CORRECTION ET TERMINAISON DES ALGORITHMES

Rendu de TD

Les réponses aux exercices sont à rendre sur votre dépôt `github` de nom « M415_Prenom_NOM » (https://github.com/uns-iut-info/), en respectant les consignes suivantes :

- ☐ tous les fichiers seront placés sous le répertoire `M415/TD_CORRECTION` (en majuscules) ;
- ☐ **dans le répertoire** `M415/TD_CORRECTION`**, vous devez créer un fichier vide (en respectant la typographie sans accents) :** `M415/TD_CORRECTION/TD_CORRECTION_Prenom1_NOM1_et_Prenom2_NOM2`
- ☐ un fichier `README.md` qui décrit votre avancement : exercices terminés, en cours, difficultés éventuelles rencontrées, ...
- ☐ quand c'est demandé, fournir un document texte de nom `exercice_num.md` avec les réponses de l'exercice numéro num ;
- ☐ les fichiers sources Java **commentés avec votre/vos noms en en-tête** doivent utiliser le codage UTF-8 et respecter les noms de l'énoncé ;
- ☐ chaque programme doit proposer dans la fonction `main()` des démonstrations en mode silencieux (aucune entrée/sortie interactive), les tests peuvent être fournis à part sous forme de modules `Junit 5`.
- ☐ Vous pouvez utiliser les squelettes Java fournis sur https://github.com/uns-iut-info/m415-skel-td1.git : `git clone https://github.com/uns-iut-info/m415-skel-td1.git`

---

1 Boucles imbriquées

Soit les deux méthodes Java suivantes :

```java
public static void bonjour1(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println("Bonjour 1");
        }
    }
}
```

```java
public static void bonjour2(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            System.out.println("Bonjour 2");
        }
    }
}
```

1. Combien de fois la méthode `bonjour1(5)` affiche « Bonjour 1 » ?
2. Combien de fois la méthode `bonjour2(5)` affiche « Bonjour 2 » ?
3. Combien de fois la méthode `bonjour1(n)` affiche « Bonjour 1 » ? Exprimez la réponse en fonction de n.
4. Combien de fois la méthode `bonjour2(n)` affiche « Bonjour 2 » ? Exprimez la réponse en fonction de n.

2 Tri de tableaux par sélection

Soit la méthode `triSelection` qui implante l'algorithme de tri suivant :

```
1  /**
2   * Preconditions :
3   * Postconditions :
4   */
5  public static void triSelection(int[] tab) {
6      for (int i = 0; i < tab.length; i++) {
7          // A1 : tab[0..i-1] est trié
8          int indiceMin = i;
9          for (int j = i + 1; j < tab.length; j++){
10             // A2 : tab[indiceMin] <= tab[k] pour tout i <= k <j
11             if (tab[indiceMin] > tab[j]) {
12                 indiceMin = j;
13             }
14         }
15         int aux = tab[i];
16         tab[i] = tab[indiceMin];
17         tab[indiceMin] = aux;
18     }
19 }
```

1. Déroulez « à la main » `triSelection(tab)` avec le tableau d'entiers `tab` suivant :
   `int[] tab = { 3, 25, 50, 8, 1, 3, 49 };`
   Vous devez donner les valeurs du tableau `tab` à chaque étape de l'algorithme (ici ligne 7 du code).
2. On considère qu'à chaque étape de l'algorithme, on peut lire un élément du tableau, le comparer à une valeur stockée en mémoire et procéder à des affectations d'un nombre constant de variables (en gros un passage dans une boucle). Quel est le nombre maximum d'étapes pour trier un tableau de taille n avec la méthode `triSelection` ? On compte donc le nombre de fois où on passe dans la comparaison `if` ligne 11.
3. Donnez un exemple de tableau avec $n = 7$ qui atteint ce nombre.
4. Quel est le nombre minimum d'étapes pour trier un tableau de taille n avec la méthode `triSelection` ?
5. Précisez les preconditions, postconditions et expliquez les assertions (ici A_1 et A_2 données dans le code) utiles à se convaincre que la méthode termine et donne la bonne réponse au probléme posé (le tri).

3 Recherche dichotomique dans un tableau

On considère la méthode Java `rechercheVite(int[] tab, int x)` donnée dans le listing de la figure 3 qui renvoie la position de x dans `tab` si $x \in$ tab, -1 sinon. Cette méthode s'applique uniquement à un tableau trié (ordre croissant).

1. Exécutez « à la main » la méthode `rechercheVite` du nombre 17 sur le tableau suivant :
   `int[] tab = {3, 6, 15, 21, 30, 33, 35, 40};`
   Quelles sont les valeurs des variables $x, gauche, droite$ et $milieu$ au cours de l'exécution de cette méthode ? Donnez un tableau avec une ligne par étape et une colonne par variable.
2. Quel est le nombre minimum d'étapes pour la `rechercheVite` d'un entier x dans un tableau de taille n ? Donnez au moins un exemple.

```java
public static int rechercheVite(int[] tab, int x) {
    int gauche = 0;
    int droite = tab.length - 1;
    int milieu;
    while (gauche <= droite) {
        milieu = (gauche + droite) / 2;
        if (x == tab[milieu])
            return milieu;
        if (x < tab[milieu])
            droite = milieu - 1;
        else
            gauche = milieu + 1;
        }
    return -1;
}
```

FIGURE 1 – Méthode rechercheVite en Java

3. Quel est le nombre maximum d'étapes pour la `rechercheVite` d'un entier x dans un tableau de taille n ? Donnez des exemples.
4. Dans la méthode `rechercheVite` on remplace la ligne 10 « `droite = milieu - 1;` » par « `droite = milieu;` ». Expliquez quel problème de correction peut se poser. Par exemple, vous exécuterez la méthode `rechercheVite` du nombre 34 sur ce tableau : `int[] tab = {3, 6, 15, 21, 30, 33, 35, 40};`

4 Recherche du maximum dans un tableau

1. Écrivez la méthode Java `max(int tab[])` qui respecte les preconditions et postconditions données dans le listing suivant. L'algorithme attendu est un simple parcours linéaire du tableau.

```java
/*
 * max
 *
 * Preconditions : tab est un tableau de n entiers quelconques
 *
 * Postconditions : le résultat est l'indice de l'élément le plus grand
 * de tab
 *
 */
public static int max(int[] tab) {

    // à compléter

    return max;
}
```

2. Soit un tableau d'entiers distincts ordonnés avec un premier segment croissant et un second segment décroissant. Trois exemples de tableaux de ce type sont : `{ 5, 8, 9, 11, 7, 6, 4 }`, `{ 5, 8, 9, 11 }` et `{ 7, 6, 4 }`.
   Écrivez la méthode Java `maxTrie(int[] tab)` qui respecte les preconditions et postconditions données dans le listing suivant. L'algorithme attendu est un simple parcours linéaire du tableau.

```java
/*
 * maxTrie
 *
 * Preconditions : tab est un tableau de n entiers distincts avec une
 * première partie triée dans l'ordre croissant et une deuxième partie
 * triée dans l'ordre décroissant
 *
 * Postconditions : le résultat est l'indice de l'élément le plus grand
 * de tab
 *
 */
public static int maxTrie(int[] tab) {

    // à compléter

    return max;
}
```

3. Écrivez la méthode Java `maxTrieDicho(int[] tab)` qui respecte les preconditions et postconditions données dans le listing suivant. L'algorithme attendu est un parcours dichotomique du tableau.

```java
/*
 * maxTrieDicho
 *
 * Preconditions : tab est un tableau de n entiers distincts avec une
 * première partie triée dans l'ordre croissant et une deuxième partie
 * triée dans l'ordre décroissant
 *
 * Postconditions : le résultat est l'indice de l'élément le plus grand
 * de tab
 *
 */

public static int maxTrieDicho(int[] tab) {

    // à compléter

    return max;
}
```

4. Donnez des tests significatifs pour ces méthodes (tests unitaires Junit ou simple appels de méthodes depuis `main()`.

5 Fusion de deux tableaux triés

1. Écrivez la méthode `int[] fusionTrie(int[] tab1, int[] tab2)` qui respecte les preconditions et postconditions données dans le listing suivant et qui minimise le nombre d'étapes nécessaires à cette fusion.
2. Donnez ce nombre d'étapes en fonction de n_1 et n_2.

```java
/*
 *
 * Preconditions : deux tableaux d'entiers triés, tab1 de taille
 * n1 et tab2 de taille n2
 *
 * Postconditions : un tableau d'entiers trié tab3 de taille n1+n2
```

```
 7   * contenant tous les éléments de tab1 et tab2
 8   *
 9   */
10 public static int[] fusionTrie(int[] tab1, int[] tab2) {
11     int[] tab3 = new int[tab1.length + tab2.length];
12     return tab3;
13 }
```

6 Palindrome

Un palindrome est un mot qui se lit aussi bien de gauche à droite que de droite à gauche. Par exemple, « rever » et « ressasser » sont des palindromes, « carotte » n'est pas un palindrome. Notez que dans cet exercice on ne traite ni les accents ni la casse.

Complétez et tester la méthode donnée dans le listing suivant qui détermine si la chaîne de caractères c est un palindrome. Pour écrire cette méthode vous devez obligatoirement compléter les « ... » dans le code avec des opérations élémentaires sur les indices ou les caractères de la chaîne c (méthode `charAt(int i)` de la classe String).

Attention : à chaque étape de la boucle, vous devez respecter l'assertion A_1

Version bonus (uniquement si vous avez fini tous les exercices du TP) : modifiez la méthode `palindrome(c)` pour produire une version `phrasePalindrome(c)` qui ignore les séparateurs comme les espaces. Dans ce cas, les chaînes telles que « elu par cette crapule » ou « engage le jeu que je le gagne » sont valides (ainsi que les exemples de Georges Perec `http://homepage.urbanet.ch/cruci.com/lexique/palindrome.htm`).

```
1  public boolean palindrome(String c) {
2    int i = 0;
3    int j = c.length()-1;
4    while ((i<j) && ..................)) {
5      //  A1 : si C1 est la sous-chaîne de c allant de l'indice 0 à l'indice i et
6      //  si C2 est la sous-chaîne de c allant de l'indice j à c.length()-1
7      //  alors C1C2 est un palindrome
8              ......................
9              ......................
10   }
11   return ..............;
12 }
```

7 Tri à bulle

Soit la méthode Java `triBulle` qui implante l'algorithme de tri donné ci-après.

1. Étudiez l'algorithme de la méthode `triBulle` en montrant le déroulement des étapes sur l'exemple du tableau `int[] tab = { 3, 25, 50, 8, 1, 3, 49 }`. Vous devez simplement donner les différentes valeurs de `tab`, étape par étape (entrée de la boucle `while`)
2. Complétez les lignes Preconditions, Postconditions ainsi que les assertions A1, A2 et A3 avec ce qui vous paraît le plus adapté pour nous convaincre de la terminaison et de la correction de cet algorithme. A1 concerne `tab[j]`, A2 et A3 concernent `tab` par rapport à l'indice j.
3. Améliorez l'algorithme pour qu'il termine dès qu'une itération (boucle `for`) sans échange est effectuée.

4. Donnez un exemple de tableau qui représente un pire cas possible pour ce dernier algorithme (qui maximise le nombre d'étapes).
5. Donnez un exemple de tableau qui représente un meilleur cas possible pour ce dernier algorithme (qui minimise le nombre d'étapes).
6. Est-ce que le tri à bulle vous semble meilleur que le tri par sélection présenté dans l'exercice 2 ?

```
1  // Preconditions : .....................................................
2  // Postconditions : ....................................................
3  public static void triBulle(int[] tab) {
4      int j = tab.length - 1;
5      while (j > 0) {
6          for (int i = 0; i < j; i++) {
7              if (tab[i] > tab[i + 1]) {
8                  int tmp = tab[i];
9                  tab[i] = tab[i + 1];
10                 tab[i + 1] = tmp;
11             }
12         }
13         // A1 : ..........................................................
14         // A2 : ..........................................................
15         // A3 : ..........................................................
16         j--;
17     }
18 }
```

Attention à ne pas oublier la ligne 16 !