Trabalho 1 - Ordenação - CIX56

Paulo R. Lisboa de Almeida

1^o Semestre - 2024

1 Descrição

Implemente os seguintes algoritmos em C:

Versões **recursivas** de: *Busca Sequencial* (ingênua), *Busca Binária*, *Insertion Sort*, *Selection Sort* e *Merge Sort*.

Versões **iterativas** de: *Busca Sequencial* (ingênua), *Busca Binária*, *Insertion Sort* e *Selection Sort*.

O trabalho tem peso de 20% da nota do semestre.

2 Template

Os algoritmos devem ser implementados utilizando como base o template disponibilizado no Moodle.

O arquivo *ordenacao.h* contém os protótipos (assinaturas) das funções que obrigatoriamente devem ser implementadas, no arquivo *ordenacao.c*.

Não é permitida a mudança nos protótipos de funções disponibilizados no arquivo *ordenacao.h*. É permitida a criação de funções auxiliares se necessário, mas **não** é permitida a criação de arquivos de header (.h) ou de implementação (.c) extras.

É obrigatória a inclusão de um arquivo *main.c* com alguns testes nos algoritmos implementados. A definição dos testes fica a cargo do aluno.

O arquivo deve compilar sem erros ou avisos através do comando *make*.

O nome do binário gerado deve ser *trab1SeuGRR* (e.g., *trab1grr1234*). Ajuste isso no *makefile*. O binário deve executar em sistemas Linux executando o programa sem a passagem de nenhum parâmetro (e.g., *./trab1grr1234*).

3 Programação Segura e Boas Práticas

É obrigatório que os programas sigam as normas e boas práticas de programação em C segura, como o uso de tipos de dados corretos para definir os tamanhos dos vetores, inteiros de tamanho fixo quando necessário, e verificação dos vetores alocados (o *malloc* realmente conseguiu alocar?).

É de responsabilidade do aluno pesquisar sobre esses conceitos, e aplicá-los corretamente. Uma boa referência é Seacord, R. C. (2020). Effective C: An Introduction to Professional C Programming.

Alguns exemplos e experimentos também estão disponíveis em: <https://prlalmeida.com.br/knowledge-base>.

4 Arquivos a serem entregues

Você deve incluir em um diretório compactado tar.gz (é obrigatório que o arquivo seja .tar.gz – arquivo *tarball* compactado via *Gzip*) de nome trab1SeuGRR.tar.gz. Se, por exemplo, seu GRR é 1234, o diretório contendo os arquivos deve se chamar trab1grr1234. Compacte esse diretório, sendo que a versão compactada vai se chamar trab1grr1234.tar.gz. O diretório deve conter o seguinte:

- arquivos de código fonte .c e .h do programa;
- makefile;

Não inclua quaisquer outros arquivos irrelevantes para a compilação do projeto. Por exemplo, não inclua binários compilados, ou arquivos objeto (.o). A inclusão de arquivos irrelevantes pode acarretar em descontos de nota.

5 Entrega

O trabalho deve ser entregue via UFPRVirtual. A data limite para o envio está estipulada no link de entrega. Não serão aceitas entregas em atraso, exceto para os casos explicitamente amparados pelas resoluções da UFPR.

6 Dicas

6.1 Contando o tempo

Para contar o tempo gasto por determinada função em C, inclua a biblioteca *time.h*. Veja a seguir um exemplo para contar o tempo em segundos necessário para se executar uma função.

```
#include <time.h>

//...
clock_t start, end;//variáveis do tipo clock_t
double total;

start = clock();//start recebe o "ciclo" corrente
minhaFuncao();//chama a função que desejamos medir o tempo
end = clock();//end recebe o "ciclo" corrente
//o tempo total é a diferença dividia pelos ciclos por segundo
total = ((double)end - start)/CLOCKS_PER_SEC;
//total agora possui o tempo em segundos
printf("Tempo total: %f", total);
```

6.2 Função clock

A especificação em C da função clock()[1] é clara em dizer que a função deve retornar o tempo de CPU (*CPU time*)[2], mas algumas fontes citam que o Windows ignora a especificação e conta o tempo de parede (*Wall clock time*)[3]. Para simplificar, será aceito tanto o *CPU time* quanto o *Wall Clock time*, mas deixe claro em seu relatório qual sistema operacional você usou.

Para tornar os testes mais confiáveis, execute os programas com "tudo fechado" em sua máquina. Por exemplo, não abra o navegador web enquanto executar os testes, já que o navegador pode competir por recursos, e gerar diferenças nos tempos, especialmente para o *wall clock time*.

7 Distribuição da Nota

Alguns descontos padrão, considerando uma nota entre 0 e 100 pontos para o trabalho:

- Plágio: perda total da pontuação para todos os envolvidos. Isso é válido mesmo para casos onde o plágio se refere a apenas um trecho do código.
- Algoritmos não recursivos ou não iterativos (dependendo da versão solicitada), ou implementados em linguagem diferente de C, serão desconsiderados.
- Não submissão via UFPRVirtual acarreta na perda total dos pontos.
- Falta de algum arquivo requisitado: desconto de 10 a 100 pontos.
- Inclusão de arquivos desnecessários (lixo): desconto de 5 a 20 pontos.
- Erros e avisos de compilação: desconto de 5 a 100 pontos.
- Nomes de arquivo incorretos: 5 pontos por arquivo.
- Arquivo com formato incorreto: 5 pontos por arquivo.
- Más práticas de programação (ex.: código obscuro, tipos incorretos de variáveis, má identação, ...): 5 a 50 pontos.

8 Demais Regras

- Dúvidas ou casos não especificados neste documento podem ser discutidos com o professor até a **data de entrega do trabalho**. **Não** serão aceitas reclamações após a data da entrega.
- Os alunos podem (e devem) procurar o professor, ou seus colegas de classe para tirar dúvidas quanto ao trabalho.
- O descumprimento das regras dispostas nesse documento podem acarretar na perda parcial ou total da nota do trabalho.

[1] Ver Seção 7.23.2.1 de <http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf>
[2] <https://en.wikipedia.org/wiki/CPU_time>
[3] <https://en.wikipedia.org/wiki/Elapsed_real_time>