TD/TP 4 Lecture dans un fichier et dichotomie

I. Lecture dans un fichier

On considère la classe **`Lectcolonne`** suivante :

```
import java.util.*;
import java.io.*;

public class Lectcolonne{

    private static double getNthColumnDouble(String ligne, int k) throws Exception {
        StringTokenizer st = new StringTokenizer(ligne, ";");
        if (st.countTokens() >= k) {
           String sval=null;
           for (int i=0; i < k; i++) {
              sval = st.nextToken();
           }
           try {
           return(new Double(sval));
           }
           catch(NumberFormatException e) {
              throw new Exception("La valeur n'est pas un double");
           }
        } else {
           throw new Exception("Il n' a que " + st.countTokens() +
                                                  " valeurs sur la ligne");
        }
    }

    public static Double[] lire(String nf,int column) {
       int i = 0;
       String ligne;
       Vector<Double> result = new Vector<Double>();

       try {
          FileReader fr = new FileReader(nf);
          BufferedReader bf = new BufferedReader(fr);
          ligne = bf.readLine();
          while (ligne != null) {
             i++;
             result.add(getNthColumnDouble(ligne, column));
             ligne = bf.readLine();
          }
          bf.close();
          fr.close();
          return(result.toArray(new Double[1]));
       } catch (Exception e) {
          System.err.println("Problème à la lecture du fichier "+nf+" ligne " +i);
          e.printStackTrace();
          return(null);
       }
   }
}
```

On va utiliser un fichier nommé `test.txt` pour tester la méthode `lire` ci-dessus. Le contenu du fichier `test.txt` est le suivant :

```
1; 2; 3; 4; 5; 3
1; 2; 3; 4; 5; 5; 6
6; 8; 9; toto; 10; 11
6; 7; 8; 9; 10
```

Q.1) - En utilisant vos notes de cours, expliquer ce que donne le code JAVA suivant :

```
Double[] tab1 = Lectcolonne.lire("test.txt", 3);
Double[] tab2 = Lectcolonne.lire("test.txt", 4);
Double[] tab3 = Lectcolonne.lire("test.txt", 6);
```

II. Recherche dichotomique

Rappelons le principe de la recherche dichotomique : pour rechercher une valeur dans un tableau trié de taille au moins 2, il est possible de la rechercher dans une des deux moitiés de ce tableau.

Q.2) - Donner la forme récursive de l'algorithme de recherche dichotomique. Cet algorithme doit retourner la position de la première occurrence de la valeur dans le tableau ou -1 si la valeur n'y est pas présente.

Q.3) - Démontrer que votre algorithme retourne le bon résultat.

Q.4) - Calculer la complexité en fonction de la taille du tableau.

Q.5) - Compléter votre algorithme afin qu'il retourne également la position où il faudrait insérer la valeur dans le tableau pour que celui-ci reste trié.

Q.6) - Donner le code JAVA pour des tableaux triés de `Double`. (En TP, cette fonction sera implémentée comme une méthode d'une nouvelle classe **`Dichotomie`**.)

III. EN TP

Q.7) - Récupérer sur la page du cours le dossier `tp4.zip`. Lancer **Netbeans** puis importer le projet à partir du **zip** récupéré. Exécuter la classe **`Principale`** du projet.

Q.8) - Dans la classe **`Principale`**, afficher maintenant la 3ème colonne du fichier `mat.txt`.

Q.9) - Trier le tableau obtenu avec la méthode `java.util.Arrays.sort`.

Q.10) - Tester votre algorithme récursif de recherche dichotomique sur ce tableau.

Q.11) - Compléter votre classe **`Dichotomie`** par une méthode effectuant le tri par insertion d'un tableau de `Double` à l'aide de la recherche dichotomique.

Q.12) - Tester votre méthode sur une colonne du fichier `iris.txt` en la comparant avec `java.util.Arrays.sort`.