BCC202 – Estruturas de Dados I (2023-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (www.decom.ufop.br/)

AULA PRÁTICA 10

- **Data de entrega: Até 15 de dezembro às 23:55.**

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Ordenação Híbrida

O estado de Minas Gerais lançou um concurso para a área de educação. Eles exigiram o uso do RG para o cadastro. O número de pessoas que se cadastraram é gigantescos. Para facilitar a procura pelo RG das pessoas, primeiro o conjunto de dados deve ser ordenado de forma crescente.

Sua tarefa é justamente ordenar os dados. Como você é um programador sagaz, você irá utilizar o *Merge Sort* para ordenar o vetor. Contudo, você irá adicionar uma melhoria. Quando os sub-vetores chegarem ao tamanho de 10 ou menos, você irá usar o *Insertion Sort*.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Você deve implementar o *Merge Sort* para vetores maiores que 10, e o *Insertion Sort* para vetores menores ou iguais que 10.

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivos-fonte.
- Há vários casos de teste. Você terá acesso (entrada e saída) de casos específicos para realizar os seus testes.

Especificação da Entrada e da saída

A primeira linha da entrada é um número inteiro *n* com o número de RGs e depois seguem os *n* RGs.

A saída é composta pelo número de vezes que o *Insertion Sort* foi chamado na primeira linha e a segunda, os cinco primeiros valores do vetor ordenado.

Entrada	Saída
10 7 2 1 4 9 10 5 6 3 8	1 1 2 3 4 5

Diretivas de Compilação

```
$ gcc -c sort.c -Wall
$ gcc -c pratica.c -Wall
$ gcc sort.o pratica.o -o exe
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta valgrind. Um exemplo de uso é:

```
gcc -g -o exe *.c -Wall; valgrind --leak-check=yes -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
==38409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed:  0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.