TD n°4

Algorithmes de tri

L'objet de ce TD est de se familiariser avec les invariants de boucle, de manipuler différents algorithmes de tris et d'étudier leur complexité. On se donne à chaque fois un tableau d'entiers et le problème consiste à modifier l'ordre des éléments de telle sorte qu'ils soient triés après l'application de l'algorithme.

1 Deux algorithmes de tri

Exercice 1 ***Tri par Sélection***

Le principe de ce tri est de sélectioner le plus petit élément du tableau, de le repositionner au début du tableau, puis de recommencer cette opération avec le reste du tableau.

1. Faites tourner l'algorithme à la main sur le tableau | 5 | 3 | 4 | 1 | 2 |. Combien a-t-on effectué de comparaisons et d'échanges ?
2. En général, combien fait-on de comparaisons ? Combien fait-on d'échanges si le tableau est déjà trié (noter que c'est le meilleur cas) ? et si le tableau est | 2 | 3 | ... | n | 1 | (noter que c'est le pire cas) ?
3. Quel est l'invariant de boucle qui permet d'assurer que l'algorithme termine et renvoie bien un tableau trié ?
4. Écrire une méthode `triSelection(int[] t)` qui effectue le tri par sélection.

Exercice 2 ***Tri par Insertion***

Le principe de ce tri est de piocher un à un les éléments du tableau et de les insérer au bon endroit dans le sous-tableau trié constitué des éléments que l'on a déjà piochés. Ainsi, à la `i`ème étape, on sait que le sous-tableau constitué des `i-1` premièrs éléments est trié, et on y insère à sa place le `i`ème élément.

Pour cet algorithme, il faut savoir insérer un élément `a` dans un sous-tableau trié `sousTableau`. On pourrait parcourir `sousTableau` de gauche à droite (c'est-à-dire dans l'ordre croissant) jusqu'à trouver la place de `a`, puis décaler d'une case vers la droite tous les éléments de `sousTableau` plus grand que `a`. Dans la pratique, on parcourt en fait `sousTableau` de droite à gauche (c'est-à-dire dans l'ordre décroissant), en décalant chaque élément de `sousTableau` d'une case vers la droite tant que `a` est plus petit que lui. Finalement, dès que `a` est plus grand que l'un des éléments de `sousTableau`, on insère `a` à la place laissé vacante par les éléments qui ont été décalés.

1. Appliquer le tri par insertion à la main sur le tableau | 5 | 3 | 4 | 1 | 2 |. Combien a-t-on effectué de comparaisons et d'échanges ?

2. En général, combien fait-on de comparaisons et d'échanges dans le pire cas et dans le meilleur cas ?
3. Quel est l'invariant de boucle qui permet d'assurer que l'algorithme termine et renvoie bien un tableau trié ?
4. Écrire une méthode `triInsertion(int[] t)` qui effectue le tri par insertion.

2 Utiliser des tableaux triés

Exercice 3 ***Recherche dichotomique***

Le but est d'écrire une fonction de recheche d'une valeur dans un tableau d'entiers.

1. Écrire une méthode `recherche(int[] t, int a)` qui prend un tableau `t` et une valeur entière `a` et qui retourne une position de `a` dans `t`. Si `a` n'est pas dans `t`, la méthode retourne `-1`.
2. Dans le cas où le tableau est trié par ordre croissant, il est possible de retrouver plus rapidement la valeur recherchée `a` par dichotomie. Le principe est de comparer à `a` la valeur stockée au milieu du tableau : si elle est plus grande que `a` , il faut chercher dans la moitié inférieur du tableau, et sinon dans la moitié supérieure. On recommence alors sur la portion du tableau de taille moitié et ainsi de suite jusqu'à trouver la valeur. Écrire la méthode `rechercheDichotomie(int[] t, int a)` basée sur cette approche. Identifier un invariant.
3. Pour les deux algorithmes, exhiber un invariant qui permet d'assurer que l'algorithme termine et renvoie bien la position de `a` dans le tableau `t` ?
4. Comparer la complexité de nos deux algorithmes.

Exercice 4 ***Comparaison des tableaux***

On dit que deux tableaux sont
– *égaux* si ils contiennent les mêmes éléments aux mêmes positions ;
– *similaires* si ils contiennent les mêmes éléments mais pas forcément aux mêmes positions ;
– *comparables* si l'ensemble des valeurs qu'ils contiennent est le même.

Par exemple les tableaux | 5 | 3 | 4 | 1 | 2 | 5 | 2 | et | 2 | 4 | 5 | 2 | 3 | 5 | 1 | sont similaires mais pas égaux, et les tableaux | 5 | 3 | 4 | 1 | 2 | 5 | 2 | et | 2 | 4 | 3 | 5 | 1 | sont comparables mais pas similaires.

1. Écrire une méthode qui teste si deux tableaux sont égaux.
2. Écrire une méthode qui teste si deux tableaux sont similaires.
3. Écrire une méthode qui teste si deux tableaux sont comparables (on pourra écrire une méthode qui efface dans un tableau trié toutes les répétitions).