaliar o desempenho do método de ordenação QuickSort (recursivo) considerando as seguintes entradas:

- Os elementos a serem ordenados são inteiros armazenados em um vetor de tamanho N.
- Os elementos a serem ordenados são inteiros armazenados em uma lista duplamente encadeada com N elementos.

- Os elementos a serem ordenados são registros armazenados em um vetor de tamanho N. Cada registro contém:
  – Um inteiro, a chave para ordenação.
  – Dez cadeias de caracteres (strings), cada uma como 200 caracteres.
  – 1 booleano
  – 4 números reais.

Para cada tipo de dado, você deverá implementar o Quick Sort Recursivo (como apresentado em sala de aula) que recebe, como entrada, o conjunto de elementos a serem ordenados (uma Lista ou um vetor) e o número de elementos a serem ordenados. Você deverá instrumentar os algoritmos para contabilizar o número de comparações de chaves (linhas 7 e 8 do algoritmo apresentado em sala de aula) e o tempo total gasto na ordenação. Estes números deverão ser impressos ao final de cada ordenação para posterior análise. Você ainda deverá implementar funções para criação dos conjuntos de elementos aleatórios. Estas funções devem ser chamadas uma vez para cada um dos N elementos a serem ordenados. Para o caso dos elementos serem registros, a função de criação deve inicializar todos os campos do registro com valores aleatórios. Note que dois elementos podem ter a mesma chave.

Análise:

O algoritmo Quick Sort deverá ser aplicado a entradas aleatórias com diferentes tamanhos (parâmetro N). Os valores de N deverão ser 1000, 10.000, 100.000, 1.000.000, 10.000.000. Para cada valor de N você deverá gerar 5 (cinco) conjuntos de dados diferentes. O resultado de suas métricas será a média aritmética desses 5 valores, exemplo:

| **Quick Sort** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **N** | **1000** | | **10.000** | | **100.000** | | **1.000.000** | | **10.000.000** | |
| Comp = Comparações<br>Temp = Tempo Total | Comp1 | | Comp1 | | Comp1 | | Comp1 | | Comp1 | |
| | Temp1 | | Temp1 | | Temp1 | | Temp1 | | Temp1 | |
| | Comp2 | | Comp2 | | Comp2 | | Comp2 | | Comp2 | |
| | Temp2 | | Temp2 | | Temp2 | | Temp2 | | Temp2 | |
| | Comp3 | | Comp3 | | Comp3 | | Comp3 | | Comp3 | |
| | Temp3 | | Temp3 | | Temp3 | | Temp3 | | Temp3 | |
| | Comp4 | | Comp4 | | Comp4 | | Comp4 | | Comp4 | |
| | Temp4 | | Temp4 | | Temp4 | | Temp4 | | Temp4 | |
| | Comp5 | | Comp5 | | Comp5 | | Comp5 | | Comp5 | |
| | Temp5 | | Temp5 | | Temp5 | | Temp5 | | Temp5 | |
| | CompM | | CompM | | CompM | | CompM | | CompM | |
| | TempM | | TempM | | TempM | | TempM | | TempM | |

O seu programa principal deve ser organizado da seguinte maneira:

- Recebe o valor de N e o arquivo de saída para os resultados. Estes parâmetros devem ser passados pela linha de comando (argc e argv em C), como no exemplo:

quicksort 10000 saida10.txt

Resultados:

Apresente gráficos e/ou tabelas para as duas métricas pedidas, número de comparações e tempo de execução (tempo de processamento), comparando o desempenho médio do Quick Sort para os três tipos de estruturas de dados e diferentes valores de N. Discuta seus resultados.

**Cenário 2: Variações do MergeSort**

Você deverá comparar o desempenho de diferentes variações do Mergesort para ordenar um conjunto de N inteiros armazenados em um vetor. As variações do Mergesort a serem implementadas e avaliadas são:

- MergeSort Recursivo: este é o MergeSort recursivo apresentado em sala de aula.
- Mertsort Iterativo: esta é uma versão iterativa do MergeSort apresentado em sala de aula.
- MergeSort Insercao(m): esta variação modifica o MergeSort Recursivo para utilizar o algoritmo de Inserção para ordenar partições (isto é, pedaços do vetor) com tamanho menor ou igual a m. Experimente com m = 10 e m = 100.
- Merge Sort com Multiway Merge(k): Nesta variação, em vez de dividir o vetor em duas partes, você pode dividir em k partes (onde k é um número fixo maior que 2, como 3 ou 4). Cada uma das partes é ordenada recursivamente e, em seguida, todas são mescladas de uma só vez. Isso pode ser mais eficiente em termos de número de comparações em algumas situações. Experimente com k = 5 e k = 10.

Realize experimentos com as quatro variações considerando vetores aleatoriamente gerados com tamanho N = 1000, 5000, 10000, 50000, 100000, 500000 e 1000000, no mínimo. Estruture o seu programa principal como sugerido acima para facilitar a coleta e posterior análise das estatísticas de desempenho.

Apresente gráficos e/ou tabelas com os resultados obtidos. Discuta os resultados e conclusões obtidos. Qual variação tem melhor desempenho, considerando as diferentes métricas. Por quê? Qual o impacto das variações nos valores de k e de m nas versões MergeSort Insercao(m) e Merge Sort com Multiway Merge(k)?

**Cenário 3: Heap Sort X Aluno Sort**

Neste cenário você deverá analisar o Heapsort e um outro algoritmo de ordenação elaborado por você para ordenar um conjunto de N inteiros positivos, aleatoriamente gerados, armazenados em um vetor. Você deverá também apresentar, no seu relatório o segundo algoritmo escolhido, explicitando o seu funcionamento.

Realize experimentos considerando vetores aleatoriamente gerados com tamanho N=1000, 5000, 10000, 50000, 100000, 500000, 1000000, no mínimo. A comparação deve focar apenas no tempo médio de execução. Apresente gráficos e/ou tabelas com os resultados obtidos. Discuta os resultados e conclusões obtidas. Qual algoritmo tem melhor desempenho. Por quê?

**Documentação**

Além da implementação do trabalho, deverá ser escrito um resumo expandido que o documente. Para elaborá-lo de forma organizada deverá ser utilizado um dos *templates* disponíveis no site da Sociedade Brasileira de Computação (SBC), mais precisamente na seguinte URL: http://www.sbc.org.br/documentos-da-sbc/category/169-templates-para-artigos-e-capitulos-de-livros. Utilize o modelo em LaTeX (overleaf.com). O resumo deverá conter entre 4 (quatro) e 6 (seis) páginas (incluindo tabelas e figuras) e a estrutura conforme a seguir. Deverão ser utilizadas citações e outras particularidades da escrita científica.

1- Resumo: apresentação breve do problema, da solução e dos resultados obtidos;
2- Introdução: deve conter uma descrição do problema a ser resolvido e uma visão geral do que foi desenvolvido;
3- Embasamento Teórico ou Fundamentação Teórica: descrição resumida dos algoritmos objetos de estudo do presente trabalho;
4- Material e Métodos: detalhamento da solução proposta;
5- Experimentos Computacionais: apresentação dos resultados obtidos. Sugere-se incluir tabelas e/ou gráficos. É necessário mencionar a arquitetura da máquina utilizada para a experimentação;

6- Conclusão: Comentários gerais sobre o trabalho, como dificuldades encontradas na implementação, entre outras. Deverá conter uma análise dos resultados obtidos;
7- Referências: bibliografia utilizada para o desenvolvimento do trabalho.

**Critérios de Avaliação e Detalhes Adicionais**

- Devido à extensão do trabalho, antes de começarem a codificar o programa, estruturem uma sequência de pequenas tarefas a serem cumpridas.
- Divida as tarefas para evitarem sobrecarga!
- Os pontos desta atividade serão distribuídos da seguinte maneira:
  - 1,0 pontos: implementação;
  - 0,5 pontos: apresentação do trabalho (avaliação individual);
  - 0,5 pontos: documentação (resumo expandido).
- Não haverá prorrogação do prazo de entrega do trabalho;
- Outros detalhes e/ou observações poderão ser adicionadas durante o desenvolvimento deste trabalho.

*Bom estudo!*

---