Comparaison de deux tris (niveau 2)

6.1 Tri bulles

> Exercice n°37 (pages 89 et 271) de l'ouvrage *C++ par la pratique*, avec une légère différence dans la version du tri implémentée.
> (pages 91 et 271 dans la 2e édition).

Le tri bulles est sûrement le plus simple des tris (il est par contre peu efficace).

L'idée est de parcourir la liste autant fois que nécessaire en échangeant 2 à 2 les éléments consécutifs qui ne sont pas dans le bon ordre.
c'est-à-dire si `t[j-1] > t[j]`, alors on échange `t[j-1]` et `t[j]`.

L'algorithme est donc le suivant (indices de 1 à `taille`) :

```
Pour i de 1 à taille-1
  Pour j de taille à i+1 (descente)
     Si t[j-1] > t[j]
       échanger t[j-1] et t[j]
```

À faire

1. Dans le fichier `tri_bulles.cc`, implémentez l'algorithme ci-dessus (attention aux indices en C++ !) dans une fonction `tri_bulles` prenant comme arguments un tableau d'entiers (statique ou dynamique, au choix)

   Pour échanger deux variables, utiliser la fonction `swap` fournie par la bibliothèque `utility` (`#include <utility>`) : `swap(x, y);`.

2. Dans la fonction `main`, déclarez un tableau d'entiers (du même type que choisi ci-dessus) et remplissez-le des éléments

   ```
   3, 5, 12, -1, 215, -2, 17, 8, 3, 5, 13, 18, 23, 5, 4, 3, 2, 1
   ```

   (18 éléments).

3. Testez votre fonction en affichant le résultat sur le tableau précédent.

4. Effectuez éventuellement d'autres tests (tableau constant, tableau vide ou à 1 élément, tableau déjà trié, tableau trié dans le sens inverse, ...)

Exemple d'exécution

```
A trier  : 3 5 12 -1 215 -2 17 8 3 5 13 18 23 5 4 3 2 1
Résultat : -2 -1 1 2 3 3 3 4 5 5 5 8 12 13 17 18 23 215
```

Note : Le nom de ce tri vient du fait que les éléments à classer «remontent» à leur place un peu comme les bulles dans un liquide remontent vers la surface.
Ajouter un affichage du tableau à chaque itération pour voir cet effet.

Par ailleurs, on implémente souvent ce tri en remplaçant l'itération « Pour i de 1 à taille-1 » par une itération «Répéter .... tant qu'il y a eu au moins 1 échange ».

6.2 tri de Shell

> Exercice n°44 (pages 98 et 288) de l'ouvrage *C++ par la pratique*.
> (pages 100 et 288 dans la 2e édition).

On cherche ici à implémenter une méthode de tri assez efficace en pratique, surtout pour des tableaux de taille faible à moyenne.
Il s'agit du tri dit « de Shell » du nom de son inventeur.

Le tri de Shell est un tri par insertion à incrément décroissant : au lieu de comparer si un élément est plus petit que son prédecesseur immédiat, on le compare à son prédecesseur à distance *k* (que l'on fait décroître au cours du temps) ; autrement dit : au lieu d'insérer chaque élément à sa place dans la sous-liste triée de tous les éléments qui le précèdent, on l'insère dans une sous-liste d'éléments qui le précèdent mais distants d'un certain incrément *k*.

L'algorithme est le suivant (indices de 1 à taille) :

```
Pour k de taille/2 à 1, en le divisant par 2
    Pour i de k+1 à taille
        j <- i-k
        Tant que j > 0
            Si t[j] > t[j+k]
                échanger t[j] et t[j+k]
                j <- j-k
            Sinon
                j <- 0
```

Comme pour le tri bulles (partie 6.1), implémentez cet algorithme en C++ (attention aux indices) et testez le sur diverses données.

Attention ! `j` peut être négatif (par `j <- j-k`)...

Exemple d'exécution

L'exemple d'exécution est le même que pour le tri bulles **ci-dessus**.

6.3 lien ICC

Comparez expérimentalement la vitesse des deux programmes sur la même liste (de plusieurs dizaines d'éléments afin de pouvoir observer une différence).

Quelle est la complexité du tri bulles ?

On peut montrer que la complexité temporelle pire cas du tri de Shell telle que présenté ici est en O(n^2), mais d'autres stratégies restent possibles pour *k* et c'est encore une question ouverte (je crois) que de savoir quelle est la meilleure stratégie pour l'évolution de *k*. On sait par contre que quelque soit la stratégie pour l'évolution de *k*, la complexité temporelle pire cas est au moins en O(n (log(n) / log(log(n)))^2).
Pour plus de détails, voir **https://en.wikipedia.org/wiki/Shellsort#Gap_sequences** et **https://en.wikipedia.org/wiki/Shellsort#Computational_complexity**