

125  
spec

#3 -----

126  
exercício

Considera um método genérico que recebe dois valores A e B e uma lista L. A e B podem ser comparados com os valores na lista (por via de `A.compareTo(valor)` ou `B.compareTo(valor)`). O inverso não é necessariamente verdade (i.e., pode não ser possível calcular `valor.compareTo(A)` ou `valor.compareTo(B)`)

Deve remover da lista todos os valores que sejam menores que B e também diferentes de A

a)

construa o protótipo do método. Soluções que não respeitem as restrições indicadas não são valorizadas.

b)

construa o corpo da função. Deve garantir o melhor desempenho possível, considerando que as listas de entrada tanto poderão ser ambas um `ArrayList` como serem ambas uma `LinkedList`. Indique qual a complexidade do método em ambos os casos. Soluções que não respeitem estas indicações não serão valorizadas.

a) `1. Protótipo <T> void remove( List<T> lista,`  
`<Comparable< ? super T > A,`  
`<Comparable< ? super T > B ) { }`

b) `<T> void remove( List<T> lista, Comparable< ? super T > A,`  
`Comparable< ? super T > B ) { }`

`I Iterator<T> it = lista.iterator();`  
`List<T> tony = new LinkedList<T>();`  
`T vl;`  
`while (it.hasNext()) {`  
 `vl = it.next();`  
 `if (vl.compareTo(B) >= 0) { tony.add(vl); }`  
 `if (vl.compareTo(A) < 0 || vl.compareTo(B) > 0) { tony.add(vl); }`

`list.clear();`  
`list.addAll(tony);`

)

Complexidade de N para o clear

3) Elabore uma função genérica:

- que recebe um parâmetro **primeiro** que é uma lista do tipo genérico T
- recebe um parâmetro **segundo** que é um valor qualquer comparável com **primeiro**

O algoritmo implementado deve ser o mais eficiente possível para ArrayList ou  
LinkedList.

92  
Oscar

1) Crie um iterador que consiga iterar os elementos de uma classe **Ponto**, que já está implementada. A classe **Ponto** recebe dois inteiros no seu construtor e os seus valores não podem ser alterados após a criação do objeto. **Trate todas as exceções que achar necessário.**

3- Considere um método `removeMaiores` que recebe duas listas de elementos. Os valores da segunda lista devem ser comparáveis com os valores da primeira lista. O método remove da segunda lista todos os valores que sejam maiores do que todos os valores da primeira lista

120  
Oscar

- Construa o protótipo do método. Soluções que não respeitem as restrições indicadas não são valorizadas.
- Construa o corpo da função. Deve garantir o melhor desempenho possível considerando que as listas de entrada tanto poderão ser um `ArrayList` como uma `LinkedList`. Indique qual a complexidade do método em ambos os casos. Soluções que não respeitem estas indicações não serão valorizadas.

Exemplo de funcionamento:

```
List<Integer> f=Arrays.asList(0,3,2,1,3);
List<Integer> g=new LinkedList<Integer>();
g.add(0); g.add(4); f.add(2);
removeMaiores(f,g); // g passa a ser {0,2}
```

(Responda no verso)

3 - Construa um método f, que recebe três parâmetros:

- um parâmetro lista, que é uma List de um tipo genérico T
- dois parâmetros min e max, que podem ser comparados com qualquer valor contido em lista, através de uma instrução como min.compareTo(lista.get(0)).

O método deve remover da lista os valores que não estejam no intervalo [min, max]. O método deve ter complexidade temporal linear independentemente do tipo de lista que é recebida (outras soluções não são valorizadas). A complexidade espacial não importa.

(Responda abaixo e / ou no verso)

29  
exercício

complexidade  
espacial  
vs  
complexidade  
temporal

static<T extends Comparable<? super T>> void f (List<T> lista, T min, T max) {

T min,  
T max);

for (T it = lista.iterator();

it.hasNext();

while (it.hasNext())

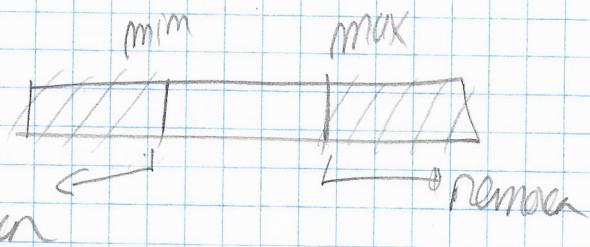
temp = it.next();

if (min.compareTo(temp) <= 0 && max.compareTo(temp) >= 0) {

it.remove();

}

}



public <T extends Comparable<? super T>> void f (List<T> lista, T min, T max) {

for (T it = lista.iterator();

list < T max = Arrays.asList(<T>());

T temp;

while (it.hasNext())

temp = it.next();

if (min.compareTo(temp) >= 0 && max.compareTo(temp) <= 0) {

max.add(temp);

}

}

lista.clear(); lista.add(max); }

- Construa um método `f`, que recebe dois parâmetros `lista1` e `lista2`:  
ambas as listas contêm valores de um tipo genérico `T` que representam elementos que podem ser  
comparados entre si (por exemplo, é possível fazer

`lista1.get(0).compareTo(lista1.get(1))` ou  
`lista2.get(0).compareTo(lista1.get(0))`.

caso as listas estão ordenadas. O objetivo do método é garantir que os menores valores ficam  
armazenados na `lista1` e que os maiores valores ficam armazenados na `lista2` (e que ambas as listas  
enviem o tamanho original e a ordenação). Por exemplo, se `lista1 = {10, 20, 30, 40, 50}` e  
`lista2 = {5, 8, 15, 25, 60}`, então no final o resultado deverá ser `lista1 = {5, 8, 10, 15, 20}` e  
`lista2 = {25, 30, 40, 50, 60}`.

todo deve ter complexidade temporal linear (em função de  $M+N$ , onde  $M$  e  $N$  são as  
sões das listas), independentemente do tipo de listas que são recebidas (outras soluções não  
serão avaliadas). A complexidade espacial não tem importância.  
nada abaixo e / ou no verso)

120/120