UNIPAM Educação que transforma	**CENTRO UNIVERSITÁRIO DE PATOS DE MINAS – UNIPAM** **BACHARELADO EM SISTEMAS DE INFORMAÇÃO** **TURMA: 4º PERÍODO – 02/2024** **ESTRUTURA DE DADOS E ALGORITMOS** **PROFESSOR RAFAEL MARINHO E SILVA**

Aluno(a):	**valor: 3 pts**

Este arquivo deve ser respondido e entregue no formato **PDF**, e enviado **individualmente** para o portal UNIPAM até o dia 04/12/2024, durante a aula.

- **Todos** os **códigos** solicitados nas atividades abaixo podem ser criados, compilados e executados usando o **VS Code, Eclipse IDE** ou o **Codespace** do GitHub.
- Utilize como referência os códigos disponíveis no seguinte link: https://drive.google.com/file/d/1gKF1HqrMUTz2P_xlsOaX7PceappjZOT0/view?usp=sharing

PRÁTICA 03

Objetivo: O objetivo dessa atividade é realizar a análise de desempenho de diferentes algoritmos de ordenação, registrando o tempo de execução para diferentes tamanhos de dados. Os alunos deverão preencher uma tabela com o tempo de execução dos algoritmos para diferentes quantidades de números e entregar o resultado em formato PDF.

Descrição da atividade:

- Nessa atividade, você irá trabalhar com 5 algoritmos de ordenação: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort e Quick Sort. Para cada um desses algoritmos, você deverá usar um conjunto de números aleatórios, e será medido o tempo de execução para diferentes quantidades de números.

- Os algoritmos devem ser executados para 5 tamanhos diferentes de entradas:
  - 100 números
  - 1.000 números
  - 100.000 números
  - 1.000.000 números
  - 10.000.000 números

- Você deve preencher a tabela abaixo com os tempos de execução de cada algoritmo para as quantidades de números especificadas.

  - A conversão de nanosegundos para segundos é simples. Como 1 segundo equivale a 1 bilhão de nanosegundos, a conversão é feita dividindo o valor em nanosegundos por 1.000.000.000 (ou 10^9). ***Fórmula: segundos = nanosegundos / 1.000.000.000***

  - **Exemplo:** Se você tiver 500.000.000 de nanosegundos, a conversão para segundos seria:
    segundos = 500.000.000 / 1.000.000.000
    = 0,5 segundos ou seja, 500 milhões de nanosegundos equivalem a 0,5 segundos.

Quantidade de Números	Bubble Sort (segundos)	Insertion Sort (segundos)	Selection Sort (segundos)	Merge Sort (segundos)	Quick Sort (segundos)
100	0,0068089 s	0,000396389 s	0,000520982 s	0,000377234 s	0,000490505 s
1.000	0,009679176 s	0,004956156 s	0,008129417 s	0,001732422 s	0,002249406 s
100.000	15,200847413 s	0,679088108 s	5,272640104 s	0,028816081 s	0,023637891 s
1.000.000	-	-	-	0,169934733 s	0,124298138 s
10.000.000	-	-	-	2,108353506 s	1,298405634 s

- Compilar e executar o código: Compile e execute os códigos fornecidos em anexo. Eles contêm a implementação dos algoritmos de ordenação (Bubble Sort, Insertion Sort, Selection Sort, Merge Sort e Quick Sort) e os arquivos com os números desordenados.

Insira os prints abaixo para a execução de cada algoritmo:

- Print para 100 números

```
@eivini →/workspaces/SI_UNIPAM/Estruturas de dados e algoritmos/aulaPratica03 (main) $ java SortTest
BubbleSort time: 680890 nanoseconds
Números ordenados com bubble foram salvos em: numeros_tamanho100_bubble.txt
InsertionSort time: 396389 nanoseconds
Números ordenados com insertion foram salvos em: numeros_tamanho100_insertion.txt
SelectionSort time: 520982 nanoseconds
Números ordenados com selection foram salvos em: numeros_tamanho100_selection.txt
MergeSort time: 377234 nanoseconds
Números ordenados com merge foram salvos em: numeros_tamanho100_merge.txt
QuickSort time: 490505 nanoseconds
Números ordenados com quick foram salvos em: numeros_tamanho100_quick.txt
```

- Print para 1.000 números

```
@eivini →/workspaces/SI_UNIPAM/Estruturas de dados e algoritmos/aulaPratica03 (main) $ java SortTest
BubbleSort time: 9679176 nanoseconds
Números ordenados com bubble foram salvos em: numeros_tamanho1000_bubble.txt
InsertionSort time: 4956156 nanoseconds
Números ordenados com insertion foram salvos em: numeros_tamanho1000_insertion.txt
SelectionSort time: 8129417 nanoseconds
Números ordenados com selection foram salvos em: numeros_tamanho1000_selection.txt
MergeSort time: 1732422 nanoseconds
Números ordenados com merge foram salvos em: numeros_tamanho1000_merge.txt
QuickSort time: 2249406 nanoseconds
Números ordenados com quick foram salvos em: numeros_tamanho1000_quick.txt
```

- Print para 100.000 números

```
@eivini →/workspaces/SI_UNIPAM/Estruturas de dados e algoritmos/aulaPratica03 (main) $ java SortTest
BubbleSort time: 15200847413 nanoseconds
Números ordenados com bubble foram salvos em: numeros_tamanho100000_bubble.txt
InsertionSort time: 679088108 nanoseconds
Números ordenados com insertion foram salvos em: numeros_tamanho100000_insertion.txt
SelectionSort time: 5272640104 nanoseconds
Números ordenados com selection foram salvos em: numeros_tamanho100000_selection.txt
MergeSort time: 28816081 nanoseconds
Números ordenados com merge foram salvos em: numeros_tamanho100000_merge.txt
QuickSort time: 23637891 nanoseconds
Números ordenados com quick foram salvos em: numeros_tamanho100000_quick.txt
```

- Print para 1.000.000 números

```
@eivini →/workspaces/SI_UNIPAM/Estruturas de dados e algoritmos/aulaPratica03 (main) $ java SortTest
MergeSort time: 169934733 nanoseconds
Números ordenados com merge foram salvos em: numeros_tamanho1000000_merge.txt
QuickSort time: 124298138 nanoseconds
Números ordenados com quick foram salvos em: numeros_tamanho1000000_quick.txt
```

- Print para 10.000.000 números

```
@eivini →/workspaces/SI_UNIPAM/Estruturas de dados e algoritmos/aulaPratica03 (main) $ java SortTest
MergeSort time: 2108353506 nanoseconds
Números ordenados com merge foram salvos em: numeros_tamanho10000000_merge.txt
QuickSort time: 1298405634 nanoseconds
Números ordenados com quick foram salvos em: numeros_tamanho10000000_quick.txt
```