TD3 : Tri

Dans ce TD, nous supposerons que les fonctions demandées sont les méthodes d'une classe Sort dont les attributs suivants sont membres et par conséquent accessibles depuis chaque fonction membre de la classe:

```
int         *x;        // Pointeur vers le premier élément du tableau à trier
int         NbElements;// Nombre d'éléments dans le tableau
```

Nous supposerons également que l'allocation mémoire et le remplissage du tableau sont réalisés lors de l'appel du constructeur de la classe.

Exercice 1

a. Ecrire la méthode Permute qui permet de permuter les éléments i1 et i2 du tableau x.

```
// Permute les élements i1 et i2 du tableau
void Sort::Permute(int i1,int i2)
```

b. Ecrire une méthode qui retourne l'indice du plus petit élément du tableau x compris entre les indices [Start ; End[.

```
// Retourne l'indice du plus petit élément du tableau
// Compris entre les indices Start (inclus) et End (non inclus)
int Sort::Min(int Start,int End)
```

c. Ecrire la méthode Tri_Selection qui permet de trier le tableau x dans l'ordre croissant en utilisant la méthode de tri par sélection.

```
// Trie les élements de x dans l'ordre croissant
// en utilisant la méthode de tri par sélection
void Sort::Tri_Selection()
```

Exercice 2

a. Ecrire la méthode décale qui décale tous les éléments compris entre [Start;Stop[de une cellule vers la droite. La valeur située à l'indice stop est retournée par la méthode. Nous supposerons que Start est inférieur à Stop.

```
// Décale tous les élément de 1 cellule
// de start vers stop. x[stop] est perdu
// Start < Stop
int Sort::Decale(int Start,int Stop)
```

b. Ecrire une méthode qui insert l'élément x[Stop] à sa place dans le tableau entre les éléments [Start;Stop[en supposant que cette portion du tableau est déjà ordonnée dans l'ordre croissant.

```
// Insere l'élément x[Stop] dans le tableau
// entre les éléments Start et Stop
// Les éléments sont décalés vers l'indice Stop
void Sort::Insert(int Start,int Stop)
```

c. Ecrire la méthode Tri_Insertion qui permet de trier le tableau x dans l'ordre croissant en utilisant la méthode de tri par insertion.

```
// Trie les élements de x dans l'ordre croissant
// en utilisant la méthode de tri par insertion
void Sort::Tri_Insertion()
```

Exercice 3

a. Ecrire la méthode Tri_Bulles qui permet de trier le tableau x dans l'ordre croissant en utilisant la méthode de tri à bulles.

```
// Trie les élements de x dans l'ordre croissant
// en utilisant la méthode de tri à bulles
void Sort::Tri_Bulles()
```

b. Déterminer le nombre de permutations dans le meilleur cas sur un tableau de n éléments. En déduire la complexité dans le meilleur cas.

c. Déterminer le nombre de permutations dans le pire cas sur un tableau de n éléments. En déduire la complexité dans le pire cas.

Exercice 4

a. Ecrire la méthode Pivote qui prend l'élément x[Start] comme pivot et le place à son emplacement définitif. Tous les éléments avant le pivot doivent être inférieurs et tous les élements après le pivot doivent être supérieurs au pivot. La fonction retourne l'indice du pivot après traitement.

```
// Prends l'élément x[Start] comme pivot et
// le place de façon définitive afin que tous les
// éléments avant le pivot soient plus petits,
// et tous les élements après le pivot soient plus
// grands. Le fonction retourne l'indice du pivot.
int Sort::Pivote(int Start, int Stop)
```

b. Ecrire la fonction QuickSort_Recursif qui permet de trier le tableau x dans l'ordre croissant en utilisant la méthode de tri rapide sur l'intervalle [Start;Stop[.

```
// Trie les élements de x dans l'ordre croissant
// en utilisant la méthode de tri rapide
void Sort::QuickSort_Recursif(int Start, int Stop)
```

c. Ecrire la fonction QuickSort qui permet de trier l'ensemble du tableau x dans l'ordre croissant en utilisant la méthode de tri rapide.

```
// Trie les élements de x dans l'ordre croissant
// en utilisant la méthode de tri rapide
void Sort::QuickSort()
```