Algoritmos e Estruturas de Dados III
"Ordenação Interna"

1^a Avaliação (Programa) - Valor: 2,5 pontos

Flávio Augusto de Freitas

30 de maio de 2022

1 Objeto da Avaliação

Obrigatoriamente seu programa deve fazer o seguinte:

1. Ordenar 20 inteiros com 3 algoritmos de ordenação, a saber: Insertion Sort, Merge Sort e Quick Sort;
   (a) No caso específico do relatório final em tela, você deve ordenar 5000, 10000 e 30000 valores inteiros.
2. O mesmo vetor de inteiros deve ser usado nos algoritmos de ordenação requeridos, com o objetivo de comparação;
3. Para todas as ordenações, você precisa colher as seguintes informações:
   - Quantidade de acessos ao vetor que está sendo ordenado;
   - Quantidade de comparações entre elementos do vetor que está sendo ordenado;
   - No caso específico de Insertion Sort:

    - Quantidade de inserções de elementos do vetor;
- No caso específico de Quick Sort:
    - Quantidade de vezes que um pivô foi selecionado;
    - Quantidade de trocas entre elementos do vetor;
- Tempo total em milisegundos (com precisão de 1 milisegundo) para ordenação em cada algoritmo de ordenação.

2 Funções que você deve implementar

Escolha uma das linguagens para implementar seu código:

- Subseção 2.1 Caso escolha a linguagem C ANSI: ver página 2;
- Subseção 2.2 Caso escolha a linguagem Java: ver página 4.

2.1 Caso escolha a linguagem C ANSI

2.1.1 Especificações Obrigatórias

Não há quantidade máxima de inteiros a ordenar, mas para este trabalho em específico, o máximo deve ser 20 inteiros para apresentação em laboratório. Isto é importante, se você planeja implementar as animações das ordenações. Para o relatório final em tela, você deve ordenar 5000, 10000 e 30000 valores inteiros. Para todas as medições para o relatório final em tela deve ser usado o mesmo vetor original aleatório. Ou seja, o mesmo vetor aleatório original deve ser usado para ordenar os 5000, 10000 e 30000 valores inteiros. A sugestão é gerar um vetor de 30000 valores inteiros aleatórios e então ordenar os 5000, 10000 e 30000 valores iniciais para cada fase de medição.

```
#define MAX_INTEIROS 20
```

2.1.2 Funções Obrigatórias

Implemente em seu programa, **obrigatoriamente usando os nomes abaixo**, as funções cujos protótipos estão dados a seguir.

`void copia_vetor(int vin[], int vout[], int n);`
Copia os `n` elementos do vetor `vin` para o vetor `vout`. Assim, você pode ordenar sempre um vetor idêntico todas as vezes.

`void imprime_vetor(int v[], int n);`
Imprime os `n` elementos iniciais do vetor `v`. Assim, pode-se acompanhar as alterações que estão sendo realizadas no vetor durante a ordenação.

`void le_vetor(int v[], int n);`
Lê `n` elementos, fornecidos pelo usuário, para o vetor `v`.

`void ordena_vetor_insertion_sort(int v[], int n);`
Ordena os `n` elementos iniciais do vetor `v` usando o método Insertion Sort. Dentro deste procedimento imprima uma mensagem informando qual algoritmo foi escolhido, por exemplo:

```
cout << "Ordenacao InsertionSort..." << endl;
```

ou

```
printf("Ordenacao InsertionSort...\n");
```

`void ordena_vetor_merge_sort(int v[], int n);`
Ordena os `n` elementos iniciais do vetor `v` usando o método Merge Sort. Dentro deste procedimento imprima uma mensagem informando qual algoritmo foi escolhido, por exemplo:

```
cout << "Ordenacao MergeSort..." << endl;
```

ou

```
printf("Ordenacao MergeSort...\n");
```

`void ordena_vetor_quick_sort(int v[], int n);`

Ordena os `n` elementos iniciais do vetor `v` usando o método Quick Sort. Dentro deste procedimento imprima uma mensagem informando qual algoritmo foi escolhido, por exemplo:

```
cout << "Ordenacao QuickSort..." << endl;
```

ou

```
printf("Ordenacao QuickSort...\n");
```

`void imprime_resultados();`

Imprime os resultados obtidos durante as ordenações. Neste procedimento deve ser impressa uma tabela para comparações de todos os valores obtidos durante a execução do programa, por exemplo:

```
Vetor original: {  3  5  7  1  3  2  9  8 10  8  1  9 17 22  6  4  0 -4 33 12 ... }
Vetor ordenado: { -4  0  1  1  2  3  3  4  5  6  7  8  8  9  9 10 12 17 22 33 ... }

ORDENACAO                                          VETOR
----------------------------  ------------------------------------------------------
METODO      TEMPO TOTAL (ms)      ACESSOS           COMPARACOES          TROCAS
             5000 10000 30000   5000 10000 30000   5000 10000 30000   5000 10000 30000
=========  ===== ===== =====  ===== ===== =====  ===== ===== =====  ===== ===== =====
Insertion  3.567 5.333 7.567    100   120   150    100   100   100     50    50    50 (20 insercoes)
Merge      1.333 2.567 3.333     88   100   120     50    50    50     33    45    65
Quick      0.567 1.567 2.567     99   110   140    120   150   145     22    30    45 (30 pivos selecionados)
```

2.2 Caso escolha a linguagem **Java**

2.2.1 Especificações Obrigatórias

Não há quantidade máxima de inteiros a ordenar, mas para este trabalho em específico, o máximo deve ser 20 inteiros para apresentação em laboratório. Isto é importante, se você planeja implementar as animações das ordenações. Para o relatório final em tela, você deve ordenar 5000, 10000 e 30000 valores inteiros. Para todas as medições para o relatório final em tela deve ser usado o mesmo vetor original aleatório. Ou seja, o mesmo vetor aleatório original deve ser usado para ordenar os 5000, 10000 e 30000 valores inteiros. A

sugestão é gerar um vetor de 30000 valores inteiros aleatórios e então ordenar os 5000, 10000 e 30000 valores iniciais para cada fase de medição.

2.2.2 Funções Obrigatórias

Implemente em seu programa, **obrigatoriamente usando os nomes abaixo**, as funções cujos protótipos estão dados a seguir.

`imprimeVetor(int vetor[], int n)`
Imprime os `n` elementos iniciais do vetor `vetor`. Assim, pode-se acompanhar as alterações que estão sendo realizadas no vetor durante a ordenação.

`ordenaVetorInsertionSort(int vetor[], int n)`
Ordena os `n` elementos iniciais do vetor `vetor` usando o método Insertion Sort. Dentro deste procedimento imprima uma mensagem informando qual algoritmo foi escolhido, por exemplo:

```
System.out.printf("Ordenacao InsertionSort...\n");
```

`ordenaVetorMergeSort(int vetor[], int n)`
Ordena os `n` elementos iniciais do vetor `vetor` usando o método Merge Sort. Dentro deste procedimento imprima uma mensagem informando qual algoritmo foi escolhido, por exemplo:

```
System.out.printf("Ordenacao MergeSort...\n");
```

`ordenaVetorQuickSort(int vetor[], int n)`
Ordena os `n` elementos iniciais do vetor `vetor` usando o método Quick Sort. Dentro deste procedimento imprima uma mensagem informando qual algoritmo foi escolhido, por exemplo:

```
System.out.printf("Ordenacao QuickSort...\n");
```

`imprimeResultados()`

Imprime os resultados obtidos durante as ordenações. Neste procedimento deve ser impressa uma tabela para comparações de todos os valores obtidos durante a execução do programa, por exemplo:

```
Vetor original: {  3  5  7  1  3  2  9  8 10  8  1  9 17 22  6  4  0 -4 33 12 ... }
Vetor ordenado: { -4  0  1  1  2  3  3  4  5  6  7  8  8  9  9 10 12 17 22 33 ... }

         ORDENACAO                                 VETOR
----------------------------  -------------------------------------------------------
METODO      TEMPO TOTAL (ms)      ACESSOS          COMPARACOES          TROCAS
             5000 10000 30000   5000 10000 30000   5000 10000 30000   5000 10000 30000
=========  ===== ===== =====  ===== ===== =====  ===== ===== =====  ===== ===== =====
Insertion  3.567 5.333 7.567    100   120   150    100   100   100     50    50    50 (20 insercoes)
Merge      1.333 2.567 3.333     88   100   120     50    50    50     33    45    65
Quick      0.567 1.567 2.567     99   110   140    120   150   145     22    30    45 (30 pivos selecionados)
```

3 O Programa

Usando as funções acima, escreva um programa obrigatoriamente usando, ou a linguagem C ANSI (ou C padrão) — não pode ser C++ — ou a linguagem Java. **Outras linguagens também não poderão ser usadas, como C# ou Python.** que ordene o vetor de inteiros usando os três algoritmos de ordenação requeridos e colha informações de desempenho durante as ordenações.

A saída do programa será uma tabela — como a mostrada anteriormente — comparando o desempenho de cada algoritmo de ordenação escolhido.

4 Observações

O seu programa deve começar com um cabeçalho (uma sequência de linhas de comentários) como o seguinte:

```
/*************************************************************/
/* Aluno: Fulano de Tal                                      */
/* Matrícula: 9999-22                                        */
/* Curso: Ciência da Computação                              */
/* 1º Trabalho Prático -- Ordenação Interna                  */
/* DCC288 -- 2022 -- IFSEMG, 3o.                             */
/* Prof. Flávio Augusto de Freitas                           */
/* Compilador: ... (gcc ou Code::Blocks) versão ...          */
/* Sistema Operacional: ...                                  */
/*************************************************************/
```

O trabalho prático é estritamente em dupla. Veja a política do Departamento Acadêmico de Ciência da Computação para casos de plágio ou cola.

5 Entrega

A entrega do trabalho será no dia 30 de maio de 2022 até 23h:59m:59s (data e hora que o SIGAA encerrará o período de envio), impreterivelmente. **Exercícios atrasados NÃO serão aceitos pelo SIGAA e não aceitarei envios por e-mail institucional, nem através de redes sociais.**

Programas com erros de sintaxe (ou seja, erros de compilação) receberão nota ZERO. Seu programa deve ser compilável sem erros ou *warnings*, da maneira especificada abaixo (que usa o compilador num modo em que quase todos os *warnings* são emitidos).

Para compilar seu programa em linguagem C ANSI, use o `gcc` ou o `Code::Blocks` (que na verdade chama o `gcc` para fazer a compilação). Caso

você use diretamente o `gcc`, passe ao compilador (na linha de comando) as seguintes opções:

```
-Wall -ansi -pedantic -O2 -U_FORTIFY_SOURCE
```

Exemplo 5.1 (Compilando `tp1-999922-888822.c` para `tp1.exe`) *Supondo que seu programa-fonte* `tp1-999922-888822.c` *em C ANSI esteja salvo na pasta* `trabalho` *no diretório raiz da unidade* `C:` *do seu HD, digite:*

```
C:\trabalho>gcc tp1-999922-888822.c -o tp1.exe -Wall -ansi
-pedantic -O2 -U_FORTIFY_SOURCE
```
□

Caso você use o **Code::Blocks**, entre em Settings » Compiler and debugger... » Compiler settings » Compiler Flags, selecione as quatro opções correspondentes a **-Wall**, **-ansi**, **-pedantic** e **-O2**, e clique em OK. Entre também em Settings » Compiler and debugger... » Compiler settings » Other options, digite `-U_FORTIFY_SOURCE` na caixa de texto **Other options** e clique OK.

Caso você use Java, use qualquer IDE de sua preferência.

Para compilar seu programa em linguagem Java, use o `javac` ou alguma IDE de sua preferência, como Eclipse, IntelliJ IDEA, NetBeans etc. Caso você use diretamente o `javac`, passe ao compilador (na linha de comando) as seguintes opções:

Exemplo 5.2 (Compilando `tp1-999922-888822.java` para `tp1.class`) *Supondo que seu programa-fonte* `tp1-999922-888822.java` *em Java esteja salvo na pasta* `trabalho` *no diretório raiz da unidade* `C:` *do seu HD, digite:*

```
C:\trabalho>javac tp1-999922-888822.java tp1.class
```
□

Seu programa deve estar bem endentado, documentado e organizado. A endentação deve deixar clara a estrutura de subordinação dos comandos. Os comentários devem ser esclarecedores. Toda função deve ser precedida de um comentário que diz o que a função faz. As funções devem ser razoavelmente pequenas, na medida do possível, e cada uma delas deve ter um propósito

bem definido. A saída do programa deve ser clara. A avaliação levará em conta todos estes pontos! Uma apresentação ruim, ou a falta de clareza do programa ou da saída do programa, poderá prejudicar sua nota.

O PROGRAMA DEVE SER ENTREGUE ATRAVÉS DO SISTEMA SIGAA.

Entregue apenas o **programa-fonte em linguagem C ANSI** ou Java, num arquivo com o nome **tp1-<sua-matrícula>-<matrícula-do-colega>.c** ou **tp1-<sua-matrícula>-<matrícula-do-colega>.java**.

Exemplo 5.3 (Trabalho em C ANSI) *Se seu número de matrícula for 9999-17 e a de seu colega for 8888-16, você deverá entregar um arquivo com o nome* `tp1-999917-888816.c` *ou* `tp1-999917-888816.zip` *caso tenha vários arquivos. (Note que não há espaços no nome do arquivo; no caso de arquivo compactado, não compacte o* `.exe`*, pois isso só fará o tamanho do arquivo de envio aumentar desnecessariamente.)*

Exemplo 5.4 (Trabalho em Java) *Se seu número de matrícula for 9999-17 e a de seu colega for 8888-16, você deverá entregar um arquivo com o nome* `tp1-999917-888816.java` *ou* `tp1-999917-888816.zip` *caso tenha vários arquivos. (Note que não há espaços no nome do arquivo; no caso de arquivo compactado, não compacte o* `.class`*.)*

A primeira versão do programa entregue até o prazo final de entrega será considerada como a entrega do trabalho, não mais sendo aceitas outras versões. Então, só envie quando tiver certeza de que o programa atende todas as especificações. Encerrado o prazo, não será aceita nova entrega do trabalho, desconsiderando qualquer envio posterior. **Não deixe para entregar seu programa na última hora!**

Guarde uma cópia do seu programa pelo menos até o final do semestre.