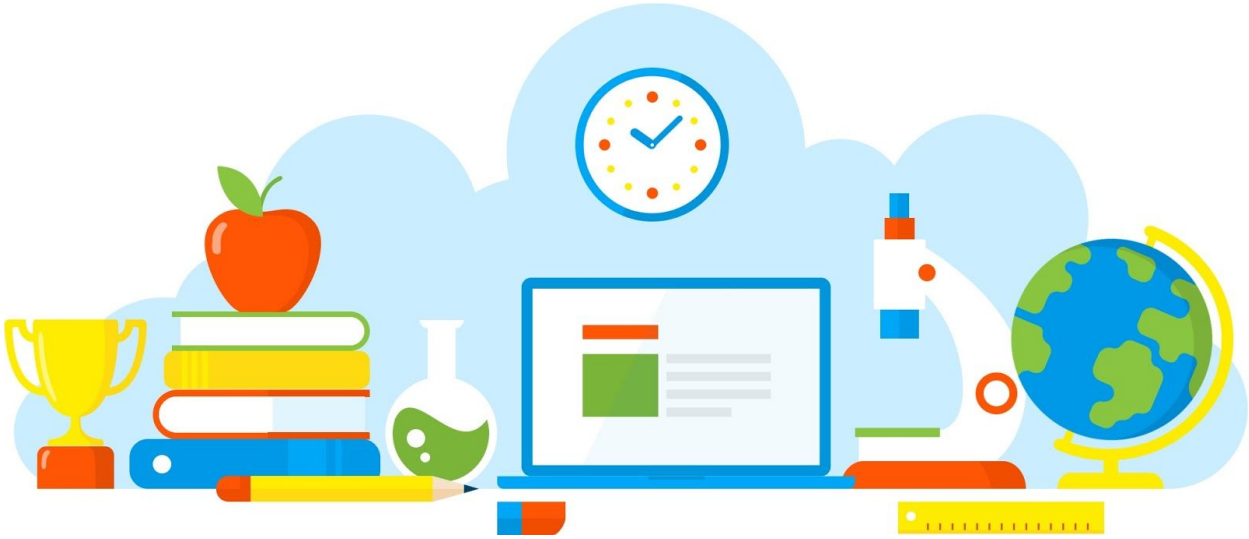


Rapport du projet “Algostat”

Présentation rapide du site et explications des algorithmes

Par : Martin Barreau, Corentin Briand et Alexandre Guillemot

Le : 30/01/2018

Introduction rapide sur le site :

Avant toute chose toutes les sources nous ayant permis de réaliser le site que ce soit sur la partie technique comme sur la partie théorique seront présentes à la fin de ce document.

1) Quelques détails importants à connaître avant utilisation du site.

Le site web permettant de tester les différents algorithmes de tri est disponible sur l'intra de **l'ETNA via la dépôt SVN** correspondant à notre projet, **ainsi qu'à l'adresse suivante : http://188.226.199.194/algostat/ .**

A noter qu'une base de données contenant l'historique de chacuns des tris effectués est en place et quelle n'est stockée que sur le serveur que nous avons mis en place en plus du dépôt SVN.

En effet le code utilisant cette base de données pointe directement vers le serveur, **il est donc recommandé d'avoir une connexion internet et de le tester via le lien ci-dessus si vous voulez utiliser le site sans avoir a configurer manuellement** un serveur pour qu'il puisse marcher avec une base de données en local.

Au niveau du code chaque tri a sa propre classe qui étends la classe Tri.

Cela permet de redéfinir la fonction TriTableau pour chacun des tri. A chaque soumission d'un formulaire, si la chaîne à trier est au bon format (uniquement composée de chiffres et de ",", ";", ".", "-" (seulement si "-" est devant les chiffres), alors on transforme la chaine en un tableau de valeurs.

Ensuite on lance le tri, on affiche les résultats et on sauvegarde tout ça dans la base de données.

Au niveau du calcul des itérations, ne sont comptés que les tours de boucles.

Les codes pour phpmyadmin sont user : root, mdp aucun.

2) Guide rapide d'utilisation :

Lorsque vous arrivez à la racine du site, vous tombez sur une page d'accueil qui contient les courbes représentant la complexité théorique moyenne d'exécutions des différents algorithme de tri que vous pourrez tester sur le site.

Pour tester un des différents algorithmes, un lien vous est proposé en dessous du message de bienvenue, sinon sur le menu en haut à droite vous avez un lien intitulé benchmark qui vous envoie sur la page de test.

Une fois sur cette page, choisissez un algorithme, puis lancez le tri avec des valeurs numériques, séparées par des virgules/point virgules. (Pour les nombres flottants mettre un . entre la partie décimale et la partie après la virgule. Pour les signes mettre un '-' si le nombre est négatif.

Pour voir les résultats des précédentes exécutions sur chacuns des algorithmes, vous pouvez cliquer sur la page analyse dans le menu.

Ce lien vous emmène vers une page qui vous présente un formulaire. Dans ce formulaire indiquez sur quel tri vous souhaitez voir les statistique lui correspondant.

Une fois cela fait vous aurez alors un graphique présentant le temps moyen d'exécution du tri en fonction du nombre de valeurs.

Présentation des tris que nous avons implémentés :

1) Le tri à bulles :

Explication du tri, partie théorique :

Le tri à bulles est un tri dont la stratégie est de parcourir un tableau de son premier élément au dernier élément non trié de celui-ci (donc la dernière case du tableau au premier tour), et en avançant d'une case à la fois de permuter l'élément de la case suivante avec celui sur la case actuelle si la valeur courante est supérieure à celle d'après.

Ainsi à la première occurrence on ramène la plus grande valeur présente dans le tableau à sa dernière case et on “marque” la dernière case non triée du tableau comme triée. On recommence les opérations jusqu’à ce qu’on ai trié n-1 premiers éléments ou n est le nombre d’éléments dans le tableau.

En pseudo code, l’algorithme implémenté ressemble à cela :

```
pour i allant de taille de T - 1 à 1
        pour j allant de 0 à i - 1
                si T[j+1] < T[j]
                échanger(T[j+1], T[j])

                fin si

        fin pour

fin pour

Avec T un tableau de valeurs, i un entier et j un entier également.
```

Ici quoi qu’il arrive on parcourra n-1 * n-1 fois le tableau, les itérations autres que les parcours de tableau étant insignifiantes comparés aux tours de boucles, on obtient donc une complexité $O(n^2)$.

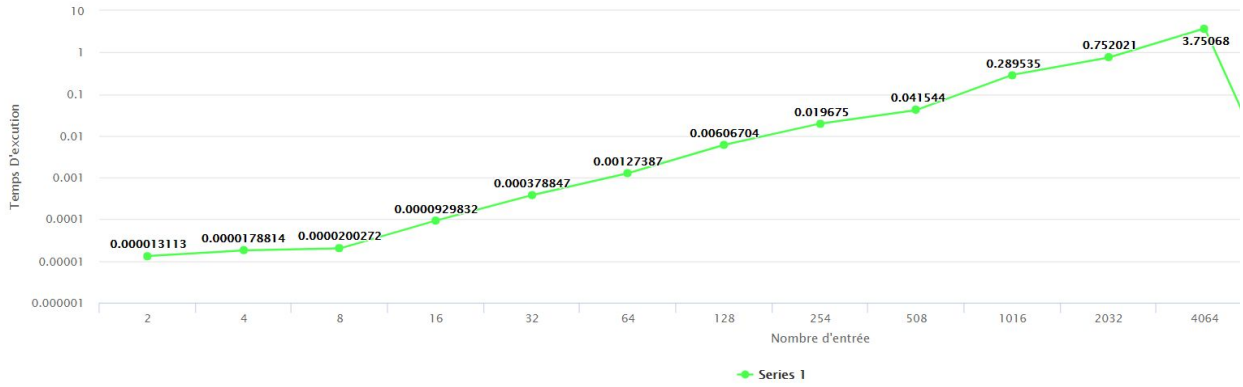
A noter, qu’étant donné que ce tri n’a aucune condition d’arrêt au cas ou on a trié le tableau avant le parcours des deux boucles, Il est donc théoriquement le plus lent de tous les algorithmes de tris que nous avons mis en place.

```php
public function triTableau()
{
  $timeD = microtime(true);
  $this->nbelem = count($this->tabNb);
  $arrayTab = $this->tabNb;
          $this->itNb = 0;
  $tab_tri = false;
  for ($i = $this->nbelem; $i > 1 && $tab_tri == false; $i--)
  {
    $tab_tri = true;
    for ($j = 0; $j < $i - 1; $j++)
    {
      if ($arrayTab[$j + 1] < $arrayTab[$j])
      {
        $tmp = $arrayTab[$j + 1];
        $arrayTab[$j + 1] = $arrayTab[$j];
        $arrayTab[$j] = $tmp;
        $tab_tri = false;
      }
              $this->itNb++;
    }
            $this->itNb++;
  }
  $this->tabTri = array_merge($arrayTab);
  $timeF = microtime(true);
  $this->timeExec = $timeF - $timeD;
}
```

Explication de notre implémentation du tri :

Pour cette implémentation, on a repris le principe de base du tri à bulles, mis à part qu'on a utilisé un système de booléen afin de permettre à l'algorithme de sortir plus vite au on détecte que le tri est fini. Cela permet d'améliorer sensiblement les performances du tri.

Sur nos tests effectués et recueillis dans la base de donnée il fallait environ 3.75 s pour un tri sur 4064 valeurs (suite aléatoire).



Ci-dessus : Courbe des temps d'exécutions (en us) en fonction du nombre d'itérations dans le contexte d'un tri à bulles.

2) Le tri par insertion :

Explication du tri "de base" :

Le tri par insertion consiste à prendre le parcourir le tableau du début + 1 à la fin et de décaler le i - ème élément à sa place vers si il le faut la partie triée du tableau, en décalant les autres éléments un à un jusqu'à ce qu'on obtienne un "trou" ou notre élément rentre (si il est supérieur aux éléments à sa droite et si il est inférieur aux éléments à sa gauche).

L'avantage est que si le tableau est presque trié ou même trié on aura une complexité proche de O(n) mais le problème est que dans le pire des cas il peut devenir très coûteux et tourner autour de $O(n^2)$.

Puisque le pseudo code vaut mieux que des mots, le voici juste ci dessous :

```
pour i de 1 à n - 1
        x ← T[i]
        j ← i
        tant que j > 0 et T[j - 1] > x
                T[j] ← T[j - 1]
                j ← j - 1
        fin tant que
```

```
        T[j] ← x
    fin pour
```

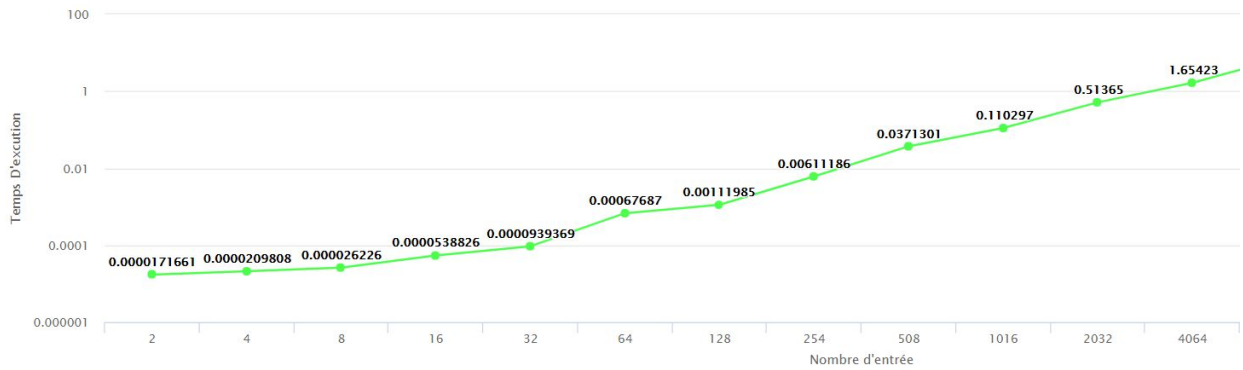
Notre calcul de complexité s'effectuant dans le pire des cas, on note donc que si le tableau est inversement trié on fait une première boucle que l'on exécute n - 1 fois et cette même boucle en lance une autre qui dure au maximum n fois soit une complexité dans le pire des cas de $O(n^2)$.

Partie technique :

```php
public function triTableau()
{
  $deb = microtime(true);
  $this->itNb = 0;
  $this->nbelem = count($this->tabNb);
  if (isset($this->tabNb[0]))
  {
    $i = 0;
    $tmp = 0;
    $tmpTab = $this->getTabNb();
    $lenght = count($tmpTab);
    for ($i = 0; $i < $lenght; $i++)
    {
      $tmp = $tmpTab[$i];
      $j = $i;
      while ($j > 0 && $tmpTab[$j - 1] > $tmp)
      {
        $tmpTab[$j] = $tmpTab[$j - 1];
        $j--;
        $this->itNb++;
      }
      $tmpTab[$j] = $tmp;
      $this->itNb++;
    }
    $this->setTabTri($tmpTab);
  }
  $this->timeExec = microtime(true) - $deb;
}
```

Dans notre implémentation, nous avons repris l'algorithme de base et la aussi nous avons modifié quelques trucs pour qu'il soit fonctionnel avec php. Nous avons utilisé un tableau intermédiaire pour ne pas modifier le tableau original, Ainsi que des quelques lignes en plus pour réaliser les échanges de place des variables et pour l'initialisation des variables avant le tri.

On notera aussi que pour 4064 valeurs, choisies aléatoirement, nous obtenons, en moyenne, un score d'environ 1.65 us soit un temps bien inférieur au tri à bulles alors que la complexité est différente. Ce qui montre que ce tri est en moyenne plus rapide et efficace qu'un tri à bulles.



Ci-dessus : Courbe des temps d'exécutions (en us) en fonction du nombre d'itérations dans le contexte d'un tri par insertion.

3) Le tri rapide :

Partie technique :

Le tri rapide est le tri le plus efficace en terme de complexité dans le pire des cas que vous trouverez dans notre projet. Il n'était pas demandé mais il nous tenait à coeur d'implémenter un tri plus performant que les autres et que nous n'avions pas développé par le passé. Certains de nos membres ayant déjà implémenté un tri fusion, nous avons donc choisi d'implémenter ce tri.

Il fonctionne avec une stratégie de pivot, dans notre implémentation, le premier élément du tableau passé en argument est choisis comme le pivot.

Ensuite on place tous les éléments du tableau initial et qui sont inférieur à ceux du pivot dans un tableau nommé left et ceux qui y sont supérieurs dans un tableau right.

On fait cette lance cette fonction récursivement jusqu'à ce que qu'on ai des tableaux de taille 1, ce qui fait qu'en fusionnant l'ensemble des tableaux triés de taille un que l'on obtient, on a donc un tableau trié.

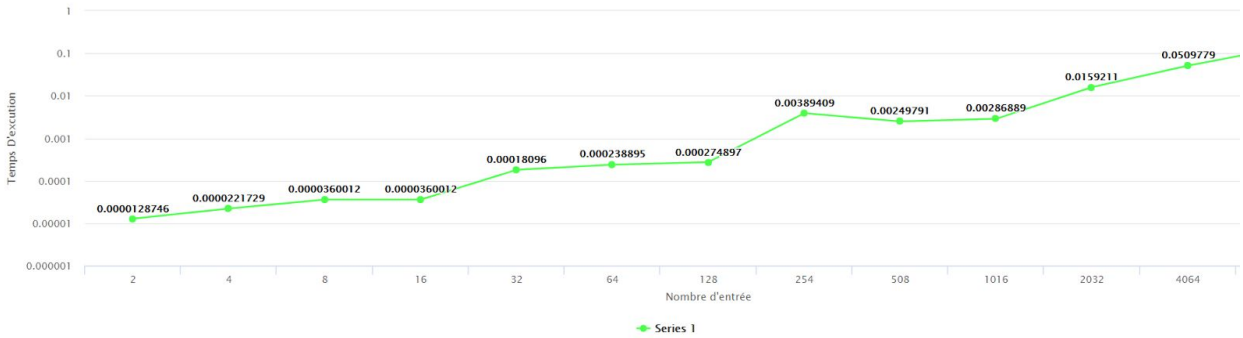
En terme de complexité, cet algorithme offre une complexité moyenne O(n log n)

```php
public function triTableau()
{
          $this->itNb = 0;
  $this->nbelem = count($this->tabNb);
  $timeD = microtime(true);
  $arrayTab = $this->tabNb;
  $this->tabTri = array_merge($this->quickSort($arrayTab));
  $timeF = microtime(true);
  $this->timeExec = $timeF -$timeD;
}


public function quickSort($array)
{

$length = count($array);
if($length <= 1){
  return $array;
}
else{
  $pivot = $array[0];
  $left = $right = array();
  for($i = 1; $i < count($array); $i++)
  {
    if($array[$i] < $pivot){
      $left[] = $array[$i];
    }
    else{
      $right[] = $array[$i];
    }
            $this->itNb++;
  }
  return array_merge($this->quickSort($left), array($pivot), $this->quickSort($right));
}
}
```

ci-dessus notre implémentation du tri rapide (quick sort) en langage php.

Cependant dans le pire des cas, on obtient une complexité de l'ordre $O(n^2)$.

Par rapport aux autres algorithmes, le temps d'exécution du tri rapide est largement inférieur : pour 4064 valeurs aléatoires, ils nous à fallu 0.05 us en moyenne pour les trier. Soit un temps presque 70 fois inférieur au temps obtenu pour le tri à bulles.



Ci-dessus : Courbe des temps d'exécutions (en us) en fonction du nombre d'itérations dans le contexte d'un tri rapide.

4) Le tri par sélection :

Partie théorique :

Le tri par sélection est un tri consistant à trouver l'élément du tableau ayant le plus petite valeurs non triée, pour la mettre à la place de la première valeur non triée dans le tableau, en terme de position.

Par exemple lorsque l'on commence le tri on cherche le plus petit élément de tout le tableau et on le met en T[0]. On "marque donc T[0] comme trié et on cherche donc le second élément le plus petit pour le mettre en T[1]. Et ainsi de suite jusqu'à ce qu'on ait un tableau trié. On sait quand le tableau est trié grâce à la valeur du marqueur.

En effet quand celui-ci vaut la taille N du tableau - 1 on sait qu'on est arrivé à la dernière case du tableau. Il est donc trié.

Ci-dessous : le pseudo-code de l'algo

```
pour i de 1 à n - 1
        min ← i
```

```
            pour j de i + 1 à n
                    si t[j] < t[min], alors min ← j fin si
             fin pour
            si min ≠ i, alors échanger t[i] et t[min] fin si
    fin pour
```

On voit donc qu'on a encore deux boucles, cependant la particularité de ce tri est qu'il a une complexité qui ne varie que très peu selon les cas d'exécutions.

Un exemple de pire des cas est celui ou on à 1, 2, 3, … , NBMAX, 0 comme suite de valeurs à trier car on va changer à chaque fois la valeurs de min ce qui va ralentir sensiblement l'exécution.

Pour en revenir à notre complexité, on fait N tours de boucle sur la première et au maximum N^2 tours pour la seconde ce qui nous donne une complexité $O(n^2)$.

Partie technique :

```php
public function triTableau()
{
  $timeD = microtime(true);
  $this->itNb = 0;
  $array  = $this->tabNb;
  $this->nbelem = count($array);
  for($i = 0; $i < $this->nbelem - 1; $i++)
  {
    $min = $i;
    $tmp = $array[$i];
    for($j = $i + 1; $j < $this->nbelem; $j++)
    {
      if ($array[$j] < $array[$min])
      {
        $min = $j;
        $tmp = $array[$i];
      }
      $this->itNb++;
    }
    if ($min != $i)
    {
      $array[$i] = $array[$min];
      $array[$min] = $tmp;
    }
    $this->itNb++;
  }
  $this->tabTri = array_merge($array);
  $timeF = microtime(true);
  $this->timeExec = $timeF - $timeD;
}
```

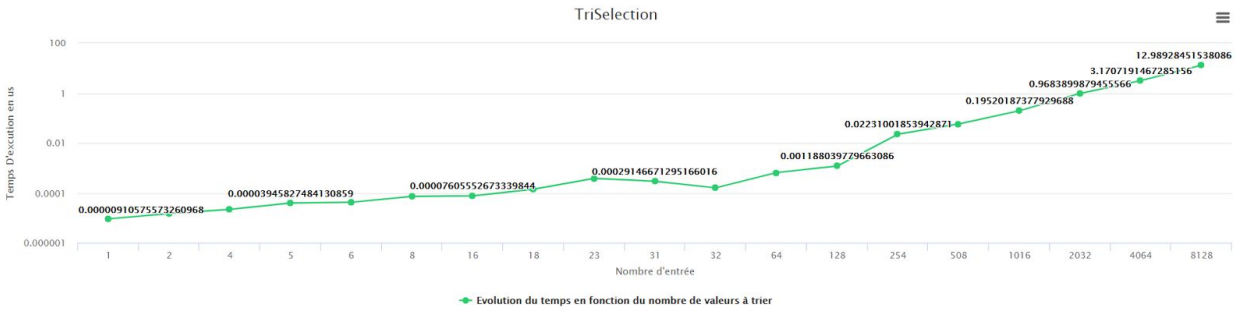
Pour réaliser le tri, nous avons repris l'algorithme de base tel qu'expliqué ci dessus. Et nous l'avons adapté pour qu'il puisse tourner sous php.

On utilise juste un tableau intermédiaire pour pouvoir trier le tableau original sans modifier la variable tabNb qui est vouée à conserver le tableau original que l'on a passé dans le formulaire.

Pour le reste on reprend la même base que la partie théorique avec les deux boucles : la première qui sert à avancer à chaque élément que l'on a trié et l'autre qui prend la position du plus petit élément non trié entre l'élément de $array[$i] et la fin du tableau $array. Une fois cette boucle effectuée on échange les positions entre la valeur minimum et la valeur de $array[$i] si $i et $min sont différents.

On répète donc les opérations jusqu'à ce qu'on ai fini le tri, soit que $i vaille la taille du tableau $array - 1.

Concernant les temps d'exécutions pour les 4064 valeurs aléatoires obtenues par notre execution de notre algorithme en moyenne on obtient : 3.17072 us ce qui est sensiblement mieux que le tri à bulles mais qui est quand même inférieur au tri par insertion.



Ci-dessus : Courbe des temps d'exécutions (en us) en fonction du nombre d'itérations dans le contexte d'un tri par sélection.

Sources :

1) Sources concernant la partie théorique :

Tous les membres de notre groupes ayant eu un passé universitaire (DUT / Licence) Nous avons tous eu des cours sur ces algorithmes par le passé. Ils font donc partie des sources,notamment pour les **calculs de complexité** ,les universités concernées sont :

- ➔ Versailles Saint Quentin en Yvelines (78), département Informatique de l'ufr des sciences.
- ➔ Reims Champagnes Ardennes (51) département informatique de l'IUT de Reims - Châlons - Charleville
- ➔ Université de Paris - Est - Créteil, (77) département informatique de l'IUT de Sénart - Fontainebleau

➔ Pseudo code des algorithmes :

- ◆ Tri à bulles : https://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles
- ◆ Tri par insertion : https://fr.wikipedia.org/wiki/Tri_par_insertion
- ◆ Tri rapide : https://fr.wikipedia.org/wiki/Tri_rapide
- ◆ Tri par sélection : https://fr.wikipedia.org/wiki/Tri_par_s%C3%A9lection
- ◆ Autre source : *The Art of Computer Programming de Donald Knuth*

2) Sources concernant la partie développement :

➔ PHP
➔ MySQL
➔ Javascript
➔ Apache
➔ Materialize
➔ PHPMyAdmin
➔ https://highcharts.com