Université des Sciences et de la Technologie Houari Boumediene


Conception et Complexité des Algorithmes

---

# Rapport de Travaux Pratiques N°5
# Projet : Algorithmes de Tri.

---

*Binôme*
DJEBRI Mounira
RAMOUL Samy Rayan

13 décembre 2018

**Matériel**

Tout d'abord il est important de noter que les résultats obtenus dépendent de la machine utilisée, ici nous travaillons sur une machine ayant les caractéristiques suivantes :
Intel Core i7-8550U de 8e génération (8 Mo de mémoire cache, jusqu'à 4 GHz) avec 16 Go de mémoire DDR4. OS : Windows 10.

**Pile sous Windows**

Pour les besoins du TP nous avons dû augmenter la taille de la pile attribuée par défaut par Windows, pour cela nous avons suivi les étapes suivantes : Dans CodeBlocks : aller dans les paramètres - $compilateur - Accéder à l'onglet Linker Settings et ajouter la commande" - Wl, -stack, taille"$

# 1 Les algorithmes de Tri

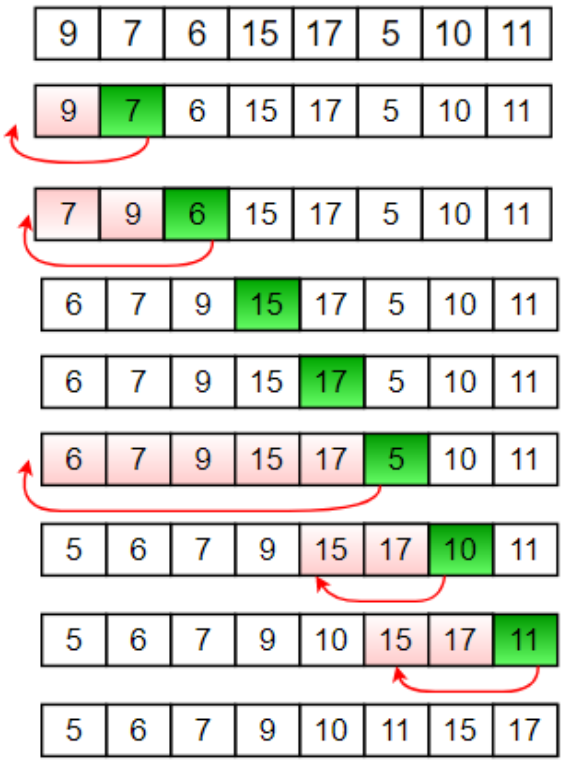
En informatique, un algorithme de tri est un algorithme qui place les éléments d'une liste dans un certain ordre. Le tri est extrêmement important en informatique pour la même raison qu'il l'est au quotidien. Il est plus facile et plus rapide de rechercher des éléments dans une liste triée que dans une liste non triée. Trier une liste d'éléments par ordre croissant ou décroissant peut aider un humain ou un ordinateur à trouver rapidement les éléments dans cette dernière. La plupart des données que nous traitons, qu'il s'agisse de chiffres ou de texte, impliquent la saisie et la récupération d'informations pertinentes de manière efficace. Le tri devient crucial dans de tels cas. Il existe plusieurs algorithmes de tri pouvant être utilisés dans un programme pour trier un tableau. Lors de la conception ou du choix d'un algorithme de tri, l'un des objectifs est de minimiser la quantité de travail nécessaire pour trier la liste des éléments. Le coût du tri peut être mesuré par le nombre d'éléments de tableau devant être comparés les uns aux autres ou par le nombre de fois où deux éléments de tableau doivent être permutés. Dans certains cas, les comparaisons coûtent plus cher que les échanges alors que dans d'autres cas, les échanges peuvent être plus coûteux que les comparaisons. Dans ce projet nous étudierons 5 méthodes de tri (Tri par Insertion, Tri à Bulles, Tri Fusion, Tri Rapide et enfin le Tri par Tas), que nous comparerons afin de voir lequel choisir selon nos besoins.

# 2 PartieI :

# 3 Tri par Insertion

## 1 Description de la méthode de tri par insertion :

Le tri par sélection est un algorithme de tri simple qui consiste à diviser une liste en deux parties, la partie triée à l'extrémité gauche et la partie non triée à l'extrémité droite. Initialement, la partie triée est vide et la partie non triée est la liste complète. Le plus petit élément est sélectionné dans le tableau non trié et échangé avec l'élément le plus à gauche, et cet élément devient une partie du tableau trié. Ce processus continue de déplacer les limites de tableau non triées d'un élément vers la droite.




## 2 Développement de l'algorithme.

```
Algorithme Tri Insertion ;
Entrée: tab[1..n] d'entier;
Sortie: tab trié;
VAR i,j,v,stock :entier;

DEBUT
    i=0;
    int v:=0;
    int j:=0;
    int stock:=0;
   Pour (i=1 à n)
    Faire

        SI(tab[i]<tab[i-1]) Alors

            j:=i-1;
            stock:=tab[i];
            Tantque(tab[j]>stock && j>=0)
            Faire
                tab[j+1]:=tab[j];
                j--;
            Fait
            tab[j+1]:=stock;

        Fsi
    Fait
FIN ;
```

## 3 Calcul de Complexité :

### 3.1 Calcule des complexités temporelles en notation asymptotique de Landau O (Grand O :

1. Au Pire Cas : Le pire cas est quand tableau est trié dans l'ordre inverse.
   Il y a au pire i comparaisons pour chaque i variant de 2 à n, (i étant la longueur de la partie déjà rangée) La complexité au pire cas est donc égale à la somme i = 2+3+...+N.

   CT(n) = 2 + 3 + 4 +...+ N = $\frac{N(N+1)}{2} - 1$
   CT(n) = $\frac{N^2+N-2}{2}$
   La complexité au pire cas en nombre de comparaison est de de l'ordre de $N^2$.
   Ce qui donne en notation Landau $O(N^2)$.

2. Au Meilleur Cas :
   Le meilleur des cas est lorsque le tableau est déjà trié. Pour l'itération, le premier élément restant de l'entrée est comparé à l'élément le plus à droite de la sous-liste triée du tableau uniquement. Dans ce cas l'algorithme ne rentre jamais dans la boucle tant que, il y a donc N-1 comparaisons et au pire N affectations. CT(n) = N + (N+1) = 2N + 1

   La complexité au meilleur cas en nombre de comparaison est de l'ordre de N,
   Ce qui donne en notation landau O(N).

3. La complexité moyenne :
   Pour i=2 à N, le nombre moyen total de comparaison est égal à : CT(n) = $\frac{1}{2}\sum_{i=2}^{n} i + 1$
   CT(n) = $\frac{N^2+3N-4}{4}$

   La complexité moyenne est de l'ordre de $N^2$.
   Ce qui donne en notation landau O($N^2$).

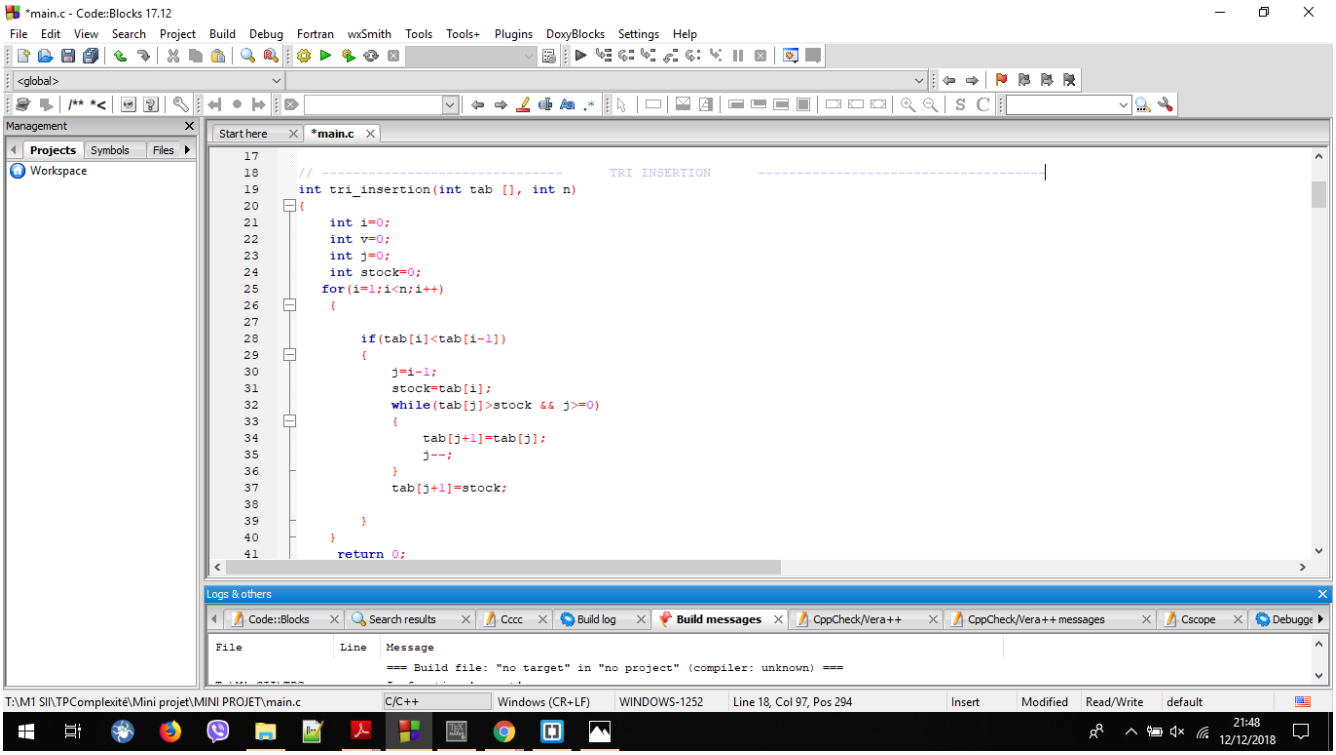
### 3.2 Calcule des complexités spatiales en notation exacte et en notation asymptotique de Landau O (Grand O)

L'algorithme de tri par insertion n'utilise que le tableau à trier de taille N ainsi que 4 variables de type entier. Sur CodeBlocks un entier est sur 8octets. Nous obtenons la complexité spatiale suivante :
CS(N) = 8(N + 4) octets
Ainsi CS(n) = O(N).

## 4 Développement de programme correspondant avec le langage C.

```
int tri_insertion(int tab [], int n)
{
    int i=0;
    int v=0;
    int j=0;
    int stock=0;
   for(i=1;i<n;i++)
    {

        if(tab[i]<tab[i-1])
        {
            j=i-1;
            stock=tab[i];
```

```
            while(tab[j]>stock && j>=0)
            {
                tab[j+1]=tab[j];
                j--;
            }
            tab[j+1]=stock;

        }
    }
     return 0;
}
```

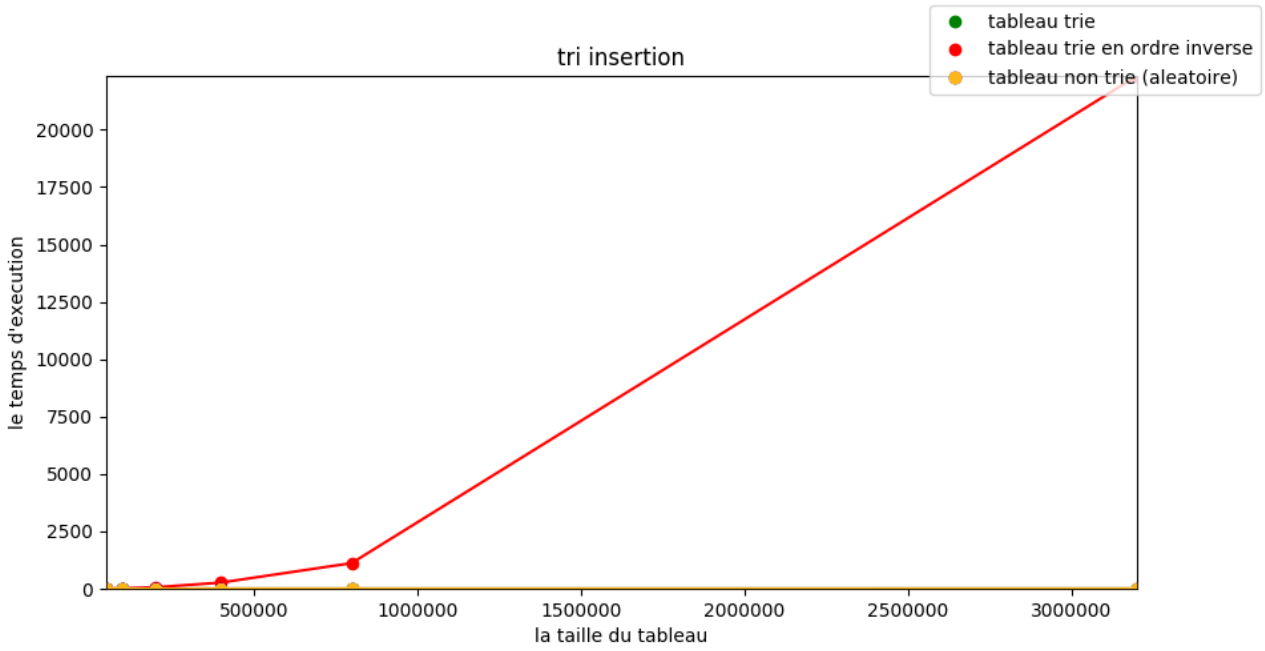



## 5 Mesure des temps d'exécution.

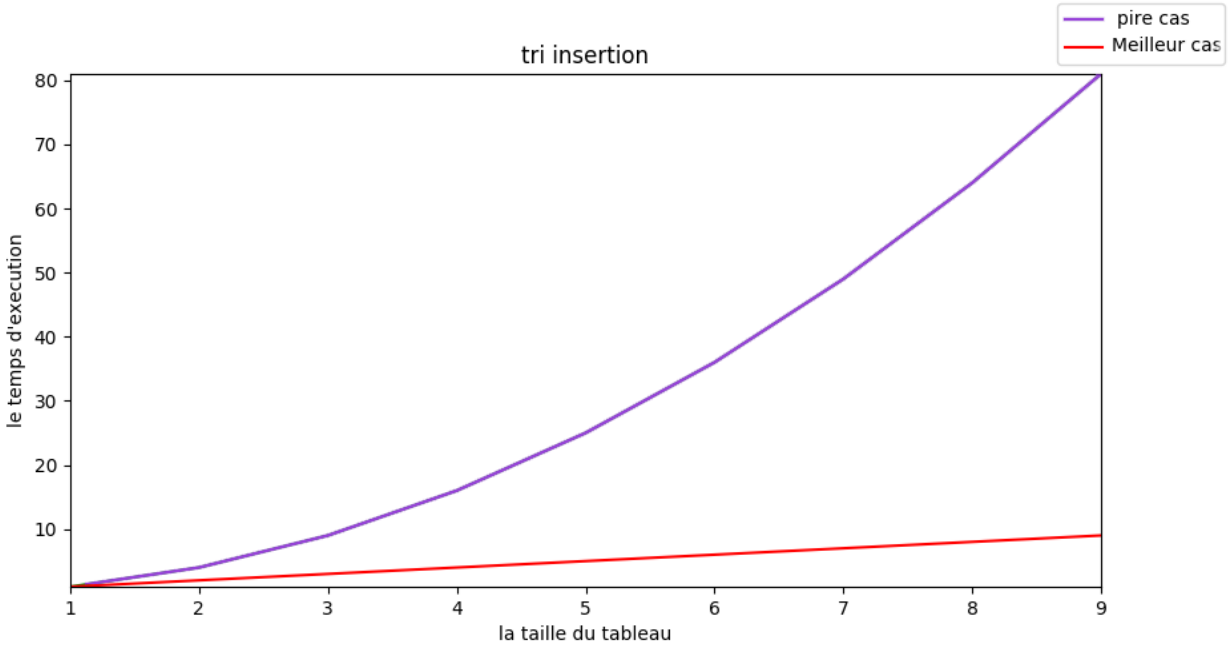
| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Temps (bon ordre) : | 0.000115 | 0.000190 | 0.000896 | 0.001033 | 0.003880 | 0.006765 |
| Temps (Aléatoire) : | 0.000701 | 0.003007 | 0.006067 | 0.016099 | 0.033002 | 0.086016 |
| Temps (inverse) : | 3.943 | 16.847 | 62.723 | 267.101 | 1099.756 | 4700.372 |

| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
|---|---|---|---|---|---|---|---|
| Temps (bon ordre) : | 0.0160000 | 0.032600 | 0.052272 | 0.167922 | 0.400770 | 0.74150 | 1.48089 |
| Temps (Aléatoire) : | 0.179991 | 0.358023 | 0.746764 | 1.65260 | 3.794566 | 7.589934 | 16.182 |
| Temps (inverse) : | 19904.098 | 86700.329 | 346755.319 | 1508777.938 | 6789653.703 | | |

### 5.1 Représentation des graphes de la variation du temps d'exécution :




### 5.2 Représentation du graphe Gf de la complexité théorique :




## 6 Interprétation des résultats.

### 6.1 Remarque et déduction d'une fonction T(n) reliant n au temps d'exécution.

On peut constater que les temps d'exécution T sont à peu près multipliés par 4 lorsque N est multiplié par 2 dans le cas des tableaux en ordre inverse.

$T1(x * N) = x^2 * T(N), N appartenant à l'intervalle [50000 - 2048000000]$

Alors que dans le cas d'un tableau dans le bon ordre ou aléatoire lorsque la taille du tableau est doublée, le temps d'exécution est lui aussi doublé.

On en déduit que le temps d'exécution est proportionnel a N, ce que l'on peut représenter par la formule suivante :

$$T2(x * N) = x * T(N) \text{ , N appartenant à l'intervalle [50000 - 2048000000]}$$

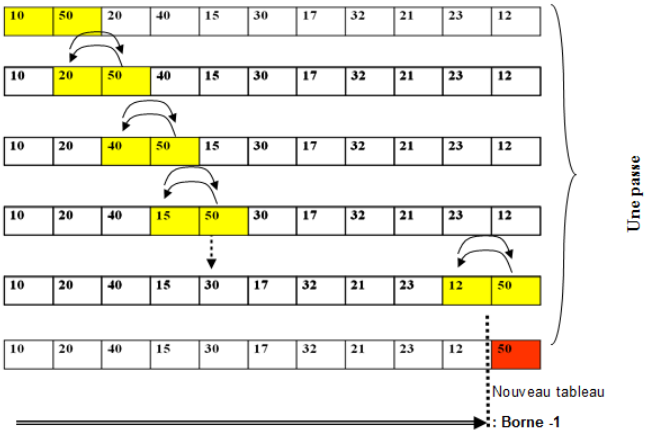
### 6.2 Comparaison de la complexité théorique et expérimentale.

Comme les résultats expérimentales sont majorées par les données du pire cas et minorées par les données au meilleur cas et se rapprochent donc de la complexité moyenne, on en conclue que les mesures expérimentales sont conforment au modèle théorique étudié.

# 4 Tri à Bulles

## 7 Description de la méthode de tri à bulles :

Une variante du tri par sélection est le tri à bulle. L'algorithme de tri à bulle est un algorithme basé sur la comparaison qui effectue plusieurs passages dans une liste. Il compare les éléments adjacents et échange ceux qui ne sont pas en ordre. Chaque passage dans la liste place la valeur suivante la plus grande à sa place. Autrement dit, l'algorithme suit les étapes suivantes :
-Il parcourt le tableau et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
-Après un premier parcours complet du tableau, le plus grand élément est forcément en fin de tableau, à sa position définitive.
-On reparcourt le tableau à nouveau, en s'arrêtant à l'avant-dernier élément.
-Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive.
Il faut donc répéter les parcours du tableau, jusqu'à ce que tous les éléments sont à leurs positions




## 8 Développement de l'algorithme.

```
Algorithme Tri à Bulles ;
Entrée: tab[1..n] d'entier;
Sortie: tab trié;
VAR i,j,v,stock :entier;

DEBUT
      i:=0;
```

```
      v:=0;
      j:=0;
      stock:=0;
   Pour(i=1 à n)
    Faire
        SI(tab[i]<tab[i-1])
        Alors
            stock:=tab[i];
            tab[i]:=tab[i-1];
            tab[i-1]:=stock;
            i:=0;

        Fsi
    Fait
FIN ;
```

# 9 Calcul de Complexité :

## 9.1 Calcule des complexités temporelles en notation asymptotique de Landau O (Grand O :

1. Au Pire Cas :
   Le pire cas (N itérations) est atteint lorsque le plus petit élément est à la fin du tableau, c-à-d lorsque le tableau est trié dans l'ordre inverse. En effet, N-1 comparaisons seront faites sur lors du premier passage, N-2 lors du second, et ainsi de suite. CT(n) = $\sum_{n=2}^{N} i - 1$
   CT(n) = $\frac{N^2-N}{2}$
   Ce qui donne en notation landau O($N^2$).

2. Au Meilleur Cas :
   Le meilleur cas est atteint quand le tableau est déjà trié. Dans ce cas, le tri à bulle fera N comparaisons et 0 échange. Par conséquent, la complexité est linéaire. Ce qui donne en notation landau O(N).

3. La complexité moyenne :
   Le nombre total d'itérations de l'algorithme est égal à : CT(n) = $\sum_{i=2}^{n} i$
   CT(n) = $\frac{N^2+N-2}{2}$

   En effet, le nombre d'échanges de paires d'éléments successifs est égal au nombre d'inversions, c'est-à-dire de couples (i,j) tels que i ¡ j et T(i) ¿ T(j). Ce nombre est indépendant de la manière d'organiser les échanges. Lorsque l'ordre initial des éléments du tableau est aléatoire, il est en moyenne égal à n(n-1)/4 . Donc la complexité moyenne est de l'ordre de $N^2$.
   Ce qui donne en notation landau O($N^2$).

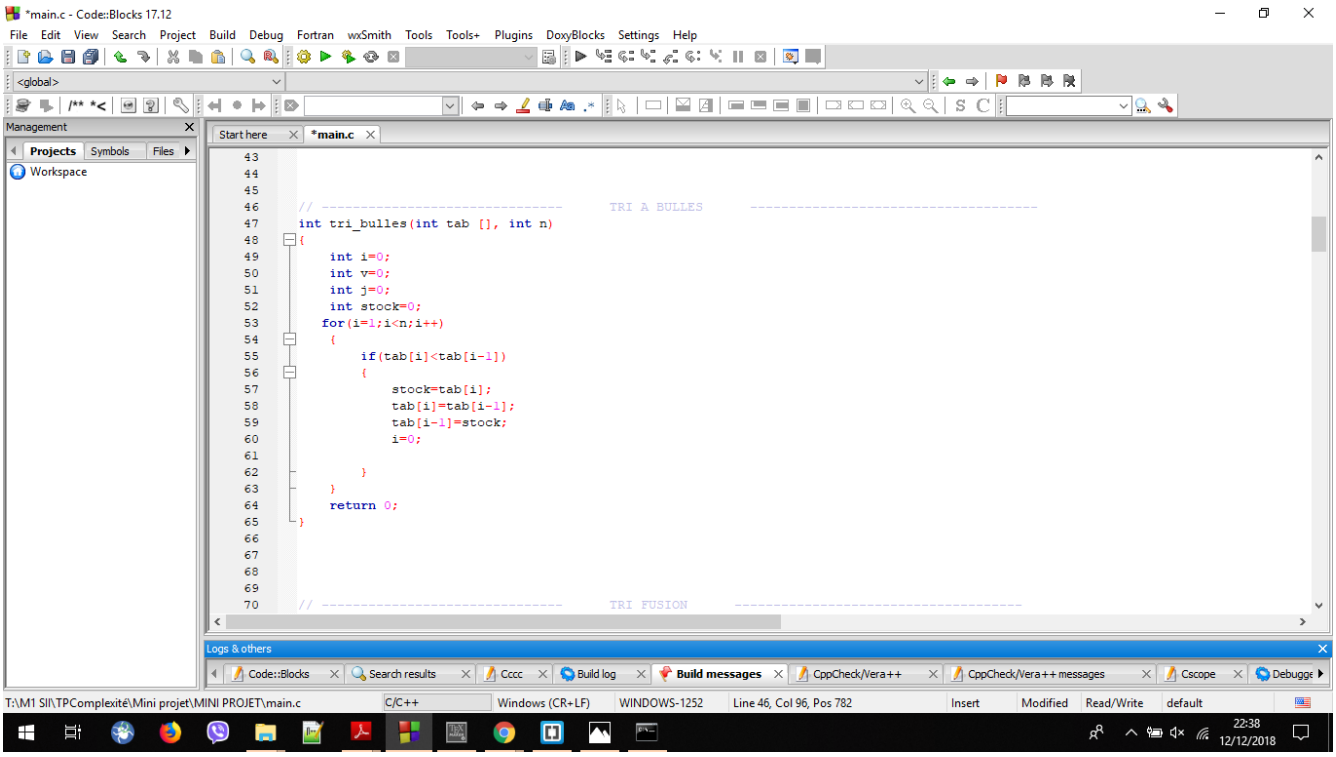
## 9.2 Calcule des complexités spatiales en notation exacte et en notation asymptotique de Landau O (Grand O)

L'algorithme de tri par bulles n'utilise que le tableau à trier de taille N ainsi que 4 variables de type entier. Sur CodeBlocks un entier est sur 8octets. Nous obtenons la complexité spatiale suivante :
CS(N) = 8(N + 4) octets
Ainsi CS(n) = O(N).

# 10 Développement de programme correspondant avec le langage C.

```
int tri_bulles(int tab [], int n)
{
```

```
    int i=0;
    int v=0;
    int j=0;
    int stock=0;
   for(i=1;i<n;i++)
    {
        if(tab[i]<tab[i-1])
        {
            stock=tab[i];
            tab[i]=tab[i-1];
            tab[i-1]=stock;
            i=0;

        }
    }
    return 0;
}
```

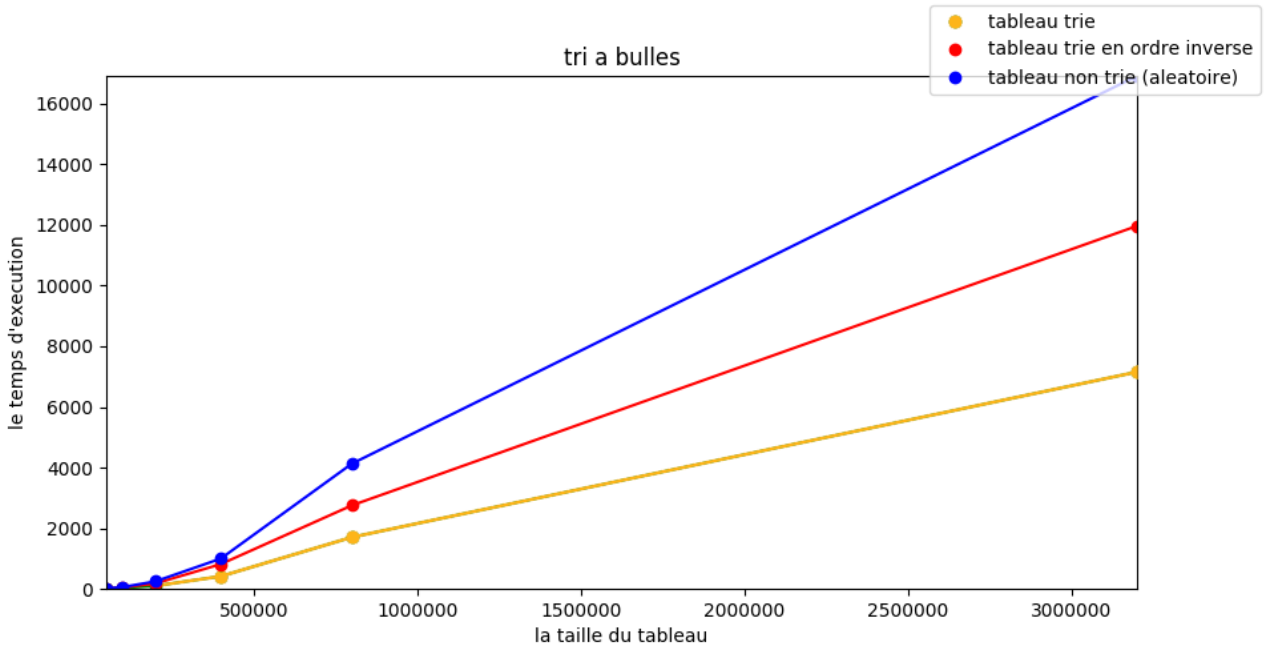



## 11 Mesure des temps d'exécution.

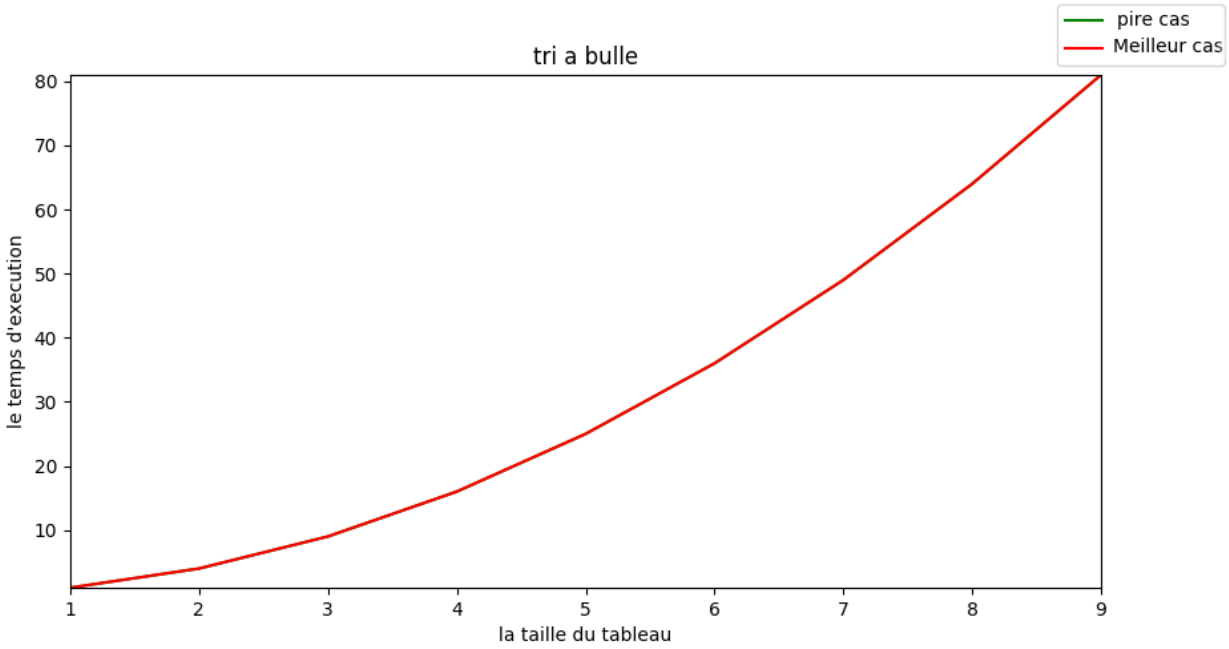
| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Temps (bon ordre) : | 6.402 | 23.349 | 100.354 | 390.419 | 1698.248 | 7152.6874 |
| Temps (Aléatoire) : | 15.629 | 60.603 | 248.038 | 1008.726 | 4110.610 | 16915.0017 |
| Temps (inverse) : | 10.7899 | 51.5167 | 160.093 | 816.74278 | 2749.2962 | 11968.141 |

| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 |
|---|---|---|---|---|---|
| Temps (bon ordre) : | 30100.023 | 120468.012 | 503126.6300 | 2124209.017 | 8098673.092 |
| Temps (Aléatoire) : | 66013.204 | 279403.476 | 1108020.906 | 4349074.095 | 17396596.370 |
| Temps (inverse) : | 47902.567 | 191096.632 | 870844.156 | 3188029.524 | 12309318.120 |

### 11.1 Représentation des graphes de la variation du temps d'exécution :




### 11.2 Représentation du graphe Gf de la complexité théorique :




## 12 Interprétation des résultats.

### 12.1 Remarque et déduction d'une fonction T(n) reliant n au temps d'exécution.

On peut constater que les temps d'exécution T sont à peu près multipliés par 4 lorsque N est multiplié par 2 et cela quelque soit l'ordre des éléments du tableau(ordonnés, inverses ou aléatoire)

$$\mathrm{T(x * N)} = \mathrm{x}^2 * T(N), avec N \in [50000 - 2048000000]$$

(x etant la tangente d'un point sur le graphe).

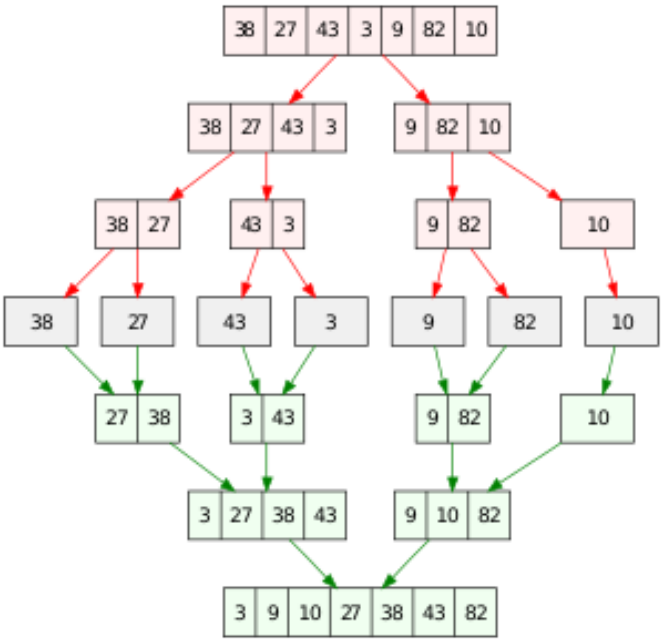
### 12.2 Comparaison de la complexité théorique et expérimentale.

Comme les résultats expérimentale sont majorées par les données du pire cas et minorées par les données au meilleur cas et se rapprochent donc de la complexité moyenne, on en conclue que les mesures expérimentales sont conforment au modèle théorique étudié.

# 5 Tri par Fusion

## 13 Description de la méthode de tri par fusion :

Le tri par fusion suit la règle de Diviser pour Régner. Lors du tri par fusion, la liste non triée est divisée en N sous-listes, chacune comportant un élément, car une liste d'un élément est considérée comme triée. Ensuite, il fusionne à plusieurs reprises ces sous-listes pour produire de nouvelles sous-listes triées, et enfin, une liste triée est produite. Le principe de tri par fusion est résumé dans ces points :
-On divise en deux moitiés la liste à trier jusqu'à ce qu'elle ne puisse plus être divisée.
-On trie chacune d'entre elles.
-On fusionne les deux moitiés obtenues pour reconstituer la liste triée.
Le tri fusion repose fondamentalement sur l'algorithme de fusionnement de deux listes triées qui est de complexité linéaire.




## 14 Développement de l'algorithme.

```
Algorithme Tri par Fusion ;
Entrée: tab[1..n] d'entier; debut, fin: entiers;
Sortie: tab trié;
VAR i :entier;

DEBUT
 SI (debut < fin)
    Alors
         i := debut+(fin-debut)/2;

        tri_fusion(tab, debut, i);//Appel de la fonction Tri par Fusion
        tri_fusion(tab, i+1, fin);//Appel de la fonction Tri par Fusion
        fusion(tab, debut, i, fin);//Appel de la fonction Fusion qui se charge de fusioner les tableaux tri
    Fsi
```

```
FIN;
```

## 15 Calcul de Complexité :

### 15.1 Calcule des complexités temporelles en notation asymptotique de Landau O (Grand O :

Comme le Tri par Fusion a une complexité stable, il n'y a ni pire ni meilleur cas. Le tri par fusion est un algorithme récursif et la complexité temporelle peut être exprimée par la relation de récurrence suivante.

$$T(n) = 2T(n/2) + O(n)$$

On peut aussi voir le problème comme s'il sagisait d'un algorithme itératif, ainsi on a : Le tri par fusion est un algorithme qui se base sur la politique de ≪ diviser pour régner ≫. On résume ça en 3 étapes – La première étape de division calcule le milieu du tableau. Chacune de ces étapes se fait en O (1) fois. L'étape ≪ conquérir ≫ trie récursivement deux sous-tableaux de N/2 éléments. L'étape de fusion fusionne n éléments, ce qui prend un temps O (N). pour chaque niveau de haut en bas, la méthode de fusion au niveau 2 s'effectue sur 2 sous-tableaux de longueur n / 2 chacun. La complexité est ici de 2 * (cN / 2) = cN .La fusion au niveau 3 s'effectue sur 4 sous-tableaux de longueur N / 4 chacun. La complexité ici est 4 * (cN / 4) = cN et ainsi de suite ... Maintenant, la hauteur de cet arbre est (logN + 1) pour un n donné. La complexité globale est donc de (logN + 1) * (cN). C'est O(NlogN) pour l'algorithme de tri par fusion. Ce qui donne en notation landau O(Nlog(N)).

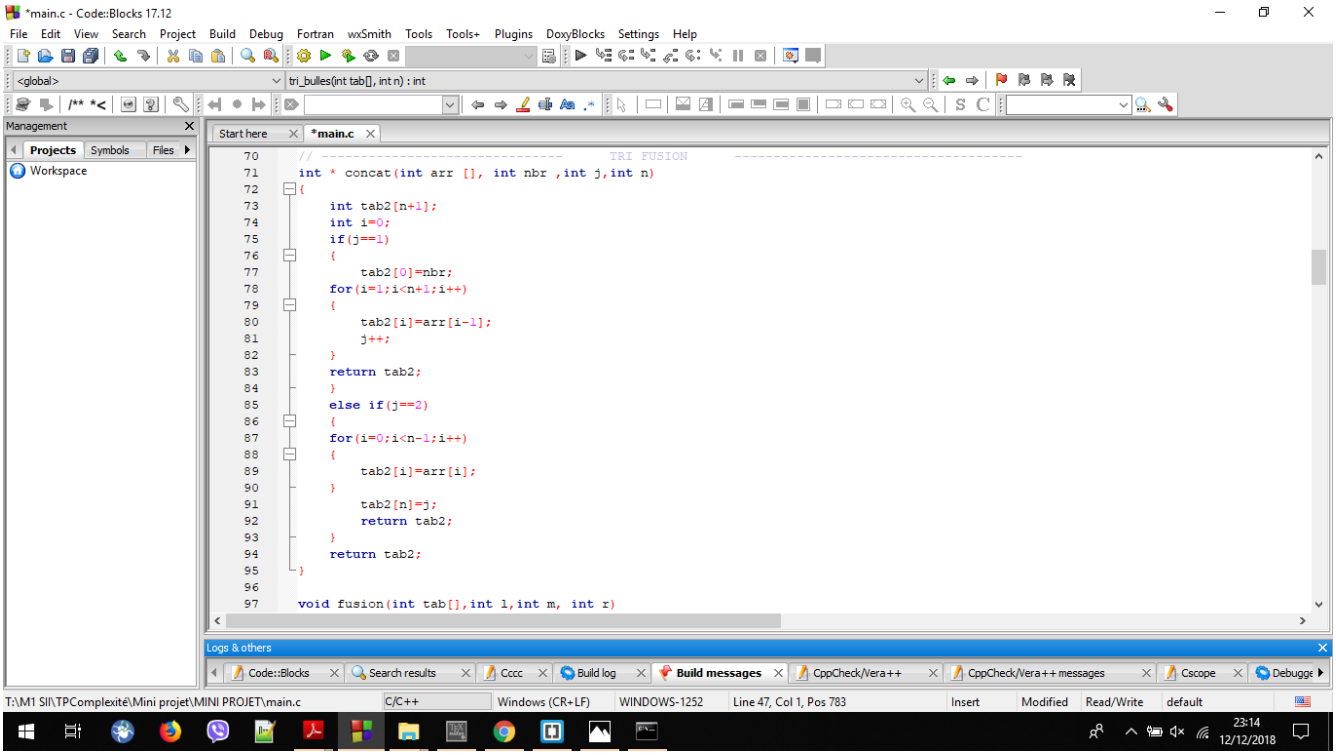
### 15.2 Calcule des complexités spatiales en notation exacte et en notation asymptotique de Landau O (Grand O)

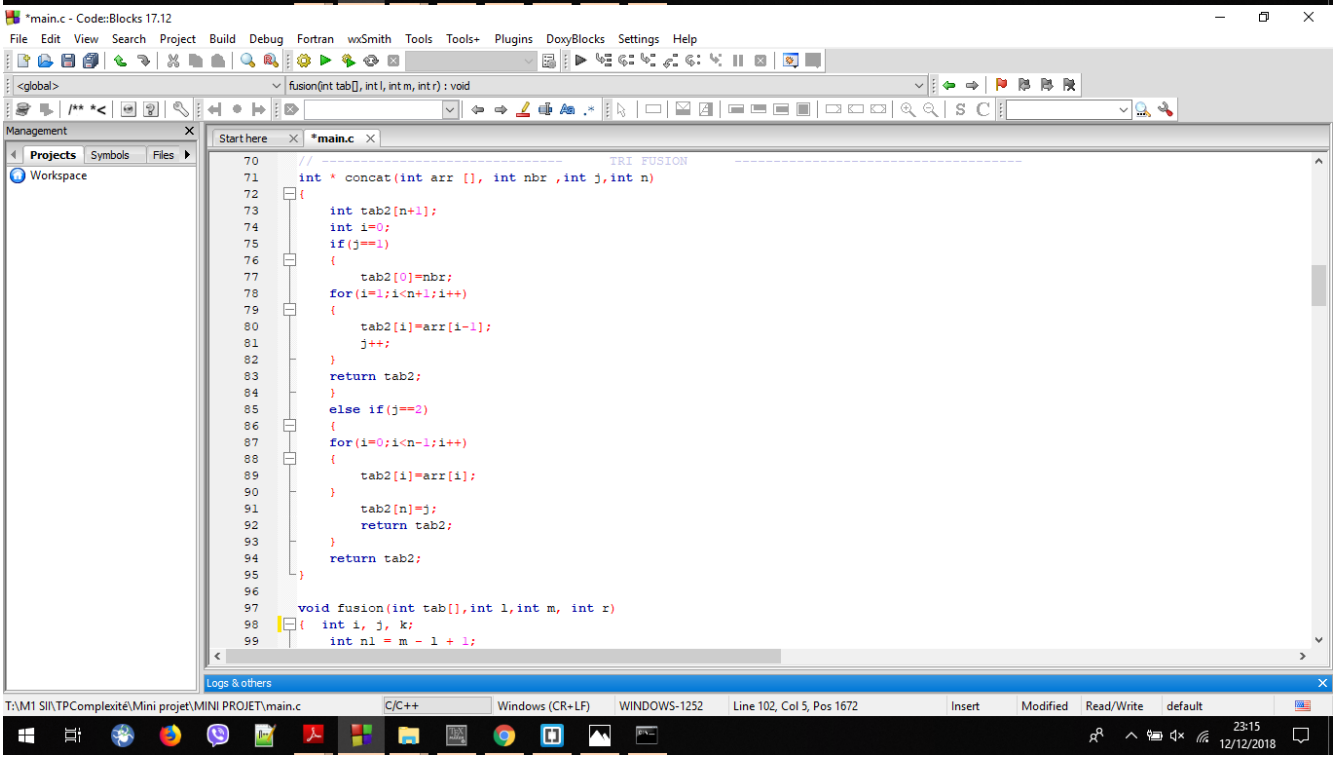
L'algorithme de tri par fusion n'utilise que le tableau à trier de taille N ainsi que 4 variables de type entier.
Sur CodeBlocks un entier est sur 8octets. Nous obtenont la complexité spatiale suivante :
CS(N) = 8(N + 3) octets
Ainsi CS(n) = O(N).

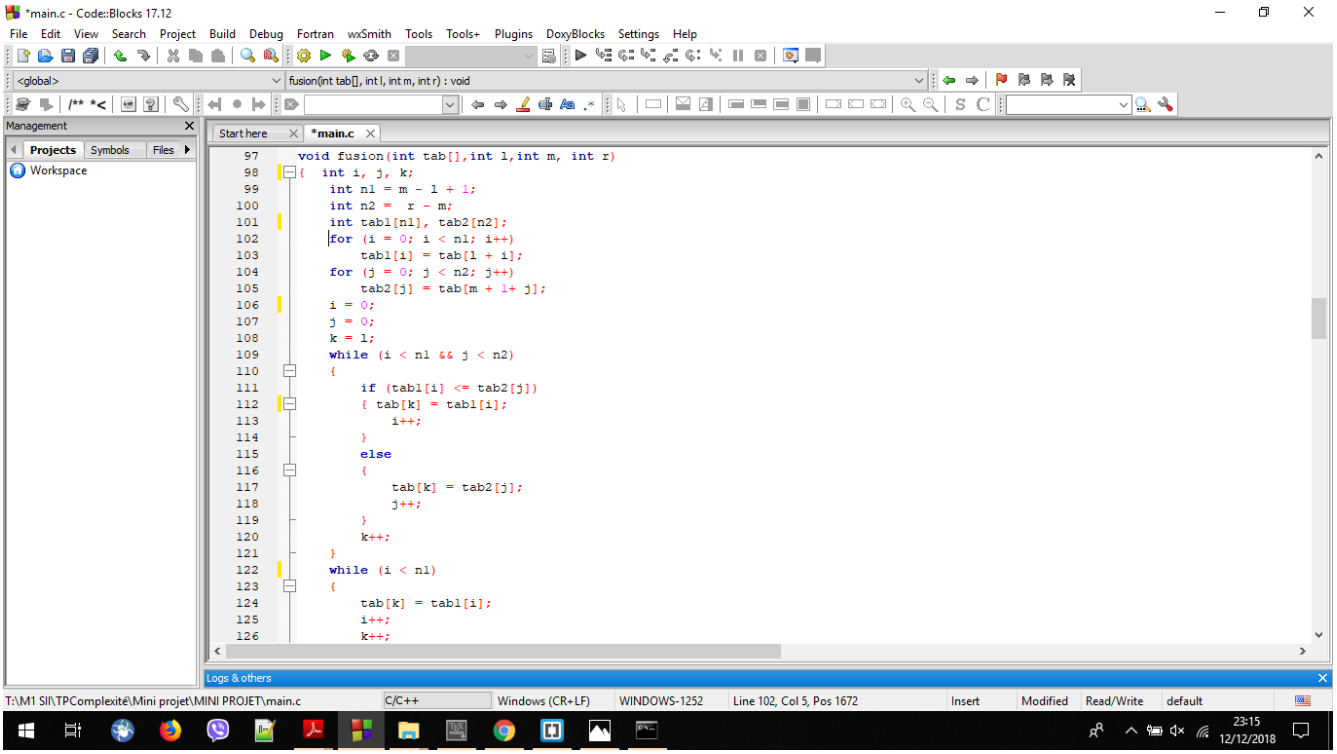
## 16 Développement de programme correspondant avec le langage C.

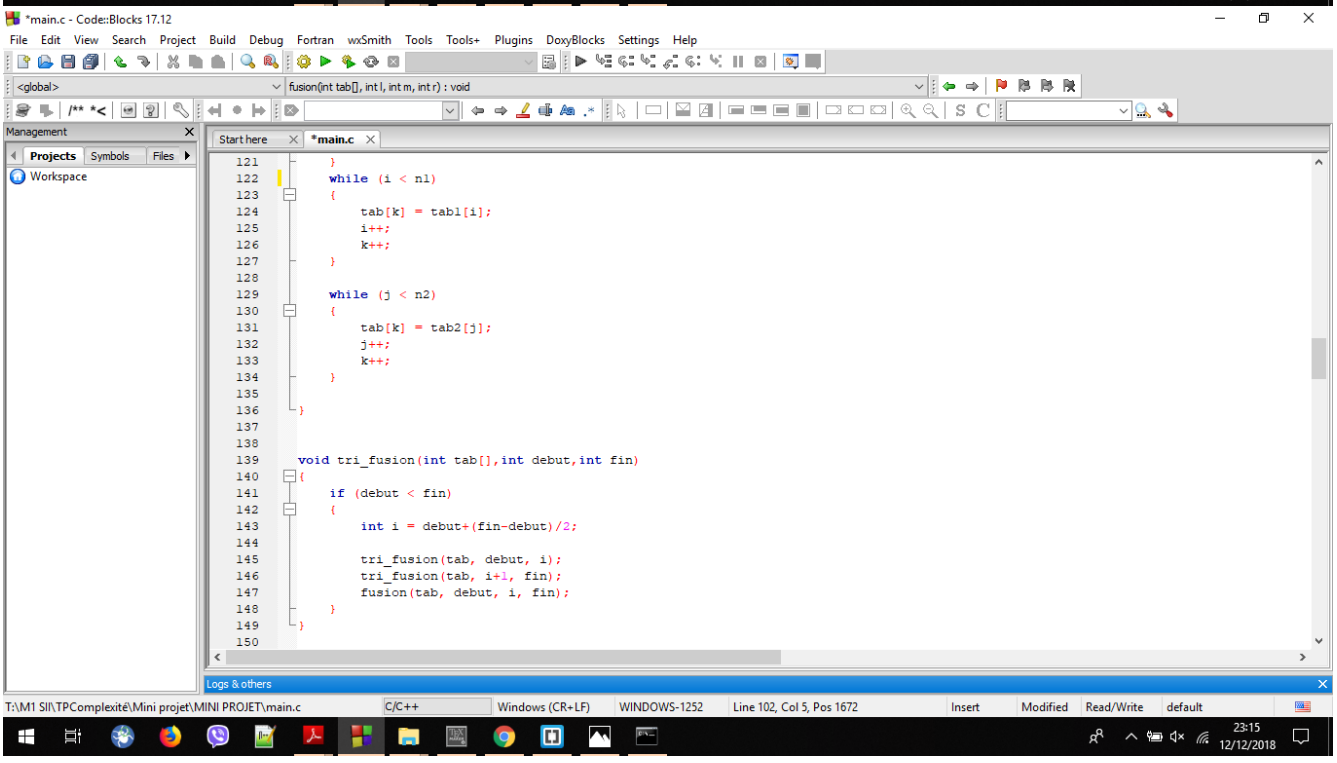
```
int * concat(int arr [], int nbr ,int j,int n)
{
    int tab2[n+1];
    int i=0;
    if(j==1)
    {
        tab2[0]=nbr;
    for(i=1;i<n+1;i++)
    {
        tab2[i]=arr[i-1];
        j++;
    }
    return tab2;
    }
    else if(j==2)
    {
    for(i=0;i<n-1;i++)
    {
        tab2[i]=arr[i];
    }
        tab2[n]=j;
        return tab2;
    }
```

```
    return tab2;
}

void fusion(int tab[],int l,int m, int r)
{  int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;
    int tab1[n1], tab2[n2];
    for (i = 0; i < n1; i++)
        tab1[i] = tab[l + i];
    for (j = 0; j < n2; j++)
        tab2[j] = tab[m + 1+ j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (tab1[i] <= tab2[j])
        { tab[k] = tab1[i];
            i++;
        }
        else
        {
            tab[k] = tab2[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        tab[k] = tab1[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        tab[k] = tab2[j];
        j++;
        k++;
    }

}


void tri_fusion(int tab[],int debut,int fin)
{
    if (debut < fin)
    {
        int i = debut+(fin-debut)/2;

        tri_fusion(tab, debut, i);
        tri_fusion(tab, i+1, fin);
        fusion(tab, debut, i, fin);
    }
}
```


*main.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
tri_bulles(int tab[], int n) : int
Management
Projects Symbols Files
Workspace
Start here
*main.c
70 // -------------------------------- TRI FUSION ----------------------------------
71 int * concat(int arr [], int nbr ,int j,int n)
72 {
73 int tab2[n+1];
74 int i=0;
75 if(j==1)
76 {
77 tab2[0]=nbr;
78 for(i=1;i<n+1;i++)
79 {
80 tab2[i]=arr[i-1];
81 j++;
82 }
83 return tab2;
84 }
85 else if(j==2)
86 {
87 for(i=0;i<n-1;i++)
88 {
89 tab2[i]=arr[i];
90 }
91 tab2[n]=j;
92 return tab2;
93 }
94 return tab2;
95 }
96
97 void fusion(int tab[],int l,int m, int r)
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
Debugge
T:\M1 SII\TPComplexité\Mini projet\MINI PROJET\main.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 47, Col 1, Pos 783
Insert
Modified
Read/Write
default
23:14
12/12/2018


*main.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
fusion(int tab[], int l, int m, int r) : void
Management
Projects Symbols Files
Workspace
Start here
*main.c
70 // -------------------------------- TRI FUSION ----------------------------------
71 int * concat(int arr [], int nbr ,int j,int n)
72 {
73 int tab2[n+1];
74 int i=0;
75 if(j==1)
76 {
77 tab2[0]=nbr;
78 for(i=1;i<n+1;i++)
79 {
80 tab2[i]=arr[i-1];
81 j++;
82 }
83 return tab2;
84 }
85 else if(j==2)
86 {
87 for(i=0;i<n-1;i++)
88 {
89 tab2[i]=arr[i];
90 }
91 tab2[n]=j;
92 return tab2;
93 }
94 return tab2;
95 }
96
97 void fusion(int tab[],int l,int m, int r)
98 { int i, j, k;
99 int n1 = m - l + 1;
Logs & others
T:\M1 SII\TPComplexité\Mini projet\MINI PROJET\main.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 102, Col 5, Pos 1672
Insert
Modified
Read/Write
default
23:15
12/12/2018



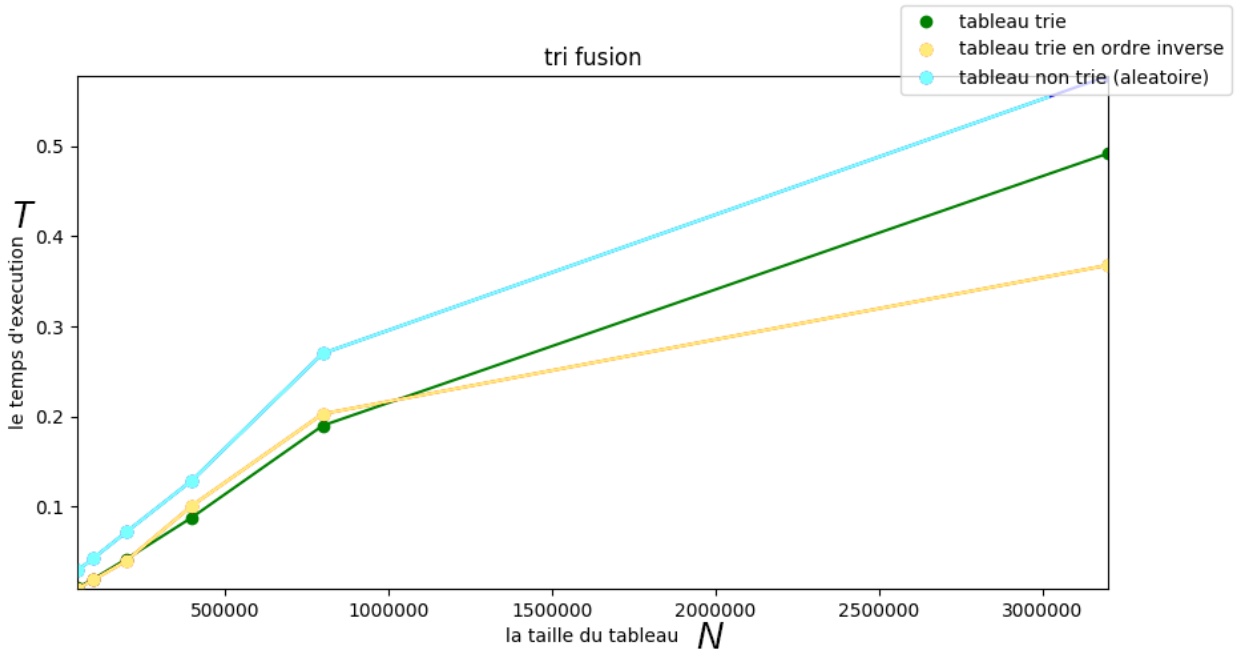




## 17 Mesure des temps d'exécution.

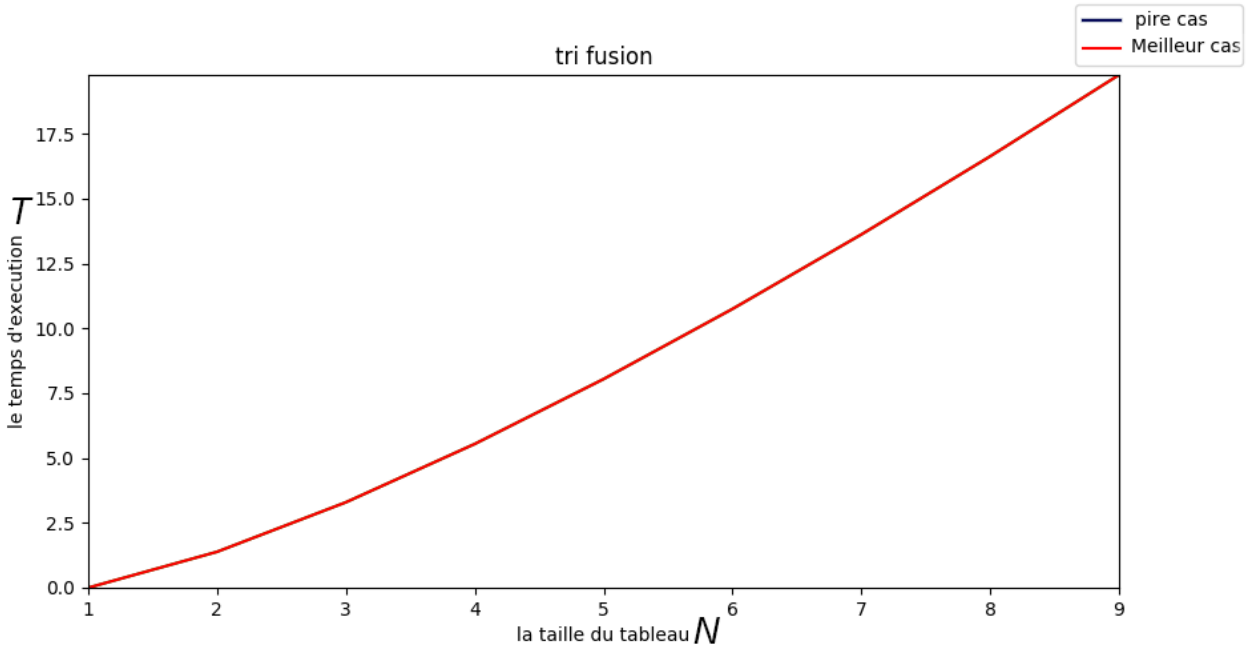
| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Temps (bon ordre) : | 0.07 | 0.020 | 0.042 | 0.088 | 0.190 | 0.492 |
| Temps (Aléatoire) : | 0.019 | 0.043 | 0.042 | 0.139 | 0.270 | 0.577 |
| Temps (inverse) : | 0.001 | 0.019 | 0.030 | 0.101 | 0.201 | 0.408 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
| Temps (bon ordre) : | 1.379 | 2.682 | 4.060 | 7.011 | 13.612 | 24.624 | 52.248 |
| Temps (Aléatoire) : | 1.160 | 2.415 | 5.543 | 10.062 | 27.367 | 54.734 | 109.462 |
| Temps (inverse) : | 0.729 | 1.465 | 3.0902 | 7.004 | 15.544 | 29.18 | 58.336 |

### 17.1 Représentation des graphes de la variation du temps d'exécution :




### 17.2 Représentation du graphe Gf de la complexité théorique :



## 18 Interprétation des résultats.

### 18.1 Remarque et déduction d'une fonction T(n) reliant n au temps d'exécution.

On peut constater que les temps d'exécution sont approximativement multipliés par 2 lorsque N est multipliés par 2 pour les tableaux en ordre, en ordre inverse et même en ordre aléatoire.

$$T1(x * N) = x * T(N) \text{ , avec N appartient à [50000 - 2048000000]}$$

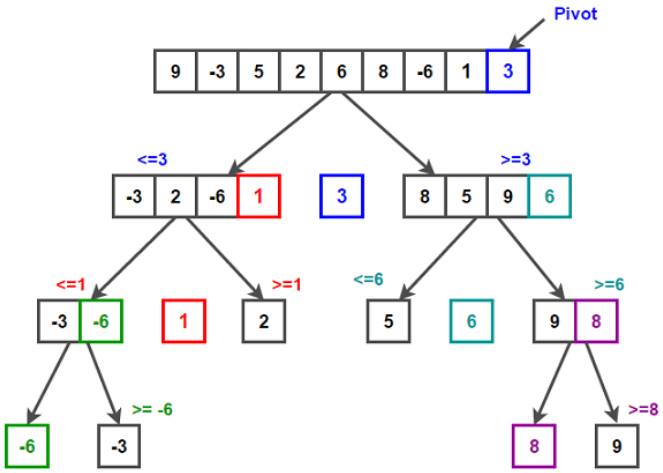
### 18.2 Comparaison de la complexité théorique et expérimentale.

Comme les résultats expérimentale se rapproche de la complexité vue plus haut, on en conclue que les mesures expérimentales sont conforment au modèle théorique étudié.

# 6 Tri Rapide

## 19 Description de la méthode de tri rapide :

Comme le tri par fusion, le tri rapide est un algorithme "Diviser pour Régner". Il sélectionne un élément en tant que pivot et partitionne le tableau donné autour du pivot sélectionné. Il existe de nombreuses versions de tri rapide qui sélectionnent le pivot de différentes manières. Le principe de cet algorithme peut être résumé comme suit :
-Placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le partitionnement.
-Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.




## 20 Développement de l'algorithme.

```
Algorithme Tri Rapide ;
Entrée: tab[1..n] d'entier; premier, dernier: entiers;
Sortie: tab trié;
VAR pivot :entier;

DEBUT
  SI(premier < dernier)
        Alors
           pivot = partitionner(tab, premier, dernier);
           tri_rapide(tab, premier, pivot-1);
```

```
            tri_rapide(tab, pivot+1, dernier);
        Fsi

FIN;
```

## 21 Calcul de Complexité :

### 21.1 Calcule des complexités temporelles en notation asymptotique de Landau O (Grand O :

1. Au Pire Cas : Le pire des cas se produit lorsque le pivot choisi est le plus petit ou le plus grand. Dans ce cas, le tri d'un tableau de N éléments se divise en trois catégories : un tableau de 0 élément et un tableau de N-1 éléments. Puis pour trier un tableau de N-1 on divise en un tableau de taille 0 et un tableau de taille N-2, et ainsi de suite...
   CT(n) = $\sum_{i=1}^{n-1} i$ CT(n) = $\frac{N^2-N}{2}$
   La complexité au pire cas est de de l'ordre de $N^2$.
   Ce qui donne en notation Landau $O(N^2)$.

Au Meilleur Cas : Le meilleur des cas se produit lorsque le processus de scission choisit toujours la médiane comme pivot. Dans ce cas, nous obtenons le même arbre d'appel que celui du type de fusion. La complexité au meilleure cas est de de l'ordre de $Nlog(N)$.
Ce qui donne en notation Landau $O(Nlog(N))$.

Au cas Moyen : Le nombre moyen de comparaison effectuées par l'algorithme de tri rapide est de :$O(Nlog(N))$

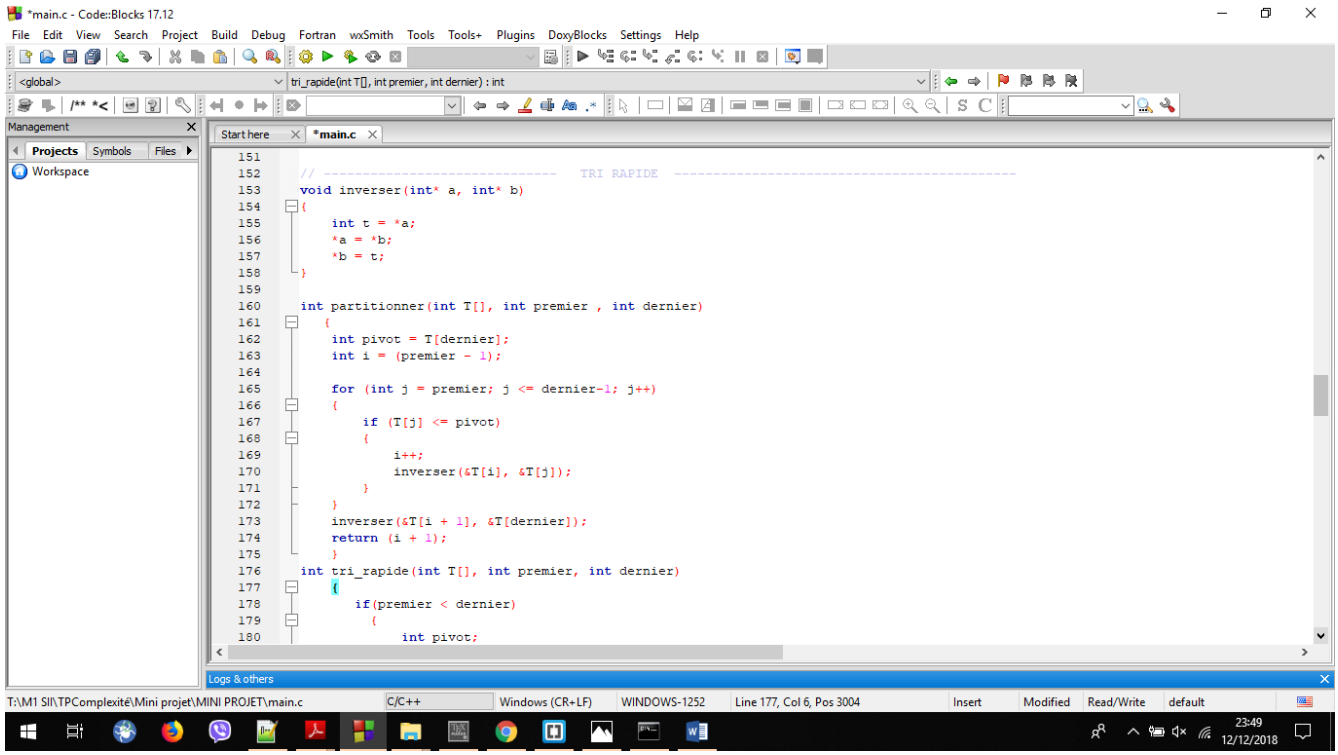
### 21.2 Calcule des complexités spatiales en notation exacte et en notation asymptotique de Landau O (Grand O)

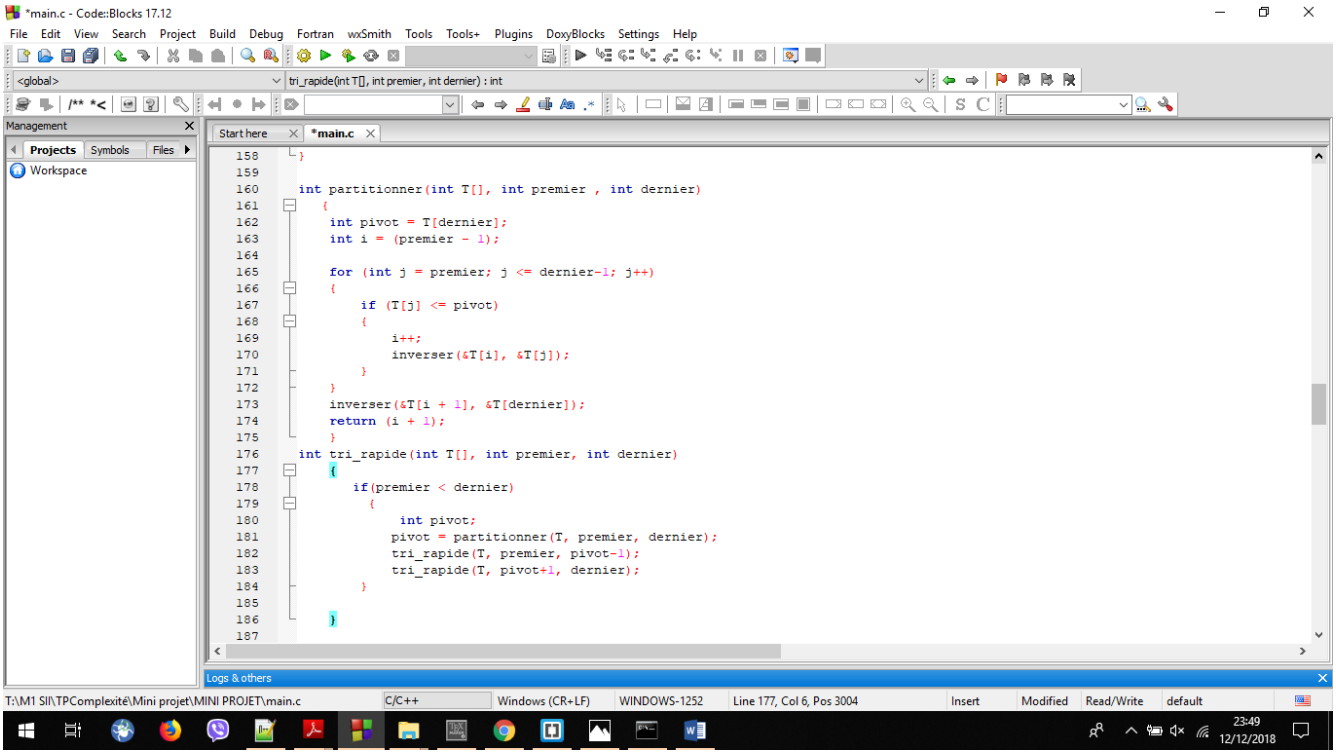
L'algorithme de tri rapide n'utilise que le tableau à trier de taille N ainsi que 4 variables de type entier.
Sur CodeBlocks un entier est sur 8octets. Nous obtenons la complexité spatiale suivante :
CS(N) = 8(N + 3) octets
Ainsi CS(n) = O(N).

## 22 Développement de programme correspondant avec le langage C.

```
void inverser(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partitionner(int T[], int premier , int dernier)
   {
    int pivot = T[dernier];
    int i = (premier - 1);

    for (int j = premier; j <= dernier-1; j++)
    {
        if (T[j] <= pivot)
        {
            i++;
            inverser(&T[i], &T[j]);
        }
    }
```

```
    inverser(&T[i + 1], &T[dernier]);
    return (i + 1);
    }
int tri_rapide(int T[], int premier, int dernier)
    {
       if(premier < dernier)
         {
             int pivot;
           pivot = partitionner(T, premier, dernier);
           tri_rapide(T, premier, pivot-1);
           tri_rapide(T, pivot+1, dernier);
        }

    }
```



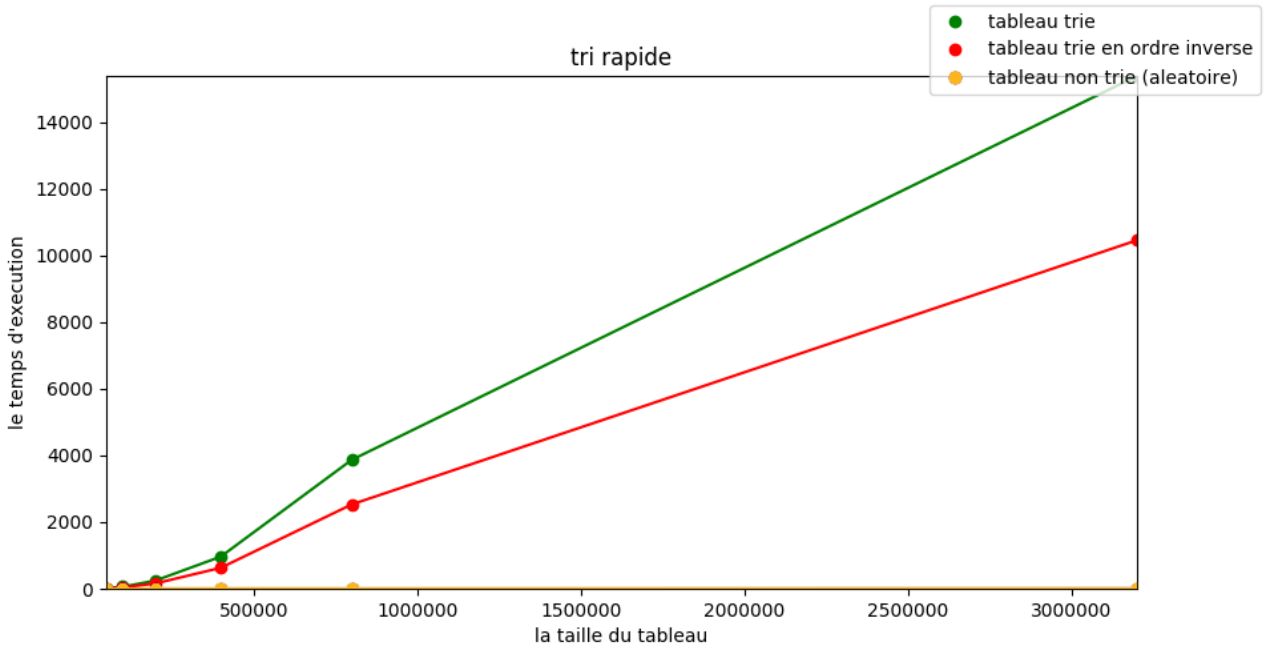



## 23 Mesure des temps d'exécution.

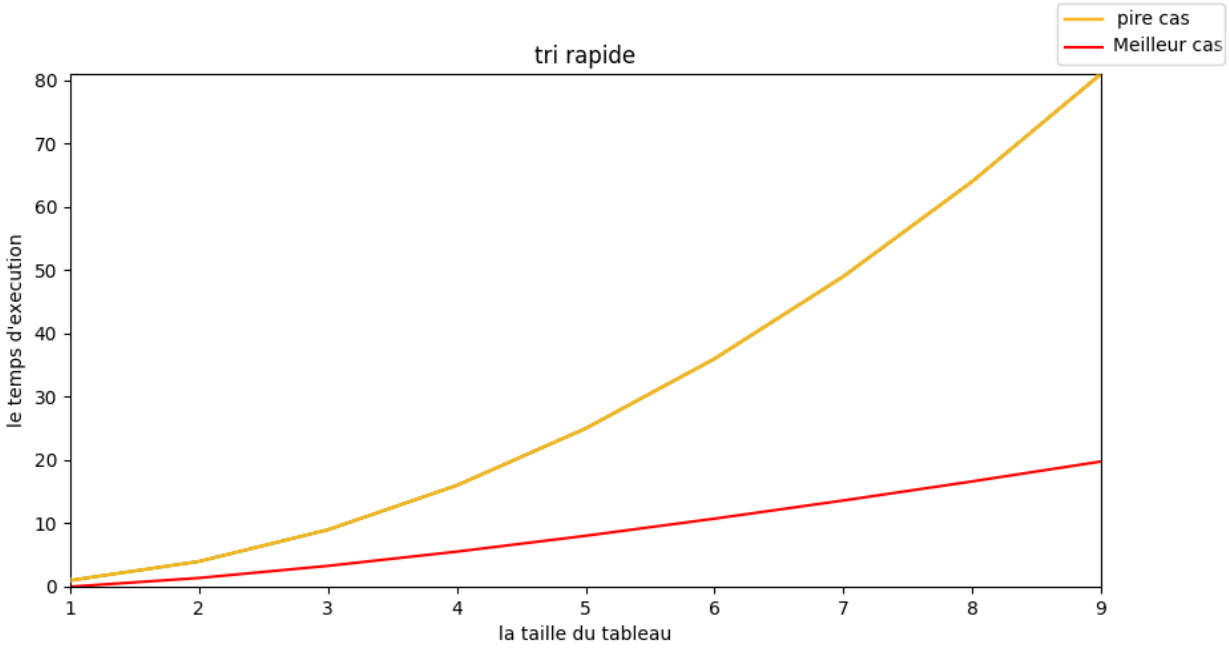
| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Temps (bon ordre) : | 14.062 | 58.707 | 240.606 | 962.812 | 3780.310 | 15400.922 |
| Temps (Aléatoire) : | 0.0098 | 0.023 | 0.062 | 0.132 | 0.325 | 2.900 |
| Temps (inverse) : | 8.787 | 38.963 | 156.1527 | 617.741 | 2537.841 | 10240.6722 |

| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 |
|---|---|---|---|---|---|
| Temps (bon ordre) : | 61509.878 | 243618.654 | 979347.255 | 3917389.023 | 15638210.000 |
| Temps (Aléatoire) : | 9.100 | 32.848 | 124.024 | 458.888 | 1762.132 |
| Temps (inverse) : | 39947.022 | 181972.6423 | 768980.415 | 3107743.419 | 12770221.4320 |

### 23.1 Représentation des graphes de la variation du temps d'exécution :




### 23.2 Représentation du graphe Gf de la complexité théorique :




## 24 Interprétation des résultats.

### 24.1 Remarque et déduction d'une fonction T(n) reliant n au temps d'exécution.

On peut constater que les temps d'exécution sont approximativement multipliés par 4 lorsque N est multipliés par 2 pour les tableaux en ordre, et en ordre inverse.

On en déduit que le temps d'exécution est proportionnel a N, ce que l'on peut représenter par la formule suivante :

$$T(x * N) = x^2 * T(N), N \in [50000 - 2048000000]$$

Toutefois, on remarque que lorsque la taille N du tableau en ordre aléatoire est doublé , le temps d'exécution est lui aussi multiplié par 2.

### 24.2 Comparaison de la complexité théorique et expérimentale.

Comme les résultats expérimentale se rapproche de la complexité moyenne et sont majoré par le pire cas et minoré par le meilleur cas, on en conclue que les mesures expérimentales sont conforment au modèle théorique étudié.

$$T(x * N) = x * T(N) \ , N \in [50000 - 2048000000]$$

# 7 Tri par Tas

## 25 Description de la méthode de tri par tas :

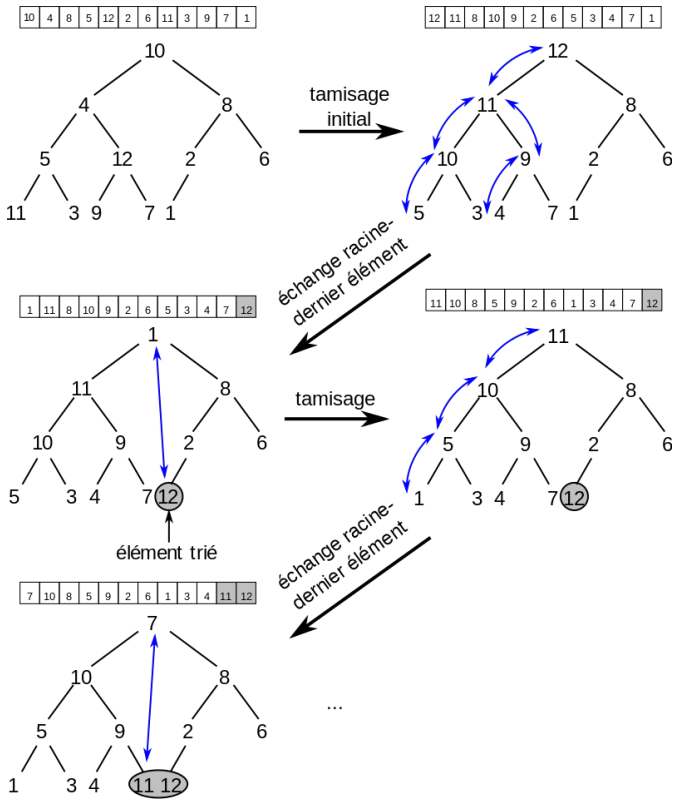
Le tri par tas est l'une des meilleures méthodes de tri en place et ne présente aucun scénario d'un pire cas qui dégénère. L'algorithme de tri de tas est divisé en deux parties fondamentales :
-Création d'un tas de la liste non triée.
-Tri du tableau en supprimant à plusieurs reprises le plus grand élément du tas et en l'insérant dans le tableau. Le tas est reconstruit après chaque enlèvement.

Un tas, tout d'abord, est un arbre binaire dans lequel chaque élément est inférieur à tous ses descendants (selon un ordre déterminé par l'application). Un tas permet l'insertion d'un élément arbitraire et la suppression d'un élément sur une place connue avec une complexité O(logN).

Ce fait nous donne une idée simple d'algorithme de tri. On suppose que les objets à trier sont initialement dans un ensemble quelconque. Le résultat doit être une séquence triée par ordre croissant (déterminé par une fonction passée en argument). La technique comporte deux étapes.

Dans la première étape, l'ensemble initial est parcouru dans un ordre quelconque, et chaque élément est inséré dans un tas initialement vide. Le résultat de cette étape est donc un tas dont la racine (le premier élément de la représentation sous la forme de vecteur) est le plus petit élément (ou l'un des plus petits éléments).

Dans la deuxième étape, une séquence, initialement vide, est créée à partir des éléments du tas obtenu dans la première étape. Le premier élément de la séquence est la racine du tas, puis la racine du tas est supprimée. Cela donne un nouveau tas, la nouvelle racine du tas étant le second élément dans l'ordre, car il est le plus petit parmi les éléments restants. L'opération est répétée tant que le tas n'est pas vide.




## 26 Développement de l'algorithme.

```
Algorithme Tri par Tas ;
Entrée: tab[1..n] d'entier;n: entiers;
Sortie: tab trié;
VAR i,j :entier;
DEBUT
    Pour(i:=n/2 à 0 avec pas:=-1)
    Faire
        tamiser(tab,n,i);
    Fait

    Pour(i:=n-1 à 0 avec pas:=-1)
    Faire
        inverser(tab[i],tab[0]);
        tamiser(tab,i,0);
    Fait

 FIN;
```

## 27 Calcul de Complexité :

### 27.1 Calcule des complexités temporelles en notation asymptotique de Landau O (Grand O :

Au pire, Meilleur et Moyen cas il s'agit de la même complexité, tel que

Dans cet algorithme nous faisons appel à la fonction tamiser : N fois. et dans tamiser le parcours du tableau,sous forme d'un arbre, se fait deux éléments en log(n). Ainsi nous obtenant un complexité asymptotique de Landau :

CT(n) = O(NlogN) ;

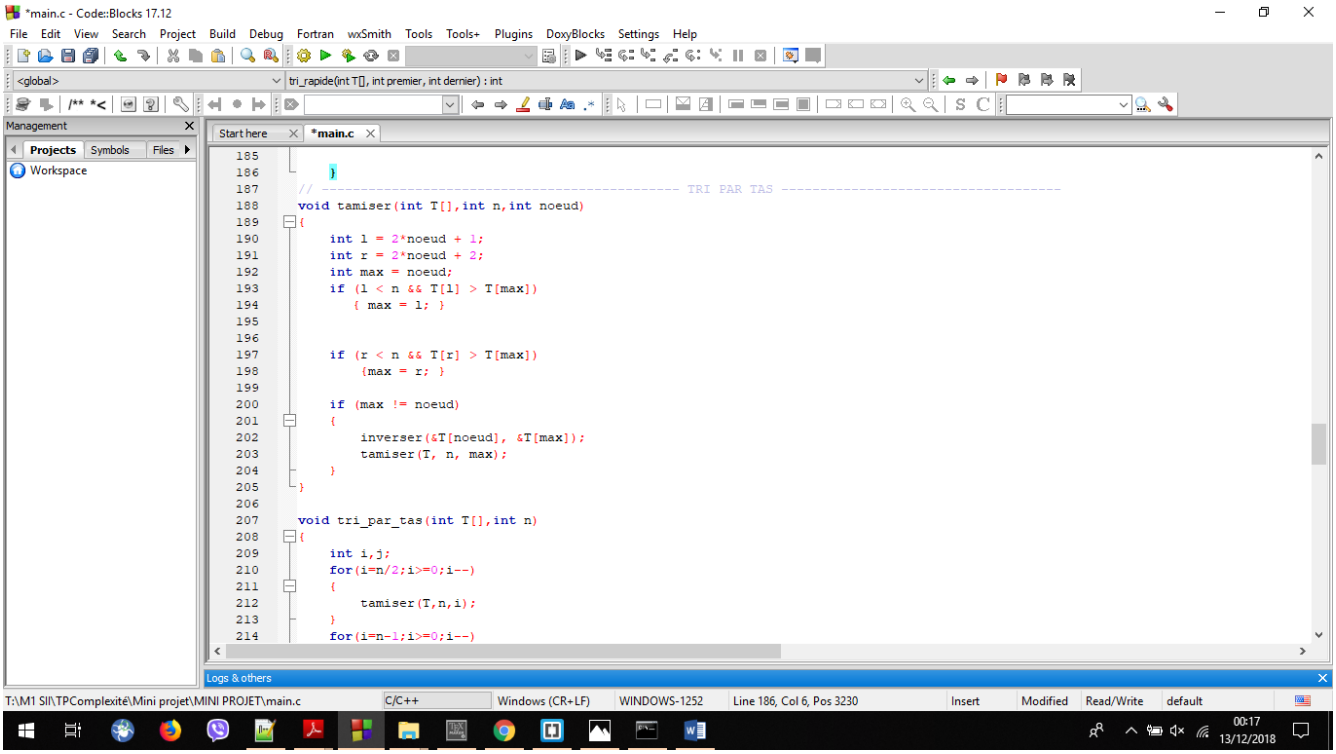
### 27.2 Calcule des complexités spatiales en notation exacte et en notation asymptotique de Landau O (Grand O)

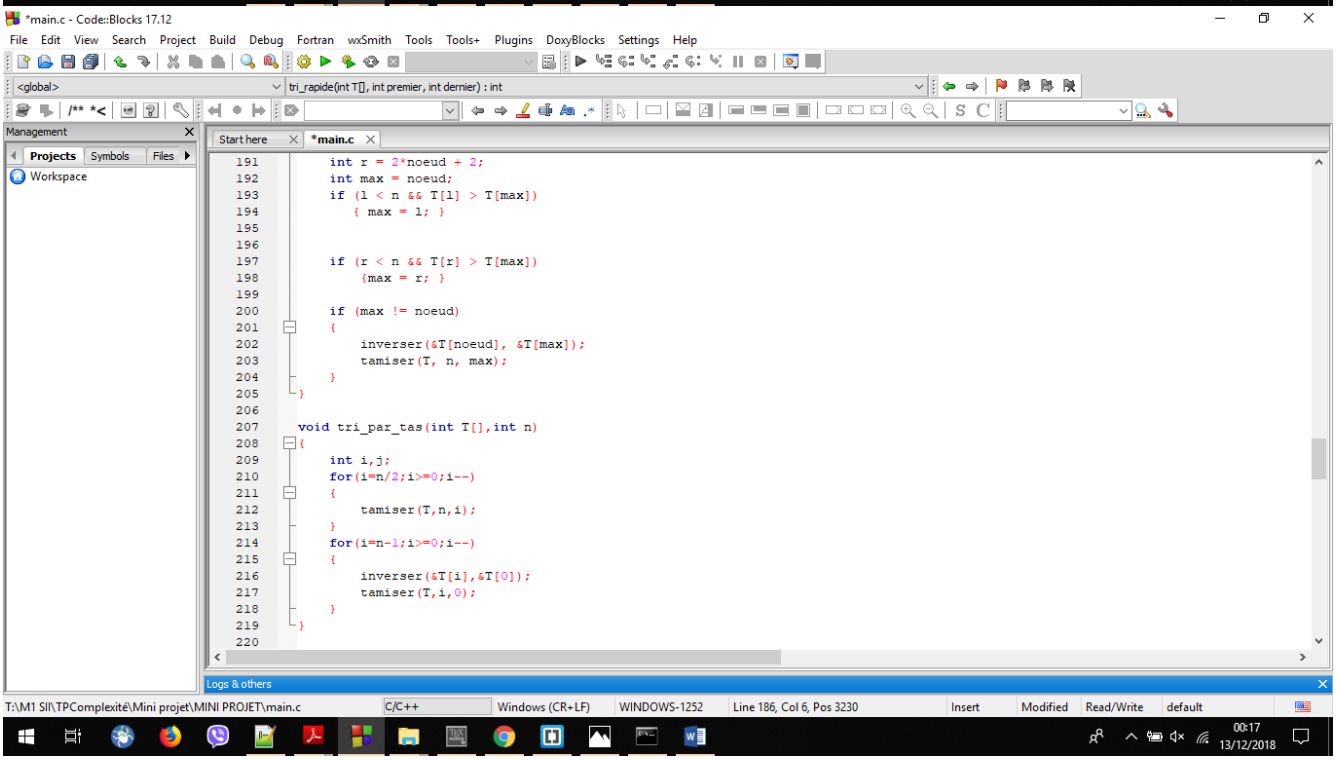
Dans la fonction Tri$_{T}asnousavons$ : $tabdetailleN, l'entierneti$;

Dans la fonction tamiser nous avons : tab de taille N, l'entier n,l,r,max ainsi que noeud ;
on considérant que la taille d'un entier est sur 8 Octets,ainsi :
CS(n) = 8(2N + 7) Octets
Ainsi CS(n) = O(n).

## 28 Développement de programme correspondant avec le langage C.

```
void tamiser(int T[],int n,int noeud)
{
    int l = 2*noeud + 1;
    int r = 2*noeud + 2;
    int max = noeud;
    if (l < n && T[l] > T[max])
       { max = l; }


    if (r < n && T[r] > T[max])
        {max = r; }

    if (max != noeud)
    {
        inverser(&T[noeud], &T[max]);
        tamiser(T, n, max);
    }
}

void tri_par_tas(int T[],int n)
{
    int i,j;
    for(i=n/2;i>=0;i--)
    {
        tamiser(T,n,i);
    }
    for(i=n-1;i>=0;i--)
    {
        inverser(&T[i],&T[0]);
        tamiser(T,i,0);
    }
}
```



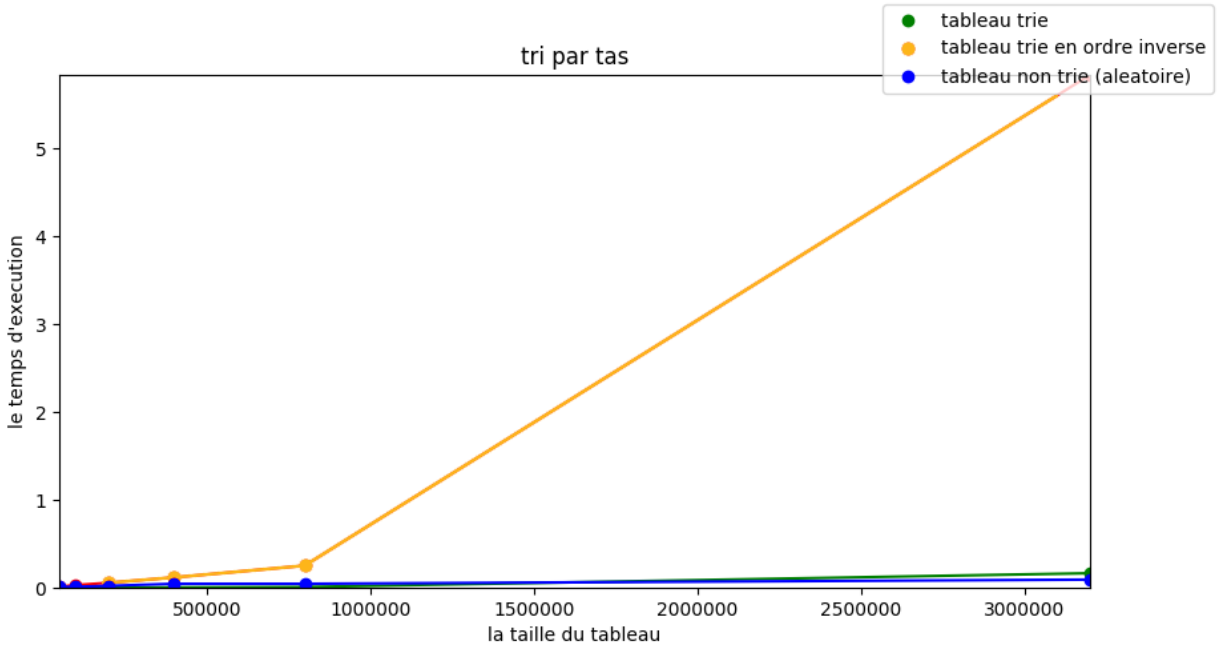




## 29 Mesure des temps d'exécution.

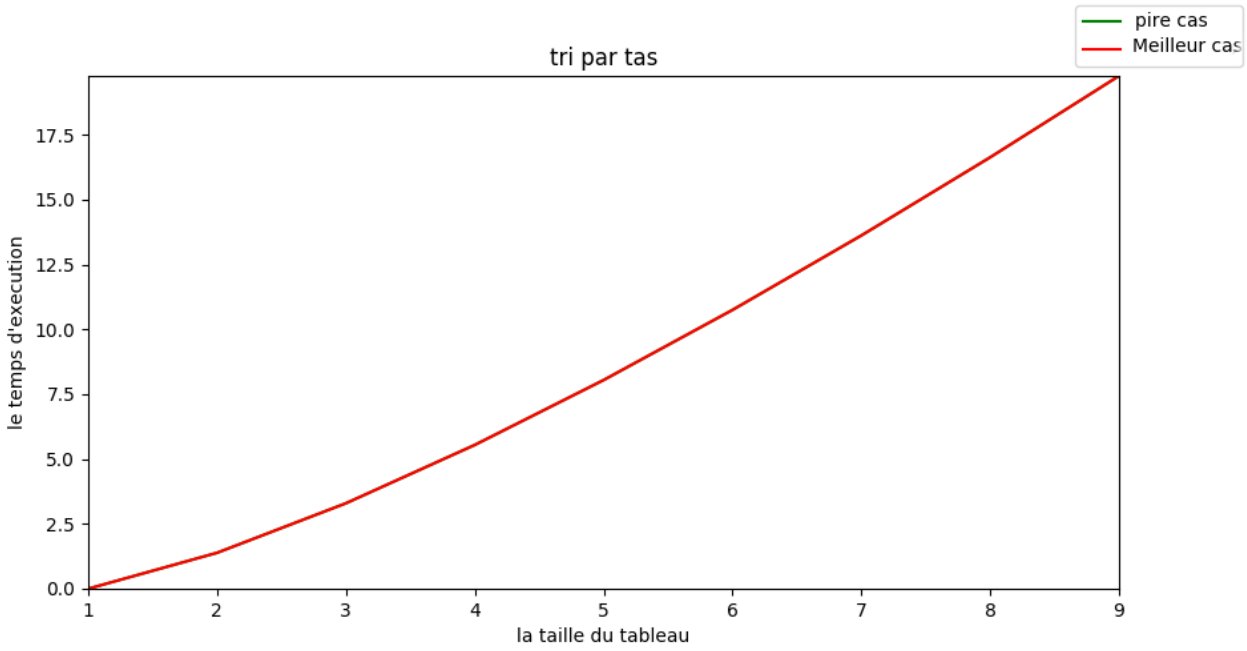
| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Temps (bon ordre) : | 0.0005485 | 0.001030 | 0.002083 | 0.004566 | 0.008513 | 0.159864 |
| Temps (Aléatoire) : | 0.007636 | 0.011022 | 0.021991 | 0.044333 | 0.044333 | 0.089964 |
| Temps (inverse) : | 0.013000 | 0.030700 | 0.056678 | 0.117674 | 0.241907 | 5.906609 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
| Temps (bon ordre) : | 0.331600 | 0.662114 | 1.326228 | 2.397210 | 3.819536 | 7.439002 | 14.27804 |
| Temps (Aléatoire) : | 2.399831 | 5.330937 | 10.343 | 23.820600 | 47.164800 | 94.319446 | 188.6490 |
| Temps (inverse) : | 11.060000 | 23.115268 | 50.001054 | 94.977215 | 200.365220 | 400.73044 | 811.4600 |

### 29.1 Représentation des graphes de la variation du temps d'exécution :



### 29.2 Représentation du graphe Gf de la complexité théorique :




## 30 Interprétation des résultats.

### 30.1 Remarque et déduction d'une fonction T(n) reliant n au temps d'exécution.

On peut constater que quelque soit l'ordre des éléments du tableau, les temps d'exécution sont approximativement doublés lorsque N est multipliés par 2.

On en déduit que le temps d'exécution est proportionnel a N, ce que l'on peut représenter par la formule suivante :

$$T(x * N) = x * T(N),\ N \in [50000 - 2048000000]$$

Toutefois, on remarque que lorsque la taille N du tableau en ordre aléatoire est doublé , le temps d'exécution est lui aussi multiplié par 2.

### 30.2 Comparaison de la complexité théorique et expérimentale.

Comme les résultats expérimentale se rapproche de la complexité moyenne/pire cas/meilleur cas, on en conclue que les mesures expérimentales sont conforment au modèle théorique étudié.

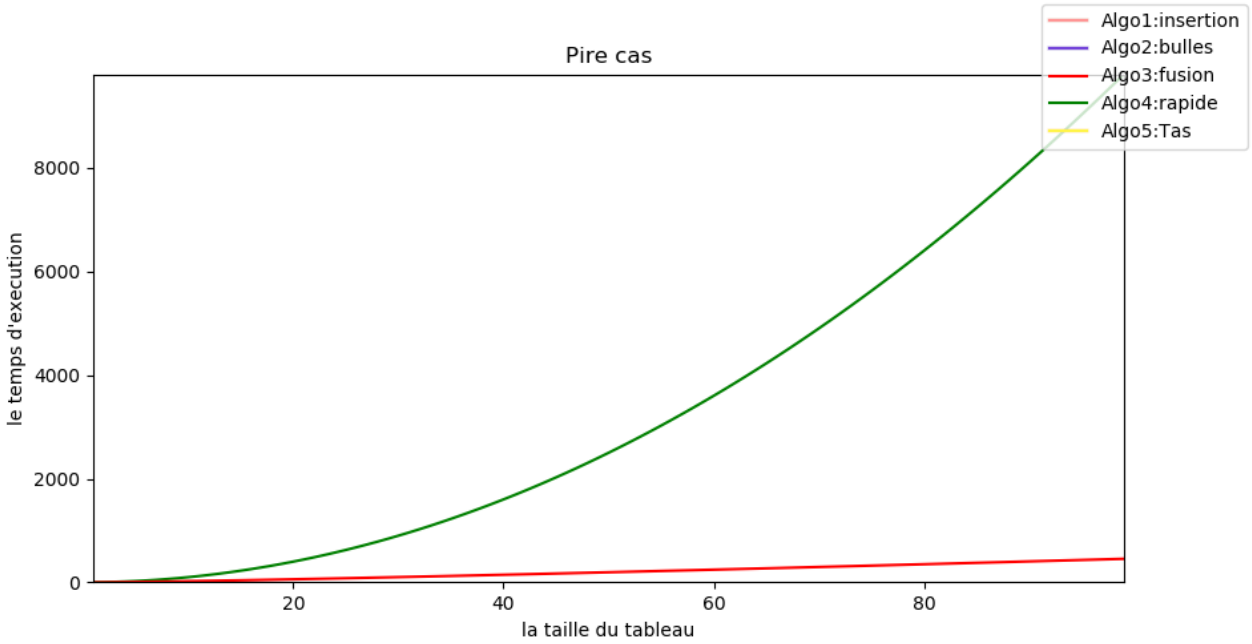
$$T(x * N) = x * T(N)\ ,\ N \in [50000 - 2048000000]$$

# 8 PartieII :

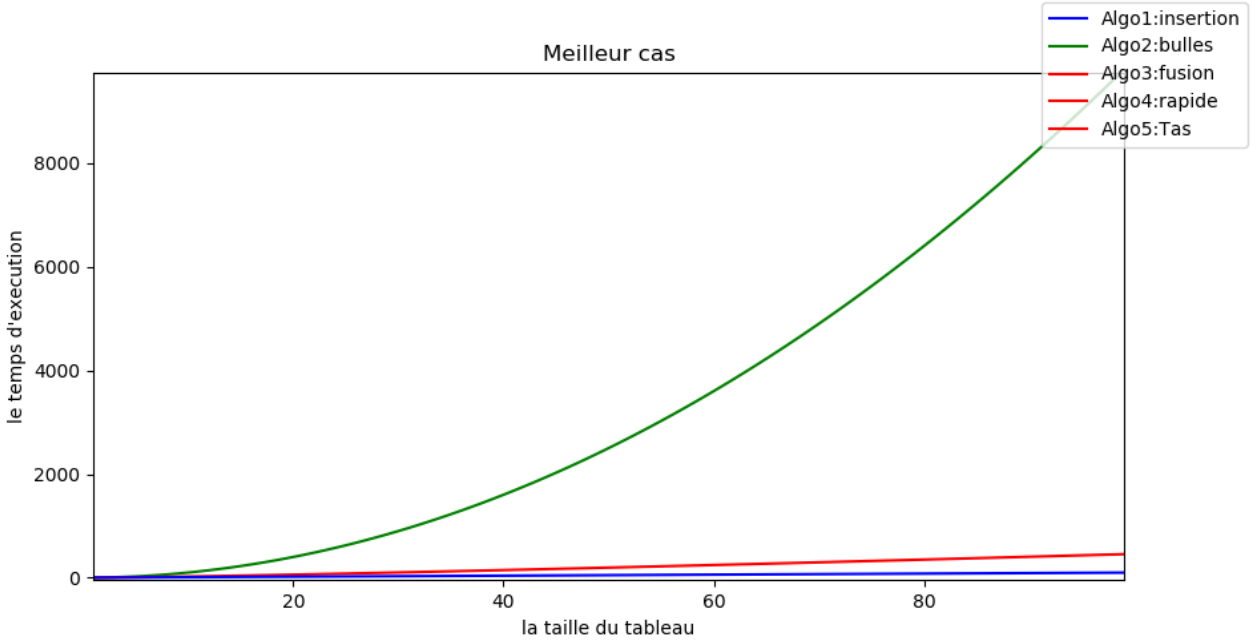
## 31 Représentation des graphes des complexités des cinq (5) algorithmes.

### 31.1 Complexité théorique.

Complexité théorique au pire cas pour les 5 algorithmes :

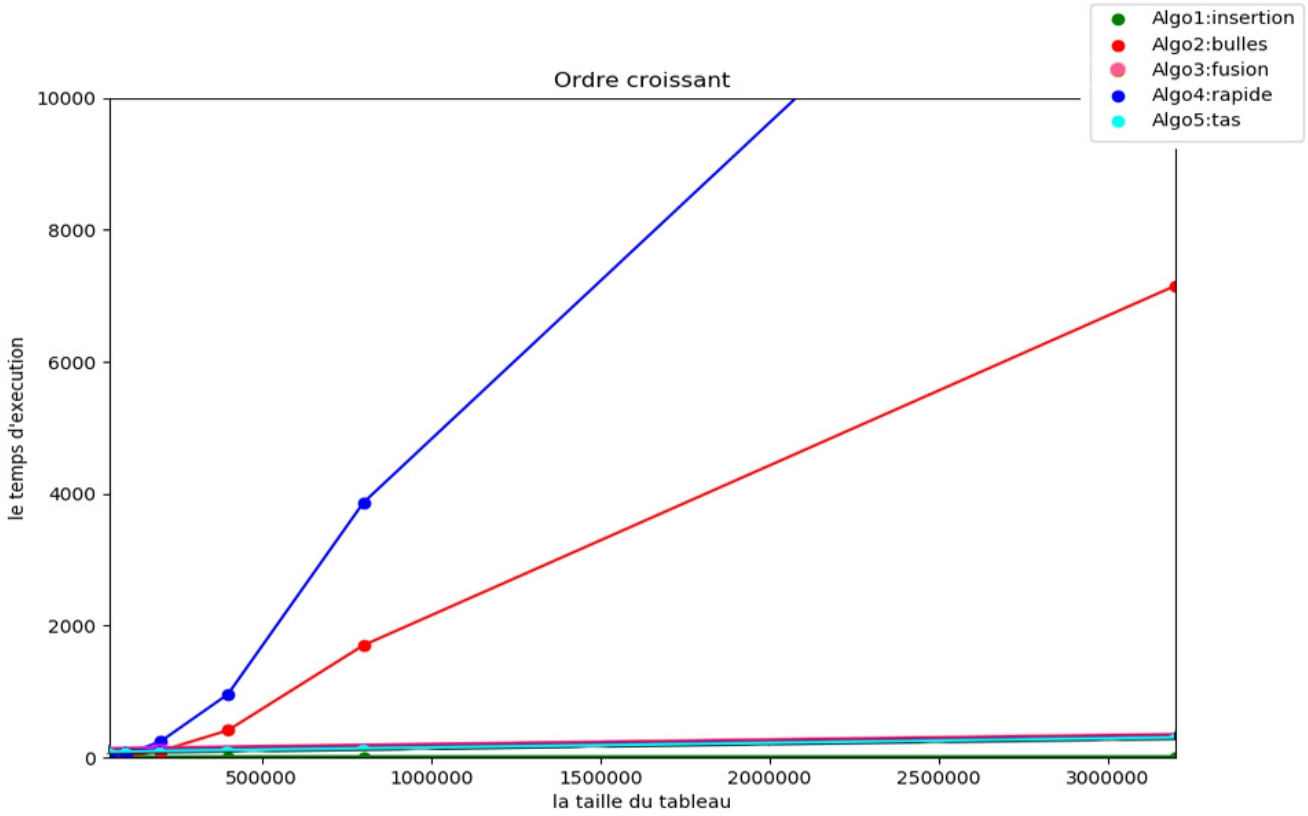



Complexité théorique au meilleur cas pour les 5 algorithmes :

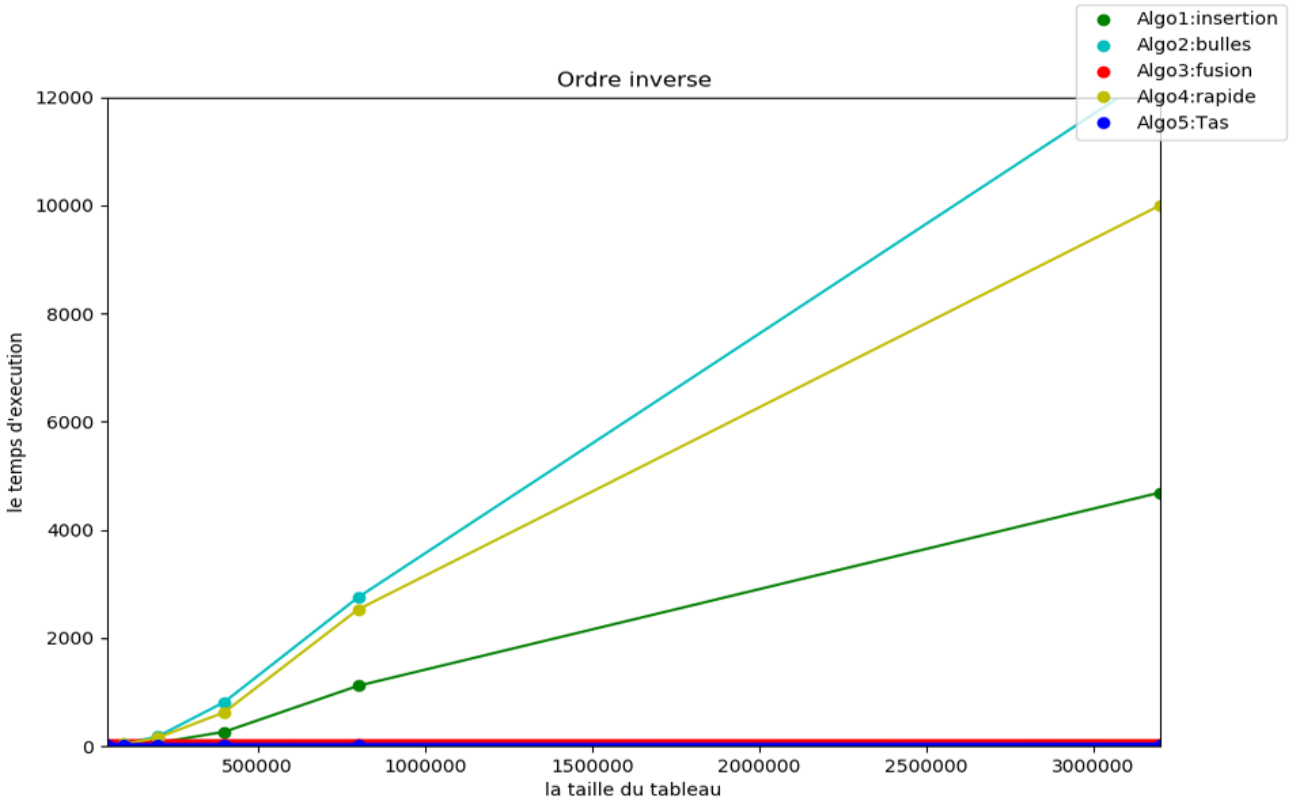



### 31.2 Complexité expérimentale.

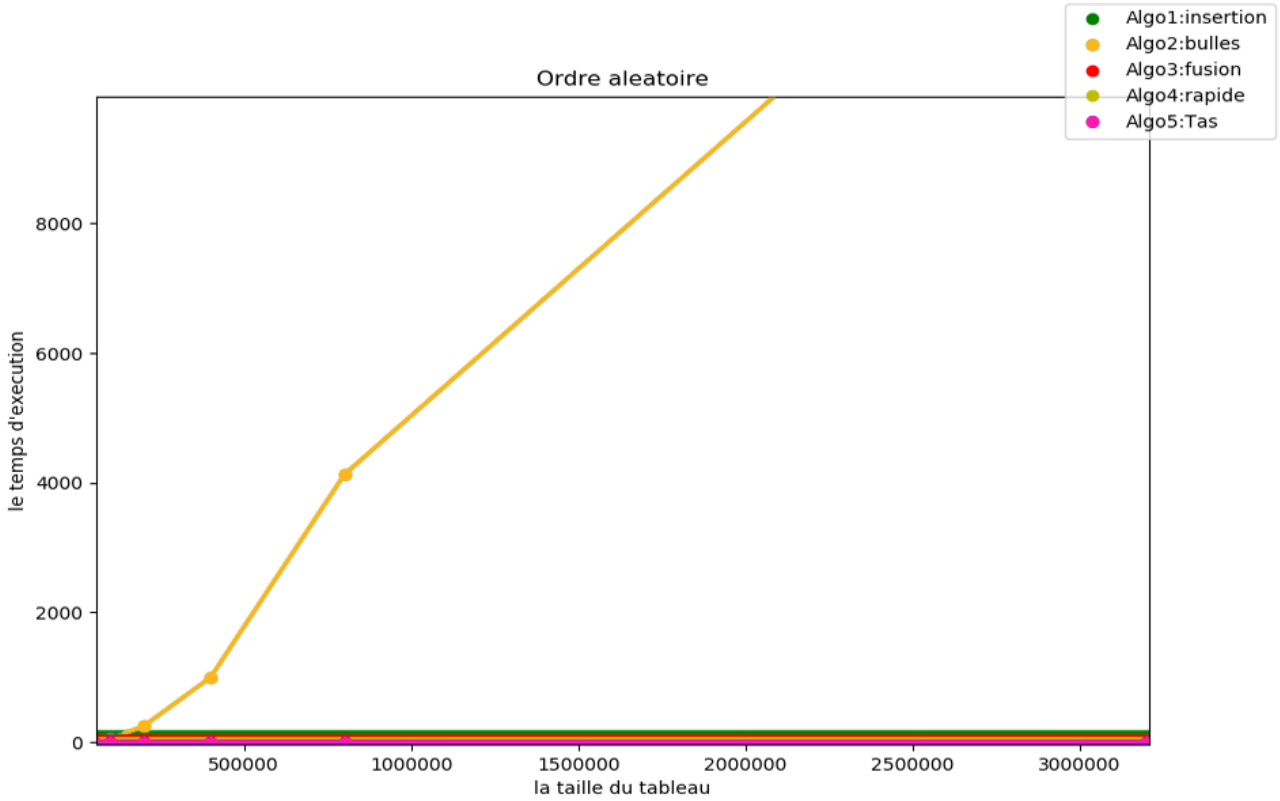
Complexité expérimentale dans le cas d'un tableau ordonné pour les 5 algorithmes :




Complexité expérimentale dans le cas d'un tableau dans l'ordre inverse pour les 5 algorithmes :




Complexité expérimentale dans le cas d'un tableau ordonné aléatoirement pour les 5 algorithmes :




# 32 Tableaux comparatifs : des complexités et des temps d'exécution des 5 algorithmes.

## 32.1 Complexité théorique.

| Algorithme | Complexité au meilleur cas | Complexité au cas moyen | Complexité au pire cas | Stabilité |
|---|---|---|---|---|
| Tri par insertion | N | $N^2$ | $N^2$ | Stable |
| Tri à bulles | N | $N^2$ | $N^2$ | Stable |
| Tri fusion | N * $log_2$(N) | N * $log_2$(N) | N * $log_2$(N) | Stable |
| Tri rapide | N * $log_2$(N) | N * $log_2$(N) | $N^2$ | NonStable |
| Tri par tas | N * $log_2$(N) | N * $log_2$(N) | N * $log_2$(N) | NonStable |

## 32.2 Temps d'exécution.

1. Pire Cas :

| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Tri par insertion | $2.5*10^9$ | $10^{10}$ | $4*10^{10}$ | $1.6*10^{11}$ | $6.4*10^{11}$ | $2.56*10^{12}$ |
| Tri à bulles | $2.5*10^9$ | $10^{10}$ | $4*10^{10}$ | $1.6*10^{11}$ | $6.4*10^{11}$ | $2.56*10^{12}$ |
| Tri fusion | 540988,914 | 1151292,546 | 2441214,529 | 5159687,930 | 10873893,605 | 22856822,699 |
| Tri rapide | $2.5*10^9$ | $10^{10}$ | $4*10^{10}$ | $1.6*10^{11}$ | $6.4*10^{11}$ | $2.56*10^{12}$ |
| Tri par tas | 540988,914 | 1151292,546 | 2441214,529 | 5159687,930 | 10873893,605 | 22856822,699 |

| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
|---|---|---|---|---|---|---|---|
| Tri par insertion | $1.24*10^{13}$ | $4.096*10^{13}$ | $1.6384*10^{14}$ | $6.5536*10^{14}$ | $2.62144*10^{15}$ | 1048576000000 | 4194304000000 |
| Tri à bulles | $1.24*10^{13}$ | $4.096*10^{13}$ | $1.6384*10^{14}$ | $6.5536*10^{14}$ | $2.62144*10^{15}$ | 1048576000000 | 4194304000000 |
| Tri fusion | 47931716,376 | 100299574,70 | 9209471433,32 | 9436687434,48 | 1908864004,60 | 86154547,1555 | 12925603,74231 |
| Tri rapide | $1.24*10^{13}$ | $4.096*10^{13}$ | $1.6384*10^{14}$ | $6.5536*10^{14}$ | $2.62144*10^{15}$ | 1048576000000 | 4194304000000 |
| Tri par tas | 47931716,376 | 100299574,70 | 9209471433,32 | 9436687434,48 | 1908864004,60 | 86154547,1555 | 12925603,74231 |

2. Meilleur cas :

| Taille du Tableau (N) : | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
|---|---|---|---|---|---|---|
| Tri par insertion | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
| Tri à bulles | 50000 | 100000 | 200000 | 400000 | 800000 | 1600000 |
| Tri fusion | 540988,914 | 1151292,546 | 2441214,529 | 5159687,930 | 10873893,605 | 22856822,699 |
| Tri rapide | 540988,914 | 1151292,546 | 2441214,529 | 5159687,930 | 10873893,605 | 122856822,699 |
| Tri par tas | 540988,914 | 1151292,546 | 2441214,529 | 5159687,930 | 10873893,605 | 22856822,699 |

| Taille du Tableau (N) : | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
|---|---|---|---|---|---|---|---|
| Tri par insertion | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
| Tri à bulles | 3200000 | 6400000 | 12800000 | 25600000 | 51200000 | 1024000 | 2048000 |
| Tri fusion | 47931716,376 | 100299574,70 | 9209471433,32 | 9436687434,48 | 1908864004,60 | 86154547,1555 | 912925603.7423181 |
| Tri rapide | 47931716,376 | 100299574,70 | 9209471433,32 | 9436687434,48 | 1908864004,60 | 86154547,1555 | 912925603.7423181 |
| Tri par tas | 47931716,376 | 100299574,70 | 9209471433,32 | 9436687434,48 | 1908864004,60 | 86154547,1555 | 912925603.7423181 |

### 33 Comparaison des 5 Algorithmes.

Comme on peut le constater des graphes, les résultats sont approximativement similaires pour le tri par insertion et le tri à bulle. En revanche, le temps d'exécution du tri par fusion, tri rapide et tas, reste proche de zéro même pour les très grands ensembles de données. Cela correspond parfaitement à leurs complexités théoriques. Depuis, le tri par insertion et le tri à bulle ont une complexité quadratique ; ils sont les plus inefficaces en termes d'exécution.

En effet, le Tri par Insertion et Tri à bulles sont les plus instinctifs et donc les plus simples à implémenter. Comparé aux trois autres qui demandent un peu plus de maitrise mais restent toutefois les moins rapide en temps d'exécution et donc les moins performants.

Quant au tri fusion il est nettement plus performant et donne de meilleurs résultats que les précédents algorithme,suivie du Tri qui peut quand à lui dégénérer vu que son efficacité dépend du choix du pivot, car si ce dernier est mal choisi on se retrouve avec un temps d'exécution similaire à celui des deux premiers algorithmes traités.

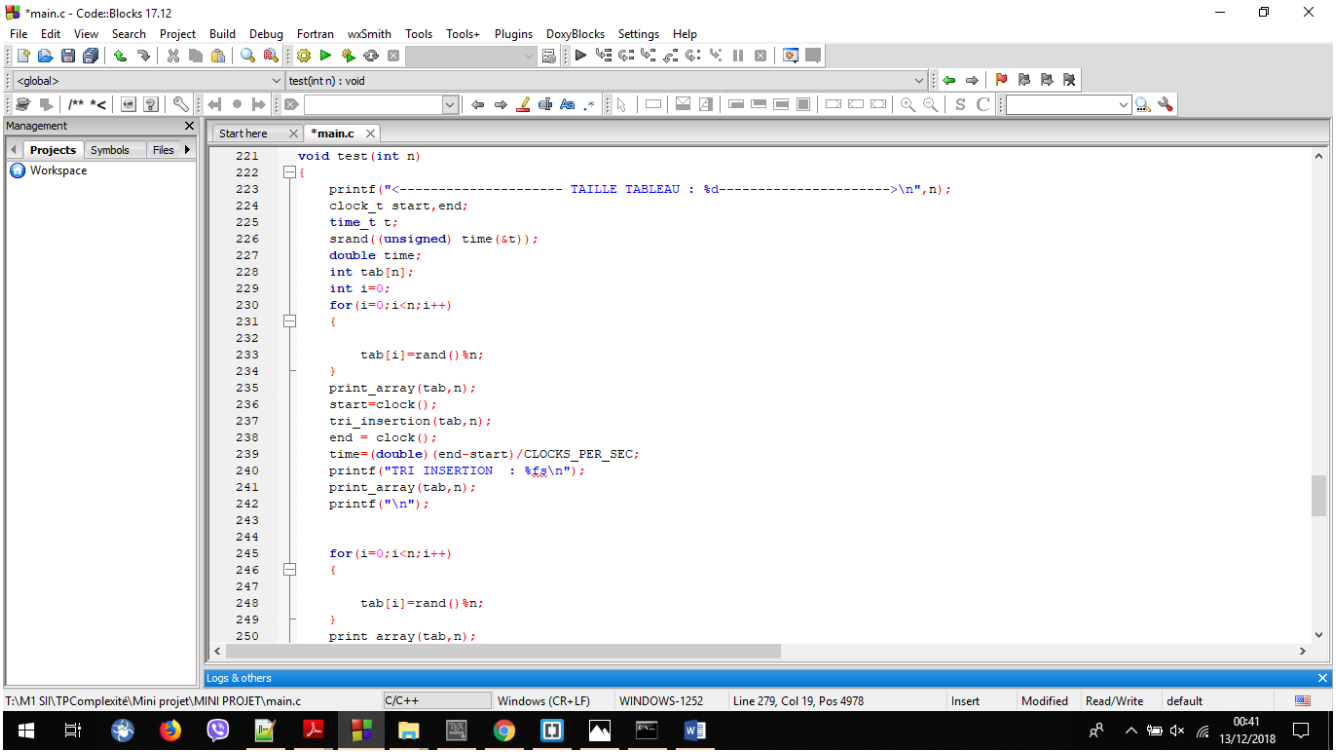
Enfin, le tri par tas est jusqu'au jour d'aujourd'hui considéré comme le plus intéressant des algorithme de tri , il donne les meilleurs temps d'exécution !

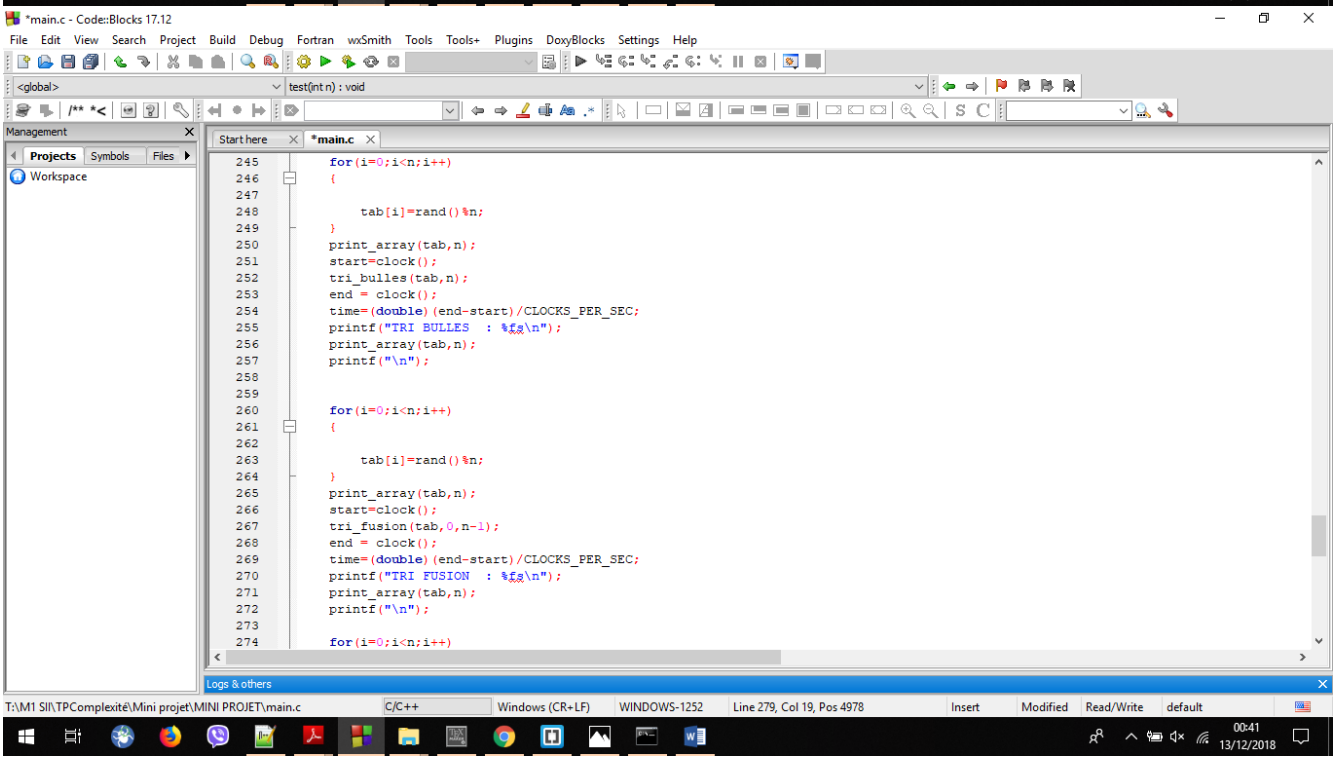
## 9 Programme Principal :

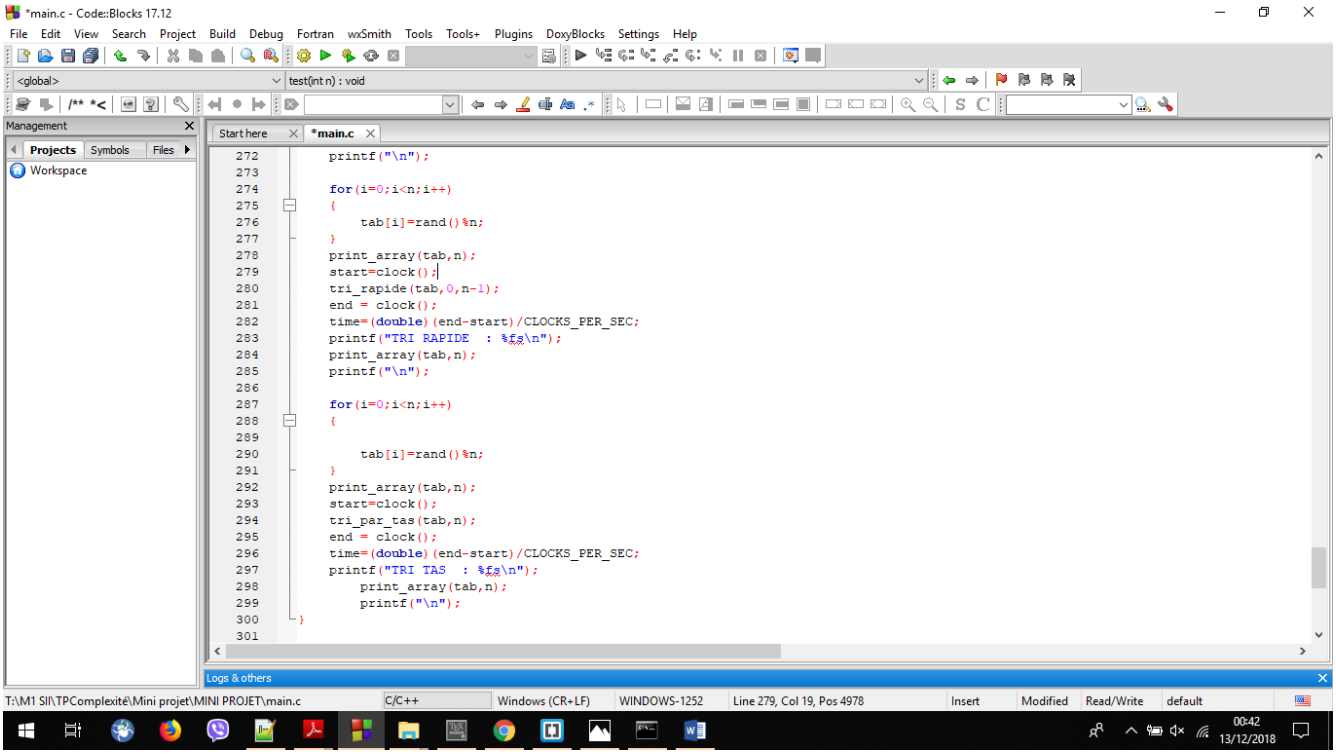
```
void test(int n)
{
    printf("<---------------------- TAILLE TABLEAU : %d---------------------->\n",n);
clock_t start,end;
    time_t t;
    srand((unsigned) time(&t));
double time;
    int tab[n];
    int i=0;
```

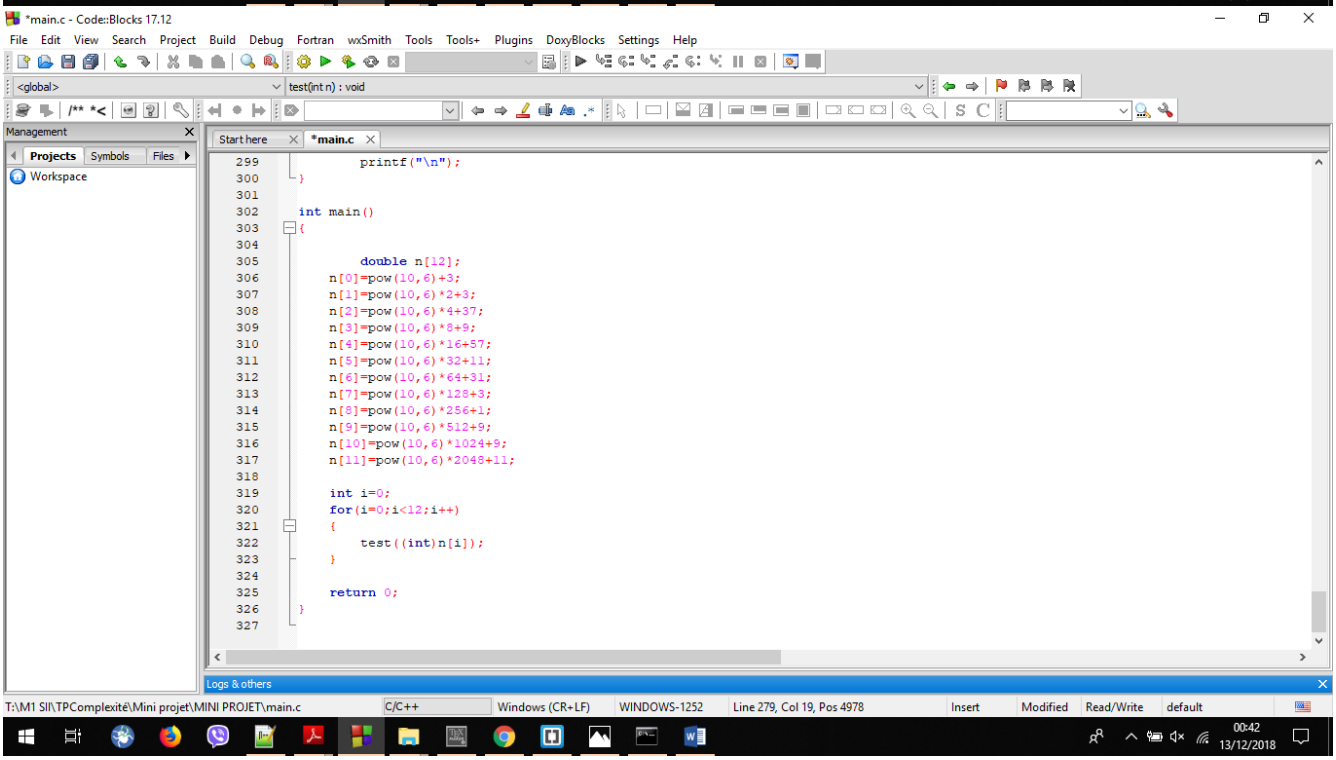
```
for(i=0;i<n;i++)
{

    tab[i]=rand()%n;
}
print_array(tab,n);
start=clock();
tri_insertion(tab,n);
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
printf("TRI INSERTION  : %fs\n");
print_array(tab,n);
printf("\n");


for(i=0;i<n;i++)
{

    tab[i]=rand()%n;
}
print_array(tab,n);
start=clock();
tri_bulles(tab,n);
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
printf("TRI BULLES  : %fs\n");
print_array(tab,n);
printf("\n");


for(i=0;i<n;i++)
{

    tab[i]=rand()%n;
}
print_array(tab,n);
start=clock();
tri_fusion(tab,0,n-1);
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
printf("TRI FUSION  : %fs\n");
print_array(tab,n);
printf("\n");

for(i=0;i<n;i++)
{
    tab[i]=rand()%n;
}
print_array(tab,n);
start=clock();
tri_rapide(tab,0,n-1);
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
printf("TRI RAPIDE  : %fs\n");
print_array(tab,n);
printf("\n");
```

```
    for(i=0;i<n;i++)
    {

        tab[i]=rand()%n;
    }
    print_array(tab,n);
    start=clock();
    tri_par_tas(tab,n);
    end = clock();
    time=(double)(end-start)/CLOCKS_PER_SEC;
    printf("TRI TAS  : %fs\n");
        print_array(tab,n);
        printf("\n");
}

int main()
{

     double n[12];
n[0]=pow(10,6)+3;
n[1]=pow(10,6)*2+3;
n[2]=pow(10,6)*4+37;
n[3]=pow(10,6)*8+9;
n[4]=pow(10,6)*16+57;
n[5]=pow(10,6)*32+11;
n[6]=pow(10,6)*64+31;
n[7]=pow(10,6)*128+3;
n[8]=pow(10,6)*256+1;
n[9]=pow(10,6)*512+9;
n[10]=pow(10,6)*1024+9;
n[11]=pow(10,6)*2048+11;

    int i=0;
    for(i=0;i<12;i++)
    {
        test((int)n[i]);
    }

    return 0;
}
```


*main.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
test(int n) : void
Management
Projects Symbols Files
Workspace
Start here
*main.c
221 void test(int n)
222 {
223 printf("<---------- TAILLE TABLEAU : %d---------->\n",n);
224 clock_t start,end;
225 time_t t;
226 srand((unsigned) time(&t));
227 double time;
228 int tab[n];
229 int i=0;
230 for(i=0;i<n;i++)
231 {
232
233 tab[i]=rand()%n;
234 }
235 print_array(tab,n);
236 start=clock();
237 tri_insertion(tab,n);
238 end = clock();
239 time=(double)(end-start)/CLOCKS_PER_SEC;
240 printf("TRI INSERTION : %fs\n");
241 print_array(tab,n);
242 printf("\n");
243
244
245 for(i=0;i<n;i++)
246 {
247
248 tab[i]=rand()%n;
249 }
250 print array(tab,n);
Logs & others
T:\M1 SI\TPComplexité\Mini projet\MINI PROJET\main.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 279, Col 19, Pos 4978
Insert
Modified
Read/Write
default
00:41
13/12/2018


*main.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
test(int n) : void
Management
Projects Symbols Files
Workspace
Start here
*main.c
245 for(i=0;i<n;i++)
246 {
247
248 tab[i]=rand()%n;
249 }
250 print_array(tab,n);
251 start=clock();
252 tri_bulles(tab,n);
253 end = clock();
254 time=(double)(end-start)/CLOCKS_PER_SEC;
255 printf("TRI BULLES : %fs\n");
256 print_array(tab,n);
257 printf("\n");
258
259
260 for(i=0;i<n;i++)
261 {
262
263 tab[i]=rand()%n;
264 }
265 print_array(tab,n);
266 start=clock();
267 tri_fusion(tab,0,n-1);
268 end = clock();
269 time=(double)(end-start)/CLOCKS_PER_SEC;
270 printf("TRI FUSION : %fs\n");
271 print_array(tab,n);
272 printf("\n");
273
274 for(i=0;i<n;i++)
Logs & others
T:\M1 SI\TPComplexité\Mini projet\MINI PROJET\main.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 279, Col 19, Pos 4978
Insert
Modified
Read/Write
default
00:41
13/12/2018


*main.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
test(int n) : void
Management
Projects Symbols Files
Workspace
Start here
*main.c
printf("\n");
for(i=0;i<n;i++)
{
tab[i]=rand()%n;
}
print_array(tab,n);
start=clock();
tri_rapide(tab,0,n-1);
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
printf("TRI RAPIDE : %fs\n");
print_array(tab,n);
printf("\n");
for(i=0;i<n;i++)
{
tab[i]=rand()%n;
}
print_array(tab,n);
start=clock();
tri_par_tas(tab,n);
end = clock();
time=(double)(end-start)/CLOCKS_PER_SEC;
printf("TRI TAS : %fs\n");
print_array(tab,n);
printf("\n");
}
Logs & others
T:\M1 SI\TPComplexité\Mini projet\MINI PROJET\main.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 279, Col 19, Pos 4978
Insert
Modified
Read/Write
default
00:42
13/12/2018


*main.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
test(int n) : void
Management
Projects Symbols Files
Workspace
Start here
*main.c
printf("\n");
}
int main()
{
double n[12];
n[0]=pow(10,6)+3;
n[1]=pow(10,6)*2+3;
n[2]=pow(10,6)*4+37;
n[3]=pow(10,6)*8+9;
n[4]=pow(10,6)*16+57;
n[5]=pow(10,6)*32+11;
n[6]=pow(10,6)*64+31;
n[7]=pow(10,6)*128+3;
n[8]=pow(10,6)*256+1;
n[9]=pow(10,6)*512+9;
n[10]=pow(10,6)*1024+9;
n[11]=pow(10,6)*2048+11;
int i=0;
for(i=0;i<12;i++)
{
test((int)n[i]);
}
return 0;
}
Logs & others
T:\M1 SI\TPComplexité\Mini projet\MINI PROJET\main.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 279, Col 19, Pos 4978
Insert
Modified
Read/Write
default
00:42
13/12/2018