Universidade Federal de Campina Grande
Centro de Engenharia Eletrica e Informática
Departamento de Sistemas e Computação
Graduação em Ciência da Computação

Exercícios sobre Algoritmos de Ordenação por Comparação (Parte I)

Objetivo: Implementar alguns algoritmos de ordenação por comparação.

O endereço do sistema de submissão é o **https://les.dsc.ufcg.edu.br:8443/EasyLabCorrection**. Entre no sistema de submissão (acessível também via link no site da disciplina – menu Cronograma). Selecione o roteiro e baixe o arquivo .zip contendo as classes necessárias.

Atividades necessárias antes de iniciar o exercicio:

1. Crie um projeto no Eplipse chamado LEDA, por exemplo (pode ser qualquer outro nome que lhe convier);
2. Descompacte o arquivo baixado na pasta dos fontes (normalmente **src**) do seu projeto LEDA criado no seu workspace. O arquivo baixado tem a seguinte estrutura:
   -sorting
   --Sorting.java (INTERFACE CONTENDO A ASSINATURA DO METODO DE ORDENAÇÃO)
   --SortingImpl.java (IMPLEMENTAÇÃO BASE A SER HERDADA POR TODAS AS IMPLEMENTAÇÕES)
   -- Util.java (CLASSE AUXILIAR CONTENDO O METODO DE SWAP A SER USADO NAS IMPLEMENTACOES)
   --simpleSorting
   --- Bubblesort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
   --- Selectionsort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
   --- Insertionsort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
   --variationsOfSelectionsort
   --- BidirectionalSelectionsort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
3. No Eclipse, selecione a pasta dos fontes no projeto LEDA e faça um refresh (apertar F5). Note que deve aparecer a estrutura de pacotes e os arquivos mencionados acima.

Observe a interface Sorting.java. Ela contém a assinatura do método de ordenação a ser implementado segundo diferentes estratégias. Observe também a existência implementação incompleta SortingImpl. Ela é uma classe abstrata que tem uma implementação do método `sort(int[] array)`, que NÃO deve ser modificaco. Essa implementação simplesmente faz uso de um outro método de ordenação que recebe três parâmetros `sort(int[] array, int leftIndex, int rightIndex)`, cujo propósito é ordenar um array de um ponto inicial até um ponto final, inclusive.

Obs: NÃO modifique a assinatura dos métodos. Caso contrário os testes não funcionarão.

Agora voce está pronto para começar a trabalhar nas seguintes atividades:

1. Implemente os métodos de ordenação nas classes Bubblesort.java, Selectionsort.java e Insertionsort.java. Procure ser fiel a estratégia de cada algoritmo.
2. Observe a classe BidirectionalSelectionsort. Ela representa uma variação do selectionsort que deve funcionar da seguinte forma: em fazer uma passagem pelo array (da esquerda para a direita) selecionando o

menor e colocando-o na posição mais a esquerda. Depois disso, é feita uma passagem da direita para a esquerda selecionando o maior elemento e colocando-o na posição mais à direita. Nas próximas passagens os elementos são colocados na segunda (mais à esquerda) e penúltima (mais à direita) posições, e assim sucessivamente. Os percursos *indo* e depois *voltando* continuam até que o array esteja todo ordenado. Dessa forma, o algoritmo aplica uma iteração do selectionsort no sentido crescente trazendo para o começo o menor elemento, e outra iteração do selectionsort no sentido decrescente levando para o final o maior elemento até que todo array esteja ordenado. Note que na primeira passagem de *ida* e na primeira de *volta*, os elementos das extremidades (array[0] e array[n-1]) ficam fixos (array[0] já contém o menor e array[n-1] já contém o maior). Nas próximas passagens os elementos (array[1] e array[n-2] ficam fixos) e assim por diante.

Uma ilustração de execução desse algoritmo é a seguinte:

[02, 05, 06, 04, 13, 03, 12, 19] – INICIO DA PASSAGEM *INDO* COMEÇANDO NO ELEMENTO 2
[**02**, 05, 06, 04, 13, 03, 12, 19] – ELEMENTO 2 (MENOR) FICOU FIXO NA PRIMEIRA POSICAO
[**02**, 05, 06, 04, 13, 03, 12, 19] – INICIO DA PASSAGEM *VOLTANDO* COMEÇANDO NO ELEMENTO 19
[**02**, 05, 06, 04, 13, 03, 12, **19**] – ELEMENTO 19 (MAIOR) FICOU FIXO NA ULTIMA POSICAO
[**02**, 05, 06, 04, 13, 03, 12, **19**] – INICIO DA PASSAGEM *INDO* COMEÇANDO NO ELEMENTO 5
[**02**, **03**, 06, 04, 13, 05, 12, **19**] – ELEMENTO 3 (MENOR) FICOU FIXO NA SEGUNDA POSICAO
[**02**, **03**, 06, 04, 13, 05, 12, **19**] – INICIO DA PASSAGEM *VOLTANDO* COMEÇANDO NO ELEMENTO 12
[**02**, **03**, 06, 04, 12, 05, **13, 19**] – ELEMENTO 13 (MAIOR) FICOU FIXO NA PENULTIMA POSICAO
[**02**, **03**, 06**,** 04, 12, 05, **13, 19**] – INICIO DA PASSAGEM *INDO* COMEÇANDO NO ELEMENTO 6
[**02**, **03**, **04,** 06, 12, 05, **13, 19**] – ELEMENTO 4 (MENOR) FICOU FIXO NA TERCEIRA POSICAO
[**02**, **03**, **04,** 06, 12, 05, **13, 19**] – INICIO DA PASSAGEM *VOLTANDO* COMEÇANDO NO ELEMENTO 5
[**02**, **03**, **04,** 06, 05, **12, 13, 19**] – ELEMENTO 12 (MAIOR) FICOU FIXO NA SEXTA POSICAO
[**02**, **03**, **04,** 06, 05, **12, 13, 19**] – INICIO DA PASSAGEM *INDO* COMEÇANDO NO ELEMENTO 6
[**02**, **03**, **04, 05,** 06, **12, 13, 19**] – ELEMENTO 5 (MENOR) FICOU FIXO NA QUARTA POSICAO
[**02**, **03**, **04, 05,** 06, **12, 13, 19**] – INICIO DA PASSAGEM *VOLTANDO* COMEÇANDO NO ELEMENTO 6
[**02**, **03**, **04, 05, 06, 12, 13, 19**] – ELEMENTO 6 (MAIOR) FICOU FIXO NA QUINTA POSICAO
[**02**, **03**, **04, 05, 06, 12, 13, 19**] – NENHUMA MUDANÇA. ARRAY JÁ ORDENADO

3. Concentre-se em implementar conforme descrito na interface e pense em cenários para testar suas implementações. Alguns cenários interessantes são: testar ordenar um array vazio, array unitário, array com numeros positivos, array com numeros negativos, array com numeros positivos e negativos, array com numeros iguais, etc. Enfim, voce precisa também pensar nas demais situações em que seu algoritmo precisa ser testado para se certificar de que implementou corretamente. Uma sugestao é usar Junit ao invés de um método main em sua classe.

Instruções para o envio

Ao terminar o exercício, você precisa enviar apenas os arquivos que voce implementou, compactados, seguindo a mesma estruturas de pacotes do original. Ou seja, seu arquivo compactado deve ter a seguinte estrutura:

-sorting
--Sorting.java (INTERFACE CONTENDO A ASSINATURA DO METODO DE ORDENAÇÃO)
--SortingImpl.java (IMPLEMENTAÇÃO BASE A SER HERDADE POR TODAS AS IMPLEMENTAÇÕES)

-- Util.java (CLASSE AUXILIAR CONTENDO O METODO DE SWAP A SER USADO NAS IMPLEMENTACOES)
--simpleSorting
--- Bubblesort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
--- Selectionsort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
--- Insertionsort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
--variationsOfSelectionsort
--- BidirectionalSelectionsort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)

Obs: a compactação DEVE ser feita a partir do diretório raiz de seus fontes de forma a preservar a estrutura de pastas que refletem a estrutura dos pacotes (package) Java. Uma boa dica é compactar a pasta "sorting" e depois remover os arquivos que nao devem ser enviados. Certifique-se de que seu arquivo compactado tem a estrutura de pacotes requerida antes de envia-lo. Se suas classes estiverem empacotadas (com informação de package) e você compactá-las diretamente sem a estrutura de pastas correta, seu código não vai compilar. Modifique o nome do arquivo compactado para NOME_COMPLETO_DO_ALUNO.ZIP.

Observações finais:

- **A interpretação do exercício faz parte do roteiro.**
- **O roteiro é individual. É como se fosse uma prova prática e a conversa entre alunos é proibida.**
- **É proibido coletar códigos prontos e adaptar. Implemente as questões. Isso é para seu aprendizado.**
- **Caso você observe qualquer problema no sistema de submissão, contacte o professor imediatamente.**