

**REPUBLIQUE TUNISIENNE**

\*\*\*\*\*

**Ministère De L'enseignement  
Supérieur Et De La Recherche  
Scientifique**



المعهد العالي للإعلامية وتقنيات  
الاتصال بحمام سوسة

**UNIVERSITE DE SOUSSE**

\*\*\*\*\*

**Institut Supérieur d'Informatique  
Et des Techniques de  
Communication Hammam Sousse**

## **Projet : Algorithmique Avancée**

Réalisé par

**Ikram BEN SELMA**

**Projet Algorithmique avancée**

**Thème : Analyse & complexité des algorithmes**

Enseignant :

**Mr.Lotfi Ben Romdhane**

Classe :

**2DNI2**

Année Universitaire : 2020- 2021

---

# I. Partie théorique :

## Introduction :

- Dans ce projet, nous allons examiner un aspect extrêmement important d'un algorithme, à savoir sa *performance*.
- Il est important pour un informaticien de choisir l'algorithme le plus rapide. Il y a là un problème difficile : la vitesse d'exécution d'un algorithme dépend de l'ordinateur (du processeur, du type et de la quantité de mémoire, de l'organisation et la taille du cache et éventuellement de la rapidité du disque).
- Comment peut-on comparer des algorithmes du point de vue de la performance si l'on ne connaît pas l'ordinateur sur lequel va tourner l'algorithme ?
- De plus, le temps d'exécution d'un algorithme dépend bien sûr de la taille du problème à résoudre.
- Pour comparer deux algorithmes du point de vue de la performance, il faudrait alors donner la taille du problème à résoudre. Le résultat de cette comparaison peut varier selon la taille du problème.

## Un algorithme ?

C'est un ensemble d'instructions permettant de transformer un ensemble de données en un ensemble de résultats et ce, en un nombre fini étapes.

- Pour atteindre cet objectif, un algorithme utilise deux ressources d'une machine : **le temps** et l'espace **mémoire**.

## La complexité temporelle d'un algorithme ?

Est le temps mis par ce dernier pour transformer les données du problème considéré en un ensemble de résultats.

## Complexité asymptotique ?

- ✚ De façon générale : La complexité d'un algorithme est une mesure de sa performance asymptotique dans le pire des cas.
  - asymptotique ?
  - on s'intéresse à des données très grandes parce que les petites valeurs ne sont pas assez informatives.

- 
- **Pire des cas ?**
  - on s'intéresse à la performance de l'algorithme dans les situations où le problème prend le plus de temps à résoudre parce qu'on veut être sûr que l'algorithme ne prendra jamais plus de temps que ce qu'on a estimé.

### *On va s'intéresser des algorithmes suivants :*

- *Tri Fusion*

Il s'agit de décomposer une liste en deux sous-listes chacune deux fois plus petites, de les trier séparément, puis de fusionner les résultats en une liste triée.

- *Tri rapide*

Le tri rapide ou tri pivot est un algorithme de tri fondé sur la méthode de conception divisé pour régner.

Principe de la méthode

Choisir un élément du tableau appelé pivot,

Ordonner les éléments du tableau par rapport au pivot

Appeler récursivement le tri sur les parties du tableau à gauche et à droite du pivot.

- *Tri bulle*

Principe : Sélectionner le minimum du tableau en parcourant le tableau de la fin au début et en échangeant tout couple d'éléments consécutifs non ordonnés.

- *Tri insertion*

On suppose que les  $i - 1$  premiers éléments de la liste sont déjà classés. On insère à sa place le  $i$ -ième élément parmi les  $i - 1$  premiers ; de la sorte, les  $i$  premiers éléments sont classés. On itère jusqu'à insérer le  $n$ -ième.

- *Tri Fusion*

Diviser : diviser la liste de  $n$  éléments à trier en deux sous-listes de  $n/2$  éléments

Régner : trier les deux sous-listes récursivement à l'aide du tri par fusion

Combiner : fusionner les deux sous-listes triées pour produire la réponse triée

## I. Partie pratique :

On va tester les algorithmes de tri ci-dessus sur de grands tableaux, on va découvrir que certains étaient plus rapides que d'autres, plus efficaces. C'est ce que l'on appelle l'efficacité d'un algorithme. Pour la mesurer, la quantifier, on s'intéresse davantage à son contraire, la complexité.

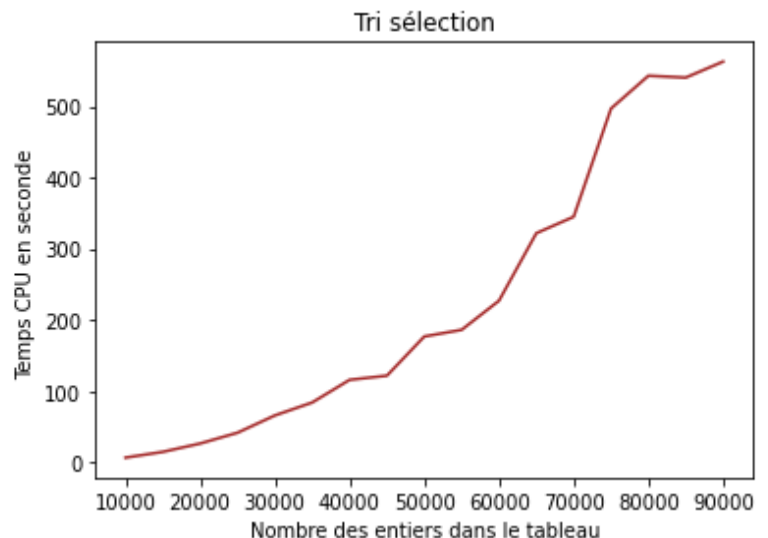
En effet, pour mesurer la complexité d'un algorithme il «suffit» de mesurer la quantité de ressources qu'il exige : mémoire ou temps-processeur (c'est ce qu'on va mesurer).

### 1. tri sélection :

#### ✓ Tableau & graph & complexité asymptotique :

Complexité asymptotique théorique :  
 $O(n^2)$

<i>N</i>	<i>Temps CPU (seconde)</i>
10000	6.382199
15000	14.48654
20000	26.24159
25000	41.41061
30000	65.68911
35000	83.89686
40000	115.70967
45000	121.56293
50000	176.65348
55000	185.99470
60000	226.90789
65000	321.81685
70000	344.94016
75000	497.02503
80000	543.08678
85000	465.50561
90000	562.96635

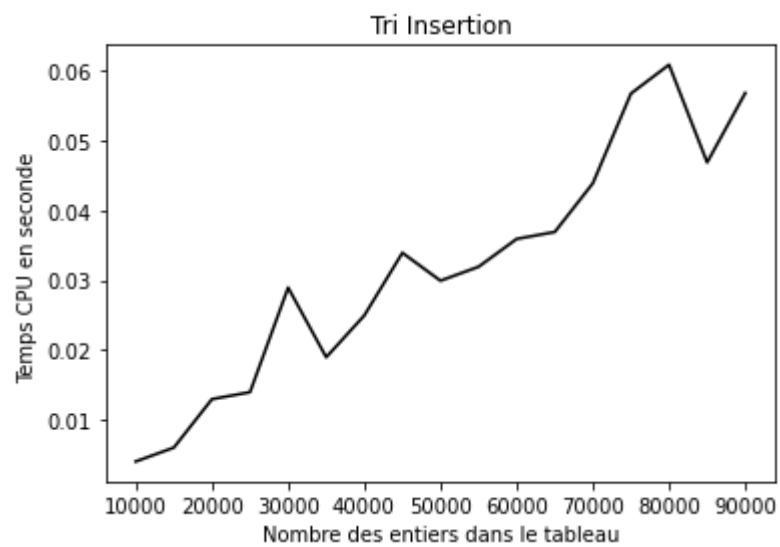


## 2. Tri par insertion

✓ Le tableau & graph & complexité asymptotique

Complexité asymptotique théorique :  
 $O(n^2)$

<i>N</i>	<i>Temps CPU (seconde)</i>
10000	0.003988
15000	0.00598
20000	0.01293
25000	0.01393
30000	0.02892
35000	0.01894
40000	0.02493
45000	0.03391
50000	0.02991
55000	0.03191
60000	0.03590
65000	0.03690
70000	0.04388
75000	0.05675
80000	0.06088
85000	0.04687
90000	0.05682

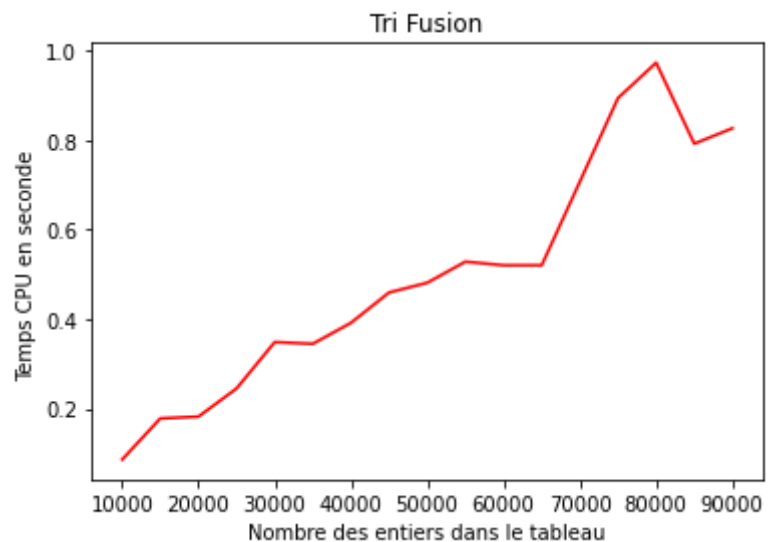


### 3. Tri par fusion

✓ Le tableau & graph & complexité asymptotique

Complexité asymptotique théorique :  
 $O(n \cdot \log(n))$

<i>N</i>	<i>Temps CPU (seconde)</i>
10000	0.086766
15000	0.17852
20000	0.18251
25000	0.24540
30000	0.34896
35000	0.34511
40000	0.39196
45000	0.45976
50000	0.48140
55000	0.52849
60000	0.52048
65000	0.52048
70000	0.70708
75000	0.89370
80000	0.97271
85000	0.79187
90000	0.82634

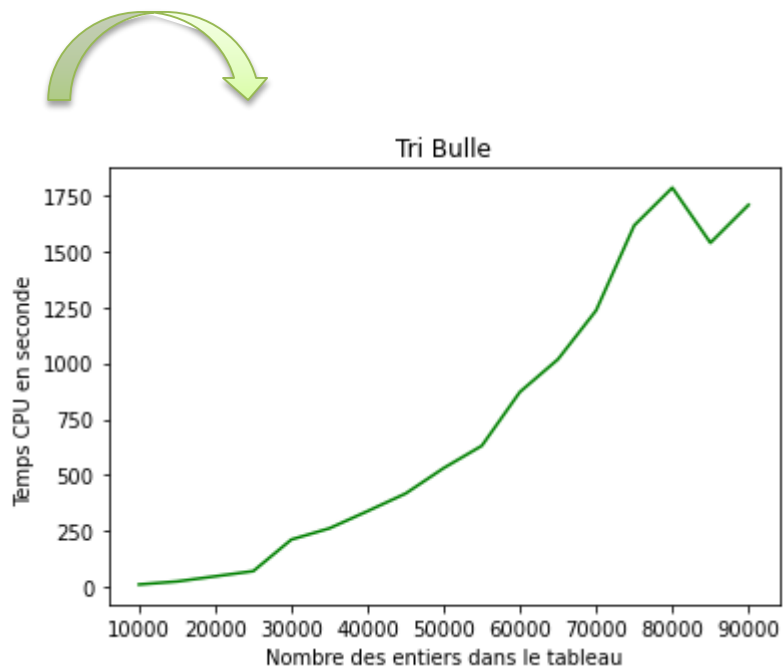


#### 4. Tri à bulle

✓ Le tableau & graph & complexité asymptotique

Complexité asymptotique théorique :  
 $O(n(n-1)/2)$

<i>N</i>	<i>Temps CPU (seconde)</i>
10000	11.423344
15000	24.92694
20000	47.91472
25000	71.10167
30000	212.2228
35000	262.3620
40000	339.0891
45000	417.8861
50000	531.6373
55000	631.0671
60000	871.5127
65000	1017.3513
70000	1235.32282
75000	1615.11323
80000	1783.43528
85000	1537.38083
90000	1706.58093



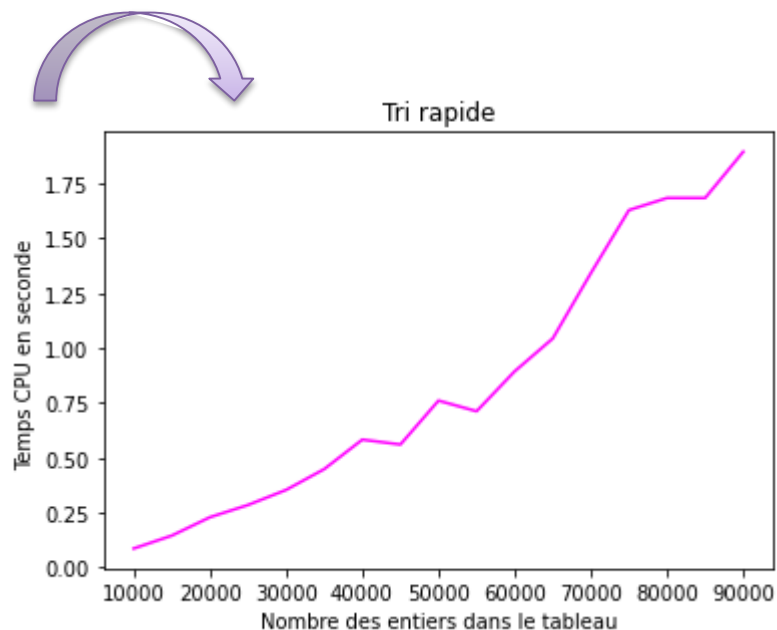
## 5. Tri rapide

✓ Le tableau & graph & complexité asymptotique

Complexité asymptotique théorique :

$$O(n \cdot \log(n))$$

<i>N</i>	<i>Temps CPU (seconde)</i>
10000	0.085798
15000	0.14463
20000	0.22838
25000	0.28423
30000	0.35306
35000	0.44791
40000	0.58229
45000	0.56004
50000	0.76012
55000	0.71169
60000	0.89371
65000	1.043584
70000	1.34158
75000	1.62879
80000	1.68415
85000	1.62565
90000	1.89504



## 6. Interprétation :




**Avant même de comparer les performances de nos algorithmes**, on peut remarquer une certaine irrégularité : des pics et des creux apparaissent et semblent (en général) avoir lieu pour tous les algorithmes.

Ces pics et ces creux correspondent aux pires et aux meilleurs cas possibles. En effet, si notre programme de génération aléatoire de tableaux vient à créer un tableau quasi-trié, alors le temps mis par certains algorithmes chutera drastiquement. Inversement, certaines configurations peuvent être



---

réellement problématiques à certains algorithmes : prenez un tableau quasi-trié mais dans le mauvais ordre ! Ce genre de tableau est catastrophique pour un algorithme de tri à bulles qui devra effectuer un nombre de passages excessif

 **Selon nos résultats**, on peut voir qu'en moyenne, l'algorithme le plus efficace est le tri rapide, puis vient le tri fusion et le tri insertion.

Le tri fusion reste, en moyenne, en dessous de 0,48 seconde, et pour le tri insertion, en moyenne il reste au-dessous de 0.05 seconde pour un tableau à 90 000 éléments sur mon ordinateur, ce qui est très loin des performances des trois autres algorithmes.

Le tri rapide est capable de trier rapidement des grands tableaux (comme son nom l'indique).

Pour les trois plus lents, on remarque que le «pire» d'entre eux est, en moyenne, le tri à bulles, puis viennent le tri par sélection (avec des écarts de performance toutefois très nets entre eux trois)