

# SAÉ S1.02 – Comparaison d’approches algorithmiques

## Présentation du contexte

En informatique, une même tâche peut être réalisée par différents algorithmes. Dans ce cas, les algorithmes partagent les mêmes entrées et produisent les mêmes sorties mais se différencient par d’autres critères comme leur complexité temporelle (= vitesse d’exécution), leur complexité spatiale (= quantité de mémoire nécessaire) ou tout autre critère spécifique à la tâche réalisée.





Dans ce travail de SAÉ, la tâche à réaliser sera de trier les éléments d’un tableau par ordre croissant. Différents algorithmes seront à implémenter et à comparer entre eux. Ce travail de comparaison devra permettre de conclure sur quel algorithme choisir selon les spécificités des données d’entrée et des objectifs à atteindre.



**IL EST FORTEMENT CONSEILLÉ DE LIRE L’ENSEMBLE DE CE DOCUMENT AVANT DE COMMENCER À CODER !**




## 1 La base

Deux fichiers *fonctions.h* et *fonctions.cpp* vous sont fournis dans l’archive *VEtudiant.zip*. Ils contiennent un lot de fonctions **initTab\*** permettant chacune de créer un tableau de nombres entiers. Ces différentes fonctions offrent des caractéristiques variées pour les tableaux comme détaillé dans l’annexe 1. Une fonction **verifTri** est également disponible dans ces fichiers et devra être appelée après chaque tri afin de vous assurer que l’algorithme de tri est correct.




-  Pour commencer, créez une nouvelle solution Visual Studio de type application console et ajoutez-y les deux fichiers fournis.
-  Implémentez un [tri par sélection](#) selon les indications fournies sur Wikipédia dans une fonction **triSelection**.
  -  Ajoutez en paramètre de sortie de cette fonction un entier non signé qui, en sortie, sera égal au nombre de comparaisons d’éléments réalisés lors de l’exécution de l’algorithme de tri.
-  Testez cette fonction sur un tableau aléatoire et vérifiez que le tableau est correctement trié.


## 2 L’étude des performances du tri

### 2.1 Le tri par sélection

-  Dans votre programme, faites varier automatiquement la taille  $N$  du tableau aléatoire et reportez dans la console le nombre de comparaisons qui a été nécessaire pour le trier.
-  Recommencez la même opération pour les autres types d’initialisation de tableau (cf. Annexe 1)
-  Adaptez votre programme pour que le nombre de comparaisons soit reporté dans un tableau texte au format CSV dont les colonnes seraient :

N	Aleat Sélect.	PresqueTri Sélect.	PresqueTriDeb Sélect.	PresqueTriDebFin Sélect.	PresqueTriFin Sélect.
---	---------------	--------------------	-----------------------	--------------------------	-----------------------



-  Adaptez une dernière fois le programme pour enregistrer automatiquement ce tableau CSV dans un fichier.
-  A l’aide d’un logiciel comme Excel, tracez les courbes du nombre de comparaisons en fonction de la taille  $N$  du tableau.
  -  Tracez également la courbe  $N^2$  et  $N \cdot \ln(N)$  et déterminez ainsi visuellement l’allure de chaque courbe de nombres de comparaisons. L’allure s’entend à un facteur constant prêt.


 Toutes les fonctions d’initialisation de tableau ont la même signature. Elles peuvent donc être référencées dans un tableau de pointeurs vers fonction afin de rendre votre code plus succinct et automatique.

## 3 D’autres algorithmes de tri

### 3.1 Le tri à bulles


Sur la page Wikipédia du [tri à bulles](#), vous trouverez un pseudo code de cet algorithme.

-  Implémentez cet algorithme dans une nouvelle fonction de tri nommée **triBulles**.
-  Reprenez le code de mesure des performances du tri de la section 2.1 pour ajouter au tableau CSV précédent les nombres de comparaisons nécessaires du tri à bulles. Le tableau CSV comporte donc maintenant 11 colonnes : la colonne  $N$  et 5 colonnes par algorithme étudié pour les 5 types d’initialisation de tableau.



 Toutes les fonctions de tri sont supposées avoir la même signature. Elles peuvent donc être référencées dans un tableau de pointeurs vers fonction afin de rendre votre code plus succinct et automatique.

### 3.2 Le tri à bulles optimisé



Sur la page Wikipédia du [tri à bulles](#), vous trouverez une version optimisée de cet algorithme.

-  Ajoutez à votre étude cette version optimisée.


### 3.3 Autres algorithmes

-  Ajoutez à votre étude l’algorithme de [tri à peigne](#).
-  Ajoutez à votre étude l’algorithme de [tri rapide](#).




## 4 Préparez votre rapport

-  A l’aide des courbes que vous pouvez tracer à partir du tableau CSV, concluez sur quel type d’algorithme de tri est préférable en fonction des cas de tableaux à trier.
-  Y a-t-il un ou des algorithmes qui se démarquent des autres ?

## 5 Autres algorithmes

-  En navigant sur Wikipédia, vous trouverez d’autres [algorithmes de tri](#). Implémentez ceux que vous pouvez (dans le temps de la SAÉ)

## Remise de votre travail

-  Remettez dans l’espace du cours en ligne un fichier compressé contenant :
  -  Votre code source en ayant pris soin de supprimer l’inutile
  -  Un fichier PDF du rapport de conclusions sur les comparaisons d’algorithmes de tri. Vous prendrez le plus grand soin dans la rédaction de ce rapport et y présenterez, courbes pertinentes à l’appui, vos résultats.

# Annexes

## Annexe 1 Référence des fonctions fournies

### A1.1 `std::vector<int> initTabAleat(size_t N)`

Crée un tableau d'entiers dont tous les éléments sont choisis aléatoirement.

Un tel tableau peut par exemple être 30968 28073 31177 2882 6140 17999 13828 20039 2310 24865.

#### A1.1.1 Paramètres

Direction	Nom	Description
in	<i>N</i>	taille du tableau

#### A1.1.2 Renvoi

Le tableau initialisé

### A1.2 `std::vector<int> initTabPresqueTri(size_t N)`

Crée un tableau d'entiers presque triés. Chaque élément est quasiment à sa place définitive, échangé éventuellement d'une place.

Un tel tableau peut par exemple être 1 3 2 4 6 5 8 7 9 10.

#### A1.2.1 Paramètres

Direction	Nom	Description
in	<i>N</i>	taille du tableau

#### A1.2.2 Renvoi

Le tableau initialisé

### A1.3 `std::vector<int> initTabPresqueTriDeb(size_t N)`

Crée un tableau d'entiers presque triés. Seuls le premier et le deuxième élément sont échangés.

Un tel tableau peut par exemple être 2 1 3 4 5 6 7 8 9 10.

#### A1.3.1 Paramètres

Direction	Nom	Description
in	<i>N</i>	taille du tableau

#### A1.3.2 Renvoi

Le tableau initialisé

### A1.4 `std::vector<int> initTabPresqueTriDebFin(size_t N)`

Crée un tableau d'entiers presque triés. Seuls le premier et le dernier élément sont échangés.

Un tel tableau peut par exemple être 10 2 3 4 5 6 7 8 9 1.

**A1.4.1 Paramètres**

Direction	Nom	Description
in	<i>N</i>	taille du tableau

**A1.4.2 Renvoi**

Le tableau initialisé

**A1.5 `std::vector<int> initTabPresqueTriFin(size_t N)`**

Crée un tableau d'entiers presque triés. Seuls le dernier et l'avant dernier élément sont échangés.

Un tel tableau peut par exemple être 1 2 3 4 5 6 7 8 10 9.

**A1.5.1 Paramètres**

Direction	Nom	Description
in	<i>N</i>	taille du tableau

**A1.5.2 Renvoi**

Le tableau initialisé

**A1.6 `void verifTri(const std::vector<int>& tab, const std::string& algoName)`**

Vérifie qu'un tableau est correctement trié. Si le tableau est mal trié, un message d'erreur est affiché sur le flux d'erreur et le programme est terminé.

**A1.6.1 Paramètres**

Direction	Nom	Description
in	<i>tab</i>	Le tableau à vérifier
in	<i>algoName</i>	Le nom de l'algorithme de tri qui a été utilisé. Ce paramètre est optionnel.