Universidade Federal do Maranhão
Departamento de Informática
Disciplina: ***Estrutura de Dados II 2025.2***
Prof.: ***João Dallyson Sousa de Almeida***

Atividade Prática 1 e 2 (Individual)
Valor: 20% das avaliações 1 e 2

DESCRIÇÃO

O objetivo desta atividade é implementar algoritmos de ordenação não-estáveis (QuickSort, HeapSort e SelectSort), vistos em aulas, e aplicar as estruturas de dados Tabela Hash, Árvore AVL e Árvore Rubro-Negra, também vistos em aula, para resolver problemas de estabilidade destes algoritmos com a finalidade de preservar o custo computacional da solução.

Assim, considere que em um sistema de controle de versão como Git, commits podem possuir o mesmo timestamp devido a: processamento em batch, fusão de branches, sincronização de repositórios e commits automáticos. A ordem relativa desses commits deve ser preservada para manter a integridade do histórico. Algoritmos de ordenação não-estáveis podem alterar essa ordem, causando inconsistências.
Em sistemas como Git, SVN, ou Mercurial, o histórico de commits é fundamental para: saber a sequência exata das mudanças, encontrar quando um bug foi introduzido, voltar para versões anteriores com precisão (rollback), combinar branches corretamente (merge). Assim, você precisa implementar uma solução para ordenar o histórico de commmits por "timestamp".

Qual o problema dos Timestamps iguais?
Considere os commits a seguir:

```
new Commit("A", "dev1", "Adiciona feature X", timestamp, 1),  // Primeiro
new Commit("B", "dev2", "Remove feature Y", timestamp, 2),    // Segundo
new Commit("C", "dev1", "Corrige bug em X", timestamp, 3)     // Terceiro
```

Utilizando um algoritmo de ordenação instável, a saída errada deve ser:
Timestamp: 2024-01-15 10:00:00 → Commit C → Commit A → Commit B

Utilizando um algoritmo de ordenação estável a saída correta deve ser:
Timestamp: 2024-01-15 10:00:00 → Commit A → Commit B → Commit C

Sobre a implementação

Cada algoritmo de ordenação dever ter a versão original (apresentada em aula) e a versão estabilizada utilizando uma das estruturas, sendo que as estrutura de dados não podem ser utilizadas repetidamente. Exemplo de combinação: 1 – SelectSort + Hash, 2-QuickSort + AVL e HeapSort + AVL. Fique à vontade para escolher a combinação.

Você também deverá testar sua implementação com arquivos de entrada em formato .json com tamanhos de 1000, 10.000 e 100.000 registros. Cada arquivo de entrada terá a seguinte estrutura:

```
[
  {
    "hash": "d5fefc5",
    "autor": "pedro",
    "mensagem": "docs: atualiza documentação de serviço de email",
    "timestamp": "2024-01-07 12:38:00",
    "ordem_original": 588,
    "branch": "main",
    "arquivos_alterados": [
      "docs/OrderManager.yml",
      "docs/UtilFactory.js",
      "scripts/AuthHelper.md"
    ]
  },
  {
    "hash": "342d86b",
    "autor": "maria",
    "mensagem": "feat: adiciona funcionalidade banco de dados",
    "timestamp": "2024-01-26 02:26:00",
    "ordem_original": 927,
    "branch": "develop",
    "arquivos_alterados": [
      "src/ProductService.xml"
    ]
  },
```

OBS1: Será disponibilizada a implementação para gerar a base sintética commits.

OBS2: A chave a ser ordenada é “timestamp”. Assim, o objetivo é garantir que commits com mesmo timestamp mantenham a ordem original. Para isso, você utilizar as estruturas de dados Hash e árvores balanceadas para implementar versões estáveis dos algoritmos de ordenação requeridos.

Exemplo Prático:

Considere os dados de entrada a seguir:

```
List<Commit> commits = Arrays.asList(
    // Timestamp: 10:00:00
    new Commit("A", "alice", "msg1", timestamp10, 3), // ordem_original = 3
    new Commit("B", "bob",   "msg2", timestamp10, 1), // ordem_original = 1
    new Commit("C", "carol", "msg3", timestamp10, 2), // ordem_original = 2

    // Timestamp: 09:00:00
    new Commit("D", "david", "msg4", timestamp9,  4)  // ordem_original = 4
);
```

Resultado da ordenação correta ordenando por “timestamp”

```
Commit D: 09:00:00, ordem_original=4  ← Mais antigo (timestamp menor)
Commit A: 10:00:00, ordem_original=3  ←
Commit B: 10:00:00, ordem_original=1  ← Mesmo timestamp: PODE ter ordem qualquer
Commit C: 10:00:00, ordem_original=2  ← Mas deve ser ESTÁVEL entre execuções
```

OBS3: Uma dica é utilizar as estruturas de dados para agrupar por “timestamp”, preservando a ordem original em uma lista na estrutura. Depois ordenada pela chave principal. É apenas uma ideia, você pode implementar de outra forma.

2. Entrega

- Código fonte do programa em Java (bem identado).
- Arquivos de Entrada e saída ordenados
- Relatório com os resultados do tempo de execução comparando as implementações nos diferentes arquivos de entrada.
- Upload no SIGAA.