

L'objectif de ce TP est :

1. d'étudier et d'expérimenter l'écriture d'algorithmes étudiés sous forme itérative, en algorithmes écrits sous forme récursive
2. de découvrir et coder un nouvel algorithme de tri, le tri par fusion


CONTEXTE : PAYS DU MONDE (STATISTIQUES 2019)

Comme dans le TP6(A) : un pays sera représenté par son nom, le continent où il se situe, sa superficie et son nombre d'habitants

Nous vous fournissons dans un projet TP8 accessible à partir du répertoire `/users/info/pub/1a/R1.01/`

- le fichier texte *Monde2019.txt*
- la classe **InitMonde** qui contient une fonction qui, à partir du fichier *Monde2019.txt*, crée un `ArrayList<Pays>`
- la classe **Pays** (dans laquelle vous définirez une relation d'ordre entre deux pays)

Avant de commencer...

- Ouvrez un terminal et placez-vous dans votre répertoire `R1.01`
 - ✓ Exécutez la commande : `cp -r /users/info/pub/1a/R1.01/TP8_Files .`
 - ✓ Lancez **IJ** et créez un projet TP8
- Ouvrez le dossier `TP8_Files` que vous avez copié dans votre répertoire `R1.01` et effectuez les actions suivantes :
 1. **copie dans votre projet 8 du fichier *Monde2019.txt***
 - copiez le fichier *Monde2019.txt* (clic sur le fichier + `CTRL + C`)
 - dans la fenêtre de votre projet TP8, sélectionnez avec le bouton droit de la souris, l'icône du projet 
 - collez le contenu du presse-papiers (`CTRL + V`) puis validez (clic sur `OK`)
 2. **copie dans le répertoire `src` de votre projet TP8 de la classe java *Pays***
 - copiez le fichier *Pays.java* (clic sur le fichier : `CTRL + C`)
 - dans la fenêtre de votre projet TP8, sélectionnez avec le bouton droit de la souris, l'icône du répertoire `src`
 - collez le contenu du presse-papiers (`CTRL + V`) puis validez (clic sur `OK`)
 3. **copie dans le répertoire `src` de votre projet TP8 de la classe java *InitMonde***
 - copiez le fichier *InitMonde.java* (clic sur le fichier + `CTRL + C`)
 - dans la fenêtre de votre projet TP8, sélectionnez avec le bouton droit de la souris, l'icône du répertoire `src`
 - collez le contenu du presse-papiers (`CTRL + V`) puis validez (clic sur `OK`)

1. Classe Pays

On définit un NOUVEL ORDRE NATUREL sur la classe **Pays** :

Les objets de type **Pays** sont ordonnés sur le continent auquel ils appartiennent, et à continent égal, sur leur nom : `ORDRE(continent, nom)`

Dans la méthode `compareTo` de la classe **Pays** : remplacez l'instruction de retour (`return ...`) par le code approprié (cf. spécification de la méthode)

1. Classe utilitaire : Tri d'un vecteur de Pays et vérification du tri

1.1. Créez une classe Utilitaire dans laquelle vous coderez les fonctions suivantes :

a) Vecteur de pays trié selon l'ordre (continent, nom) produit à partir d'un vecteur de pays non vide

```
public static ArrayList<Pays> TriBulle(ArrayList<Pays> vPays) {  
    // { vPays non vide }  
    // => { résultat = vecteur de Pays trié selon l'ORDRE(continent, nom)  
    //      en utilisant la méthode du TRI À BULLES AMÉLIORÉ }
```

b) Vérification du tri

```
public static boolean verifTri(ArrayList<Pays> vPays) {  
    // { } => { résultat = vrai si vPays trié selon l'ORDRE(continent, nom) }
```

2. Initialisation et vérification du tri d'un vecteur de Pays

2.1. Créez une classe Monde et ajoutez-y une procédure main dans laquelle vous écrirez, puis testerez :

- la déclaration suivante : `ArrayList<Pays> leMonde = InitMonde.creerMonde();`
- les instructions suivantes :
 - ✓ Tri par la méthode du tri à bulles amélioré du vecteur leMonde selon l'ORDRE(continent, nom)
 - ✓ Vérification du tri (un message sera affiché pour indiquer à l'utilisateur si le vecteur a bien été trié selon l'ordre convenu)

3. Recherche dichotomique d'un pays de continent et de nom donné

3.1. Dans la classe Utilitaire, ajoutez et codez les fonctions suivantes :

a) Recherche dichotomique d'un pays de continent et de nom donnés – FORME ITÉRATIVE

```
public static int rechDichoIter(ArrayList<Pays> vPays, String contP, String nomP) {  
    // { vPays trié selon l'ORDRE(continent, nom) } =>  
    // { résultat = * indice du pays de continent contP et de nom nomP  
    //              s'il existe dans le vecteur vPays  
    //              * -1 si non trouvé  
    // LA RECHERCHE EST DICHOTOMIQUE ITÉRATIVE }
```

b) Recherche dichotomique d'un pays de continent et de nom donnés – FORME RÉCURSIVE

b1 – "Le Modèle"

```
public static int rechDichoRec(ArrayList<Pays> vPays, String contP, String nomP) {  
    // { vPays trié selon l'ORDRE(continent, nom) } =>  
    // { résultat = * indice du pays de continent contP et de nom nomP  
    //              s'il existe dans le vecteur vPays  
    //              * -1 si non trouvé }
```

b2 – "Le Worker"

```
public static int rechDichoWorker(ArrayList<Pays> vPays, String contP, String nomP,  
    int inf, int sup) {  
    // { vPays trié selon l'ORDRE(continent, nom) } =>  
    // { résultat = indice du pays de continent contP et de nom nomP, s'il existe  
    //              dans le vecteur vPays[inf .. sup], ou -1 si non trouvé  
    // LA RECHERCHE EST DICHOTOMIQUE RÉCURSIVE }
```

3.2. Dans la procédure main de la classe Monde, ajoutez les instructions suivantes :

- Saisies d'un continent et d'un pays
- Recherche ITÉRATIVE de l'existence dans le vecteur trié, d'un pays du nom saisi, dans le continent saisi
Selon le résultat de la recherche, affichage d'un message d'échec ou affichage des caractéristiques du pays cherché
- Recherche RÉCURSIVE de l'existence dans le vecteur trié, d'un pays du nom saisi, dans le continent saisi
Selon le résultat de la recherche, affichage d'un message d'échec ou affichage des caractéristiques du pays cherché

NOTE : les résultats des deux formes de recherche, doit bien sûr être le même...

4. Recherche du pays ayant la plus grande superficie

Parmi les pays du fichier Monde2019, il existe un pays qui a une superficie plus grande que tous les autres...

4.1. Dans la classe Utilitaire déclarez et codez les fonctions suivantes :

a) Recherche itérative

```
public static Pays plusGrandPaysIter(ArrayList<Pays> vPays) {  
    // {vPays non vide} => {résultat = élément de vPays ayant  
    //                          la plus grande superficie  
    //                          RECHERCHE ITÉRATIVE}
```

b) Recherche récursive – Diviser Pour Régner

b1 – "Le Modèle"

```
public static Pays plusGrandPaysDPR(ArrayList<Pays> vPays) {  
    // {vPays non vide} => {résultat = élément de vPays ayant  
    //                          la plus grande superficie}
```

b2 – "Le Worker"

```
public static Pays maxPaysDPRWorker(ArrayList<Pays> vPays,  
                                     int inf, int sup) {  
    // {vPays non vide} => {résultat = élément de vPays[inf..sup] ayant  
    //                          la plus grande superficie  
    //                          RECHERCHE RÉCURSIVE }
```

4.2. Dans le corps de la procédure main de la classe Monde, ajoutez les instructions suivantes :

- Recherche ITÉRATIVE, puis affichage du pays de plus grande superficie
- Recherche RÉCURSIVE, puis affichage du pays de plus grande superficie

4.3. Exécutez et testez

5. Recherche de l'indice du pays le moins peuplé

Parmi les pays du fichier Monde2019, il existe un pays qui a moins d'habitants que tous les autres...

5.1. Dans la classe Utilitaire déclarez et codez les fonctions suivantes :

a) Recherche itérative

```
public static int indMinPopSeq(ArrayList<Pays> vPays) {  
    // { vPays non vide } =>  
    //      { résultat = indice dans vPays de l'élément  
    //                        dont le nombre d'habitants est le plus faible  
    //                        RECHERCHE ITÉRATIVE }
```

b) Recherche récursive

b1 – "Le Modèle"

```
public static int indMinPopDPR(ArrayList<Pays> vPays) {  
    // { vPays non vide } =>  
    //      { résultat = indice de l'élément de vPays dont  
    //                        le nombre d'habitants est le plus faible}
```

b2 – "Le Worker"

```
public static int indMinPopDPRWorker(ArrayList<Pays> vPays,  
                                     int inf, int sup) {  
    // { vPays non vide } =>  
    //      { résultat = indice dans vPays[inf..sup]  
    //                        de l'élément dont le d'habitants est le plus faible  
    //                        RECHERCHE RÉCURSIVE }
```

- 5.2. Dans le corps de la procédure `main` de la classe `Monde`, ajoutez les instructions suivantes :
- Recherche `ITÉRATIVE`, puis affichage de l'indice du pays le moins peuplé, de son nom et de son nombre d'habitants
 - Recherche `RÉCURSIVE`, puis affichage de l'indice du pays le moins peuplé, de son nom et de son nombre d'habitants
- 5.3. Exécutez et testez

6. Tri par fusion d'un `ArrayList<E>`

PREMIÈRE APPROCHE DU TRI FUSION

- 6.1. Créez une nouvelle classe `TestTriFusion` dans laquelle vous ajouterez :

- les procédures suivantes :

a) *Tri d'une tranche de vecteur, par fusion de ses parties gauche et droite, chacune d'elle étant triée*

```
public static void fusionTabGTabD(ArrayList<Integer> vInt,
                                int inf, int m, int sup) {
    //{ inf <= sup, m = (inf+sup)/2, vInt[inf..m] trié, vInt[m+1..sup] trié
}
// => { vInt[inf..sup] trié }
```

INDICATIONS POUR CODER LA PROCÉDURE `fusionTabGTabD`

ÉTAPE 1 : déclarer une variable `temp` de type `ArrayList` de `T` (`T` = type du vecteur paramètre de la procédure)

ÉTAPE 2 : ajouter à `temp` les éléments de `vInt[inf..m]` et de `vInt[m+1..sup]` de façon à ce que `temp` soit trié

MÉTHODE : comparer le 1^{er} élément de `vInt[inf..m]` qui n'a pas déjà été ajouté à `temp`,
au 1^{er} élément de `vInt[m+1..sup]` qui n'a pas déjà été ajouté à `temp`
puis ajouter à `temp` le plus petit de ces deux éléments

ÉTAPE 3 : ajouter à `temp` les éléments de `vInt[inf..sup]` qui n'ont pas été ajoutés lors de l'étape 2
(ils peuvent appartenir à l'un des deux sous-vecteurs)

ÉTAPE 4 : copier, élément par élément, le contenu de `temp` dans `vInt[inf..sup]`

b) *Tri récursif d'une tranche de vecteur non triée à l'origine (code fourni à copier et coller)*

```
public static void triFusion(ArrayList<Integer> vInt, int inf, int sup) {
    //{vInt[inf..sup] non vide} => {vInt[inf..sup] trié}
    if (inf < sup) {
        int m = (inf + sup) / 2;
        triFusion(vInt, inf, m);
        triFusion(vInt, m + 1, sup);
        fusionTabGTabD(vInt, inf, m, sup);
    }
}
```

- une procédure `main`

- 6.2. Dans la procédure `main`, ajoutez les instructions suivantes :

- Déclaration d'un `ArrayList` de `Integer`, initialisé avec 10 entiers non ordonnés
- Affichage du contenu de ce vecteur
- Tri de ce vecteur par appel de `triFusion`, puis affichage du contenu du vecteur trié

- 6.3. Testez

APPLICATION À UN `ARRAYLIST` DE PAYS

- 6.4. Dans la classe `Utilitaire`, ajoutez les procédures nécessaires pour trier par superficie croissante un `ArrayList` de `Pays`, selon la méthode du *Tri Fusion*
- 6.5. Dans la classe `Monde`, ajoutez les instructions nécessaires pour tester ces nouvelles procédures