TP5 - Algorithmes de tris

PARTIE 1 : tri insertion, tri sélection, complexité

1.1 Simulations (papier/crayon)

Vous disposez de ce tableau de valeurs que l'on souhaite trier :

10	-3	17	15	-5	11
0	*1*	*2*	*3*	*4*	*5*

Simuler le tri par sélection, puis le tri par insertion, en utilisant pour cela le tableau fourni dans le fichier Excel (onglet Partie 1) et le poly d'algo (page 18).

1.2 Complexité (papier/crayon)

On souhaite comparer l'algorithme de tri sélection et l'algorithme de tri insertion (avec recherche séquentielle) dans 2 contextes particuliers : un tableau trié dans l'ordre inverse et un tableau déjà trié dans le bon ordre.

On considère donc les 2 tableaux suivants : tab1 = {6, 5, 4, 3, 2, 1} tab2 = {1, 2, 3, 4, 5, 6]

Répondre aux questions suivantes pour chaque tri et chaque tableau :

- Donner dans un tableau les étapes successives de ce tri.
- Combien de comparaisons faut-il pour trier ces 6 entiers ?
- On suppose qu'on a plus généralement les **n** premiers entiers rangés dans l'ordre décroissant. Combien le tri effectue-t-il de comparaisons ?
- Répondre à la même question pour les entiers rangés dans l'ordre croissant.
- Quelle conclusion peut-on en tirer ?

1.3 Programmation tri insertion

Dans l'exercice précédent, l'algorithme de tri par insertion a utilisé une recherche séquentielle pour trouver la position dans laquelle devrait se trouver l'élément courant. Il en résulte une complexité **$O(n^2)$**. Si on utilise une recherche dichotomique, la partie recherche de cet algorithme aurait une complexité de **$O(n \log_2 n)$**, ce qui semble bien meilleur. Mais cette complexité est négligeable face aux décalages qui nécessitent **$O(n^2)$**, donc à nouveau un algorithme de complexité **$O(n^2)$**. On peut donc faire une recherche séquentielle sans changer la complexité de l'algorithme.

Dans cet exercice on vous demande d'écrire une version du tri insertion qui effectue **en même temps** la recherche séquentielle de la position dans laquelle devrait se trouver l'élément courant **et** les éventuels décalages. Il faut démarrer les décalages à partir de la case courante, en allant "vers la gauche", et en s'arrêtant dès que tab[j-1] est plus petit que la valeur recherchée.

Reprendre votre projet avec les classes `Appartement` et `Pièce` du TP3 et ajouter à la classe `Appartement` une méthode `trier` qui permet de trier les chambres (instances de la classe `Pièce`) d'une résidence universitaire (instance de la classe `Appartement`) par ordre croissant de leur surface en utilisant l'algorithme de tri décrit précédemment. Modifier la méthode `afficher` de la classe `Pièce` pour afficher la surface des pièces plutôt que leurs dimensions. Modifier également l'initialisation de la résidence pour que les chambres aient des surfaces différentes. Tester en affichant la résidence avant et après le tri.

PARTIE 2 : Comparaisons de différents tris

Le but de cet exercice est double : implémenter 2 versions du tri à **bulles**, puis comparer la complexité de ces 2 algorithmes avec celle de 2 autres algorithmes de tri, le tri par **sélection** et le tri dit **rapide** (qui sont déjà implémentés). Le tri rapide (*quick sort*) est un algorithme récursif de type dichotomique, sa complexité est en **$O(n \log_2 n)$**. (Revoir sa description dans le poly d'algo page 22). Vous comparerez la complexité de ces 4 algorithmes en effectuant un grand nombre d'exécutions de chacun d'entre eux et en comparant la complexité moyenne, donnée par le nombre moyen de comparaisons et d'écritures effectuées sur le vecteur contenant vos données.

Créez un nouveau projet avec les fichiers disponibles sur Moodle. Ces fichiers vous fournissent la définition des algorithmes de tri par sélection et de tri rapide, ainsi qu'un programme principal permettant à l'utilisateur de choisir l'un des 4 algorithmes de tri et de l'exécuter sur des vecteurs d'entiers ayant différentes caractéristiques.

Étudiez le code fourni et testez le programme pour avoir une première idée du projet sur lequel vous allez travailler. (Vous constaterez qu'aucun traitement n'est encore effectué pour les tris à bulles).

?

2.1 Le tri à bulles "de base"

Le **tri à bulles** ou **tri par propagation** est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau (ou d'un vecteur dans le cadre de ce TP) vers les cases de fin du tableau, comme les bulles d'air remontent à la surface d'un liquide. L'algorithme parcourt le tableau et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération en s'arrêtant une case plus tôt. En effet, après le premier passage, la plus grande valeur est à sa place dans la dernière case, on peut donc pour le second passage ne considérer que les *nbVal-1* premières valeurs, et on placera la deuxième plus grande valeur dans l'avant-dernière case, et ainsi de suite. Après n-1 parcours du tableau, on est assuré d'avoir fini.

Programmez ce tri à l'emplacement prévu dans le fichier vecteurTrie.cpp. Vous manipulerez un vecteur d'entiers. N'oubliez pas les deux compteurs qui sont chargés de donner respectivement le nombre de comparaisons et le nombre d'écritures dans le vecteur.

Vérifiez votre implémentation en exécutant plusieurs fois le programme principal avec des options différentes.

2.2 Le tri à bulles optimisé

L'algorithme n'est pour l'instant pas très efficace : il se peut que le tableau devienne trié bien avant les n-1 parcours : le programme continuerait quand même. On peut modifier le programme pour que le tri s'arrête dès que l'on réalise un parcours dans lequel aucun échange d'éléments n'est effectué. Cependant, on ne se contentera pas de cette première optimisation. Une optimisation supérieure est réalisée par l'algorithme suivant, où l'on mémorise la position de la dernière permutation lors d'un parcours du tableau et on arrête à cette position le parcours de la boucle suivante.

Programmez ce tri à l'emplacement prévu, toujours sans oublier d'y ajouter les compteurs de comparaisons et d'écriture.

Procédure triBullesOptimisé(tab)
{Lors de chaque parcours, la valeur la plus grande encore non triée est déplacée en queue de tableau juste devant les valeurs déjà triées. ***finTemp*** *est l'indice de la dernière case ayant changé de valeur lors du parcours courant;* ***fin*** *est la valeur de* ***finTemp*** *à la fin du parcours courant. La valeur dans cette case est définitivement triée ainsi que celles qui suivent dans le tableau }*

```
paramètres    (D/R) tab : tableau d'entiers
variables     i, fin, finTemp: entiers
début
    fin <-- tab.taille()
    tant que fin > 1
        finTemp <-- 1
        pour i <-- 1 à fin-1 faire
            si tab[i-1] > tab[i]
                permuterLesValeursDesPositions(i-1, i)
                finTemp <-- i
            fsi
        fpour
        fin <-- finTemp
       {permet l'optimisation du nombre d'itérations de la boucle pour}
    ftq
fin
```

2.3 Comparaison d'algorithmes et complexité

Vous allez maintenant comparer les différents algorithmes implémentés, en faisant varier les paramètres d'exécution :

- l'algorithme de tri : bulles, bulles-opt, sélection ou rapide
- le nombre d'exécutions (10 par exemple)
- l'ordre initial : aléatoire, croissant ou décroissant
- le nombre de valeurs (100 par exemple)
- la valeur maximale (plus elle est faible, et plus y aura de valeurs identiques)

Pour cela, utilisez le fichier excel fourni (onglet Partie 2), qui contient un tableau de résultats analogue à celui donné ci-dessous et réalisez une étude de complexité en procédant à des expérimentations judicieuses. Réfléchissez à l'ordre de grandeur de la complexité en fonction de *n* (la taille du vecteur à trier).

Algorithme de tri	Nb exécutions	Ordre initial	Nb valeurs	Valeur maximale	Nb comparaisons	Nb écritures
...	...	...	...	...	...	...

- Est-ce que le nombre exact de comparaisons correspond au calcul théorique ?
- Est-ce que le nombre exact d'écritures correspond au calcul théorique ?
- Est-ce que la complexité moyenne obtenue est conforme à ce qui est attendu ?
- Quelles conclusions pouvez-vous faire ?