TP2 – Comparaison d'algorithmes de tri

Pour débuter, créez un nouveau projet dans Eclipse nommé **TP Tableau**.
Pour gérer la saisie de l'utilisateur, vous devrez utiliser la classe `Saisie` fournie dans **saisie.jar**

Soit un tableau T de taille N où chaque cellule c_i avec $0 \leq i \leq N-1$ contient un nombre. Trier T par ordre croissant consiste à réordonner les valeurs des cellules de manière à ce que, quel que soient i et j, on a : $0 \leq i < j \leq N-1$ et $c_i \leq c_j$

Pour passer d'un tableau non trié à un tableau trié par ordre croissant, il existe plusieurs méthodes de tri. L'objectif de ce TP est de programmer deux de ces méthodes pour pouvoir ensuite comparer leur efficacité.

Exemple

Pour illustrer les deux méthodes de tri décrites ci-dessous, nous prendrons le tableau T comprenant 5 éléments et dont les éléments sont :

8	5	7	1	2
0	1	2	3	4

Exercice 1 - Classe Tableau

Créer une classe Tableau qui contiendra un tableau statique de N entiers.
Faire trois constructeurs :

- Un constructeur sans paramètre dans lequel on demandera à l'utilisateur de saisir l'ensemble des valeurs du tableau sous la forme : N V_0 V_1 … V_{N-1} où N est le nombre de valeurs souhaité et V_0 à V_{N-1} les N valeurs.
- Un constructeur avec un entier N en paramètre, où N correspond au nombre de valeurs souhaité. Les N valeurs seront données aléatoirement par la machine et seront comprises entre 1 et 100.
- Un constructeur par recopie qui prend un objet de type Tableau en paramètre.

Ecrire une méthode qui permet d'afficher le tableau.

Exercice 2 - Tri par sélection

Le tri par sélection consiste à chercher le plus petit élément du tableau et à le placer en première position. Une fois le plus petit élément positionné en première position, on recommence la même opération en commençant à la deuxième position du tableau : on

cherche le plus petit élément compris dans le tableau entre la 2e position et la fin du tableau, puis on place cet élément à la deuxième position. On recommence ainsi en partant de la 3e position, puis de la 4e et ainsi de suite jusqu'au bout du tableau.

Soit le tableau de départ suivant	8 5 7 1 2 0 1 2 3 4
Le plus petit élément du tableau est 1. On l'insère à la première position du tableau	1 5 7 8 2 0 1 2 3 4
On recommence en partant de la 2e position : le plus petit élément est 2. On l'insère donc en deuxième position.	1 2 7 8 5 0 1 2 3 4
On recommence en partant de la 3e position : le plus petit élément est 5. On l'insère donc en troisième position.	1 2 5 8 7 0 1 2 3 4
On recommence en partant de la 4e position : le plus petit élément est 7. On l'insère donc en quatrième position.	1 2 5 7 8 0 1 2 3 4

Ecrire une méthode dans la classe Tableau qui permette de réaliser le tri par sélection

Exercice 3 - Insertion d'un élément

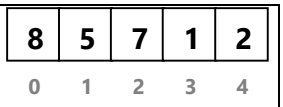
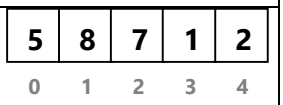
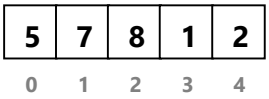
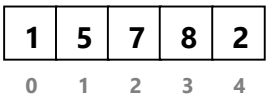
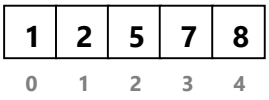
Pour réaliser l'algorithme de tri par insertion décrit ci-dessous, vous aurez besoin d'une fonction d'insertion d'un élément. Considérons que cet élément se trouve à la position i et doit être insérer à la position j (avec j < i). Pour insérer T[i] à la position j, il faut :

1. Sauvegarder la valeur de T[i] dans une variable x ;
2. Décaler d'une position tous les éléments compris entre la position j et la position i-1 ;
3. Mettre la valeur de x à la position j.

Ecrire une méthode dans la classe Tableau qui permette de faire l'insertion d'un élément.

Exercice 4 - Tri par insertion

Le tri par insertion consiste à classer les deux premiers éléments du tableau. Une fois que les deux premiers sont ordonnés, on prend l'élément qui suit et on le classe à son tour dans ce qui a déjà été classé. Pour chaque élément i du tableau, on sait que les éléments de 0 à i-1 sont déjà classés. On va chercher la position j parmi les i-1 premiers éléments de manière à ce que T[i] < T[j]. On insère alors T[i] à la position j.

Soit le tableau de départ suivant	
On commence par classer les 2 premiers éléments	
Puis on prend l'élément à la position i=2 (T[2] = 7), on cherche la position j dans le sous tableau déjà trié ([5 ; 8]) de manière à avoir j>=0, et T[j-1]<T[i]<T[j]. On insère alors l'élément de la position i à la position j. On obtient alors le résultat ci-contre.	
Puis on prend l'élément à la position i=3 (T[3] = 1), on cherche la position j dans le sous tableau déjà trié ([5 ; 7 ; 8]) de manière à avoir j>=0, et T[j-1]<T[i]<T[j]. On insère alors l'élément de la position i à la position j. On obtient alors le résultat ci-contre.	
Enfin on prend l'élément à la position i=4 (T[4] = 2), on cherche la position j dans le sous tableau déjà trié ([1 ; 5 ; 7 ; 8]) de manière à avoir j>=0, et T[j-1]<T[i]<T[j]. On insère alors l'élément de la position i à la position j. On obtient alors le résultat ci-contre.	

Ecrire une méthode dans la classe Tableau qui permette de réaliser le tri par insertion.

Exercice 5 : Comparaison des différents tris

Les méthodes de tris seront évaluées au moyen de trois critères :

- le nombre d'affectations effectuées au sein du tableau,
- le nombre de tests réalisés sur les éléments du tableau,
- le temps nécessaire pour trier le tableau.

La comparaison des tris consistera à trier un même tableau avec les 2 méthodes de tris. Pour que la comparaison entre les 2 méthodes soit pertinente, vous devrez comparer 20 fois les 2 méthodes de tri avec un tableau de taille N : à chaque itération, vous créerez un nouveau tableau qui sera utilisé pour les 2 méthodes. De plus, pour analyser l'évolution des algorithmes de tri en fonction de la taille du tableau, vous commencerez par comparer les méthodes de tri sur des tableaux de 500 éléments, puis sur des tableaux de 1000 éléments et ainsi de suite jusqu'à des tableaux de 10 000 éléments en augmentant à chaque fois de 500 éléments.

Pour réaliser cette évaluation, vous devrez créer une classe Mesure qui vous permettra de stocker les valeurs des différents critères. Une instance de Mesure devra être passée en argument des méthodes de tri pour pouvoir compter le nombre de tests et d'affectations.

Le temps peut être calculé au moyen de la méthode statique `nanoTime`() se trouvant dans la classe `System`.

L'algorithme de comparaison pourra être réalisé dans la méthode main.

BONUS – D'autres tris récursifs

Tri par fusion

Le tri par fusion est un algorithme qui propose de découper la liste en 2 parties de longueur égale (à un élément près) que l'on doit ensuite trier pour enfin fusionner ces 2 listes triées.
La fonction récursive triFusion aura donc besoin de deux fonctions :

- **partagerListeEn2** qui, étant donné un tableau, retourne un couple de tableau de longueurs équivalentes formés des éléments du tableau à partager
- **fusionner2ListesTriees** qui, étant donnés 2 tableaux triés dans l'ordre croissant, construit dans l'ordre croissant le tableau des éléments de ces 2 tableaux.

Tri rapide

Le tri rapide est un algorithme de tri très utilisé pour sa relative simplicité et sa rapidité.
On peut en donner une solution récursive :

- si le tableau a moins de 2 éléments, c'est fini, le tableau est déjà triée
- sinon
    - on choisit un élément du tableau au hasard qu'on appelle pivot; ici on prendra le premier élément du tableau pour simplifier,
    - on partitionne le tableau en 2 : le tableau des éléments qui sont inférieurs au pivot et le tableau des éléments qui sont supérieurs au pivot,
    - on trie ensuite ces 2 tableaux suivant la même méthode (2 appels récursifs)
    - on construit la liste résultat en concaténant les 2 tableaux triés sans oublier le pivot entre les 2.