Algoritmos e Estruturas de Dados
2013/14
Primeiro Teste – 4 de Novembro de 2013
Departamento de Informática, Universidade Nova de Lisboa

Atenção: Os Anexos ao teste poderão ser-lhe úteis.

1. Considere o método `alg` apresentado abaixo.

```
public static void alg(int[] a){
   int temp;
   int i = 0;
   int j = a.length-1;
   while ( i<j ) {
      temp = a[i];
      a[i++] = a[j];
      a[j--] = temp;
   }
}
```

   Determine a complexidade temporal do método `alg`, no melhor caso, no pior caso e no caso esperado, **justificando** todos os cálculos com muita clareza.

2. Considere o método `separate` apresentado abaixo. Este método recebe, como parâmetros, um vetor de números inteiros e um número inteiro, denominado `threshold`, e devolve um iterador dos números contidos no vetor, sendo que todos os números inferiores ao valor contido em `threshold` devem ser iterados antes dos restantes. Para completar a implementação de `separate`, pedimos-lhe que implemente, recursivamente, o método auxiliar `recSeparate` que utiliza, como estrutura auxiliar, uma lista duplamente ligada. Uma vez desenvolvido o método recursivo pedido, determine a complexidade temporal do método `separate`, no melhor caso, no pior caso e no caso esperado.

   Exemplo: para a chamada do método `separate` com os argumentos v = {1, -1, -4, 15, 3, 14, -7} e `threshold` = 0, o iterador resultante da chamada poderá devolver os números em questão pela seguinte ordem: -1, -4, -7, 1, 15, 3, 14. Note-se que a ordem apresentada não é obrigatória, o importante é que o método devolva os números inferiores a `threshold` antes dos restantes.

```
public static Iterator<Integer> separate(int v[], int threshold){
   List<Integer> list = new DoublyLinkedList<Integer>();
   recSeparate(v, 0, list, threshold);
   return list.iterator();
}
```

   Nota: Implemente a solução em código Java. Apenas as soluções recursivas de `recSeparate` serão avaliadas.

3. O Tipo Abstrato de Dados `WeatherStation` define as operações associadas à gestão dos valores de precipitação das estações meteorológicas de Portugal Continental. Para o estudo a desenvolver, as estações são identificadas através de um número inteiro (o código da estação) e têm associado um valor de precipitação acumulado. Todas as estações no sistema recebem, periodicamente, valores de precipitação que devem ser adicionados ao valor acumulado individual de cada estação. Durante o estudo, é necessário ainda providenciar, de forma expedita, o valor total da precipitação de todas as estações, assim como a média. Poderá ainda ser preciso apresentar o código de identificação da estação com o valor máximo de precipitação acumulada. Apresenta-se abaixo o interface para o Tipo Abstrato de Dados `WeatherStation`.

```
public interface WeatherStation {

   // Insere uma nova estação meteorológica no território de Portugal
   // Continental. A inserção só é realizada se o código da estação ainda
   // não existir no sistema.
   void addWeatherStation(int stationCode)
      throws ExistingStationException;

   // Adiciona valor de precipitação ao valor acumulado da estação
   // identificada por stationCode. A operação só é realizada se stationCode
   // já existir no sistema. Assume-se que o valor contido em rainValue
   // é válido.
   void addRainValue(int stationCode, double rainValue)
      throws NonExistingStationException;

   // Devolve o valor acumulado de precipitação para a estação meteorológica
   // identificada pelo código stationCode. A operação só é realizada se
   // stationCode já existir no sistema.
   double getRainValue(int stationCode)
      throws NonExistingStationException;

   // Devolve o valor total de precipitação nas estações do sistema.
   double totalRainValue();

   // Devolve a média da precipitação nas estações do sistema.
   double averageRainValue();

   // Devolve o código da estação com o máximo de precipitação.
   int maxRainStationCode();

}
```

Relativamente ao problema proposto:

- Explicite detalhadamente as estruturas de dados e as variáveis de instância mais adequadas para implementar a interface `WeatherStation`;
- Descreva brevemente como implementaria as seis operações e calcule as suas complexidades temporais, no melhor caso, no pior caso e no caso esperado, justificando. **Não deve desenvolver código em java, apenas fazer uma descrição da implementação das operações de acordo com a sua escolha de estruturas de dados.**

Anexo A - Recorrências

Recorrência 1

$$T(n) = \begin{cases} a & n = 0 \qquad n = 1 \\ & \textbf{ou} \\ bT(n \overset{\mapsto}{-} 1) + c & n \geq 1 \qquad n \geq 2 \end{cases} \qquad T(n) = \begin{cases} O(n) & b = 1 \\ O(b^n) & b > 1 \end{cases}$$

com $a \geq 0, \quad b \geq 1, \quad c \geq 1$ **constantes**

Recorrência 2a)

$$T(n) = \begin{cases} a & n = 0 \qquad n = 1 \\ & \textbf{ou} \\ bT(\frac{n}{2}) + O(1) & n \geq 1 \qquad n \geq 2 \end{cases} \qquad T(n) = \begin{cases} O(\log n) & b = 1 \\ O(n) & b = 2 \end{cases}$$

com $a \geq 0, \quad b = 1, 2$ **constantes**

Recorrência 2b)

$$T(n) = \begin{cases} a & n = 0 \qquad n = 1 \\ & \textbf{ou} \\ bT(\frac{n}{c}) + O(n) & n \geq 1 \qquad n \geq 2 \end{cases}$$

com $a \geq 0, \quad b \geq 1, \quad c > 1$ **constantes**

$$T(n) = \begin{cases} O(n) & b < c \\ O(n \log_c n) & b = c \\ O(n^{\log_c b}) & b > c \end{cases}$$

Anexo B – Interfaces e Classes de Apoio

```
public interface Queue<E> {
   boolean isEmpty( );
   int size( );
   void enqueue( E element );
   E dequeue( ) throws EmptyQueueException;
}
public interface Iterator<E> {
    boolean hasNext( );
    E next( ) throws NoSuchElementException;
    void rewind( );
}
public interface List<E> {
    boolean isEmpty( );
    int size( );
    Iterator<E> iterator( );
    E getFirst( ) throws EmptyListException;
    E getLast( ) throws EmptyListException;
    E get( int position ) throws InvalidPositionException;
    int find( E element );
    void addFirst( E element );

    void addLast( E element );
    void add( int position, E element )
       throws InvalidPositionException;
    //interface List<E> continua na página 5.
```

```
    //continuação do interface List<E> (página 4).

    E removeFirst( ) throws EmptyListException;

    E removeLast( ) throws EmptyListException;

    E remove( int position ) throws InvalidPositionException;

    boolean remove( E element );

}

class DListNode<E> implements Serializable {
   public DListNode( E theElement, DListNode<E> thePrevious,
      DListNode<E> theNext );

   public DListNode( E theElement );

   public E getElement( );

   public DListNode<E> getPrevious( );

   public DListNode<E> getNext( );

   public void setElement( E newElement );

   public void setPrevious( DListNode<E> newPrevious );

   public void setNext( DListNode<E> newNext );
}

public class DoublyLinkedList<E> implements List<E> {

   public boolean isEmpty( );

   public int size( );

   public Iterator<E> iterator( );

   public E getFirst( ) throws EmptyListException;

   public E getLast( ) throws EmptyListException;

   //DoublyLinkedList<E> continua na página 6.
```

```
   //continuação da DoublyLinkedList<E> (página 5).

   protected DListNode<E> getNode( int position );

   public E get( int position ) throws InvalidPositionException;

   public int find( E element );

   public void addFirst( E element );

   public void addLast( E element );

   protected void addMiddle( int position, E element );

   public void add( int position, E element )
      throws InvalidPositionException;

   protected void removeFirstNode( );

   public E removeFirst( ) throws EmptyListException;

   protected void removeLastNode( );

   public E removeLast( ) throws EmptyListException;

   protected void removeMiddleNode( DListNode<E> node );

   public E remove( int position )
      throws InvalidPositionException;

   protected DListNode<E> findNode( E element );

   public boolean remove( E element );

   public void append( DoublyLinkedList<E> list );
}

public interface Entry<K,V>{

   K getKey( );

   V getValue( );

}
```

```
public interface Comparable<T>{
   int compareTo( T object );
}
public interface Dictionary<K,V>{
   boolean isEmpty( );
   int size( );
   Iterator<Entry<K,V>> iterator( );
   V find( K key );
   V insert( K key, V value );
   V remove( K key );
}
```