Projeto: Desempenho de Algoritmos de Ordenação

A parte prática da avaliação P1 de nossa disciplina consiste na elaboração de um programa, para medição experimental do desempenho dos algoritmos de ordenação apresentados em sala de aula (aula + seminário), com a execução de um conjunto de experimentos para medição de tempo de execução para ordenação de vetores compostos por números inteiros gerados aleatoriamente. **Também deverá ser elaborado um relatório,** com os resultados coletados durante o experimento, devidamente planilhados.

O programa para medição experimental de tempo deve contar com um menu, para a escolha dos algoritmos a serem testados, e atender aos seguintes requisitos:

- O programa deverá ser implementado com os três algoritmos estudados em sala de aula e os sete apresentados nos seminários. Os nove algoritmos que farão parte do seu programa são relacionados abaixo:

    - Apresentados em sala: BubbleSort, InsertionSort, SelectionSort;
    - Apresentados no seminário: ShellSort, MergeSort, QuickSort, HeapSort, RadixSort (somente LSD), CountigSort e TimSort.

- Após o passo de escolha do algoritmo de ordenação para teste em um primeiro menu, seu programa deverá criar um sub-menu, para a escolha da quantidade de elementos a serem ordenados, que estarão disponíveis em valores pré-definidos, a saber: 1.000, 5.000, 10.000, 20.000, 50.000 e 100.000.
- Para cada bateria de testes em cada algoritmo, o programa deverá criar um vetor dinâmico (**alocação de memória obrigatória**), que será preenchido com os elementos aleatórios gerados na quantidade escolhida no item anterior, e medir o tempo de execução do algoritmo para ordená-lo. Este processo deverá ser repetido 10 vezes para cada quantidade de elementos escolhida, sempre reinicializando a alocação e o preenchimento do vetor com a geração de novos elementos e descartando sua alocação (**free**) ao final da ordenação. O objetivo é garantir não ser possível a ordenação de um mesmo vetor, ou seja, a mesma sequência de valores.
- Para geração dos elementos para preenchimento do vetor, deverá ser utilizada a função `rand()`, alterando sua "**semente**" a cada nova geração de números aleatórios – ver documento disponível na plataforma AVA.
  Não se preocupe com elementos repetidos, eles podem existir.
- Em cada uma das 10 passagens, o tempo de execução deverá ser medido e armazenado, para ao final, gerar o tempo médio das 10 ordenações. Aqui sugiro a criação de um outro vetor de 10 posições para armazenamento dos resultados, e posterior cálculo de média.
- A marcação dos instantes de tempo para a medição do tempo de execução do algoritmo de ordenação, deverá ser realizada imediatamente antes e depois de ser chamada a função de ordenação, estando fora da contagem de tempo a alocação do vetor dinâmico e a posterior liberação de memória.

- Uma vez escolhido o algoritmo e a quantidade de elementos do vetor, seu programa deverá executar tudo de forma **automática sem intervenção do usuário:** as tarefas de alocação de memória, geração de números aleatórios e armazenamento das médias de ordenação.
- Ao final da execução do bloco de ordenação, seu programa deve informar o tempo médio gasto na execução do algoritmo de ordenação testado no momento, ou seja, a média de tempo gasto com a execução deste algoritmo específico para as 10 baterias de ordenação de cada quantidade escolhida.
- Finalizadas as medições das 10 rodadas de números aleatórios, seu programa deve fazer mais duas medições: uma do tempo gasto quando a entrada do algoritmo é um vetor já ordenado de forma crescente, e outra medição quando a entrada é um vetor que tem seus elementos dispostos ordem inversa, ou seja, decrescente. O objetivo nesta etapa é registrar os **comportamentos natural e pior caso** do algoritmo, apresentando estes comportamentos no relatório. Para esta operação **não são necessárias as 10 repetições** já que os elementos estão ordenados de forma crescente e decrescente e, portanto, serão sempre os mesmos. Neste ponto, você pode seguir dois caminhos diferentes: utilize o último vetor criado, que ao final da 10ª ordenação estará em ordem crescente e teste o comportamento natural. Depois inverta-o para o teste de pior caso. O segundo caminho será criar uma função específica para gerá-lo, sempre com a mesma quantidade de elementos do teste corrente, utilizando o próprio índice do *loop* "`for`", armazenando-o para cada posição do vetor. Dessa forma teremos dois *loops* respectivamente, um crescente (`i++`) e outro decrescente (`i--`).
- Os valores de tempo medidos para cada quantidade de elementos em ordem crescente e em ordem decrescente, devem ser armazenados e apresentados em um gráfico separadamente.

Medições de tempo

Para a tarefa de coleta de tempo, utilize a função **`gettimeofday()`**, a qual considero a melhor para esta aplicação, ela faz parte da biblioteca **`sys/time.h`** - documento disponível na plataforma AVA.

Existem outras funções nativas da Linguagem C para a tomada e medição de tempo, que também podem ser utilizadas neste projeto (não as considero precisas). Você não está restrito à função **gettimeofday()**, porém neste caso **justifique brevemente sua escolha no relatório**, e por que achou relevante utilizá-la. Apenas tenha em mente a **precisão da medição** apresentando resultados em **milissegundos** no mínimo (dê preferência para microssegundos), para que seja possível a comparação entre os algoritmos.

Utilize uma variável do tipo "***double***" para obter a precisão necessária. Resultados apresentados em bases de tempo que não possibilitem a comparação direta via gráfico, não serão considerados, afinal é **essencial que haja a diferenciação entre as performances dos algoritmos**.

Dependendo da configuração utilizada, poder de processamento e a memória disponível no computador utilizado para o desenvolvimento, as quantidades de elementos para ordenação sugeridos (1.000 - 5.000 - 10.000 - 20.000 - 50.000 - 100.000), podem ser insuficientes, gerando leituras muito próximas de zero, **impossibilitando a comparação entre os algoritmos**. Neste caso, os integrantes do grupo podem aumentar essas quantidades em um fator de 10 ou 100 (isso é uma sugestão), para a obtenção de leituras de tempo que sejam **mais confortáveis**. Uma vez que estas novas quantidades terão que ser utilizadas com todos os algoritmos de ordenação, garantimos a igualdade condições para todos os testes.

Relatório

Seu relatório, que deve ser enviado em um arquivo "**.pdf**" e formatado em padrão ABNT, especifique dados do computador utilizado no projeto, tais como modelo do processador e quantidade de memória, bem como as etapas e dificuldades para a implementação do programa, seus resultados e comentários que julguem pertinentes.

Importante: Seu relatório deve conter **gráficos comparativos** de desempenho dos 11 algoritmos de ordenação e **gráficos em separado** para as ordenações de vetores ordenados e vetores invertidos (último item dos requisitos).

Por questão de padronização, utilizem os 3 algoritmos fornecidos nas aulas, e os 7 apresentados pelos grupos no seminário sobre algoritmos de ordenação que estão disponíveis na plataforma Moodle. Assim garantimos que todos utilizem os mesmos algoritmos.

Este projeto deverá ser realizado pelos mesmos integrantes dos grupos do seminário.

Todo o projeto, software completo (com **todos os arquivos** gerados pela IDE CodeBlocks) e **relatório**, devem ser entregues na plataforma AVA em data indicada na própria.