Trabalho de Análise de Algoritmos – ECOM03A

UNIFEI - Universidade Federal de Itajubá
Prof. João Paulo R. R. Leite (joaopaulo@unifei.edu.br)

Atividade **INDIVIDUAL.**

Formato: Arquivo PDF ÚNICO contendo um relatório das atividades. Qualquer outro formato será desconsiderado e o(a) aluno(a) não receberá nota.
Avaliação: O relatório valerá 2 pontos na nota N1 (20%).

Escreva um programa em C++ para a **ordenação de um vetor de palavras**, ou seja, objetos do tipo *string*. O arquivo "aurelio40000.txt", disponibilizado em conjunto com este documento, contém 40 mil palavras em português, que devem ser utilizadas para a criação dos vetores. Lembre-se de utilizar, durante o desenvolvimento, a biblioteca **<string>** de C++, que facilita (e muito) as comparações e atribuições de valores de *strings* (nada de usar *strcmp* e *strcpy* do C). A maneira que você irá utilizar para preencher os vetores com as palavras provenientes do arquivo, é por sua conta e criatividade.

Você deverá criar vetores de diferentes tamanhos (1000, 5000, 10000, 15000, 20000, 30000 e 40000 palavras) e, para cada um deles, aplicar alguns algoritmos de ordenação. São eles:

- **Bubble** Sort, **Selection** Sort, **Insertion** Sort
- **Shell** Sort, **Merge** Sort, **Quick** Sort

Você deve marcar o **tempo da execução** de cada algoritmo para cada tamanho de vetor, e criar uma **tabela** com estes dados. A seguir, a partir da tabela, e utilizando seu software de planilha eletrônica favorito (Excel, Calc), você deverá criar 3 diferentes gráficos. Neles, o eixo X será sempre o tamanho do vetor, que chamaremos de N, e o eixo Y terá os valores de tempo gasto para a execução (em milissegundos). Acrescente uma legenda, para ficar claro qual curva é relativa a qual método. São eles:

1. Gráfico de linhas com as curvas obtidas apenas para Insertion, Selection e Bubble Sort.
2. Gráfico de linhas com as curvas obtidas apenas para Shell, Merge Sort e Quick Sort.
3. Gráfico de linhas com as curvas obtidas para os 6 algoritmos.

A seguir, escreva alguns parágrafos explicando o que se pode concluir dos gráficos obtidos, comparando o desempenho de cada algoritmo, mostrando qual deles é a melhor escolha e a pior escolha e por quê. É importante especificar a ordem de complexidade de cada um deles (O(n)? $O(n^2)$?). Para suas conclusões, leve em consideração tanto seu desempenho quanto a dificuldade para sua codificação.

Não se esqueça de que o documento a ser entregue é um **RELATÓRIO**. Crie as seções obrigatórias: Título, Objetivo, Código-Fonte (com comentários, mas sem exageros), Testes e Resultados, Discussão, Conclusão. A organização e a aparência do trabalho também serão avaliadas. O trabalho é **INDIVIDUAL**.

Bom trabalho!

Como medir o tempo de execução?

Acrescente ao programa as bibliotecas, variáveis e chamadas de funções necessárias para a contagem do tempo de execução do programa. Ao final de cada chamada de método, mostre o tempo em milissegundos gasto em seu processamento. **Faça todas as medições em um mesmo computador, para que a comparação de tempo seja justa e faça sentido.** Siga o exemplo abaixo:

```
#include <ctime>

int main() {
  int tempo_ini, tempo_fin, tempo_ms;

  tempo_ini = (int) clock(); // Pega o tempo imediatamente anterior à chamada
  bubble_sort(vet, n); // faz chamada do método (por exemplo, bubble sort)
  tempo_fin = (int) clock(); // Pega o tempo imediatamente posterior à chamada

  // Calcule a diferença de tempo em milissegundos (*1000)
  tempo_ms = ((tempo_fin - tempo_ini)*1000/CLOCKS_PER_SEC);

  return 0;
}
```

1. Acrescente a biblioteca **<ctime>** (possui *clock()* e *CLOCKS_PER_SEC*)**.**
2. Utilize a função **clock***()* para retornar o tempo atual de execução do seu programa. Um inteiro retornado pela função clock deve sempre ser comparado a uma outra chamada prévia da mesma função. Seu valor absoluto é pouco significativo para nosso contexto, mas está relacionado ao tempo passado desde o início da execução. Seu valor é dado em "*clock ticks*", que são unidades de tempo constantes, mas que variam de sistema para sistema.
3. A quantidade de *clock ticks* por segundo específica do seu sistema é dada pela constante **CLOCKS_PER_SEC**. Por isso é necessária a normalização no final.

E como fazer o gráfico?

Execute o programa para os diferentes tamanhos de entrada e anote o tempo gasto por cada método. A seguir, plote um gráfico de linhas com os valores obtidos, utilizando a ferramenta de planilha eletrônica disponível em sua máquina (**Microsoft Excel** ou **Libre Office Calc**). Por exemplo:

n	Bubble	Selection
1000	2	1
5000	44	26
10000	196	100
15000	1387	623
20000	5691	2473
30000	12990	5537
40000	23206	9782

* exemplos de dados