# Trabalho sobre Métodos de Pesquisa e Ordenação em Java

## Objetivos

- Avaliar competências e conhecimentos desenvolvidos na disciplina de Pesquisa e Ordenação, por meio de um trabalho experimental.
- Oportunizar o desenvolvimento e escrita de uma atividade nos moldes de um trabalho acadêmico, bem como aperfeiçoar a capacidade de criar, planejar, trabalhar e decidir em grupo de forma cooperativa.
- Aprimorar-se na programação Java.

## Informações gerais

- Data limite para entrega: **24/11** (segunda-feira) pelo Envio de Trabalho do AVA (**devem ser postados todos os arquivos fonte, todos os arquivos gerados, texto escrito com a descrição dos métodos utilizados e a conclusão do grupo e um relatório descrevendo como cada membro do grupo participou do trabalho**).
- **O grupo que não cumprir o prazo perderá 1 ponto por dia de atraso, sendo no máximo 3 dias.**
- Grupos formados por até 5 alunos, sendo que qualquer componente do grupo deve ser capaz de apresentar/defender todas as ideias contidas no trabalho. É fundamental que o trabalho seja feito de forma cooperativa, e não, simplesmente, que cada componente cuide de uma parte do mesmo para depois juntar e formar o todo.
- A parte escrita deverá ser feita de forma bastante sucinta e deverá estar nos padrões adotados pela FAESA para apresentação de relatório de pesquisa.

## Critérios avaliativos

- Organização do documento impresso e formatação segundo as normas de trabalho acadêmico.
- Abordagem do tema central de forma clara e precisa.
- Utilização adequada de estruturas de dados e métodos de pesquisa e ordenação.
- **Não usar estruturas prontas do JAVA (só pode usar ArrayList ou LinkedList. Não pode usar nenhum tipo de Sort, HashMap, nem algo do tipo)**
- Programa correto.
- Legibilidade do código e identação.
- Entrevista individual e/ou coletiva.
- Trabalho em equipe e de forma cooperativa.
- Pontualidade na entrega (a cada dia de atraso, o trabalho valerá 1,0 ponto a menos).

| | |
|---|---|
| Texto nas normas da FAESA | 0,5 |
| Descrição dos métodos e conclusões | 1,0 |
| Implementação dos métodos | 4,0 |
| Apresentação | 1,5 |
| TOTAL | 7,0 |

## Descrição geral do trabalho

O tema da pesquisa é o estudo de métodos de pesquisa e ordenação usando chave secundária.

## Organização da parte escrita

Introdução:
O capítulo da introdução deve descrever brevemente cada um dos métodos utilizados no trabalho e a metodologia utilizada na implementação destes métodos.

Desenvolvimento:
Apresentar as estratégias utilizadas para manter a estabilidade dos algoritmos de ordenação e fazer a pesquisa em chave secundária nos algoritmos de pesquisa.
Descrever a máquina em que o programa for rodado. Descrever os arquivos usados para teste e colocar as tabelas com os tempos de cada método para cada arquivo, assim como os cometários dizendo qual rodou mais rápido e mais lento em cada caso.

Conclusão:
Apresentar as conclusões geradas, verificando se os resultados encontrados estão de acordo com a teoria estudada em sala de aula, bem como as dificuldades encontradas para realizar o trabalho.

Bibliografia:
Listar os títulos que auxiliaram o desenvolvimento do trabalho, segundo as normas da ABNT.

## Problema a ser implementado

Serão disponibilizados 12 arquivos do tipo texto, contendo 1.000, 5.000, 10.000 e 50.000 registros, que representam dados de reserva de voos e serão compostos dos campos código da reserva, nome do passageiro, código do voo, data, assento. Os registros estarão dispostos de forma aleatória, invertida e ordenada pelo nome do passageiro. Será também fornecido um arquivo com 400 nomes que serão pesquisadas.

O formato do arquivo será:

**reserva;nome;voo;data(formato dd/mm/aaaa);assento**

O trabalho consiste em:

1) Comece a contar o tempo e faça o procedimento a seguir 5 vezes:
1.1) Carregue um arquivo dado (por exemplo reserva1000alea.txt) em um vetor ou ArrayList.
1.2) Faça a ordenação pelo nome, se tiver nome igual, pelo código da reserva, de cada arquivo disponibilizado, usando o método **Heapsort**.
1.3) Grave um arquivo com o resultado da ordenação (por exemplo heap1000alea.txt). FIM DO PROCEDIMENTO.
1.4) Após rodar o procedimento acima 5 vezes, termine de contar o tempo.
1.5) Faça a diferença entre o tempo final e o inicial e divida por 5.
1.6) Para cada arquivo original, faça todo o procedimento acima, começando do 1.1.

2) Comece a contar o tempo e faça o procedimento a seguir 5 vezes:
2.1) Carregue um arquivo dado (por exemplo reserva1000alea.txt) em um vetor ou ArrayList.
2.2) Faça a ordenação pelo nome, se tiver nome igual, pelo código da reserva, de cada arquivo disponibilizado, usando o método **Quicksort**.

2.3) Gere um arquivo com o resultado da ordenação (por exemplo quick1000alea.txt). FIM DO PROCEDIMENTO.
2.4) Após rodar o procedimento acima 5 vezes, termine de contar o tempo.
2.5) Faça a diferença entre o tempo final e o inicial e divida por 5.
2.6) Para cada arquivo original, faça todo o procedimento acima, começando do 2.1.

3) Comece a contar o tempo e faça o procedimento a seguir 5 vezes:
3.1) Carregue um arquivo dado (por exemplo multa1000alea.txt) em um vetor ou ArrayList.
3.2) Faça a ordenação pelo nome, se tiver nome igual, pelo código da reserva, de cada arquivo disponibilizado, usando o método **Quicksort, porém quando a chamada recursiva tiver 20 elementos, chame o Inserção Direta para ordenar aquela parte do vetor**.
3.3) Gere um arquivo com o resultado da ordenação (por exemplo QuickIns1000alea.txt). FIM DO PROCEDIMENTO.
3.4) Após rodar o procedimento acima 5 vezes, termine de contar o tempo.
3.5) Faça a diferença entre o tempo final e o inicial e divida por 5.
3.6) Para cada arquivo original, faça todo o procedimento acima, começando do 3.1.

4) Comece a contar o tempo e faça o procedimento a seguir 5 vezes:
4.1) Carregue um arquivo dado (por exemplo reserva1000alea.txt) em uma ABB e a balanceie. **Cuidado com os nomes iguais.**
4.2) Faça a pesquisa na ABB, usando os 400 registros fornecidos pela professora, gerando um arquivo com o resultado da pesquisa (por exemplo ABB1000alea.txt), conforme observação abaixo. FIM DO PROCEDIMENTO.
4.3) Após rodar o procedimento acima 5 vezes, termine de contar o tempo.
4.4) Faça a diferença entre o tempo final e o inicial e divida por 5.
4.5) Para cada arquivo original, faça todo o procedimento acima, começando do 4.1.

5) Comece a contar o tempo e faça o procedimento a seguir 5 vezes:
5.1) Carregue um arquivo dado (por exemplo reserva1000alea.txt) em uma AVL. **Cuidado com os nomes iguais.**
5.2) Faça a pesquisa na AVL, usando os 400 registros fornecidos pela professora, gerando um arquivo com o resultado da pesquisa (por exemplo AVL1000alea.txt), conforme observação abaixo. FIM DO PROCEDIMENTO.
5.3) Após rodar o procedimento acima 5 vezes, termine de contar o tempo.
5.4) Faça a diferença entre o tempo final e o inicial e divida por 5.
5.5) Para cada arquivo original, faça todo o procedimento acima, começando do 5.1.

6) Comece a contar o tempo e faça o procedimento a seguir 5 vezes:
6.1) Carregue um arquivo dado (por exemplo reserva1000alea.txt) em um Hashing Encadeado.
6.2) Faça a pesquisa no Hashing, usando os 400 registros fornecidos pela professora, gerando um arquivo com o resultado da pesquisa (por exemplo Hash1000alea.txt), conforme observação abaixo. FIM DO PROCEDIMENTO.
6.3) Após rodar o procedimento acima 5 vezes, termine de contar o tempo.
6.4) Faça a diferença entre o tempo final e o inicial e divida por 5.
6.5) Para cada arquivo original, faça todo o procedimento acima, começando do 4.1.

**Observações:**

1) **Como imprimir o resultado da pesquisa no arquivo?**

Se o nome for encontrado, imprima o nome, a reserva, o voo, a data e o assento de cada registro, assim como o total de reservas, de acordo com o modelo abaixo.

NOME Marcos Silva:
Reserva: XXXX Voo: XXXXX Data: XX/XX/XXXX Assento: XXX
Reserva: XXXX Voo: XXXXX Data: XX/XX/XXXX Assento: XXX
Reserva: XXXX Voo: XXXXX Data: XX/XX/XXXX Assento: XXX
TOTAL: 3 reservas

Se o nome não for encontrado, mostre que ele não tem reserva.

NOME Marcos Silva:
NÃO TEM RESERVA

**2)** Cada processo deve ser rodado **5 vezes** e o tempo deve ser a **média do tempo total**.

**3)** Para computar o tempo, utilize o método **System.currentTimeMillis()** que retorna o tempo da máquina em milissegundos, ou **System.nanoTime()** que retorna o tempo da máquina em nanossegundos.

**4)** O sistema não deve ter interface com o usuário nem menu. Ele deve ser rodado em sequência.

**5)** Todos os arquivos lidos e gerados devem estar dentro do projeto no Eclipse, Netbeans ou VsCode.

**6)** Cada processo de Ordenação (Heapsort, Quicksort, Quicksort com Inserção) deve gerar 12 arquivos de saída (1 para cada arquivo de entrada). Cada processo de Pesquisa (ABB, AVL e hashing) deve também gerar 12 arquivos de saída (1 para cada arquivo de entrada).

**7) Observe que cada processo será rodado 5 vezes e gerados 5 arquivos que podem ser sobrescritos.** Isso quer dizer que, no final, terá apenas um arquivo para cada tipo (ordenação ou pesquisa) e para cada tamanho, por exemplo, haverá apenas um arquivo heap1000alea.txt, um arquivo heap5000alea.txt, etc. Como são 12 arquivos para cada processo e são 6 processos, serão gerados 72 arquivos.

**8) Enviar os códigos fonte e os 72 arquivos gerados (TODOS)**