Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

Objectifs: Algorithmes plus complexes sur les tableaux.

Exercice 1 : Trié ?

Un tablau est trié dans le sens croissant si la valeur dans chaque case est inférieure ou égale à la valeur de la case précédente. Écrire la fonction `estTrie` qui teste si un tableau d'entiers est trié. La fonction doit passer ce test.

```
1 void testEstTrie () {
2   assertEquals(true,estTrie(new int[]{2,4,6,6,7}));
3   assertEquals(false,estTrie(new int[]{2,1,6,7,8}));
4   assertEquals(true,estTrie(new int[]{}));
5   assertEquals(true,estTrie(new int[]{1}));
6 }
```

Exercice 2 : Recherche d'une valeur par dichotomie

Dichotomie : Du grec ancien dikhotomia (« division en deux parties »)

Nous nous intéressons ici à la recherche d'une valeur dans un tableau trié dans l'ordre croissant. Cette supposition va nous permettre de définir un algortihme plus efficace que pour un tableau non trié : la recherche dichotomique.

L'algorithme de recherche dichotomique correspond à la description suivante:

Comparer le nombre cherché à la valeur contenue dans la case située au milieu du tableau,
(1) si il y a égalité, retourner l'indice courant,
(2) si la valeur recherchée précède la valeur actuelle du tableau, continuer la recherche dans le demi-tableau à gauche de la position actuelle,
(3) si la valeur recherchée suit la valeur actuelle du tableau, continuer la recherche dans le demi-tableau à droite de la position actuelle.

1. Essayez cet algorithme en recherchant la valeur 80 dans le tableau à 10 valeurs ci-dessous

valeurs	10	20	30	40	50	60	70	80	90	100
indices	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

2. Essayez l'algorithme dans ce même tableau, mais en recherchant la valeur 35.
3. Comment peut-on représenter "le demi-tableau à gauche de la position actuelle" et le "demi-tableau à droite de la position actuelle" qui sont mentionnés dans les cas (1) et (2) de l'algorithme ?
4. Avec cette représentation, comment calculer l'indice de la case au milieu du tableau ?
5. Quand faut-il s'arrêter avec cet algorithme ?
6. Maintenant que vous comprenez mieux cet algorithme, écrivez la fonction `int fastSearch(int[] tab, int valeur)` qui retourne l'indice de `tab` où se trouve `valeur`, ou retourne -1 si `valeur` n'est pas présente dans `tab`.
7. Quel est à votre avis l'avantage de ce type de recherche ?

Exercice 3 : Trier un tableau: tri par sélection

Dans cet exercice vous allez écrire un algorithme de tri; c'est un algorithme qui permet de ranger les valeur d'un tableau dans l'ordre croissant. Il existe différents algorithmes permettant de trier un tableau, le *tri par sélection* est parmi les plus simples à implémenter. Le principe de cet algorithme est de sélectionner la valeur minimale du tableau et la ranger dans la première case, puis la valeur minimale parmi celles qui restent et la ranger dans la deuxième case, et ainsi de suite jusque ce que toutes las valeurs sont épuisées.

Plus précisément, à tout moment le tableau contient une partie déjà triée au début du tableau, et le reste jusque la fin du tableau est non encore trié. Au début, la partie triée est vide. À chaque itération on augmente d'une case la partie triée comme suit:

— trouver la valeur minimale dans la partie non encore triée;
— placer cette valeur minimale à la fin de la partie triée, augmentant ainsi d'une case la taille de la partie triée.

En effectuant autant d'itération que le nombre de cases du tableau (moins un), on obtient un tableau trié.

Voici sur un exemple le calcul effectué lors de la 4^{e} et 5^{e} itération. La limite entre les parties triée et non triée est signifiée par une double barre.

Au début de la 4^{e} itération, la partie triée est constituée de 3 cases:

10	20	30	80	90	40	70	50	60	100

La valeur manimale de la partie non triée est 40. On échange la valeur 40 avec la valeur qui se trouve immédiatement après la partie triée, ici 80. À la fin de la 4^{e} itération la partie triée est constitué de 4 cases, et le tableau est comme ceci:

10	20	30	40	90	80	70	50	60	100

Au début de la 5^{e} itération, le tableau est comme ci-dessus et la valeur minimale de la partie non triée est 50. Comme pour l'itération précédente, on échange la valeur minimale avec la valeur se trouvant immédiatement après la partie triée. Le tableau résultant est:

10	20	30	40	50	80	70	90	60	100

Vous allez écrire la fonction `void triSelection(int[] tab)` qui permet de trier le tableau `tab`. Commencer par répondre à ces questions:

1. Déroulez le tri par sélection sur l'exemple ci-dessus après la 5^{e}, jusque obtenir un tableau trié.
2. Pour une itération i, quelle est la taille de la partie triée ? Quel est l'indice de la première case de la partie non triée ?
3. Écrire le code permettant de trouver la valeur minimale de la partie non triée du tableau `tab` lors de l'itération i.
4. Écrire la fonction `triSelection`.

Vous trouverez sur Moodle des fichiers contenant des tests pour la recherche dichotomique et pour les algorithmes de tri. Cela vous permettra de tester votre algorithme si celui-ci diffère de la correction donnée en TD.

Prolongement

Exercice 4 : Tri par insertion

Le *tri par insertion* est un autre algorithme de tri, dont le principe est le suivant. À tout moment, le tableau est constitué d'une partie triée au début, suivie d'une partie non triée. À chaque itération on augmente la taille de la partie triée en prenant la première valeur de la partie non triée et en l'insérant à la bonne place de la partie triée.

Voici un exemple de déroulement de l'algorithme sur le tableau qui initialement contient ces valeurs:

70	90	60	30	40	80	10	100	30	50

Au début de la 4^{e} itération, les valeurs qui se trouvaient initialement dans les 4 premières cases du tableau sont triées:

30	60	70	90	40	80	10	100	30	50

Lors de la 4^{e} itération, on va insérer la première valeur de la partie non triée, ici 40, à la bonne place, c'est à dire entre 30 et 60. La partie triée est étendue d'une case, en décalant vers la droite les valeurs 60, 70 et 90:

30	40	60	70	90	80	10	100	30	50

Vous devez écrire la fonction `void triInsertion(int[] tab)` qui implémente le tri par insertion. Répondez d'abord à ces questions:

1. À l'itération i, quelle est la taille da la partie triée ? Quel est l'indice de la première case de la partie non triée ?
2. Combien d'itérations sont nécessaires pour trier un tableau de taille n ?
3. En supposant que l'indice de la première case de partie non triée de `tab` est j, écrire le code qui permet d'insérer la première valeur de la partie non triée dans la partie triée, en décalant vers la droite les valeurs plus grandes de la partie triée.
4. Écrire la fonction `triInsertion`.