IUT de BAYONNE PAYS BASQUE - DÉPARTEMENT INFORMATIQUE
Semestre 1- 2021-2022

R1.01 : Initiation au développement (partie 2)
Feuille TD n° 4

Algorithmes classiques pour d'éléments homogènes : Tris

Objectifs :
1.- Savoir appliquer un modèle d'algorithme connu.
2.- Savoir analyser un problème et identifier un modèle connu d'algorithme participant à sa résolution
3.- Sensibilisation à la notion de complexité temporelle

On souhaite écrire des sous-programmes de tri d'un tableau, (par exemple, d'entiers) qui effectuent le tri des éléments d'un tableau **`tab`**, contenant **`nbTab`** éléments, selon des stratégies différentes. On supposera que le tri doit être fait par ordre **croissant** des valeurs de **`tab`**. En voici les déclarations en C++ :

```
void triParSelectionDePlace ( int tab [], unsigned int nbTab);
// tri CROISSANT des nbTab (>0) éléments de tab (avec d'éventuels doublons)
// par la méthode de tri de même nom
void triParInsertion ( int tab [], unsigned int nbTab);
// tri CROISSANT des nbTab (>0) éléments de tab (avec d'éventuels doublons)
// par la méthode de tri de même nom
void triBulle ( int tab [], unsigned int nbTab);
// tri CROISSANT des nbTab (>0) éléments de tab (avec d'éventuels doublons)
// par la méthode de tri de même nom
```

1.- Tri de la bulle

Principe général :

- On choisit :
  - le sens de parcours, par exemple de gauche à droite.
  - le sens du tri, ici, le sujet dit croissant.
- Lors du parcours, ici démarrant au premier indice de gauche, il s'agira de faire circuler l'extremum vers la droite du tableau.
  - Compte-tenu des choix réalisés, cet extremum est le plus grand élément. On fait donc circuler le plus grand élément en comparant 2 à 2 les éléments du tableau, et en les échangeant si l'élément de gauche est plus grand que l'élément de droite.
  - Durant ce parcours, l'élément le plus à gauche sera comparé avec son premier voisin de droite, et déplacé vers la droite par échanges successifs, jusqu'à ce qu'il rencontre un élément plus grand que lui. L'élément plus grand poursuit alors le même mouvement, et ainsi de suite jusqu'à ce que la dernière place du tableau soit occupée par l'élément le plus grand du tableau
  - A la fin de ce parcours, la place la plus à droite sera occupée par le plus grand élément du tableau.
- Il faut alors recommencer un nouveau parcours, depuis la case la plus à gauche, mais cette fois, l'élément le plus grand rencontré lors du parcours sera placé à l'avant-dernière case du tableau.
- Un nouveau parcours permettra alors de faire circuler le plus grand élément du parcours à l'avant-avant-dernière case du tableau.
- A chaque itération (nouveau parcours), la portion de tableau parcourue diminue.
- Lors de la dernière itération, la portion de tableau parcourue aura 2 cases. A la fin de ce dernier parcours, le tableau sera complètement trié.

Illustration
Soit le tableau (82, 10, 95, 5, 27, 110, 70, 31, 3) à trier par ordre croissant de valeurs.

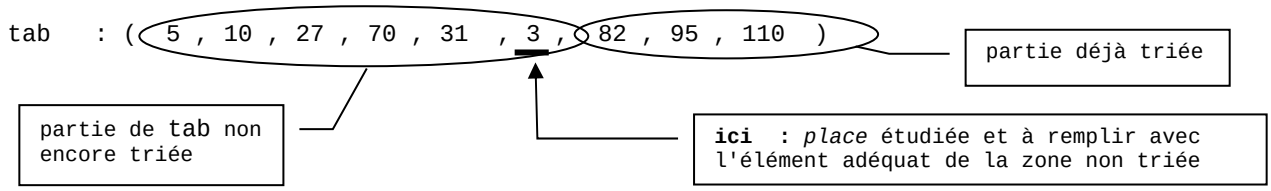
Se reporter au polycopié de cours pour suivre l'évolution du tableau lors de l'exécution pas à pas de cet algorithme.

A.- Mise en œuvre de l'Algorithme du tri Bulle

Partie 1 : *Faire monter la bulle vers la dernière case de la portion de tableau non triée*

On suppose tout d'abord que `tab` est déjà partiellement trié (par ordre croissant de valeurs) et qu'il est dans l'état suivant :

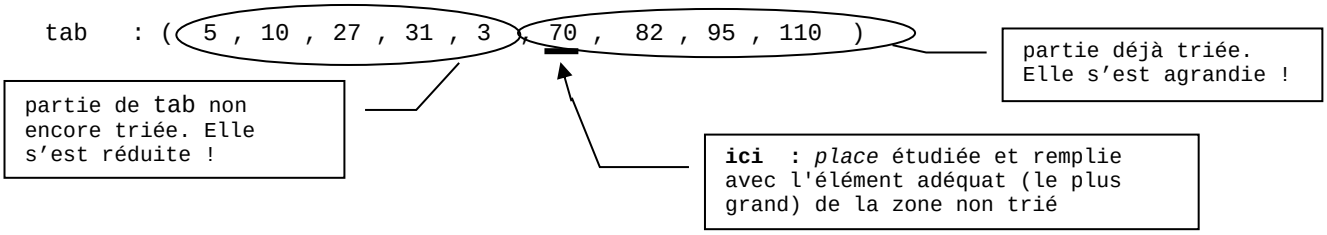
- la partie supérieure (c'est à dire la tranche `tab (ici+1..nbTab-1)`) est déjà triée
- la partie inférieure (c'est à dire la tranche `tab (bDeb..ici)`) n'est pas encore triée, avec `bDeb=0`



La stratégie pour résoudre cette problématique est la suivante :

- Parcours séquentiel complet (de gauche à droite, par indices croissants) de la zone non encore triée, avec action conditionnelle
- Action conditionnelle : **échanger élément courant et son suivant** (à l'indice +1)
  Condition : élément courant > à son suivant

Etat du tableau après ce parcours :



Travail à faire

1) Écrire en C++ la déclaration du sous-programme (fonction ? procédure ?) `faireMonterLaBulleIci` mettant en œuvre cette stratégie.
2) Écrire l'algorithme du sous-programme.

Partie 2 : *Trier tout le tableau*

Travail à faire

3) A partir de la situation décrite à l'étape 1, que faut-il faire pour trier le tableau `tab` ?
4) Écrire l'algorithme du `triBulle` en respectant la déclaration fournie dans l'encadré.
   Accompagner l'algorithme des spécifications internes appropriées.

B.- Comparaison des performances du tri Bulle et du tri par sélection de Place

(sensibilisation à la complexité temporelle)

Les 2 algorithmes ont été développés en utilisant la même technique :

a) On considère la portion de tableau `tab(0..ici)` non encore triée, la portion `tab(ici+1..nbTab-1)` déjà triée
b) On écrit l'algorithme nécessaire à placer en position `ici` le plus grand élément de la portion `tab(0..ici)`
c) On généralise l'algorithme de l'étape b) pour trier tout le tableau

On va comparer les performances des algorithmes réalisés à l'étape b) pour chacun des tris vus : tri par sélection de Place et tri de la Bulle

Travail à faire

5) Calculer :
   — le nombre de comparaisons, et le nombre correspondant d'accès au tableau
   — le nombre d'échanges dans le tableau, et le nombre correspondant d'accès au tableau

   dans les actions :
   `faireMonterLaBulleIci (bDeb..ici)` et `placerLaPlusGrandeValeurDeTab(bDeb..ici)EnPositionIci`

   avec `bDeb = 0`

6) Quelle(s) conclusions peut-on en tirer ?

C.- Généralisation à des tableaux d'informations complexes

On suppose maintenant que le tableau ne contient pas que des entiers, mais des informations complexes. Considérons par exemple une collection de personnes regroupées dans un agenda.
Pour ce faire, deux types Adresse et Personne ont été définis.

```
struct Personne
{
    string nom;
    string prenom;
    Adresse adresse;
};
```

avec

```
struct Adresse
{
    string numRue;
    string nomRue;
    unsigned short int codePostal;
    string nomVille;
};
```

Travail à faire

7) Faire les modifications qui s'imposent au niveau des ***algorithmes*** `FaireMonterLaBulleIci` et `triBulle` pour trier un agenda, implémenté sous la forme d'un tableau `tab` de `UnePersonne`, par ordre alphabétique croissant sur les noms des personnes.

8) Est-ce que les calculs de performance précédemment calculés ont changé ?

2.- Tri par insertion (ou Tri par sélection de <u>valeur</u>)

Principe

On ajoute un par un de nouveaux éléments à un tableau déjà trié en les insérant « à sa place », c'est-à-dire à une place telle que le nouveau tableau soit aussi trié. Illustrons la méthode sur un exemple.

Illustration

Soit le tableau (**82**, *<u>10</u>, 95, 5, 27, 110, 70, 31, 3*)
La tranche de tableau tab(0..0), est triée, puisqu'elle ne contient qu'un élément, la valeur 82.
On veut insérer 10 dans la tranche de tableau délimitée par les indices (0..1) de telle sorte que cette tranche soit aussi triée. Pour ce faire, il faut décaler 82 vers la droite pour pouvoir insérer la valeur 10 devant.
Le tableau résultant est : (**10, 82**, *95, 5, 27, 110, 70,31, 3*)

On veut insérer 95 dans la tranche (**10, 82**, *<u>95</u>)* : il est à sa place, on obtient : (**10, 82**, **95**, *5, 27, 110, 70,31, 3*)

On veut insérer 5 dans la tranche (**10, 82**, **95**, *<u>5</u>*) : on décale vers la droite les éléments qui lui sont plus grands, puis on insère 5 à sa place et on obtient : (**5, 10, 82**, **95,** *27, 110, 70,31, 3*)

On veut insérer 27 dans la tranche (**5, 10, 82**, **95,** *<u>27</u>*) : on décale vers la droite les éléments qui lui sont plus grands, puis on insère 27 à sa place et on obtient : (**5, 10, 27, 82**, **95,** *110, 70,31, 3*)

On veut insérer 110 dans la tranche (**5, 10, 27, 82**, **95,** *<u>110</u>)* : il est à sa place, on obtient : (**5, 10, 27, 82**, **95, 110**, *70,31, 3*)

On veut insérer 70 dans la tranche (**5, 10, 27, 82**, **95, 110**, *<u>70</u>*) : on décale vers la droite les éléments qui lui sont plus grands, puis on insère 70 à sa place et on obtient : (**5, 10, 27, 70, 82**, **95, 110**, *31, 3*)

On veut insérer 31 dans la tranche (**5, 10, 27, 70, 82**, **95, 110**, *<u>31</u>)* : on décale vers la droite les éléments qui lui sont plus grands, puis on insère 31 à sa place et on obtient : (**5, 10, 27, 31, 70, 82**, **95, 110**, *3*)

On veut insérer 3 dans la tranche : (**5, 10, 27, 31, 70, 82**, **95, 110**, *<u>3</u>*) : on décale vers la droite les éléments qui lui sont plus grands, puis on insère 3 à sa place et on obtient : (**3, 5, 10, 27, 31, 70, 82**, **95, 110**)

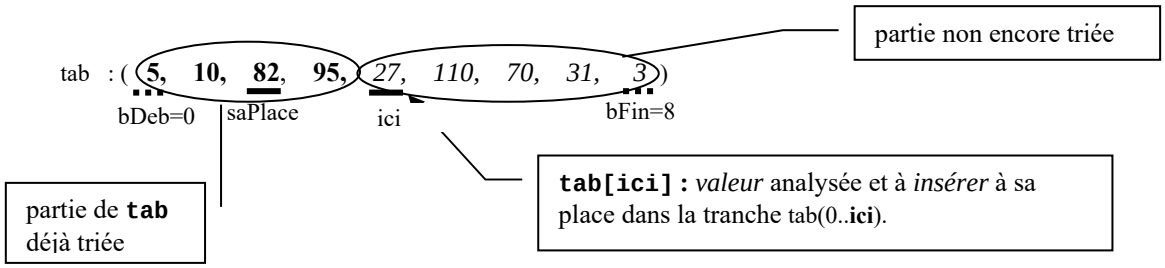
Stratégie de mise en œuvre de l'algorithme

On applique la méthodologie de décomposition du problème vue dans les tris précédents. On définit la situation ('='photo') du tableau en cours de tri (Partie 1), situation qui est amenée à être répétée (Partie 2).

Partie 1 : *Insérer tab[ici] à sa place dans la portion de tableau tab(bDeb..ici)*

On suppose tout d'abord que `tab` est déjà partiellement trié (par ordre croissant) et qu'il est dans l'état suivant :

- la partie inférieure (c'est à dire la tranche `tab (bDeb..ici-1)` entre les indices `bDeb` et `ici-1` est déjà triée.
- la partie supérieure (c'est à dire la tranche `tab (ici..bFin)` entre les indices `ici` et `bFin` n'est pas encore triée.



Le problème à résoudre dans cette Partie 1 est donc :

« insérer `tab[ici]` **à sa place dans la portion de tableau** `tab(bDeb..ici)` **»**

de sorte à étendre la portion triée du tableau à `tab(bDeb..ici)`.

Pour réaliser cette action, on peut imaginer 2 étapes :

A. tout d'abord de **déterminer la place** (que l'on nommera, par exemple, **`saPlace`**) dans la tranche **`tab(bDeb..ici),`** où devra se placer la valeur **`tab[ici]`** sélectionnée,

B. puis d'**insérer la valeur `tab[ici]`** à sa place, en décalant préalablement si nécessaire vers la droite tous les éléments de la tranche nécessaires.

Exemple : Pour un tableau en C++ de taille 9, où **`bDeb=0`** et **`bFin`**=9-1=8

> Dans la figure ci-dessus, l'indice **`ici`** = 4, et la valeur du tableau à cet indice est **`tab[ici]`** = 27.
>
> A. Déterminer la place : la valeur **`tab[ici]`** = 27 devra être insérée à l'indice **`saPlace`** = 2.
>
> B. Insertion à sa place : pour insérer **`tab[ici]`** à **`saPlace`**, les éléments du tableau situés à partir de l'indice **`saPlace`** = 2 seront tous décalés d'un cran vers la droite. Avant le décalage, il faut penser à sauvegarder la valeur contenue dans **`tab[ici].`**
>
> *fin exemple*

Amélioration de la stratégie

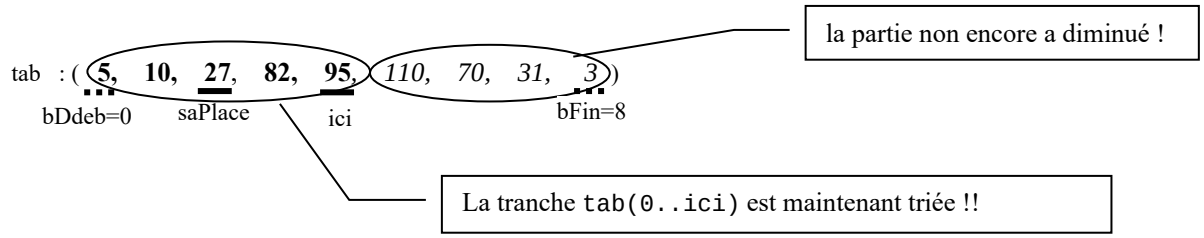
On peut optimiser cette première idée en redécoupant différemment les étapes A. et B. :

A. **`faireDeLaPlace`** dans la tranche **`tab(bDeb..ici),`** où devra se placer la valeur **`tab[ici]`** sélectionnée. En effet, puisque l'on parcourt le tableau, on peut chercher la position tout en ***décalant préalablement si nécessaire vers la droite l'élément courant évalué***.
La position cherchée sera nommée **`saPlace`**. Ici aussi, il ne faut pas oublier de sauvegarder préalablement la valeur contenue dans **`tab[ici].`**

B. puis **placer la valeur `tab[ici]`** à sa place.

Fin de l'étape 1

Une fois l'action « **insérer** `tab[ici]` **à sa place dans la portion de tableau** `tab(bDeb..ici)` » terminée, le tableau se trouve dans l'état suivant :



Les actions (A) et (B) citées ci-dessus a été concrétisées sous la forme d'**un** sous-programme dont voici la déclaration en C++ :

```
1  void insertion  (int tab [],
2                   unsigned int bDeb,
3                   unsigned int ici);
4  // étant donnée une portion de tableau tab(bDeb..ici),
5  // telle que tab(bDeb..ici -1) est triée par ordre croissant de valeurs,
6  // trouve l'indice saPlace du tableau tab compris entre deb et ici, auquel la
7  // valeur tab[ici] devra être insérée pour que la portion tab(bDeb..ici) du
8  // tableau soit encore triée par ordre croissant
```

Travail à faire

1. Écrire l'algorithme du sous-programme **`insertion`**, accompagné des spécifications internes succinctes nécessaires (nom, signification et type de chaque élément utilisé.

Partie 2 : *Trier tout le tableau*

Il s'agit d'écrire l'algorithme de tri par insertion en s'appuyant sur le précédent sous-programme.

Travail à faire

2. Écrire l'algorithme du sous-programme **`triParInsertion`**, accompagné de spécifications internes succinctes nécessaires (nom, signification et type de chaque élément utilisé).

3.- Tri par sélection de <u>place</u>

Vu en cours