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 (Dividir para Conquistar)

Objetivo: Praticar a implementacao de algoritmos de ordenação por comparação que utilizam a técnica de dividir para conquistar.

O endereço do sistema de submissão é o **https://les.dsc.ufcg.edu.br:8443/EasyLabCorrection**.

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 (exceto o PDF) 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)
   --divideAndConquer
   --- Mergesort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
   --- Quicksort.java (IMPLEMENTACAO A SER PREENCHIDA PELO ALUNO)
   --- quicksort3
   ---- QuicksortMedianOfThree.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.

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 o método sort(T[] array, int leftIndex, int rightIndex) nas classes Mergesort.java e Quicksort.java.

Observe a classe QuicksortMedianOfThree.java. Ela representa a implementação de uma variação do quicksort que funciona de forma ligeiramente diferente. Relembre que quando o pivot escolhido divide o array aproximadamente na metade, o quicksort tem uma performance perto da ótima. Para aproximar a entrada do caso ótimo, diversas abordagens podem ser utilizadas. Uma delas é usar a **mediana de 3** achar o pivot. Essa técnica consiste no seguinte:

- Comparar tres elementos do array a ser ordenado: o mais a esquerda **A[left]**, o central **A[center]** e o mais a direita **A[right]**.
- Trocar os elementos se necessário para que eles fiquem ordenados: **A[left] < A[center] < A[right]**
- Adotar o **A[center]** como pivot.
- Coloca o pivot na penultima posicao **A[right-1].**

- Aplica o particionamento considerando o vetor menor **A[left]** até **A[right - 1].**
- Aplica o quicksort mediana de 3 na metade menor que o pivot.
- Aplica o quicksort mediana de 3 na metade maior que o pivot.

Uma ilustração de execução desse algoritmo é a seguinte:

[08, 05, 06, 04, 02, 13, 03, 12, 19] – SELECIONA OS TRES ELEMENTOS
[02, 05, 06, 04, 08, 13, 03, 12, 19] – COLOCA-OS NA ORDEM CORRETA
[02, 05, 06, 04, 08, 13, 03, 12, 19] – ADOTA O NOVO PIVOT
[01, 05, 06, 04, 12, 13, 03, **08**, 19] – MOVE O PIVOT PARA A PENULTIMA POSICAO
[01, 05, 06, 04, 12, 13, 03, **08**, 19] – APLICA O PARTICIONAMENTO NESSA FAIXA
[01, 05, 06, 04, 03, **08**, 12, 13, 19] – COLOCA O PIVOT NA POSICAO CORRETA
[01, 05, 06, 04, 03, **08**, 12, 13, 19] – APLICA O ALGORITMO NOVAMENTE NESSAS DUAS PARTES

Uma descricao algoritmica (pseudo-codigo) dos procedimentos acima seria:

```
medianaDeTres(A[],left,right){
  middle = o indice do array A que esta aproximadamente  no meio de left e right
   faz trocas entre  A[left],A[middle] e A[right] de forma que A[left] < A[middle] < A[right]
}
```

```
quicksort3(A[],left, right) {
 if(left < right && left >=0){
   medianaDeTres(A,left,right) //vai deixar o pivot na posicao central (middle) entre left e right
  swap(A[middle],A[right-1]) //coloca o pivot na penultima posicao
  pivotPosition = partition(A, left, right-1) //particiona o array e coloca o pivot na posicao correta
 quicksort3(array, left, pivotPosition-1) //aplica o quicksort a primeira parte
 quicksort3(array, pivotPosition+1,right) //aplica o quicksort a segunda parte
}
```

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
--divideAndConquer
--- Mergesort.java
--- Quicksort.java
--- quicksort3
---- QuicksortMedianOfThree.java

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.**