INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: ORDENAÇÃO

Prática 07

Parte 0: Preparação

Passo 1: Crie um novo projeto chamado **Pratica7**.

O usa a biblioteca <chrono> requer versões mais recentes do C++ (C++11 em diante), o que pode requerer configuração adicional da IDE.

Passo 2: Utilize o arquivo **ordenacao.cpp** que acompanha a prática 7 ao projeto.

Esse arquivo é uma versão simplificada do código apresentado em sala de aula.

Parte 1: Implementando BubbleSort

Passo 1: Implemente a função `bubblesort()` do arquivo **ordenacao.cpp**.

Siga o algoritmo contido no material de aula. Atenção as otimizações possíveis desse algoritmo: não precisar sempre varrer o array até o final, já que a cada rodada o maior elemento acaba na sua posição certa; e caso não haja trocas, significa que o array está ordenado e já o algoritmo já pode encerrar.

Passo 2: Rode e teste a aplicação.

Verifique se a ordenação ocorreu adequadamente. Use a função `print()` e o depurador para identificar erros. **OBS.:** Aumente o tamanho do *array* (variável `size`) para testar se seu código está correto e ver o efeito no tempo de execução.

Parte 2: Implementando InsertionSort / SelectionSort

Passo 1: **Escolha** entre implementar a função `selectionsort()` ou `insertionsort()`.

Siga a descrição dos algoritmos no material de aula.

Passo 2: Rode e teste a aplicação.

Verifique o resultado e faça a depuração se necessário. Aumente o tamanho do *array* para testar se seu código está correto e ver o efeito no tempo de execução.

Parte 4: Implementando a junção do MergeSort

Passo 1: Implemente a função `merge()`, usada pela função `mergesort()`.

Essa função realiza a junção dos dois *subarrays* ordenados de `aux` (origem) em `array` (destino). A primeiro pedaço vai de `start` até `mid - 1` e o segundo de `mid` até `finish`. O parâmetro `array` é indexado de `start` até `finish` (inclusivo).

Passo 2: Rode e teste a aplicação.

Verifique o resultado e faça a depuração se necessário. Aumente o tamanho do *array* para testar se seu código está correto e ver o efeito no tempo de execução.

Parte 4: Implementando a partição do QuickSort

Passo 1: Implemente a função `partition()`, usada pela função `quicksort()`.

> Essa função deve organizar o array de forma que o elemento escolhido como pivô acabe na sua posição correta (ordenada), todos os elementos a sua esquerda são menores (ou iguais) a ele, e os da direita são maiores. A função por fim deve retornar a posição final do pivô. Tome cuidado para não violar os limites do array (`start` e `finish`). Siga a descrição da função no material de aula se necessário.

Passo 2: Rode e teste a aplicação.

> Verifique o resultado e faça a depuração se necessário. Aumente o tamanho do *array* para testar se seu código está correto e ver o efeito no tempo de execução.

Passo 3: **(Desafio/Opcional)** Implemente versões alternativas da função `partition()` usando diferentes formas de selecionar o pivô e avalie o impacto disso no desempenho do algoritmo.

> Há variações da função de partição onde o pivô não necessariamente termina na sua posição correta final no array. Isso requer modificar a função principal `quicksort()` de acordo.