Conception et Complexité des Algorithmes

Rapport travaux pratiques n° 2

Les Algorithmes de Tri

Travail présenté par :

Membres de la team 13 :

BOUADI Nassima 191931012438

FERKOUS Sarah. 191931043867

MOKHTARI Mohamed Rayane. 191931069009

GUERBAS Thinhinane. 191933000894

2022/2023

Introduction générale

Le tri est l'opération qui consiste à présenter une suite d'éléments dans un ordre précis. On s'intéresse particulièrement au problème de tri car il est souvent requis dans l'écriture de nombreux algorithmes plus complexes dont certains algorithmes de recherche, comme la recherche dichotomique. C'est la raison pour laquelle , il devient important de développer des algorithmes efficaces pour le tri surtout lorsque le nombre d'objets à trier est très grand. Les algorithmes de tri élémentaire qui servent à trier le contenu en un ordre croissant : tri par sélection , tri par insertion , tri a bulles. D'autre part, il y a des algorithmes plus performants : tri par fusion , tri par tas , tri rapide. La classification des algorithmes de tri est très importante, car elle permet de choisir l'algorithme le plus adapté au problème traité, tout en tenant compte des contraintes imposées par celui-ci. Les principales caractéristiques qui permettent de différencier les algorithmes de tri, outre leur principe de fonctionnement, sont la complexité temporelle, la complexité spatiale et le caractère stable.

Table des matières

Table des figures

Liste des tableaux

Partie 1

Développement de l'algorithme et du programme correspondant

1.1 Question 01 :

Implémenter les algorithmes de tri suivants en langage C :

- Tri par sélection
- Tri par insertion
- Tri à bulle
- Tri rapide (implémentez trois méthodes de choix du pivot)
- Tri fusion
- Tri par tas

Les implémentations se retrouveront la partie annexe

Partie 2

Mesure du temps d'exécution

2.1 Question 01 :

Les pseudos code des algorithmes de tri :

- Tri par sélection
- Tri par insertion
- Tri à bulle
- Tri rapide (implémentez trois méthodes de choix du pivot)
- Tri fusion
- Tri par tas

2.1.1 Tri par sélection :

Les nombres à trier sont placés dans un tableau. L'algorithme est structuré en une boucle et a chaque itération de la boucle, le plus petit nombre parmi ceux qui reste à trier est sélectionnée et place au debut droite du tableau, le principe de cette algorithme :

> chercher le min parmi des nombres et l'échanger avec le nombre de la première case du tableau.

> chercher le min parmi le restant des nombres à trier et l'échanger avec le nombre qui se trouve dans la 2 ème case du tableau.

> continuer cette opération jusqu'au trie complet du tableau .

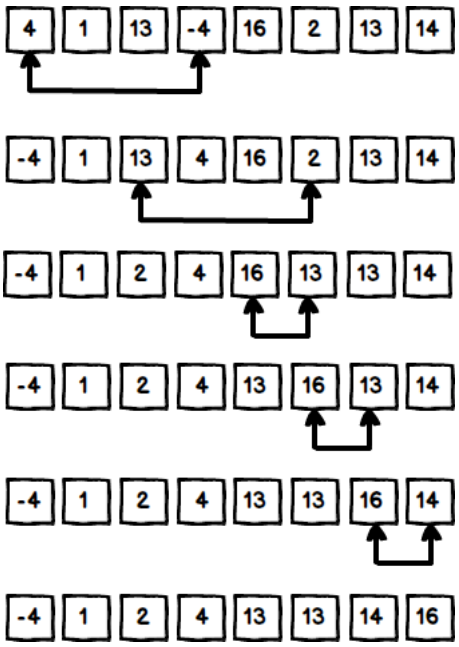


FIGURE 2.1 – Méthode tri par sélection

Complexité du tri par sélection :

Le temps pris par cet algorithme pour trier n éléments dépend de l'ordre initial des éléments. Le nombre de comparaisons effectuées entre éléments du tableau peut être une bonne mesure de la complexité de la plupart des algorithmes de tri. Dans ce cas de figure quelque soit la donnée , l'algorithme exécute entièrement les deux boucle POUR.

-> $O(n^2)$.

Pseudo code :

```
Procedure TriSelection(Tab : tableau d'entiers ,  N: entier)
Var
        i, j, min, temp : entiers;
Debut
        Pour i := 0 a n-1       (n iterations )
        Faire
                min :=i;
                        Pour j:=i+1     ((n-(i+1)+1) iterations)
                        Faire
                                Si (Tab[j]< Tab[min]) alors
                                        min := j;
                                        temp := Tab[i];
                                        Tab[i] := Tab[min];
                                        Tab[min] := temp;
                                        min := i;
                                FSI;
                        Fait;
        Fait;
Fin;
```

FIGURE 2.2 – Pseudo code du tri par sélection.

2.1.2 Tri par insertion :

Dans cette méthode on possède deux tableaux , un premier tableau Tab[1. . .i-1] trié par ordre croissant et le deuxième Tab[i. . ...n] non trier. L'algorithme consiste en deux boucles : dans la deuxième il s'agit

d'insérer le 1er élément du deuxième tableau Tab[i] a sa bonne position dans le premier tableau et veiller à garder ce dernier trier.

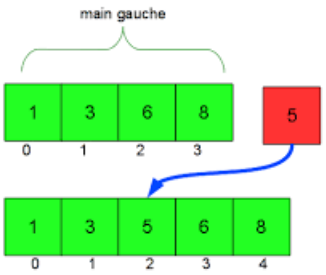


FIGURE 2.3 – Méthode tri par insertion

Complexité du tri par insertion :

Dans ce cas de figure quelque soit la donnée , l'algorithme exécute entièrement les deux boucle imbriquées .

-> $O(n^2)$.

Pseudo code :

```
Procedure TriInsertion( T :tableau d'entiers , n : entier)
        Var i, k, c, p : entiers ;
Debut
    |   Pour i:=1 à n
    |   Faire
    |       |   C :=T[i];
    |       |   P :=0;
    |       |   Tantque (T[p]<c)
    |       |   Faire
    |       |       |   P :=p+1 ;
    |       |   Fait;
    |       |   Pour k=i-1 à p pas k:=k-1
    |       |   Faire
    |       |       |   //on décale les nombres
    |       |       |   T[k+1]=T[k];
    |       |   Fait;
    |       |   T[p] :=c; //on écrit l'élément
    |   Fait ;
Fin
```

FIGURE 2.4 – Pseudo code du tri par sélection.

2.1.3 Tri par bulles :

Le principe du tri par bulles est de parcourir le tableau a trier et de comparer tous les éléments consécutifs deux à deux pour faire sortir comme une bulle , le plus grand élément. Ainsi le maximum est placé à l'extrémité droite du tableau.Le même processus pour les n-1 premiers éléments non triés et ainsi de suite jusqu'à trier tous les éléments du tableau par ordre croissant.

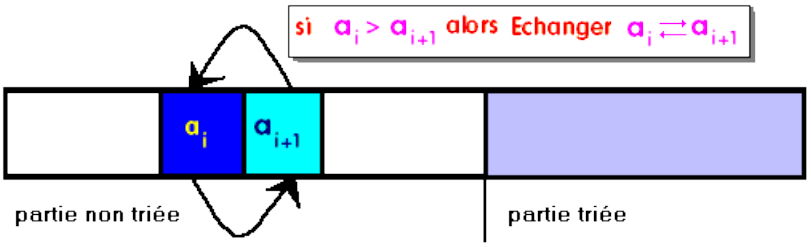


FIGURE 2.5 – Méthode tri par bulles

Complexité du tri par bulles :

Dans ce cas de figure , le pire cas se présente lorsque le plus petit élément est placé à la l'extrémité droite du tableau c'est-à-dire le dernier élément du tableau.

-> $O(n^2)$.

Pseudo code :

```
Procédure TriBulle( t :tableau d'entiers , n : entier)
    Var flag, i, c :entier ;
Debut
    flag :=1;
    Tantque (flag) faire
        flag : =0;
        i : =0;

        Tantque (i<n-1) faire (n fois)
            //si élément est supérieur au suivant
            Si (T[i]>T[i+1]) alors
                c=T[i]; //on échange
                T[i]=T[i+1]; //les deux
                T[i+1]=c; //nombres
                flag=1;
            Finsi ;
            i :=i+1 ;
        Fintq
    Fintq ;
Fin
```

FIGURE 2.6 – Pseudo code du tri par bulles.

2.1.4 Tri rapide :

Cette algorithme choisit d'abord un pivot parmi l'un des éléments du tableau a trier. Le tableau est ensuite partitionné autour du pivot : on déplace tous nos éléments pour faire en sorte que les éléments les plus petit que le pivot se trouve à gauche alors que les éléments les plus grand que le pivot se retrouveront a droite.Il suffit ensuite de faire un trie récursive sur les sous-tableaux se trouvant de part et d'autre du pivot pour avoir un tableau complètement trier.

Concrètement, pour faire le partitionnement un sous-tableau :

- le pivot est placé à la fin (arbitrairement), en échangeant avec le dernier élément du sous-tableau ;
- tous les éléments inférieurs au pivot sont placés en début du sous-tableau ;
- Le pivot est déplacé à la fin des éléments déplacés.

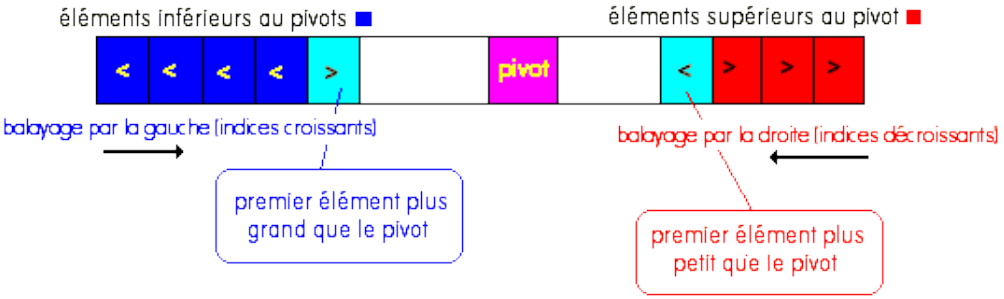


FIGURE 2.7 – Méthode du tri rapide.

Complexité du tri rapide :

Le temps mis pour trier une suite de nombres dépend de l'ordre de ces nombres dans la suite et le choix aléatoire des pivots .Le cas le plus défavorable survient lorsque le pivot choisi au hasard est le plus petit élément , et que cette situation se répète à chaque appel récursif.

-> $O(n^2)$.

Pseudo code :

```
Procedure TriRapide (E/S t : Tableau[1..MAX] d'Entier; gauche, droit : entier)
        var i, j ,pivot, x : Entiers
Début
        i  gauche;
        jdroit;
        pivotchoixpivot()// il existe plusieurs méthode pour choisir le pivot
        répéter
                Tant que (t[i] <pivot )
                        i = i+1
                fintant que
                Tant que (t[j] >pivot)
                        j=j-1
                 fintant que
                Si (i<=j) alors permutation(t[i] ,t[j]);
                        i =i+1 ;
                        j=j-1 ;
                finsi;
        jusqu'à  i> j ;
        Si (gauche < j ) alors TriRapide(t, gauche, j)
        finsi ;
        Si i<droit alors TriRapide(t, i, droit)
         finsi ;
Fin
```

FIGURE 2.8 – Pseudo code du tri rapide.

2.1.5 Tri fusion :

Le tri fusion,ou tri dichotomique, est un algorithme de tri par comparaison stable. Il suit le paradigme diviser pour régner, son principe est le suivant :

- On divise en deux moitiés le tableau à trier (en prenant par exemple , un élément sur deux pour chacun des tableaux)
- On trie chacun d'entre eux.
- On fusionne les deux moitiés obtenues pour reconstruire un tableau trié .

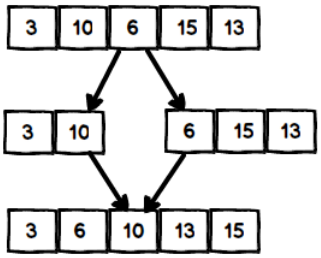


FIGURE 2.9 – Méthode tri par fusion

Complexité du tri fusion :

Etant donné que le processus récursif ne dépend pas de l'instance mais uniquement de sa taille n, donc dans notre cas la taille des données est évidemment le cardinal n du tableau à trier , il n'y a pas lieu de distinguer de meilleur ou de pire cas.

Si n=1 => l'algorithme est O(1) ;

Sinon Si n>1

Calcul de milieu O(1) ;

2 appels récursifs 2* T (n/2) ;

Fusion en O(n) ;

T(n) = O(1) si n=1 ;

2*T(n) + O(n) si n>1 ;

-> O(n log n) .

Pseudo code :

```
Procedure triFusion( i : entier, j :entier, tab[] :entier, tmp[] :entier) {
  Si(j <= i) Faire retourner fait;
   m , pg , pd : entier;
   m = (i + j) / 2;
  triFusion(i, m, tab, tmp);     //trier la moitié gauche récursivement
  triFusion(m + 1, j, tab, tmp); //trier la moitié droite récursivement
  pg = i;     //pg pointe au début du sous-tableau de gauche
  pd = m + 1; //pd pointe au début du sous-tableau de droite
  c : entier;     //compteur
// on boucle de i à j pour remplir chaque élément du tableau final fusionné
  pour c de i a j
Faire
    Si(pg == m + 1)  //le pointeur du sous-tableau de gauche a atteint la limite
    Faire
      tmp[c] = tab[pd];
      pd = pd+1;
```

FIGURE 2.10 – Pseudo code du tri par fusion.

```
    |
    Fait
    Sinon si (pd == j + 1)
          Faire //le pointeur du sous-tableau de droite a atteint la limite
            tmp[c] = tab[pg];
            pg = pg+1;
          Fait;
              sinon si (tab[pg] < tab[pd])
                    faire //le pointeur du sous-tableau de gauche pointe vers  un
              élément plus petit
                          tmp[c] = tab[pg];
                           pg=pg+1;
                    Fait;
```

FIGURE 2.11 – Pseudo code du tri par fusion.

```
                    sinon
                    Faire  //le pointeur du sous-tableau de droite pointe
                vers un élément plus petit
                        tmp[c] = tab[pd];
                        pd=pd+1;
                    fait;
  Fait;
pour c de a j
Faire //copier les éléments de tmp[] à tab[]
    tab[c] = tmp[c];
 Fait;
}
```

FIGURE 2.12 – Pseudo code du tri par fusion.

2.1.6 Tri par tas :

Pour trier un tableau donné , il faut d'abord construire un tas dans le but de récupérer le plus grand nombre du tableau au niveau de la racine. Le processus procède ensuite par la permutation du premier élément et le dernier élément du tableau. Le même processus est donc répété.

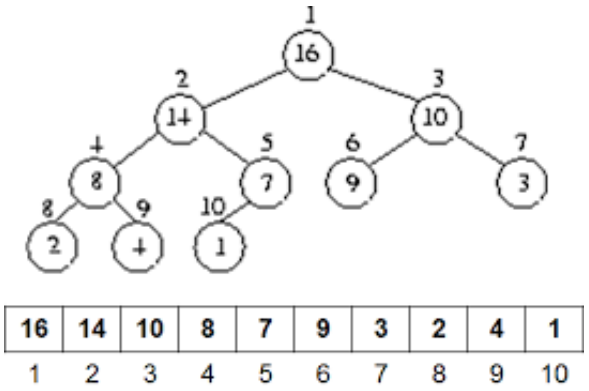


FIGURE 2.13 – Méthode tri par tas

Complexité du tri par tas :

Le tri par tas consiste à :

> transformer le tableau de n éléments en un tas : de complexité O(n)

> puis extraire les éléments un à un. O(n log n).

La complexité dans ce cas la est :

-> O(n log n) .

Pseudo code :

```
Procédure echanger(T : tableau d'entier ,i :entier,j : entier)
        Var echange:entier
Début
        Echange :=T[i]; //permutation pour rendre s fils inferieurs a leurs pere
        T[i] :=T[j];
        T[j] :=echange;
Fin
```

```
Procédure remonter (T : tableau d'entier ,n :entier,i : entier)
Debut
        Si (i=0) alors sortir finsi ;
        Si (T[i]>T[i/2]) alors
                //si la valeur du fils est inférieur on le remonte
                echanger (T, i, i/2);
                remonter (T, n, i/2);
        Finsi
Fin
```

FIGURE 2.14 – Pseudo code du tri par tas.

```
Procédure redescendre (T : tableau d'ntier ,n :entier,i : entier)
        Var imax : entier
Début
        //pour s'arrêter et mettre la plus grande valeur a la fin
        Si (2*i+1>=n) alors sortir Finsi ;
                Si (T[2*i+1]>T[2*i]) alors imax=2*i+1; // droite indice
                                        sinon imax :=2*i; //gauche
                        Si (T[imax]>T[i]) alors
                                echanger (T, imax, i);
                                redescendre (T, n, imax);
                        Finsi ;
                Finsi ;
        Finsi ;
Fin
```

FIGURE 2.15 – Pseudo code du tri par tas.

```
Procédure organiser (T : tableau d'entier ,n :entier)
        Var i : entier
Début
|       Pour i := 1 à n
|       |     remonter(T, n, i);
|       Fait ;
Fin
```

```
Procédure Tri_Arbre(T : tableau d'entier ,n :entier)
        Var i : entier
Début
|       organiser(T, n); i=n-1
|       Tantque (i>0) faire
|       |     echanger(T, 0, i); //la dernière valeur va être remplacé par la
|       |     //racine la plus grande valeur qui se trouve au début du tableau
|       |     redescendre(T, i, 0) ; i :=i-1 ;
|       Fintq
Fin
```

FIGURE 2.16 – Pseudo code du tri par tas.

2.2 Question 02 :

A quoi correspond le meilleur, moyen et pire cas pour chaque méthode de tri ? Justifiez

Algorithmes	Pire cas	Moyen cas	meilleure cas
Tri par Sélection	Le temps pris pour cette algorithme pour faire le tri dépend de l'ordre des éléments .Le nb de comparaison effectuées entre les éléments est une bonne unités de mesure ,quelque soit la donnée l'algorithme exécute entièrement des 2 boucles pour est donc $O(n^2)$.	Le temps requis pour le tri par sélection dépend de l'ordre original du tableau , le test Tab[j]< Tab[min] se fait de toute façon de même nombre de fois.La variation du temps n'est attribuable qu'au nombre de fois que les affection sont effectuées. Le temps est donc $O(n^2)$.	En ce qui concerne sa complexité, on dit que le tri par sélection est en $O(n^2)$., à la fois dans le meilleur des cas, en moyenne et dans le pire des cas, c'est-à-dire que son temps d'exécution est de l'ordre du carré du nombre d'éléments à trier
Tri par insertion	On considère une itération i de la boucle externe ,le pire cas se présente lorsque c est inférieur à Tab[j] pour chaque j entre 1 a i-1 alors comparer c avec Tab[i-2] Tab[i-3]....dans ce cas la boucle effectue i-1 comparaison est donc $O(n^2)$.	Afin de déterminer le temps utilisé par cet algorithme en moyenne, supposons que les n éléments à trier sont distincts et que chacune des permutation possible est équiprobable. dans la boucle interne pour une valeur i ,Tab[i] peut donc s'insérer n'importe où avec une proba de 1/i par rapport aux éléments Tab[1]... , le nb comparaison est égale respect a i,i-1.....,1 est donc le temps moyen est $O(n^2)$.	Dans le meilleur des cas, le tableau initial est trié et on effectue alors une comparaison à chaque insertion, on effectue donc N-1 comparaisons.
Tri a bulles	Dans le pire cas, le nombre d'itération de cet algorithme est n(n+1)/2 -1 est donc $O(n^2)$.	Dans le pire cas, les entiers du tableau sont initialement donnés dans l'ordre décroissant. Dans ce cas, on effectue l'échange à chaque comparaison, c'est-à-dire que le nombre d'itération de cet algorithme est n(n+1)/2 -1 est donc $O(n^2)$.	dans le meilleur des cas, le tableau initial est trié et il n'y a pas d'échange à faire ;

TABLE 2.1 – Pire cas , moyen cas , meilleur cas pour algorithmes de tri par sélection , insertion et par bulles.

Algorithmes	Pire cas	Moyen cas	meilleure cas
Tri rapide	Le pire cas est quand le pivot choisi aléatoirement soit le plus petit du tableau net que cette situation se répète d'une manière récursive est donc $O(n^2)$.	On suppose que les éléments S du tableau sont distincts, cette supposition va maximiser la taille de S1 et S3(sous ensemble de S) et donc le temps moyen de l'algorithme de tri rapide.	dans le meilleur des cas, en O(NlogN) ;
Tri fusion	La taille des données est évidemment le cardinal n du tableau à trier , il n'y a pas lieu de distinguer de meilleur ou de pire cas , le processus récursif ne dépend pas de l'instance mais uniquement de sa taille n .	La taille des données est évidemment le cardinal n du tableau à trier , il n'y a pas lieu de distinguer de meilleur ou de pire cas , le processus récursif ne dépend pas de l'instance mais uniquement de sa taille n .	On constate que la procédure de fusion nécessite un tableau intermédiaire aussi grand que le nombre d'éléments à interclasser. C'est là où réside le principal inconvénient du tri fusion, car si sa complexité dans tous les cas est en O(NlogN) ,
Tri par tas	La complexité du tri par tas dans le pire quand dans le moyen cas est égale à la complexité de la construction d'un tas qui est de l'ordre n log2(n) , la complexité de la boucle qui est également égale à n log2(n) elle est donc n log2(n)	La complexité du tri par tas dans le pire quand dans le moyen cas est égale à la complexité de la construction d'un tas qui est de l'ordre n log2(n) , la complexité de la boucle qui est également égale à n log2(n) elle est donc n log2(n)	La complexité du tri par tas dans le pire quand dans le moyen cas ou dans le meilleur est égale à la complexité de la construction d'un tas qui est de l'ordre n log2(n) , la complexité de la boucle qui est également égale à n log2(n) elle est donc n log2(n)

TABLE 2.2 – Pire cas , moyen cas , meilleur cas pour algorithmes de tri rapide , fusion et par tas.

2.3 Question 03 :

Mesurer les temps d'exécution de chaque algorithme avec des données pouvant se présenter en entrée selon 3 configurations :

> Les données du tableau sont triées en bon ordre.

> Les données du tableau sont triées en ordre inverse.

> Les données du tableau ne sont pas triées (c.à.d aléatoires).

2.3.1 Bon ordre :

Tri par sélection :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	33	563,000023	2203,999996	56558,99811	228468,9941	long	long	long	long

TABLE 2.3 – Temps d'éxécution du tri par selection bon ordre.

Tri par insertion :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	0	0	0	0	0	5	0	0	0

TABLE 2.4 – Temps d'éxécution du tri par insertion bon ordre.

Tri par bulles :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	12	470,999986	1802,000046	31016,00075	202733,9935	long	long	long	long

TABLE 2.5 – Temps d'éxécution du tri par bulles bon ordre.

Tri rapide :

Taille du tableau	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	0	0	3	7	50,000001	78,000002	2280.000210	5280.000210

TABLE 2.6 – Temps d'éxécution du tri rapide bon ordre.

Tri par fusion :

Taille du tableau	10^4	$5*10$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	0	5	17,999999	50,000001	126,000002	637,000024	1271,999955	14937.000275	24937.000275

TABLE 2.7 – Temps d'éxécution du tri par fusion bon ordre.

Tri par Tas :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	0	3	5	24	52,000001	326,000005	661,000013	11225.000381	31225.000381

TABLE 2.8 – Temps d'éxécution du tri par tas bon ordre.

2.3.2 Ordre inverse :

Tri par sélection :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	83,999999	2184,999943	8579,000473	220391,0065	8807501,4	long	long	long	long

TABLE 2.9 – Temps d'éxécution du tri par selection ordre inverse.

Tri par insertion :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	103	2512,000084	9887,000084	253740,9973	10178621	long	long	long	long

TABLE 2.10 – Temps d'éxécution du tri par insertion ordre inverse.

Tri par bulles :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	165,9999944	309,000015	17399,00017	443002,9907	long	long	long	long	long

TABLE 2.11 – Temps d'éxécution du tri par bulles ordre inverse.

Tri rapide :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	0	2	3	17,000001	32,000002	239,999995	397,000015	2259,000063	5408.999920

TABLE 2.12 – Temps d'éxécution du tri rapide ordre inverse.

Tri par fusion :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	2	9	18,999999	82,000002	182,999998	981,000006	1899,000049	10142,0002	23927.999496

TABLE 2.13 – Temps d'éxécution du tri par fusion ordre inverse.

Tri par Tas :

Taille du tableau	10^4	5*10	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	1	6	14	71,999997	164,000005	1019,000053	2137,000084	12373,00015	29860.000610

TABLE 2.14 – Temps d'éxécution du tri par tas ordre inverse.

2.3.3 Ordre Aléatoire :

Tri par sélection :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	104,999997	509,000003	9519,000053	55201,00021	975940,0024	long	long	long	long

TABLE 2.15 – Temps d'éxécution du tri par selection Ordre Aléatoire.

Tri par insertion :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	54,000001	294,999987	5202,000141	31066,99944	509996,0022	long	long	long	long

TABLE 2.16 – Temps d'éxécution du tri par insertion Ordre Aléatoire.

Tri par bulles :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	165,000007	2720,000029	24430,00031	285277,0081	long	long	long	long	long

TABLE 2.17 – Temps d'éxécution du tri par bulles Ordre Aléatoire.

Tri rapide :

Taille du tableau	10^4	$5*10^$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	0	1	7	21	79,000004	211,999997	455,000013	2535,000086	11739.000320

TABLE 2.18 – Temps d'éxécution du tri rapide Ordre Aléatoire.

Tri par fusion :

Taille du tableau	10^4	5*1	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	2	10	29,999999	93,999997	259,000003	908,999979	1807,999969	9253,00025	932223.999023

TABLE 2.19 – Temps d'éxécution du tri par fusion Ordre Aléatoire.

Tri par Tas :

Taille du tableau	10^4	$5*10^4$	10^5	$5*10^5$	10^6	$5*10^6$	10^7	$5*10^7$	10^8
Temps d'éxecution	1	0	20	55	219,999999	783,999979	1843,999982	13454,00047	59131.000519

TABLE 2.20 – Temps d'éxécution du tri par tas Ordre Aléatoire.

2.4 Question 04 :

Représenter ces mesures dans un tableau puis avec un graphe. Que pouvez-vous conclure ?

2.4.1 Représentation tabulaire :

Les representation dans un tableaux ont les a effectuer dans la question précédente,passons a la representation graphique :

2.4.2 Représentation graphique :

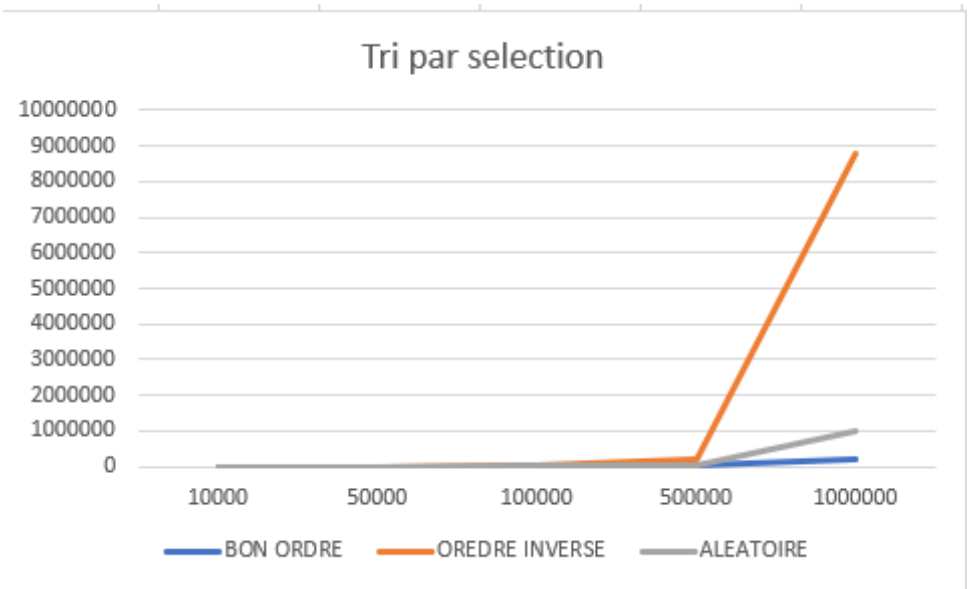


FIGURE 2.17 – Graphe de tri par sélection dans les 3 configurations demandées.

Analyse :

On remarque que dans le pire cas (orde inverse) le temps d'execution augmente trés vite car lors de la recherche de la valeur minimum du tableau l'instruction « min ← j » est presque exécutée pour chaque itération de la boucle de recherche car à chaque i++
T[i] <= T[min] et aussi dans ce cas pour pouvoir inverser un tableau de taille n (dans le but de le trier dans le bon ordre) on effectue n permutations où n-1 dans le cas ou n est impair. Malgrés la stabilité des courbes « bon ordre » et « aléatoire » les temps d'execution sont assez longs pour un tri car il effectue [n*(n-1)] / 2 comparaisons.

Conclusion :

Le tri par selection est un algorithme simple, mais considéré comme inefficace car il s'exécute en temps quadratique en le nombre d'éléments à trier, et non en temps pseudo linéaire.

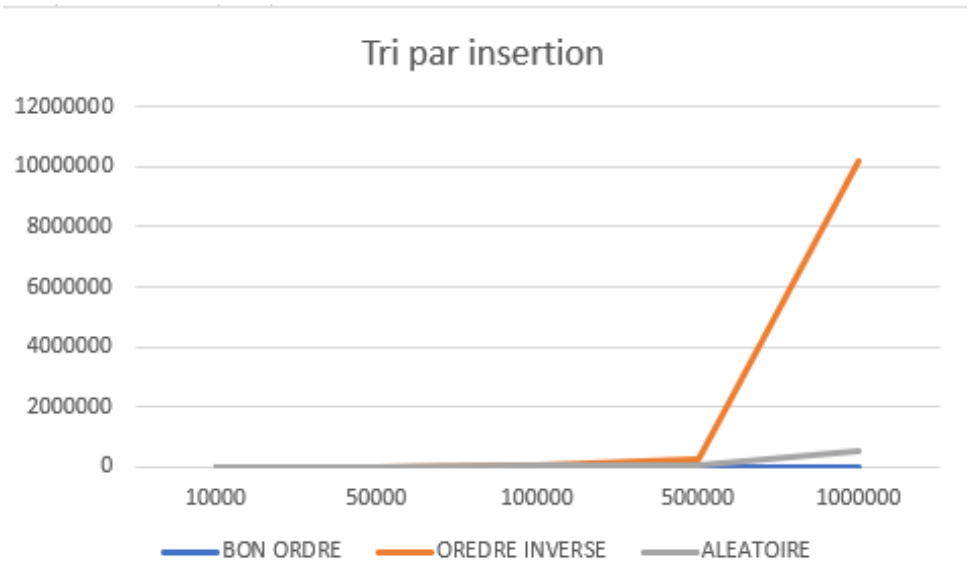


FIGURE 2.18 – Graphe de tri par insertion dans les 3 configurations demandées.

Analyse :

On remarque que le temps d'execution dans le pire cas augmente de façon exponentielle pour de grandes séquences de données et que le temps d'execution est rapide pour des séquences de données à moitié trié (aléatoire) et trié (bon ordre).

Conclusion :

Le tri par insertion est l'algorithme le plus efficace sur des entrées de petite taille. Il est aussi efficace lorsque les données sont déjà presque triées. Par contre le tri par insertion est beaucoup plus lent que d'autres algorithmes comme le tri rapide (ou quicksort) et le tri fusion pour traiter de grandes séquences, car sa complexité asymptotique est quadratique.

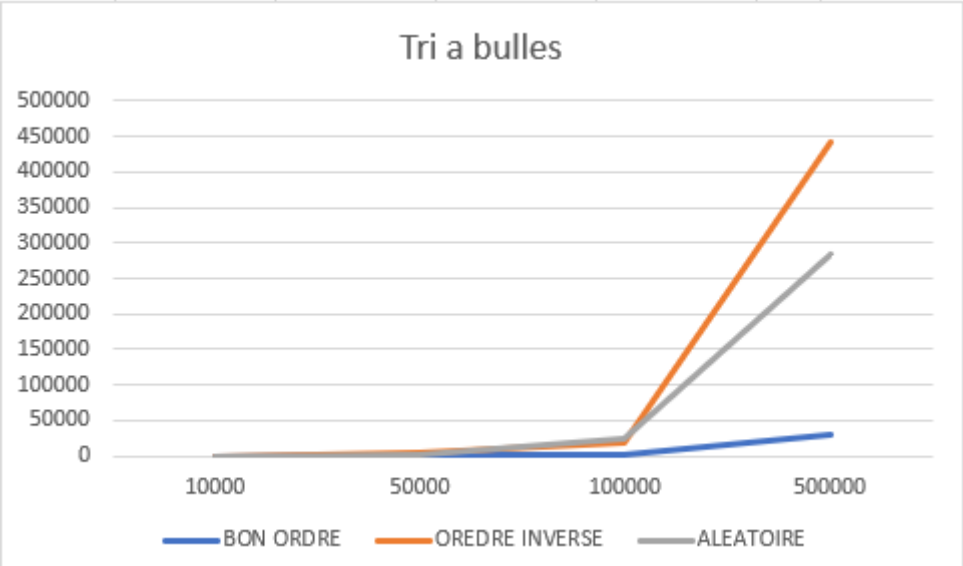


FIGURE 2.19 – Graphe de tri par bulles dans les 3 configurations demandées.

Analyse :

Le temps d'execution de cet algorithme augmente très vite que ce soit pour une séquence de données aléatoire ou trié dans l'ordre inverse. Il n'est cependant pas trop long pour une séquence de données qui est déjà trié meme si on observe une légère augmentation pour de grandes séquences de données.

Conclusion :

C'est l'algorithme le plus long des algorithmes communément enseigné ce qui explique pourquoi il n'est pas utilisé en pratique.

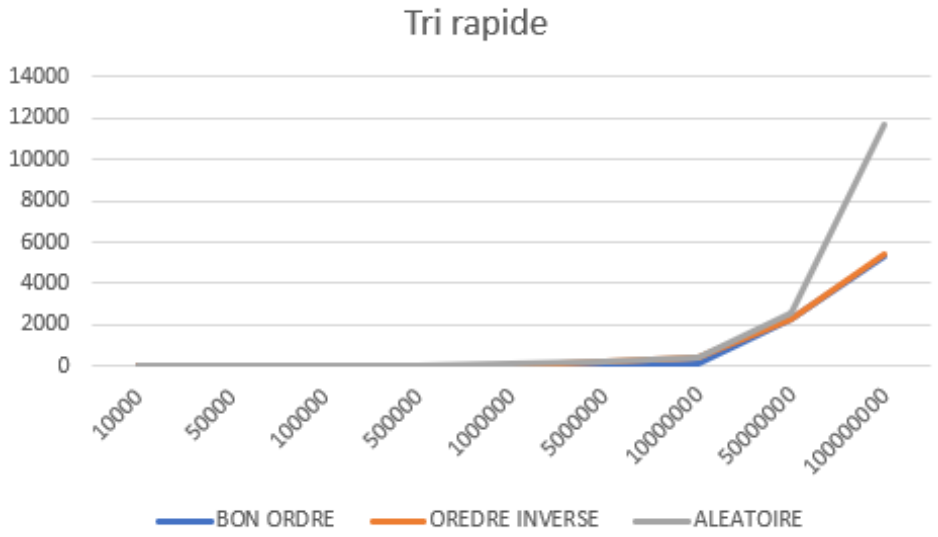


FIGURE 2.20 – Graphe de tri par rapide dans les 3 configurations demandées.

Analyse :

On remarque que les courbes « ordre inverse » et « aléatoire » sont pratiquement similaire et que la courbe « bon ordre » est assez basse et commence à augmenter quand la taille $n>=10^7$

Conclusion :

Le tri rapide ne tire pas avantage du fait que l'entrée est déjà presque triée. Dans ce cas particulier, il est plus avantageux d'utiliser le tri par insertion par exemple. Malgré ce désavantage théorique, c'est en pratique un des tris les plus rapides.

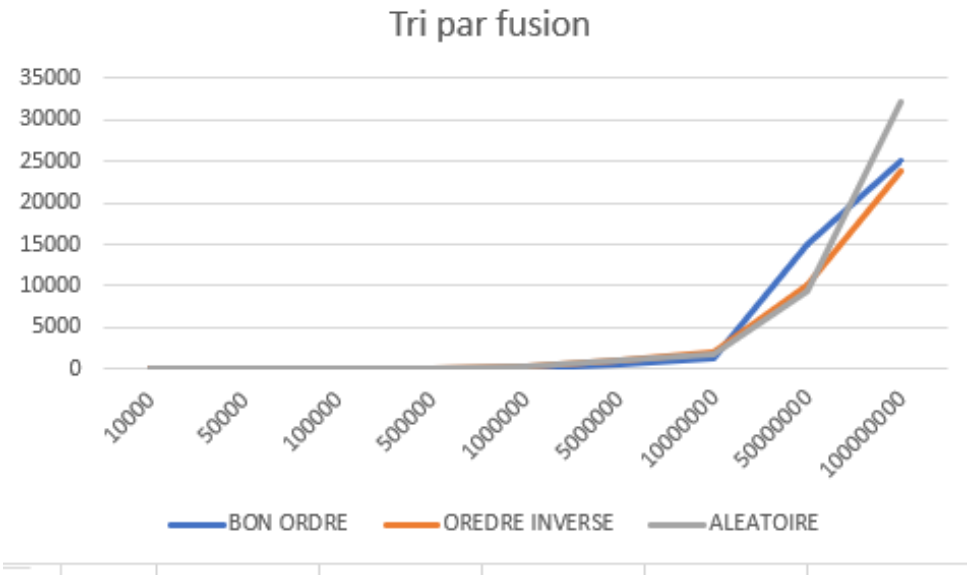


FIGURE 2.21 – Graphe de tri par fusion dans les 3 configurations demandées.

Analyse :

On remarque à partir du graphe que les trois courbes augmentent de façon exponentielle à partir de taille du tableau égal à 10^7.

Conclusion :

Le tri fusion sur les tableaux a une efficacité comparable au tri rapide, mais elle n'opère pas en place elle fait appel à une zone temporaire de données qui impacte sur la complexité spatiale de la solution. Sur les listes par contre, sa complexité serait optimale, il s'implémenterait très simplement et ne requerrait pas de copie en mémoire temporaire.

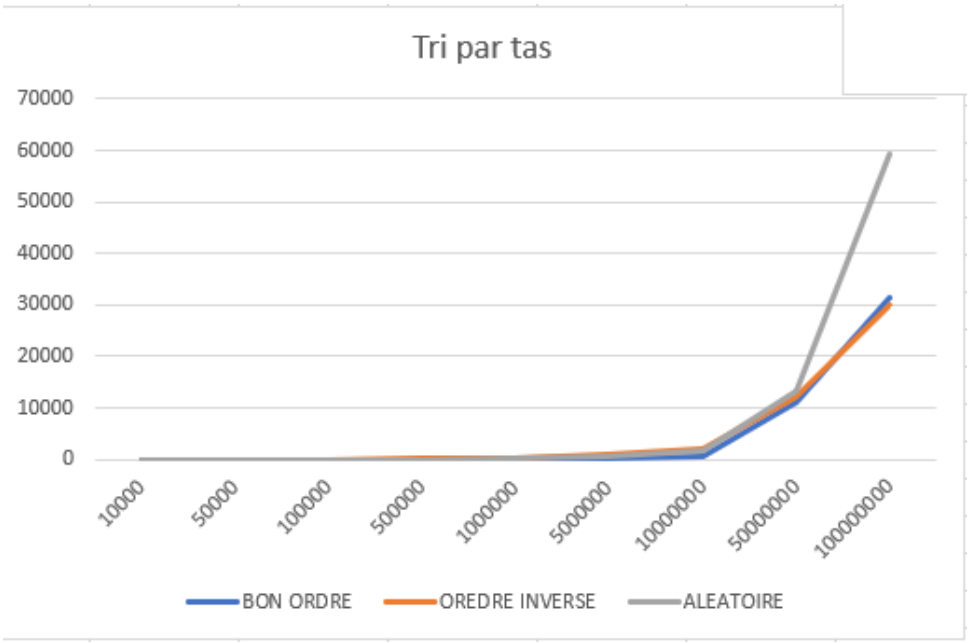


FIGURE 2.22 – Graphe de tri par tas dans les 3 configurations demandées.

Analyse :

On remarque que,
la courbe «bon ordre » augmente légèrement.
la courbe « ordre inverse » augmente vite.
la courbe « aléatoire » augmente de façon exponentielle.

Conclusion :

L'inconvénient majeur de cet algorithme est sa lenteur comparé au tri rapide (qui est en moyenne deux fois plus rapide sur un tableau de taille importante, il sera amené à traiter un nombre élevé d'emplacements mémoire dont l'éloignement peut dépasser la capacité du cache, ce qui ralentit l'accès à la mémoire et l'exécution de l'algorithme.

2.5 Question 05 :

Modifier les algorithmes dans la partie I pour qu'ils renvoient le nombre de comparaisons d'éléments du tableau effectués. Représentez les résultats dans un tableau puis avec un graphe. Que pouvez-vous conclure ?

2.5.1 Représentation tabulaire :

	Tri par selection	tri par insertion	tri par bulles	tri par tas	tri par fusion	tri rapide
Bon ordre	10000	25161126	100000000	244460	100277231	200287231
Ordre inversé	50005000	49995000	49995000	15965	267232	49995000
Ordre aléatoire	50005000	25161126	49995000	235252	267232	173935

TABLE 2.21 – Representation tabulaire pour le calcule de nb comparaison (taille 10^4.

	Tri par selection	tri par insertion	tri par bulles	tri par tas	tri par fusion	tri rapide
Bon ordre	50000	628335367	1249975000	1455438	Espace de stockage insuffisant dans la pile	Espace de stockage insuffisant dans la pile
Ordre inversé	1250025000	1249975000	1249975000	1366047	1568928	293720625
Ordre aléatoire	1250025000	628335367	1249975000	1409952	1568928	908690

TABLE 2.22 – Representation tabulaire pour le calcule de nb comparaison (taille $5 * 10^4$.

	Tri par selection	tri par insertion	tri par bulles	tri par tas	tri par fusion	tri rapide
Bon ordre	100000	Espace de stockage insuffisant dans la pile	704982704	3112517	Espace de stockage insuffisant dans la pile	Espace de stockage insuffisant dans la pile
Ordre inversé	705082704	704982704	704982704	2926640	3337856	Espace de stockage insuffisant dans la pile
Ordre aléatoire	705082704	Espace de stockage insuffisant dans la pile	704982704	3019280	3337856	2004507

TABLE 2.23 – Representation tabulaire pour le calcule de nb comparaison (taille 10^5.

2.5.2 Représentation graphique :

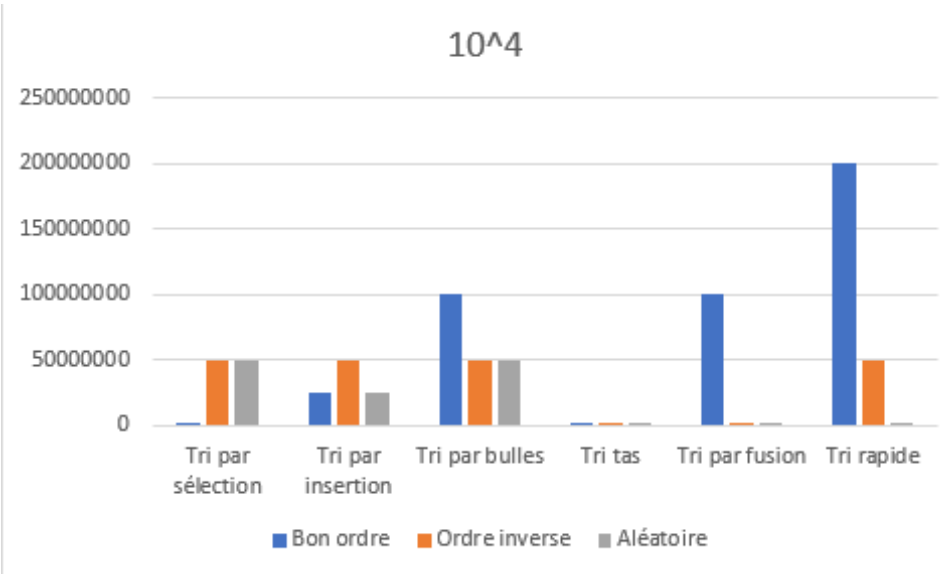


FIGURE 2.23 – Représentation graphique des algorithmes pour donner de nb de comparaison (taille 10^4)

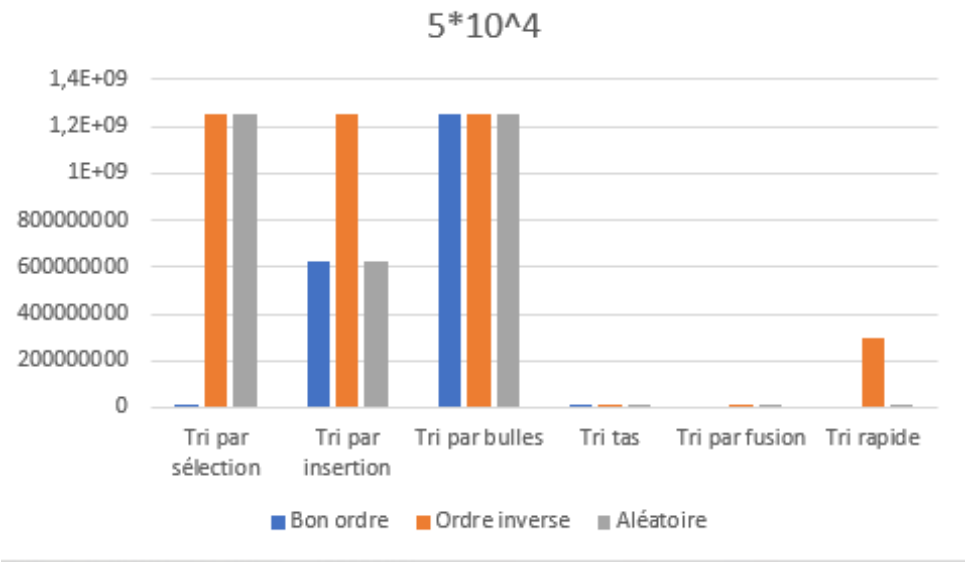


FIGURE 2.24 – Représentation graphique des algorithmes pour donner de nb de comparaison (taille $5 * 10^4$)

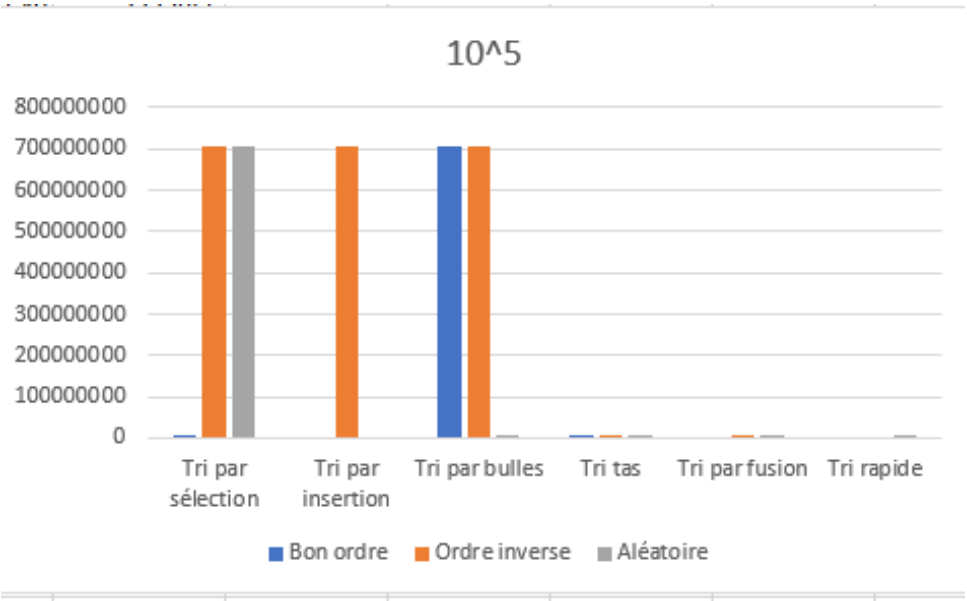


FIGURE 2.25 – Représentation graphique des algorithmes pour donner de nb de comparaison (taille 10^5)

Remarque :

Il n'y a pas de valeures nulles sur le graphes seulement les valeurs sont beaucoup plus petites que celles qui sont en haut.

Analyse :

On remarque à partir des graphes que l'algorithme qui effectue le moins de comparaisons pour :
la configuration bon ordre est le tri par selection, le tri par bulles, le tri par tas et le tri fusion,
la configuration ordre inverse est le tri par insertion et le tris par tas,
la configuration aléatoire est le tri par tas, le tri rapide et le tri fusion.
Le tri par tas effectue un nombre de comparaison qui est relativement bas sous les 3 configurations.
Le tri rapide et le tri fusion son meilleurs pour des instances de données aléatoires.
On remarque aussi que selon la configuration de la séquence de données en entrée l'algorithme de tri est plus ou moins efficace. Par exemple pour le tri par selection et le tri par bulles ils sont plus efficaces sous la configuration «bon ordre» et le sont moins sous la configuration «ordre inverse» et quant au tri par insertion c'est le contraire, par contre le tri par tas est meilleur pour les trois configurations.

Conclusion :

L'algorithme de tri par tas est l'algorithme qui effectue le moins de comparaison on en déduit donc qu'il est plus efficace que les quatres autres algorithmes de tri (par selection, par insertion, par bulles).
Les algorithmes de tri fusion et de tri rapide sont plus efficaces avec des instances de données non trié.

Partie 3

Environnement expérimental

Caractéristiques de la machine	BOUADI Nassima	FERKOUS Sarah	MOKHTARI Mohamed Rayane	GUERBAS Tinhinane
Marque	DELL i3	DELL i5	HP G5	DELL i7
Système d'exploitation	Système d'exploitation 64 bits, processeur x64.Windows 10 Professionnel.	Système d'exploitation 64 bits, processeur x64.Windows 10 Professionnel.	Système d'exploitation 64 bits, processeur x64.Windows 10 Professionnel.	Système d'exploitation 64 bits, processeur x64Windows 10 Pro N.
Processeur	Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz 2.00 GHz	Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz 2.70 GHz	Intel® Core(TM) I3-6006U CPU 2.00GHz	Intel(R) Core(TM) i7-4610M CPU @ 3.00GHz 3.00 GHz
Mémoire Ram installée	4,00 Go	8,00 Go	4,00 Go	4,00 Go

Table 3.1 – Description de l'environnement expérimental.

Version langage de programmation	BOUADI Nassima	FERKOUS Sarah	MOKHTARI Mohamed Rayane	GUERBAS Tinhinane
	code block 17.12	code blocks 20.03	code block 17.12	dev c++ 5.11

Table 3.2 – Version de langage de programmation.

Annexes

Code source des algorithmes de tri :

```
7
8   /* run this program using the console pauser or add your own getch, system("pause") or input loop */
9
10
11  //************************ Algo 1 - Tris par selection ***************************************************************************************
12  //on cherche le min a droite
13  void Tri_par_selection(int *Tab, int n){
14      int i, min, j, temp, cmpt=0;
15      for(i=0; i<n-1; i++){
16          min = i;
17          for(j=i+1; j<n; j++){
18              if(Tab[j] < Tab[min]){
19                  cmpt++;
20                  min = j;
21                  temp = Tab[i];
22                  Tab[i] =Tab[min];
23                  Tab[min] = temp;
24                  min = i;
25              }
26
27          }
28      }
29      printf("\t%d\t",cmpt);
30  }
31
```

FIGURE 3.1 – Algorithme de tri par selection.

```
31
32  //************************ Algo 2 - par insertion ********************************************************************************************
33  //parcourir 2 tab: tab trie et l'autre non
34  void tri_insertion(int *T, int n)
35  {
36      int i, k, c, p, cmpt=0;
37      for(i=1; i < n; i++)
38      {
39          c = T[i];
40          p = 0;
41          while (T[p] < c){
42              // on cherche la position p que l' on va affecter à l'élément
43              cmpt++;
44              p++;
45          }
46          for (k=i-1; k>=p; k--){
47              // on décale les nombres
48              T[k+1] = T[k];
49          }
50          // on écrit l'élément
51          T[p] = c ;
52      }
53      printf("\t%d\t",cmpt);
54  }
55
```

FIGURE 3.2 – Algorithme de tri par insertion.

```c
//************************ Algo 3 - Tris par bulle ***************************************************************************************
//comparaison 2 a 2 consecutif
void tri_bulle(int *T,int n)
{
    int flag, i, c, cmpt=0;
    flag=1;
    while (flag)
    {
        //tant qu'on modifie le tableau
        flag=0; i=0;
        while (i<n-1)
        {
            if(T[i]>T[i+1])
            { //si élément est supérieur au suivant on échange les deux nombres
                cmpt++;
                c=T[i];
                T[i]=T[i+1];
                T[i+1]=c;
                flag=1;
            }
            i++;
        }
    }
    printf("\t%d\t",cmpt);
}

```

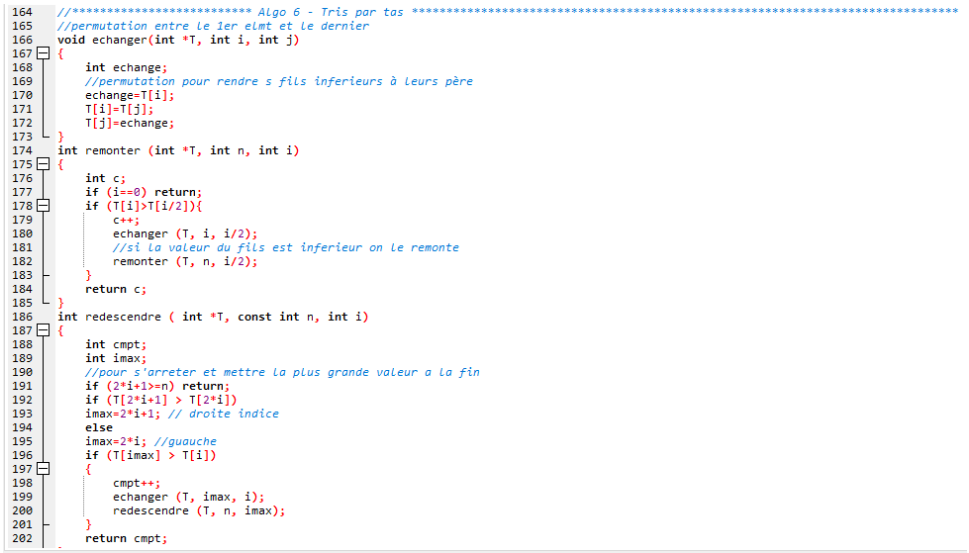
FIGURE 3.3 – Algorithme de tri par bulle.

```c
//************************ Algo 4 - Tris rapide ***************************************************************************************
//choisir un pivot : plus petit de pivot a G
void changer(int *T, int i, int j)
{
    int echange;
    echange=T[i];
    T[i]=T[j];
    T[j]=echange;
}
int partition(int*tableau, int deb, int fin)
{
    int compt=deb;
    int pivot=tableau[deb];
    int i;
    for(i=deb+1;i<=fin;i++){
        if(tableau[i]<pivot){
            compt++;
            changer (tableau, compt, i);
        }
    }
    changer (tableau, compt, deb) ;
    return (compt) ;
}

int tri_rapide_bis(int *tableau,int debut,int fin)
{
    int pivot = 0;
    if (debut<fin){
        pivot=partition(tableau, debut, fin);
        tri_rapide_bis(tableau, debut, pivot-1);
        tri_rapide_bis (tableau, pivot+1, fin);
    }
    return pivot;

}
void tri_rapide(int *tableau,int n)
{
    int cmpt = tri_rapide_bis(tableau,0,n-1);
    printf("\t%d\t",cmpt);
```

FIGURE 3.4 – Algorithme de tri rapide.

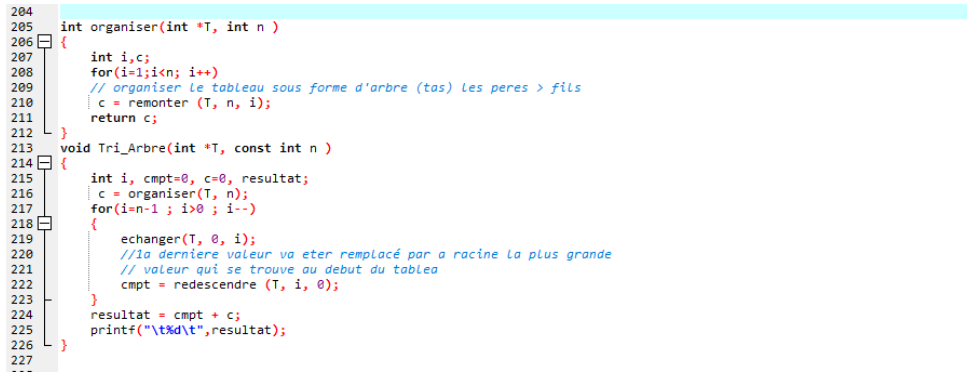
```c
//************************** Algo 5 - Tris par fusion ***************************************************************************************
//dichotomique on divise le tab en 2, on tri chaqu'un d'entre eux apr on fusionne
int triFusion(int i, int j, int tab[], int tmp[]) {
    if(j <= i){ return;}

    int m = (i + j) / 2;

    triFusion(i, m, tab, tmp); //trier la moitié gauche récursivement
    triFusion(m + 1, j, tab, tmp); //trier la moitié droite récursivement
    int pg = i; //pg pointe au début du sous-tableau de gauche
    int pd = m + 1; //pd pointe au début du sous-tableau de droite
    int c; //compteur
// on boucle de i à j pour remplir chaque élément du tableau final fusionné
    for(c = i; c <= j; c++) {
        if(pg == m + 1) { //le pointeur du sous-tableau de gauche a atteint la limite
            cmpt++;
            tmp[c] = tab[pd];
            pd++;
        }else if (pd == j + 1) { //le pointeur du sous-tableau de droite a atteint la limite
            cmpt++;
            tmp[c] = tab[pg];
            pg++;
        }else if (tab[pg] < tab[pd]) { //le pointeur du sous-tableau de gauche pointe vers un élément plus petit
            cmpt++;
            tmp[c] = tab[pg];
            pg++;
        }else { //le pointeur du sous-tableau de droite pointe vers un élément plus petit
            tmp[c] = tab[pd];
            pd++;
        }
    }
    for(c = i; c <= j; c++) { //copier les éléments de tmp[] à tab[]
        tab[c] = tmp[c];
    }
    return cmpt;
}
```

FIGURE 3.5 – Algorithme de tri par fusion.



```c
//************************** Algo 6 - Tris par tas ***************************************************************************************
//permutation entre le 1er elmt et le dernier
void echanger(int *T, int i, int j)
{
    int echange;
    //permutation pour rendre s fils inferieurs à leurs père
    echange=T[i];
    T[i]=T[j];
    T[j]=echange;
}
int remonter (int *T, int n, int i)
{
    int c;
    if (i==0) return;
    if (T[i]>T[i/2]){
        c++;
        echanger (T, i, i/2);
        //si la valeur du fils est inferieur on le remonte
        remonter (T, n, i/2);
    }
    return c;
}
int redescendre ( int *T, const int n, int i)
{
    int cmpt;
    int imax;
    //pour s'arreter et mettre la plus grande valeur a la fin
    if (2*i+1>=n) return;
    if (T[2*i+1] > T[2*i])
    imax=2*i+1; // droite indice
    else
    imax=2*i; //guauche
    if (T[imax] > T[i])
    {
        cmpt++;
        echanger (T, imax, i);
        redescendre (T, n, imax);
    }
    return cmpt;
```

FIGURE 3.6 – Algorithme de tri par tas.



```c

int organiser(int *T, int n )
{
    int i,c;
    for(i=1;i<n; i++)
    // organiser le tableau sous forme d'arbre (tas) les peres > fils
     c = remonter (T, n, i);
    return c;
}
void Tri_Arbre(int *T, const int n )
{
    int i, cmpt=0, c=0, resultat;
     c = organiser(T, n);
    for(i=n-1 ; i>0 ; i--)
    {
        echanger(T, 0, i);
        //la derniere valeur va eter remplacé par a racine la plus grande
        // valeur qui se trouve au debut du tablea
        cmpt = redescendre (T, i, 0);
    }
    resultat = cmpt + c;
    printf("\t%d\t",resultat);
}

```

FIGURE 3.7 – Algorithme de tri par tas.

Code source du main :

```c
int main(int argc, char** argv){
    int   currentNumber;
    char   *endChar;

    char *buffer11 = malloc(sizeof(char) * 10000);
    char *buffer12 = malloc(sizeof(char) * 10000);
    char *buffer13 = malloc(sizeof(char) * 10000);

    int *tableu11 = malloc(sizeof(int) * 10000);
    int *tableu12 = malloc(sizeof(int) * 10000);
    int *tableu13 = malloc(sizeof(int) * 10000);

    int *tmp = malloc(sizeof(int) * 10000);

    int i =0, j=0, k=0;

    FILE *fp11, *fp12, *fp13;  //10000

    clock_t t11, t12, t21, t22, t31, t32;
    double delta1,delta2,delta3;
    double resulta[3][6];


    printf("\nLes resultats pour la taille de 10000\n\n");


        //ouverture fichier les données du tableau sont triées en bon ordre:
        fp11 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listebo1.txt","r");
        if(NULL == fp11)
        {
          return(1);
        }

        //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
        while(NULL != fgets(buffer11, 10000, fp11))
        {
         //printf("nouvelle ligne: \n");
          currentNumber = strtol(buffer11, &endChar, 10);
```

FIGURE 3.8 – Code source de la partie main

```c
          tableu11[i] = currentNumber;
          //on entre dans la ligne pour reccuperer les nbr
          while(*endChar != (char) NULL)
          {
             char* ptr = endChar+1;
             currentNumber = strtol(ptr, &endChar, 10);
             i++;
             tableu11[i] = currentNumber;
          }
        }

       //ouverture fichier les données du tableau sont triées en ordre inverse:
        fp12 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listeio1.txt","r");
        if(NULL == fp12)
        {
          return(1);
        }

        //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
        while(NULL != fgets(buffer12, 10000, fp12))
        {
         //printf("nouvelle ligne: \n");
          currentNumber = strtol(buffer12, &endChar, 10);
          tableu12[j] = currentNumber;
          //on entre dans la ligne pour reccuperer les nbr
          while(*endChar != (char) NULL)
          {
             char* ptr = endChar+1;
             currentNumber = strtol(ptr, &endChar, 10);
             j++;
             tableu12[j] = currentNumber;
          }
        }


       //ouverture fichier les données du tableau ne sont pas triées:
         fp13 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listent1.txt","r");
        if(NULL == fp13)
        {
```

FIGURE 3.9 – Code source de la partie main

```
298
299         //ouverture fichier les données du tableau ne sont pas triées:
300           fp13 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listent1.txt","r");
301         if(NULL == fp13)
302         {
303           return(1);
304         }
305
306         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
307         while(NULL != fgets(buffer13, 10000, fp13))
308         {
309          //printf("nouvelle ligne: \n");
310           currentNumber = strtol(buffer13, &endChar, 10);
311           tableu13[k] = currentNumber;
312          //on entre dans la ligne pour reccuperer les nbr
313          while(*endChar != (char) NULL)
314          {
315             char* ptr = endChar+1;
316             currentNumber = strtol(ptr, &endChar, 10);
317             k++;
318             tableu13[k] = currentNumber;
319          }
320         }
321
```

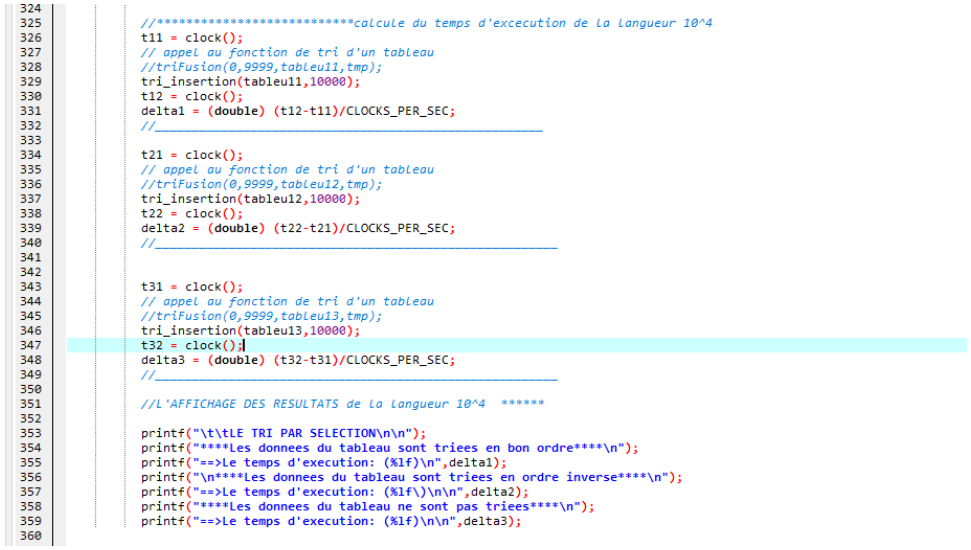
FIGURE 3.10 – Code source de la partie main

```
324
325         //**************************calcule du temps d'excecution de la langueur 10^4
326         t11 = clock();
327         // appel au fonction de tri d'un tableau
328         //triFusion(0,9999,tableu11,tmp);
329         Tri_par_selection(tableu11,10000);
330         t12 = clock();
331         delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
332         //_____________________________________________________
333
334         t21 = clock();
335         // appel au fonction de tri d'un tableau
336         //triFusion(0,9999,tableu12,tmp);
337         Tri_par_selection(tableu12,10000);
338         t22 = clock();
339         delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
340         //_____________________________________________________
341
342
343         t31 = clock();
344         // appel au fonction de tri d'un tableau
345         //triFusion(0,9999,tableu13,tmp);
346         Tri_par_selection(tableu13,10000);
347         t32 = clock();
348         delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
349         //_____________________________________________________
350
351         //L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******
352
353         printf("\t\tLE TRI PAR SELECTION\n\n");
354         printf("****Les donnees du tableau sont triees en bon ordre****\n");
355         printf("==>Le temps d'execution: (%lf)\n",delta1);
356         printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
357         printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
358         printf("****Les donnees du tableau ne sont pas triees****\n");
359         printf("==>Le temps d'execution: (%lf)\n\n",delta3);
360
```

FIGURE 3.11 – Code source de la partie main

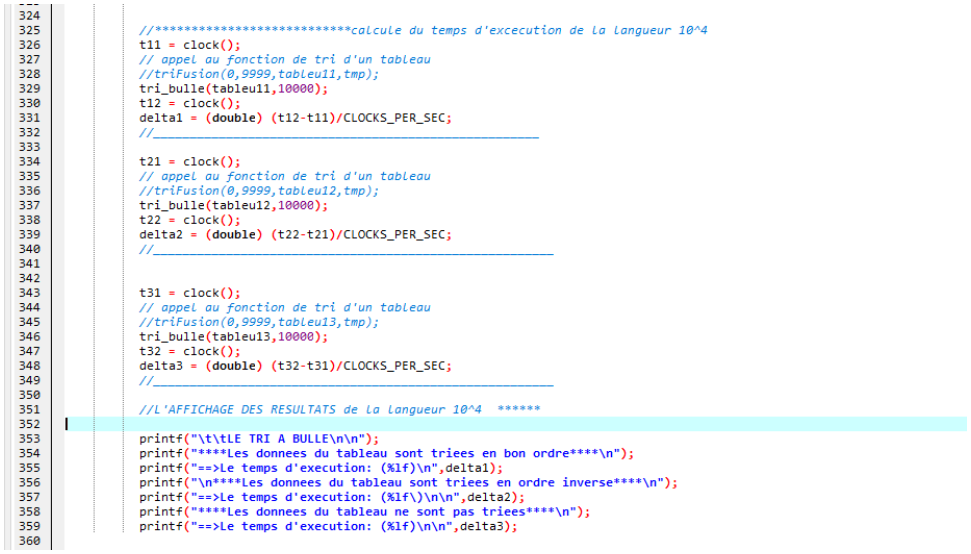
```
361
362     int rps;
363     printf("vous voullez afficher le tableau triee? choisisser 1 si c'est le cas.\t");
364     scanf("%d",&rps);
365     //l'affichage de tabkeau trie
366     if(rps == 1){
367         for(i=1;i<10000;i++){
368         printf("Le nombre courant = %d\n",tableu12[i]);
369     }
370
371   }
372   free(tableu11); free(buffer11);
373   free(tableu12); free(buffer12);
374   free(tableu13); free(buffer13);
375   free(tmp);
376
377
378
379
380     return 0;
381 }
```

FIGURE 3.12 – Code source de la partie main



```
//**************************calcule du temps d'excecution de la langueur 10^4
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(0,9999,tableu11,tmp);
tri_insertion(tableu11,10000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(0,9999,tableu12,tmp);
tri_insertion(tableu12,10000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(0,9999,tableu13,tmp);
tri_insertion(tableu13,10000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________________

//L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******

printf("\t\tLE TRI PAR SELECTION\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.13 – Code source de la partie main



```
//**************************calcule du temps d'excecution de la langueur 10^4
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(0,9999,tableu11,tmp);
tri_bulle(tableu11,10000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(0,9999,tableu12,tmp);
tri_bulle(tableu12,10000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(0,9999,tableu13,tmp);
tri_bulle(tableu13,10000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________________

//L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******

printf("\t\tLE TRI A BULLE\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

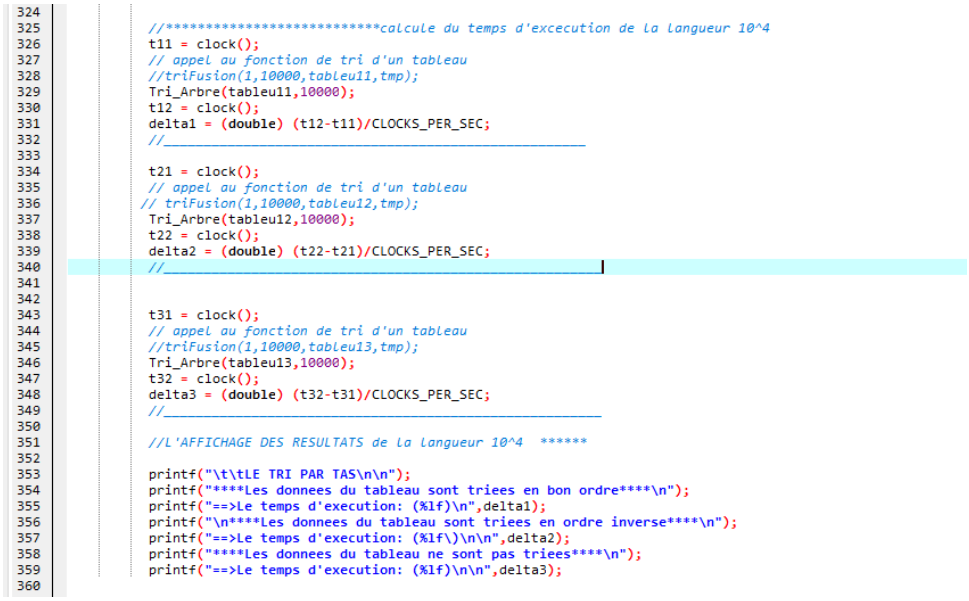
FIGURE 3.14 – Code source de la partie main

```
324
325         //**************************calcule du temps d'excecution de la langueur 10^4
326         t11 = clock();
327         // appel au fonction de tri d'un tableau
328         //triFusion(0,9999,tableu11,tmp);
329         tri_rapide(tableu11,10000);
330         t12 = clock();
331         delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
332         //_____________________________________________________
333
334         t21 = clock();
335         // appel au fonction de tri d'un tableau
336         //triFusion(0,9999,tableu12,tmp);
337         tri_rapide(tableu12,10000);
338         t22 = clock();
339         delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
340         //_____________________________________________________
341
342
343         t31 = clock();
344         // appel au fonction de tri d'un tableau
345         //triFusion(0,9999,tableu13,tmp);
346         tri_rapide(tableu13,10000);
347         t32 = clock();
348         delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
349         //_____________________________________________________
350
351         //L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******
352
353         printf("\t\tLE TRI RAPIDE\n\n");
354         printf("****Les donnees du tableau sont triees en bon ordre****\n");
355         printf("==>Le temps d'execution: (%lf)\n",delta1);
356         printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
357         printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
358         printf("****Les donnees du tableau ne sont pas triees****\n");
359         printf("==>Le temps d'execution: (%lf)\n\n",delta3);
360
```

FIGURE 3.15 – Code source de la partie main

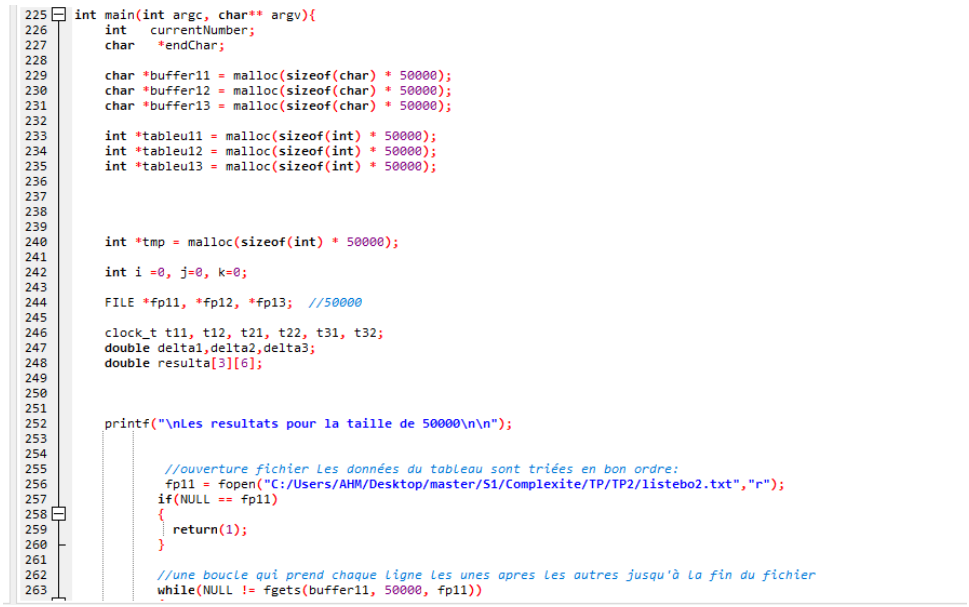
```
324
325         //**************************calcule du temps d'excecution de la langueur 10^4
326         t11 = clock();
327         // appel au fonction de tri d'un tableau
328         triFusion(1,10000,tableu11,tmp);
329         //tri_rapide(tableu11,10000);
330         t12 = clock();
331         delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
332         //_____________________________________________________
333
334         t21 = clock();
335         // appel au fonction de tri d'un tableau
336         triFusion(1,10000,tableu12,tmp);
337         //tri_rapide(tableu12,10000);
338         t22 = clock();
339         delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
340         //_____________________________________________________
341
342
343         t31 = clock();
344         // appel au fonction de tri d'un tableau
345         triFusion(1,10000,tableu13,tmp);
346         //tri_rapide(tableu13,10000);
347         t32 = clock();
348         delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
349         //_____________________________________________________
350
351         //L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******
352
353         printf("\t\tLE TRI PAR FUCSION\n\n");
354         printf("****Les donnees du tableau sont triees en bon ordre****\n");
355         printf("==>Le temps d'execution: (%lf)\n",delta1);
356         printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
357         printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
358         printf("****Les donnees du tableau ne sont pas triees****\n");
359         printf("==>Le temps d'execution: (%lf)\n\n",delta3);
360
```

FIGURE 3.16 – Code source de la partie main



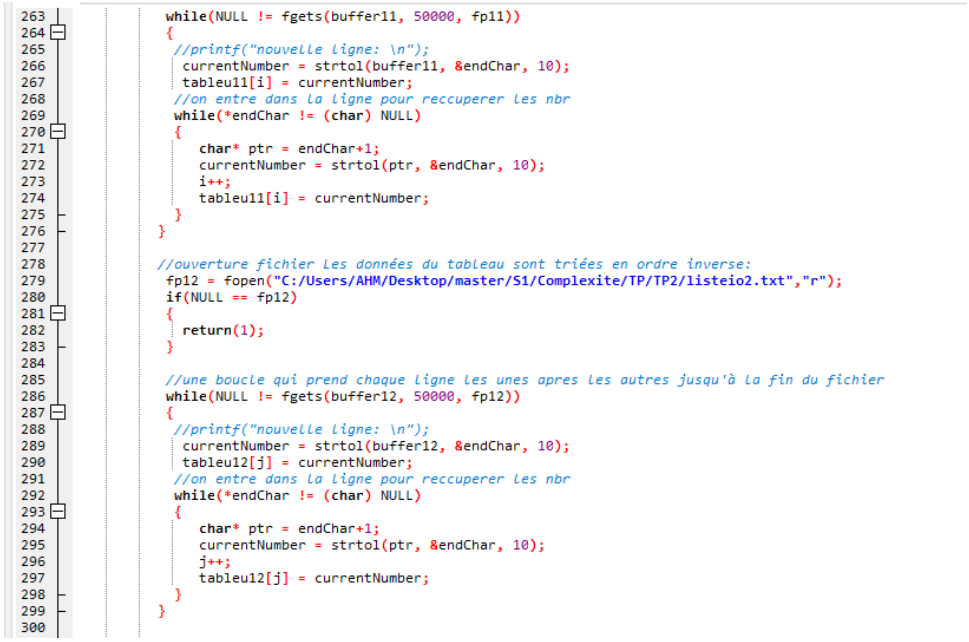
```
//**************************calcule du temps d'excecution de la langueur 10^4
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,10000,tableu11,tmp);
Tri_Arbre(tableu11,10000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
// triFusion(1,10000,tableu12,tmp);
Tri_Arbre(tableu12,10000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,10000,tableu13,tmp);
Tri_Arbre(tableu13,10000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________

//L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******

printf("\t\tLE TRI PAR TAS\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.17 – Code source de la partie main



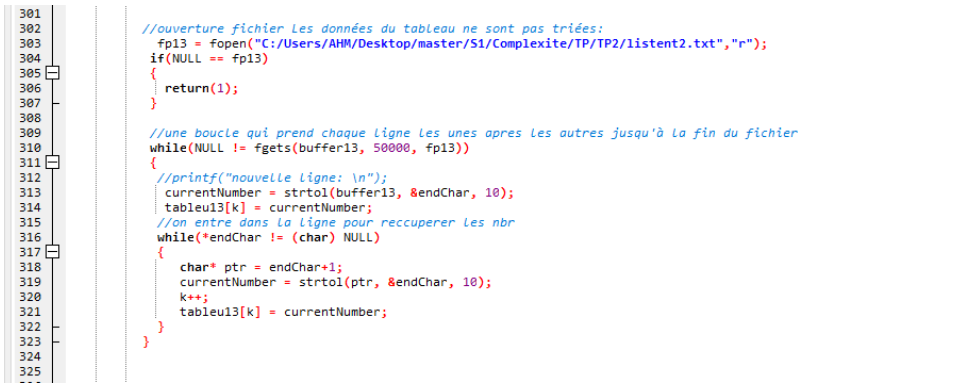
```
int main(int argc, char** argv){
    int   currentNumber;
    char   *endChar;

    char *buffer11 = malloc(sizeof(char) * 50000);
    char *buffer12 = malloc(sizeof(char) * 50000);
    char *buffer13 = malloc(sizeof(char) * 50000);

    int *tableu11 = malloc(sizeof(int) * 50000);
    int *tableu12 = malloc(sizeof(int) * 50000);
    int *tableu13 = malloc(sizeof(int) * 50000);



    int *tmp = malloc(sizeof(int) * 50000);

    int i =0, j=0, k=0;

    FILE *fp11, *fp12, *fp13;  //50000

    clock_t t11, t12, t21, t22, t31, t32;
    double delta1,delta2,delta3;
    double resulta[3][6];


    printf("\nLes resultats pour la taille de 50000\n\n");


        //ouverture fichier les données du tableau sont triées en bon ordre:
        fp11 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listebo2.txt","r");
        if(NULL == fp11)
        {
          return(1);
        }

        //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
        while(NULL != fgets(buffer11, 50000, fp11))
```

FIGURE 3.18 – Code source de la partie main



```
263         while(NULL != fgets(buffer11, 50000, fp11))
264         {
265          //printf("nouvelle ligne: \n");
266           currentNumber = strtol(buffer11, &endChar, 10);
267           tableu11[i] = currentNumber;
268          //on entre dans la ligne pour reccuperer les nbr
269          while(*endChar != (char) NULL)
270          {
271             char* ptr = endChar+1;
272             currentNumber = strtol(ptr, &endChar, 10);
273             i++;
274             tableu11[i] = currentNumber;
275          }
276         }
277
278        //ouverture fichier les données du tableau sont triées en ordre inverse:
279         fp12 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listeio2.txt","r");
280         if(NULL == fp12)
281         {
282           return(1);
283         }
284
285         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
286         while(NULL != fgets(buffer12, 50000, fp12))
287         {
288          //printf("nouvelle ligne: \n");
289           currentNumber = strtol(buffer12, &endChar, 10);
290           tableu12[j] = currentNumber;
291          //on entre dans la ligne pour reccuperer les nbr
292          while(*endChar != (char) NULL)
293          {
294             char* ptr = endChar+1;
295             currentNumber = strtol(ptr, &endChar, 10);
296             j++;
297             tableu12[j] = currentNumber;
298          }
299         }
300
```

FIGURE 3.19 – Code source de la partie main



```
301
302       //ouverture fichier les données du tableau ne sont pas triées:
303         fp13 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listent2.txt","r");
304         if(NULL == fp13)
305         {
306           return(1);
307         }
308
309         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
310         while(NULL != fgets(buffer13, 50000, fp13))
311         {
312          //printf("nouvelle ligne: \n");
313           currentNumber = strtol(buffer13, &endChar, 10);
314           tableu13[k] = currentNumber;
315          //on entre dans la ligne pour reccuperer les nbr
316          while(*endChar != (char) NULL)
317          {
318             char* ptr = endChar+1;
319             currentNumber = strtol(ptr, &endChar, 10);
320             k++;
321             tableu13[k] = currentNumber;
322          }
323         }
324
325
```

FIGURE 3.20 – Code source de la partie main

```
327
328         //**************************calcule du temps d'excecution de la langueur 10^4
329         t11 = clock();
330         // appel au fonction de tri d'un tableau
331         //triFusion(0,49999,tableu11,tmp);
332         Tri_par_selection(tableu11,50000);
333         t12 = clock();
334         delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
335         //_____________________________________________________
336
337         t21 = clock();
338         // appel au fonction de tri d'un tableau
339         //triFusion(0,49999,tableu12,tmp);
340         Tri_par_selection(tableu12,50000);
341         t22 = clock();
342         delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
343         //_____________________________________________________
344
345
346         t31 = clock();
347         // appel au fonction de tri d'un tableau
348         //triFusion(0,49999,tableu13,tmp);
349         Tri_par_selection(tableu13,50000);
350         t32 = clock();
351         delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
352         //_____________________________________________________
353
354         //L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******
355
356         printf("\t\tLE TRI PAR SELECTION\n\n");
357         printf("****Les donnees du tableau sont triees en bon ordre****\n");
358         printf("==>Le temps d'execution: (%lf)\n",delta1);
359         printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
360         printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
361         printf("****Les donnees du tableau ne sont pas triees****\n");
362         printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

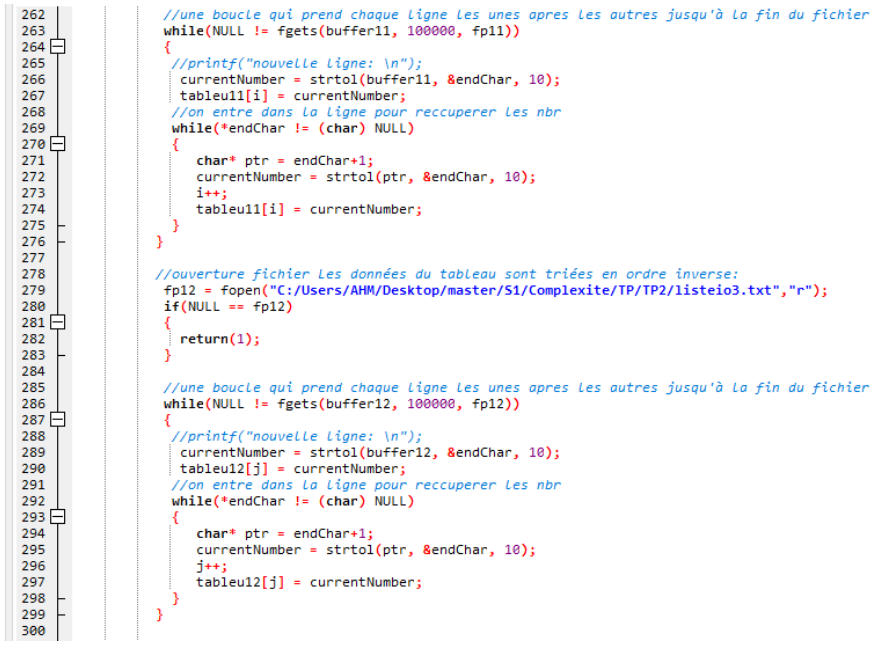
FIGURE 3.21 – Code source de la partie main

```
364
365     int rps;
366     printf("vous voullez afficher le tableau triee? choisisser 1 si c'est le cas.\t");
367     scanf("%d",&rps);
368     //l'affichage de tabkeau trie
369     if(rps == 1){
370         for(i=1;i<50000;i++){
371         printf("Le nombre courant = %d\n",tableu12[i]);
372     }
373
374   }
375   free(tableu11); free(buffer11);
376   free(tableu12); free(buffer12);
377   free(tableu13); free(buffer13);
378   free(tmp);
379
```

FIGURE 3.22 – Code source de la partie main

```
225 int main(int argc, char** argv){
226     int   currentNumber;
227     char   *endChar;
228
229     char *buffer11 = malloc(sizeof(char) * 100000);
230     char *buffer12 = malloc(sizeof(char) * 100000);
231     char *buffer13 = malloc(sizeof(char) * 100000);
232
233     int *tableu11 = malloc(sizeof(int) * 100000);
234     int *tableu12 = malloc(sizeof(int) * 100000);
235     int *tableu13 = malloc(sizeof(int) * 100000);
236
237
238
239
240     int *tmp = malloc(sizeof(int) * 100000);
241
242     int i =0, j=0, k=0;
243
244     FILE *fp11, *fp12, *fp13;  //100000
245
246     clock_t t11, t12, t21, t22, t31, t32;
247     double delta1,delta2,delta3;
248     double resulta[3][6];
249
250
251
252     printf("\nLes resultats pour la taille de 100000\n\n");
253
254
255          //ouverture fichier Les données du tableau sont triées en bon ordre:
256          fp11 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listebo3.txt","r");
257         if(NULL == fp11)
258         {
259           return(1);
260         }
261
262         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
263         while(NULL != fgets(buffer11, 100000, fp11))
```

FIGURE 3.23 – Code source de la partie main

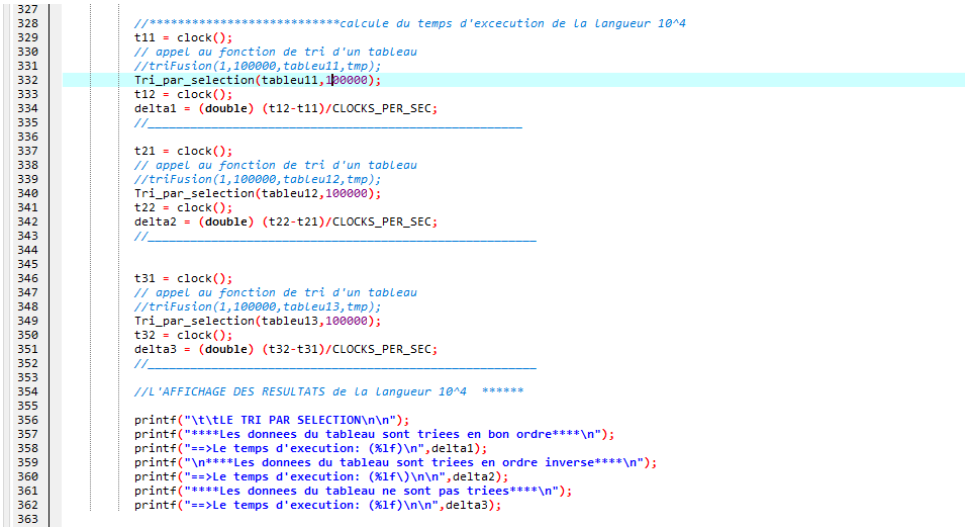


```
262         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
263         while(NULL != fgets(buffer11, 100000, fp11))
264         {
265          //printf("nouvelle ligne: \n");
266          currentNumber = strtol(buffer11, &endChar, 10);
267          tableu11[i] = currentNumber;
268          //on entre dans la ligne pour reccuperer les nbr
269          while(*endChar != (char) NULL)
270          {
271             char* ptr = endChar+1;
272             currentNumber = strtol(ptr, &endChar, 10);
273             i++;
274             tableu11[i] = currentNumber;
275          }
276        }
277
278        //ouverture fichier les données du tableau sont triées en ordre inverse:
279         fp12 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listeio3.txt","r");
280         if(NULL == fp12)
281         {
282          return(1);
283         }
284
285         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
286         while(NULL != fgets(buffer12, 100000, fp12))
287         {
288          //printf("nouvelle ligne: \n");
289          currentNumber = strtol(buffer12, &endChar, 10);
290          tableu12[j] = currentNumber;
291          //on entre dans la ligne pour reccuperer les nbr
292          while(*endChar != (char) NULL)
293          {
294             char* ptr = endChar+1;
295             currentNumber = strtol(ptr, &endChar, 10);
296             j++;
297             tableu12[j] = currentNumber;
298          }
299        }
300
```

FIGURE 3.24 – Code source de la partie main

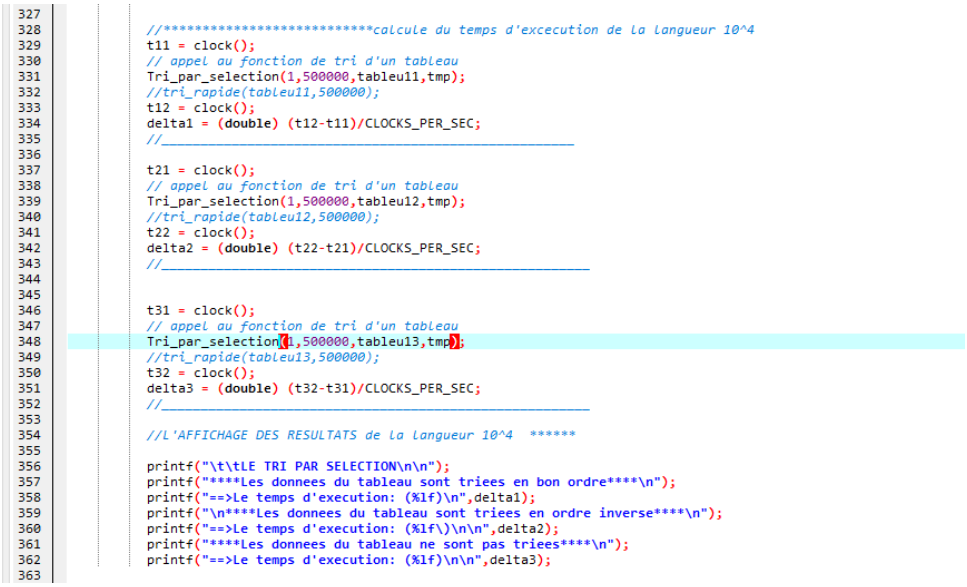
```
301
302       //ouverture fichier les données du tableau ne sont pas triées:
303         fp13 = fopen("C:/Users/AHM/Desktop/master/S1/Complexite/TP/TP2/listent3.txt","r");
304         if(NULL == fp13)
305         {
306          return(1);
307         }
308
309         //une boucle qui prend chaque ligne les unes apres les autres jusqu'à la fin du fichier
310         while(NULL != fgets(buffer13, 100000, fp13))
311         {
312          //printf("nouvelle ligne: \n");
313          currentNumber = strtol(buffer13, &endChar, 10);
314          tableu13[k] = currentNumber;
315          //on entre dans la ligne pour reccuperer les nbr
316          while(*endChar != (char) NULL)
317          {
318             char* ptr = endChar+1;
319             currentNumber = strtol(ptr, &endChar, 10);
320             k++;
321             tableu13[k] = currentNumber;
322          }
323        }
324
325
```

FIGURE 3.25 – Code source de la partie main



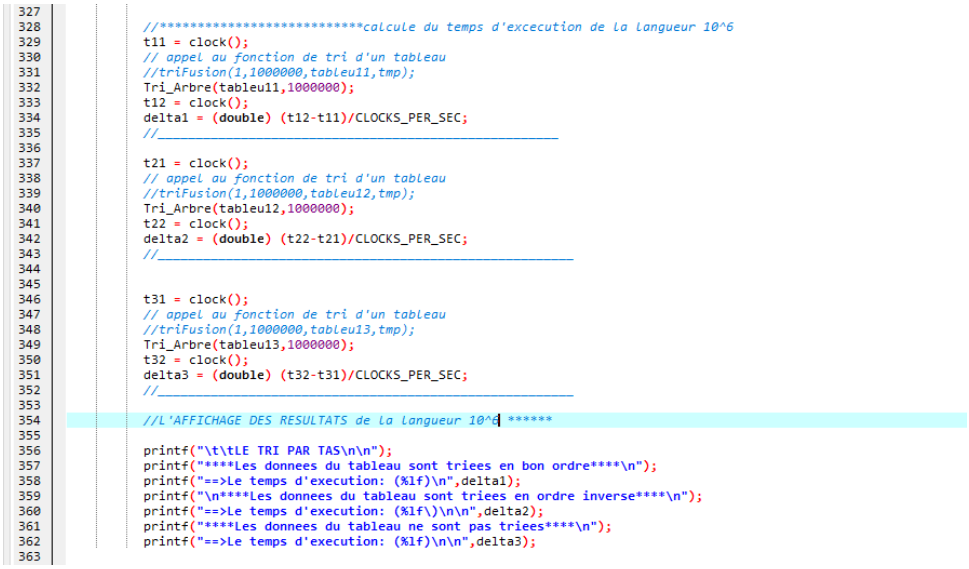
```
//*************************calcule du temps d'excecution de la langueur 10^4
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,100000,tableu11,tmp);
Tri_par_selection(tableu11,100000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,100000,tableu12,tmp);
Tri_par_selection(tableu12,100000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,100000,tableu13,tmp);
Tri_par_selection(tableu13,100000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________

//L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******

printf("\t\tLE TRI PAR SELECTION\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.26 – Code source de la partie main



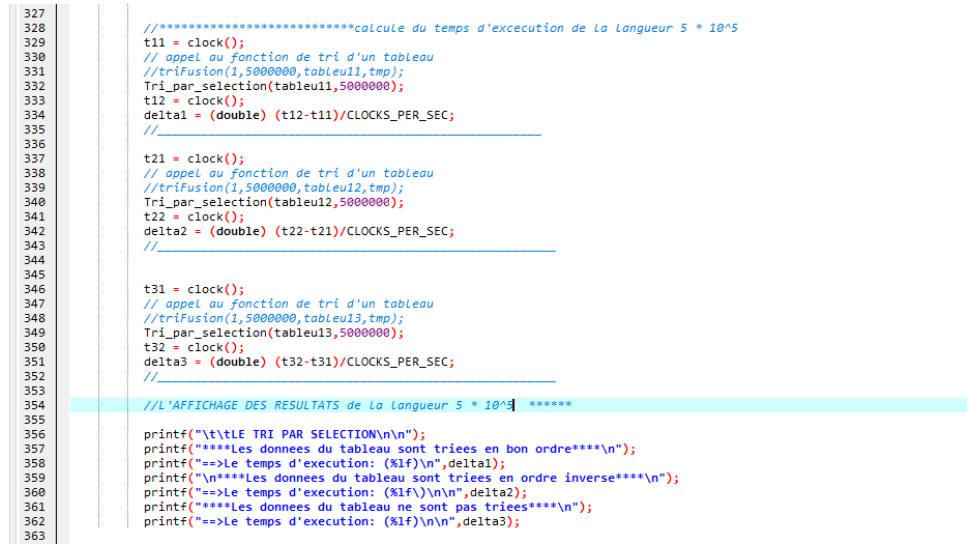
```
//*************************calcule du temps d'excecution de la langueur 10^4
t11 = clock();
// appel au fonction de tri d'un tableau
Tri_par_selection(1,500000,tableu11,tmp);
//tri_rapide(tableu11,500000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
Tri_par_selection(1,500000,tableu12,tmp);
//tri_rapide(tableu12,500000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
Tri_par_selection(1,500000,tableu13,tmp);
//tri_rapide(tableu13,500000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________

//L'AFFICHAGE DES RESULTATS de la langueur 10^4  ******

printf("\t\tLE TRI PAR SELECTION\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.27 – Code source de la partie main



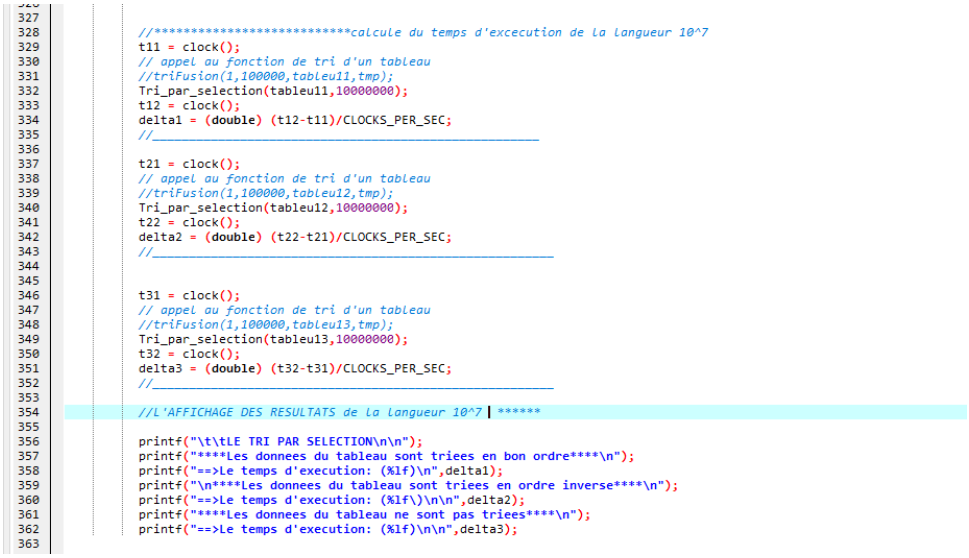
```
//**************************calcule du temps d'excecution de la langueur 10^6
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,1000000,tableu11,tmp);
Tri_Arbre(tableu11,1000000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,1000000,tableu12,tmp);
Tri_Arbre(tableu12,1000000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,1000000,tableu13,tmp);
Tri_Arbre(tableu13,1000000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________________

//L'AFFICHAGE DES RESULTATS de la Langueur 10^6 ******

printf("\t\tLE TRI PAR TAS\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.28 – Code source de la partie main



```
//**************************calcule du temps d'excecution de la langueur 5 * 10^5
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,5000000,tableu11,tmp);
Tri_par_selection(tableu11,5000000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,5000000,tableu12,tmp);
Tri_par_selection(tableu12,5000000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,5000000,tableu13,tmp);
Tri_par_selection(tableu13,5000000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________________

//L'AFFICHAGE DES RESULTATS de la Langueur 5 * 10^5  ******

printf("\t\tLE TRI PAR SELECTION\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.29 – Code source de la partie main

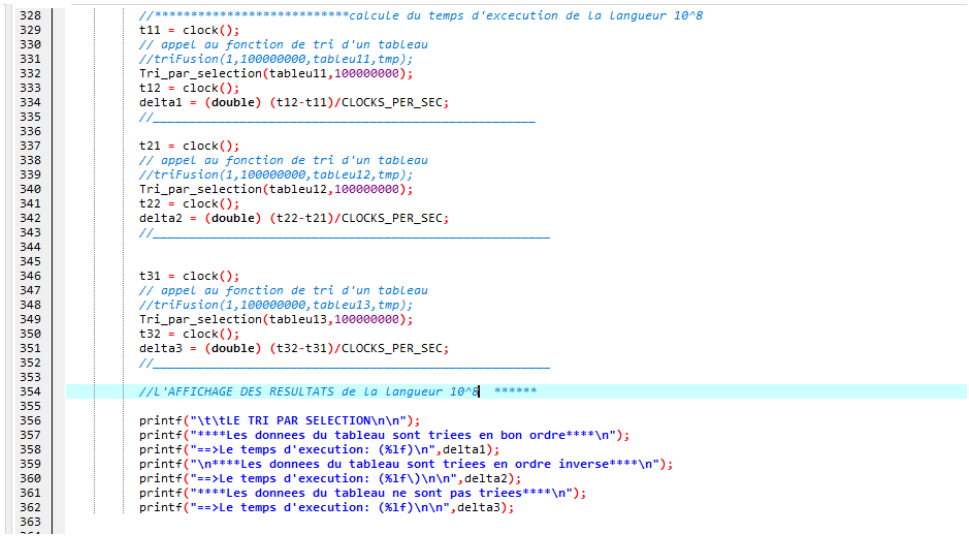


```
//***************************calcule du temps d'excecution de la langueur 10^7
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,100000,tableu11,tmp);
Tri_par_selection(tableu11,10000000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,100000,tableu12,tmp);
Tri_par_selection(tableu12,10000000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,100000,tableu13,tmp);
Tri_par_selection(tableu13,10000000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________________

//L'AFFICHAGE DES RESULTATS de la Langueur 10^7 ******

printf("\t\tLE TRI PAR SELECTION\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.30 – Code source de la partie main

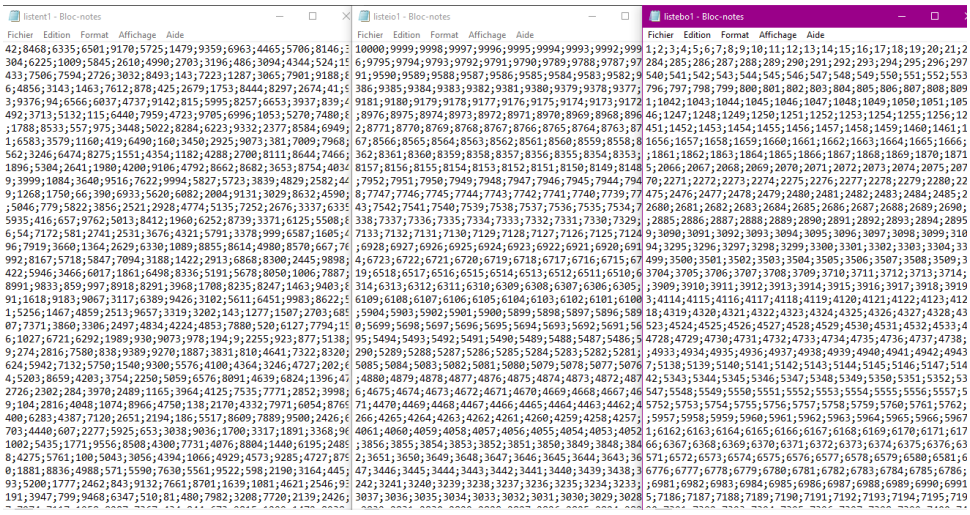
```
//***************************calcule du temps d'excecution de la langueur 10^4
t11 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,50000000,tableu11,tmp);
Tri_par_selection(tableu11,50000000);
t12 = clock();
delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
//_____________________________________________________

t21 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,50000000,tableu12,tmp);
Tri_par_selection(tableu12,50000000);
t22 = clock();
delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
//_____________________________________________________


t31 = clock();
// appel au fonction de tri d'un tableau
//triFusion(1,50000000,tableu13,tmp);
Tri_par_selection(tableu13,50000000);
t32 = clock();
delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
//_____________________________________________________

//L'AFFICHAGE DES RESULTATS de la Langueur 10^4  ******

printf("\t\tLE TRI PAR SELECTION\n\n");
printf("****Les donnees du tableau sont triees en bon ordre****\n");
printf("==>Le temps d'execution: (%lf)\n",delta1);
printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
printf("****Les donnees du tableau ne sont pas triees****\n");
printf("==>Le temps d'execution: (%lf)\n\n",delta3);
```

FIGURE 3.31 – Code source de la partie main



```
328         //**************************calcule du temps d'excecution de la langueur 10^8
329         t11 = clock();
330         // appel au fonction de tri d'un tableau
331         //triFusion(1,100000000,tableu11,tmp);
332         Tri_par_selection(tableu11,100000000);
333         t12 = clock();
334         delta1 = (double) (t12-t11)/CLOCKS_PER_SEC;
335         //_____________________________________________
336
337         t21 = clock();
338         // appel au fonction de tri d'un tableau
339         //triFusion(1,100000000,tableu12,tmp);
340         Tri_par_selection(tableu12,100000000);
341         t22 = clock();
342         delta2 = (double) (t22-t21)/CLOCKS_PER_SEC;
343         //_____________________________________________
344
345
346         t31 = clock();
347         // appel au fonction de tri d'un tableau
348         //triFusion(1,100000000,tableu13,tmp);
349         Tri_par_selection(tableu13,100000000);
350         t32 = clock();
351         delta3 = (double) (t32-t31)/CLOCKS_PER_SEC;
352         //_____________________________________________
353
354         //L'AFFICHAGE DES RESULTATS de La Langueur 10^8  ******
355
356         printf("\t\tLE TRI PAR SELECTION\n\n");
357         printf("****Les donnees du tableau sont triees en bon ordre****\n");
358         printf("==>Le temps d'execution: (%lf)\n",delta1);
359         printf("\n****Les donnees du tableau sont triees en ordre inverse****\n");
360         printf("==>Le temps d'execution: (%lf\)\n\n",delta2);
361         printf("****Les donnees du tableau ne sont pas triees****\n");
362         printf("==>Le temps d'execution: (%lf)\n\n",delta3);
363
```

FIGURE 3.32 – Code source de la partie main



listent1 - Bloc-notes

```
42;8468;6335;6501;9170;5725;1479;9359;6963;4465;5706;8146;
304;6225;1009;5845;2610;4990;2703;3196;486;3094;4344;524;1
433;7506;7594;2726;3032;8493;143;7223;1287;3065;7901;9188;
6;4856;3143;1463;7612;878;425;2679;1753;8444;8297;2674;41;
3;9376;94;6566;6037;4737;9142;815;5995;8257;6653;3937;839;
492;3713;5132;115;6440;7959;4723;9705;6996;1053;5270;7480;
;1788;8533;557;975;3448;5022;8284;6223;9332;2377;8584;6949;
1;6583;3579;1160;419;6490;160;3450;2925;9073;381;7009;7968;
562;3246;6474;8275;1551;4354;1182;4288;2700;8111;8644;7466;
1896;5304;2641;1980;4200;9106;4792;8662;8682;3653;8754;4034
9;3999;1084;3640;9516;7622;9994;5827;5723;3839;4829;2582;4
9;1268;1750;66;390;6933;5620;6082;2004;9131;3029;8632;4590;
;5046;779;5822;3856;2521;2928;4774;5135;7252;2676;3337;633
5935;416;657;9762;5013;8412;1960;6252;8739;3371;6125;5508;
6;54;7172;581;2741;2531;3676;4321;5791;3378;999;6587;1605;
96;7919;3660;1364;2629;6330;1089;8855;8614;4980;8570;667;7
992;8167;5718;5847;7094;3188;1422;2913;6868;8300;2445;9898;
422;5946;3466;6017;1861;6498;8336;5191;5678;8050;1006;7887;
8991;9833;859;997;8918;8291;3968;1708;8235;8247;1463;9403;
91;1618;9183;9067;3117;6389;9426;3102;5611;6451;9983;8622;
1;5256;1467;4859;2513;9657;3319;3202;143;1277;1507;2703;685
07;7371;3860;3306;2497;4834;4224;4853;7880;520;6127;7794;15
6;1027;6721;6292;1989;930;9073;978;194;9;2255;923;877;5138;
9;274;2816;7580;838;9389;9270;1887;3831;810;4641;7322;8320;
624;5942;7132;5750;1540;9300;5576;4100;4364;3246;4727;202;
4;5203;8659;4203;3754;2250;5059;6576;8091;4639;6824;1396;47
2726;2302;284;3970;2489;1165;3964;4125;7535;7771;2852;3998;
9;104;2816;4048;1074;8966;4750;138;2170;4332;7971;6054;8765
400;6283;4387;7120;2651;2194;186;5517;8609;7889;9500;2426;6
703;4440;607;2277;5925;653;3038;9036;1700;3317;1891;3368;96
1002;5435;1771;9556;8508;4300;7731;4076;8804;1440;6195;2485
8;4275;5761;100;5043;3056;4394;1066;4929;4573;9285;4727;87
0;1881;8836;4988;571;5590;7630;5561;9522;598;2190;3164;445
93;5200;1777;2462;843;9132;7661;8701;1639;1081;4621;2546;9
191;3947;799;9468;6347;510;81;480;7982;3208;7720;2136;3723
```

listeio1 - Bloc-notes

```
10000;9999;9998;9997;9996;9995;9994;9993;9992;999
6;9795;9794;9793;9792;9791;9790;9789;9788;9787;97
91;9590;9589;9588;9587;9586;9585;9584;9583;9582;9
386;9385;9384;9383;9382;9381;9380;9379;9378;9377;
9181;9180;9179;9178;9177;9176;9175;9174;9173;9172
;8976;8975;8974;8973;8972;8971;8970;8969;8968;896
2;8771;8770;8769;8768;8767;8766;8765;8764;8763;87
67;8566;8565;8564;8563;8562;8561;8560;8559;8558;8
362;8361;8360;8359;8358;8357;8356;8355;8354;8353;
8157;8156;8155;8154;8153;8152;8151;8150;8149;8148
;7952;7951;7950;7949;7948;7947;7946;7945;7944;794
8;7747;7746;7745;7744;7743;7742;7741;7740;7739;77
43;7542;7541;7540;7539;7538;7537;7536;7535;7534;7
338;7337;7336;7335;7334;7333;7332;7331;7330;7329;
7133;7132;7131;7130;7129;7128;7127;7126;7125;7124
;6928;6927;6926;6925;6924;6923;6922;6921;6920;691
4;6723;6722;6721;6720;6719;6718;6717;6716;6715;67
19;6518;6517;6516;6515;6514;6513;6512;6511;6510;6
314;6313;6312;6311;6310;6309;6308;6307;6306;6305;
6109;6108;6107;6106;6105;6104;6103;6102;6101;6100
;5904;5903;5902;5901;5900;5899;5898;5897;5896;589
0;5699;5698;5697;5696;5695;5694;5693;5692;5691;56
95;5494;5493;5492;5491;5490;5489;5488;5487;5486;5
290;5289;5288;5287;5286;5285;5284;5283;5282;5281;
5085;5084;5083;5082;5081;5080;5079;5078;5077;5076
;4880;4879;4878;4877;4876;4875;4874;4873;4872;487
6;4675;4674;4673;4672;4671;4670;4669;4668;4667;46
71;4470;4469;4468;4467;4466;4465;4464;4463;4462;4
266;4265;4264;4263;4262;4261;4260;4259;4258;4257;
4061;4060;4059;4058;4057;4056;4055;4054;4053;4052
;3856;3855;3854;3853;3852;3851;3850;3849;3848;384
2;3651;3650;3649;3648;3647;3646;3645;3644;3643;36
47;3446;3445;3444;3443;3442;3441;3440;3439;3438;3
242;3241;3240;3239;3238;3237;3236;3235;3234;3233;
3037;3036;3035;3034;3033;3032;3031;3030;3029;3028
```

listent1 - Bloc-notes

```
1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20;21;2
284;285;286;287;288;289;290;291;292;293;294;295;296;297
540;541;542;543;544;545;546;547;548;549;550;551;552;553
796;797;798;799;800;801;802;803;804;805;806;807;808;809
1;1042;1043;1044;1045;1046;1047;1048;1049;1050;1051;105
46;1247;1248;1249;1250;1251;1252;1253;1254;1255;1256;12
451;1452;1453;1454;1455;1456;1457;1458;1459;1460;1461;1
1656;1657;1658;1659;1660;1661;1662;1663;1664;1665;1666;
;1861;1862;1863;1864;1865;1866;1867;1868;1869;1870;1871
5;2066;2067;2068;2069;2070;2071;2072;2073;2074;2075;207
70;2271;2272;2273;2274;2275;2276;2277;2278;2279;2280;22
475;2476;2477;2478;2479;2480;2481;2482;2483;2484;2485;2
2680;2681;2682;2683;2684;2685;2686;2687;2688;2689;2690;
;2885;2886;2887;2888;2889;2890;2891;2892;2893;2894;2895
9;3090;3091;3092;3093;3094;3095;3096;3097;3098;3099;310
94;3295;3296;3297;3298;3299;3300;3301;3302;3303;3304;33
499;3500;3501;3502;3503;3504;3505;3506;3507;3508;3509;3
3704;3705;3706;3707;3708;3709;3710;3711;3712;3713;3714;
;3903;3910;3911;3912;3913;3914;3915;3916;3917;3918;3919
3;4114;4115;4116;4117;4118;4119;4120;4121;4122;4123;412
18;4319;4320;4321;4322;4323;4324;4325;4326;4327;4328;43
523;4524;4525;4526;4527;4528;4529;4530;4531;4532;4533;4
4728;4729;4730;4731;4732;4733;4734;4735;4736;4737;4738;
;4933;4934;4935;4936;4937;4938;4939;4940;4941;4942;4943
7;5138;5139;5140;5141;5142;5143;5144;5145;5146;5147;514
42;5343;5344;5345;5346;5347;5348;5349;5350;5351;5352;53
547;5548;5549;5550;5551;5552;5553;5554;5555;5556;5557;5
5752;5753;5754;5755;5756;5757;5758;5759;5760;5761;5762;
;5957;5958;5959;5960;5961;5962;5963;5964;5965;5966;5967
1;6162;6163;6164;6165;6166;6167;6168;6169;6170;6171;617
66;6367;6368;6369;6370;6371;6372;6373;6374;6375;6376;63
571;6572;6573;6574;6575;6576;6577;6578;6579;6580;6581;6
6776;6777;6778;6779;6780;6781;6782;6783;6784;6785;6786;
;6981;6982;6983;6984;6985;6986;6987;6988;6989;6990;6991
5;7186;7187;7188;7189;7190;7191;7192;7193;7194;7195;719
```

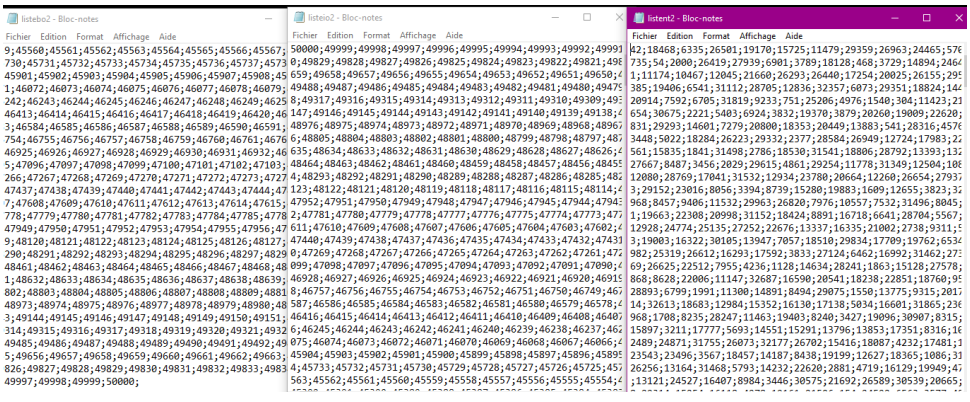
listeio2 - Bloc-notes

```
50000;49999;49998;49997;49996;49995;49994;49993;49992;49991
0;49829;49828;49827;49826;49825;49824;49823;49822;49821;498
659;49658;49657;49656;49655;49654;49653;49652;49651;49650;4
49488;49487;49486;49485;49484;49483;49482;49481;49480;4947
49315;49314;49313;49312;49311;49310;49309;493
3856;3855;3854;3853;3852;3851;3850;3849;3848;384
2;3651;3650;3649;3648;3647;3646;3645;3644;3643;36
47;3446;3445;3444;3443;3442;3441;3440;3439;3438;3
242;3241;3240;3239;3238;3237;3236;3235;3234;3233;
3037;3036;3035;3034;3033;3032;3031;3030;3029;3028
```

listent2 - Bloc-notes

```
42;18468;6335;26501;19170;15725;11479;29359;26963;24465;576
735;54;2000;26419;27939;6901;3789;18128;468;3729;14894;246
1;11174;10467;12045;21660;26293;26440;17254;20025;26155;295
385;19406;6541;31112;28705;12836;32357;6073;29351;18824;14
20914;7592;6705;31819;9233;751;25286;4976;1540;304;11423;21
654;30675;2221;5403;6924;3832;19370;3879;20260;19009;22620;
6776;6777;6778;6779;6780;6781;6782;6783;6784;6785;6786;
3448;5022;18284;26223;29332;2377;28584;26949;12724;17983;2
561;15835;1841;31498;2786;18530;31541;18806;28792;13393;13
27667;8487;3456;2029;29615;4861;29254;11778;31349;12504;108
12080;28769;17041;31532;12934;23780;20664;12260;26654;2793
3;29152;23016;8056;3394;8739;15280;19883;1609;12655;3823;3
968;8457;9406;11532;29963;26820;7976;10557;7532;31496;8045;
1;19663;22308;20998;31152;18424;8891;16718;6641;28704;5567;
12928;24774;25135;27252;22676;13337;16335;21002;2738;9311;5
3;19003;16322;30105;13947;7057;18510;29834;17709;19762;653
982;25319;26612;16293;17592;3833;27124;6462;16992;31462;27
69;26625;22512;7955;4236;1128;14634;28241;1863;15128;27578;
868;8628;22006;11147;32687;16590;20541;18238;22851;18760;9
28893;6799;1991;11300;14891;8494;29075;1550;13775;9315;201
14;32613;18683;12984;15352;16130;17138;5034;16601;31865;236
968;1708;8235;28247;11463;19403;8240;3427;19096;30907;8315;
15897;3211;17777;5693;14551;15291;13796;13853;17351;8316;16
2489;24871;31755;26073;32177;26702;15416;18087;4232;17481;1
23543;23496;3567;18457;14187;8438;19199;12627;18365;1086;31
26256;13164;31468;5793;14232;22620;2881;4719;16129;19949;47
;13121;24527;16407;8984;3446;30575;21692;26589;30539;20665;
```

FIGURE 3.33 – Exemple de fichier des données.



listeio2 - Bloc-notes

```
9;45560;45561;45562;45563;45564;45565;45566;45567;
730;45731;45732;45733;45734;45735;45736;45737;4573
45901;45902;45903;45904;45905;45906;45907;45908;45
1;46072;46073;46074;46075;46076;46077;46078;46079;
242;46243;46244;46245;46246;46247;46248;46249;4625
46413;46414;46415;46416;46417;46418;46419;46420;46
3;46584;46585;46586;46587;46588;46589;46590;46591;
754;46755;46756;46757;46758;46759;46760;46761;4676
46925;46926;46927;46928;46929;46930;46931;46932;46
5;47096;47097;47098;47099;47100;47101;47102;47103;
266;47267;47268;47269;47270;47271;47272;47273;4727
47437;47438;47439;47440;47441;47442;47443;47444;47
7;47608;47609;47610;47611;47612;47613;47614;47615;
778;47779;47780;47781;47782;47783;47784;47785;4778
47949;47950;47951;47952;47953;47954;47955;47956;47
9;48120;48121;48122;48123;48124;48125;48126;48127;
290;48291;48292;48293;48294;48295;48296;48297;4829
48461;48462;48463;48464;48465;48466;48467;48468;48
1;48632;48633;48634;48635;48636;48637;48638;48639;
802;48803;48804;48805;48806;48807;48808;48809;4881
48973;48974;48975;48976;48977;48978;48979;48980;48
3;49144;49145;49146;49147;49148;49149;49150;49151;
314;49315;49316;49317;49318;49319;49320;49321;4932
49485;49486;49487;49488;49489;49490;49491;49492;49
5;49656;49657;49658;49659;49660;49661;49662;49663;
826;49827;49828;49829;49830;49831;49832;49833;4983
49997;49998;49999;50000;
```

listent2 - Bloc-notes

```
563;45562;45561;45560;45559;45558;45557;45556;45555;45554;4
4;45733;45732;45731;45730;45729;45728;45727;45726;45725;457
45904;45903;45902;45901;45900;45899;45898;45897;45896;4589
075;46074;46073;46072;46071;46070;46069;46068;46067;46066;4
6;46245;46244;46243;46242;46241;46240;46239;46238;46237;462
635;46634;46633;46632;46631;46630;46629;46628;46627;46626;4
4;46755;46756;46755;46754;46753;46752;46751;46750;46749;467
587;46586;46585;46584;46583;46582;46581;46580;46579;46578;4
4616;46615;46614;46613;46612;46611;46610;46609;46608;46607
47440;47439;47438;47437;47436;47435;47434;47433;47432;47431
0;47269;47268;47267;47266;47265;47264;47263;47262;47261;472
099;47098;47097;47096;47095;47094;47093;47092;47091;47090;4
46928;46927;46926;46925;46924;46923;46922;46921;46920;4691
8;46757;46756;46755;46754;46753;46752;46751;46750;46749;467
635;46634;46633;46632;46631;46630;46629;46628;46627;46626;4
48464;48463;48462;48461;48460;48459;48458;48457;48456;4845
47952;47951;47950;47949;47948;47947;47946;47945;47944;4794
123;48122;48121;48120;48119;48118;48117;48116;48115;48114;4
4;48293;48292;48291;48290;48289;48288;48287;48286;48285;482
2;47781;47780;47779;47778;47777;47776;47775;47774;47773;477
611;47610;47609;47608;47607;47606;47605;47604;47603;47602;4
47440;47439;47438;47437;47436;47435;47434;47433;47432;47431
0;47269;47268;47267;47266;47265;47264;47263;47262;47261;472
099;47098;47097;47096;47095;47094;47093;47092;47091;47090;4
46928;46927;46926;46925;46924;46923;46922;46921;46920;4691
8;46757;46756;46755;46754;46753;46752;46751;46750;46749;467
587;46586;46585;46584;46583;46582;46581;46580;46579;46578;4
4616;46615;46614;46613;46612;46611;46610;46609;46608;46607
6;46245;46244;46243;46242;46241;46240;46239;46238;46237;462
075;46074;46073;46072;46071;46070;46069;46068;46067;46066;4
45904;45903;45902;45901;45900;45899;45898;45897;45896;4589
4;45733;45732;45731;45730;45729;45728;45727;45726;45725;457
563;45562;45561;45560;45559;45558;45557;45556;45555;45554;4
```

FIGURE 3.34 – Exemple de fichier des sequences de données.

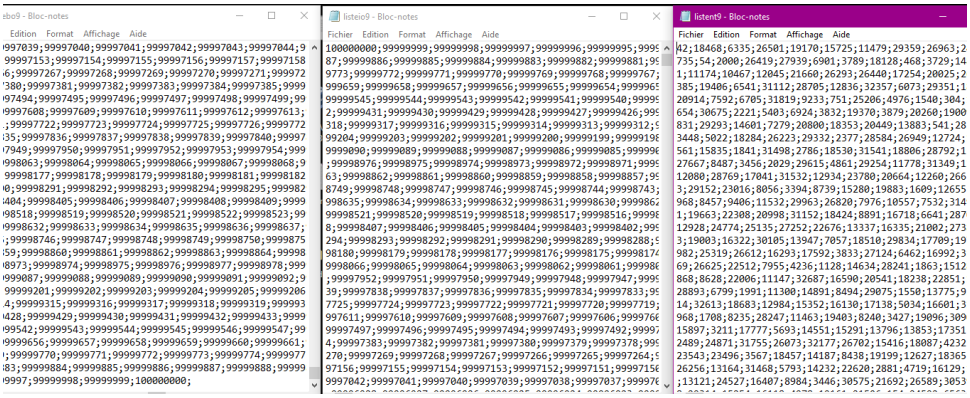


FIGURE 3.35 – Exemple de fichier des sequences de données.

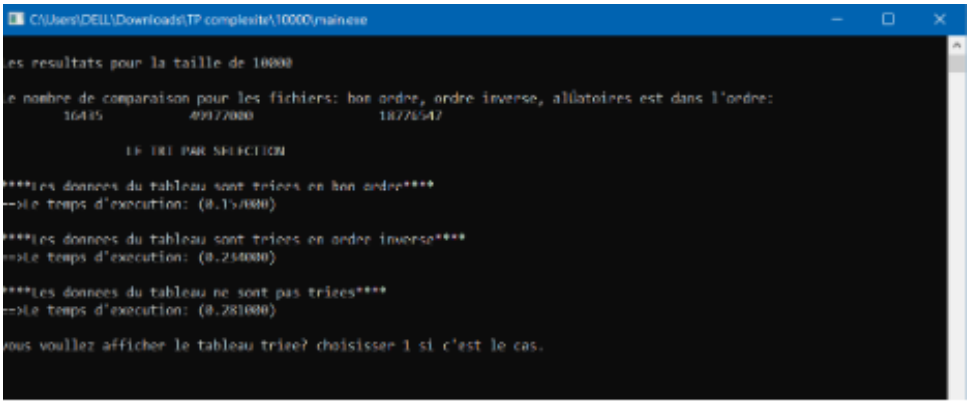


FIGURE 3.36 – Exemple de fenêtre d'éxécution.

Conclusion générale

L'algorithme le plus efficace est le tri rapide, suivi du tris par tas et du tri fusion ils ont en communs une complexité de O(n log2(n)). Les trois autres algorithmes sont beaucoup plus longs en terme de temps d'execution et sont donc beaucoup moins utilisé en pratique le pire d'entre eux est le tri à bulles, suivi du tris par séléction et du tri par insertion ils partagent tous la meme complexité qui est de O(n2). Le pire cas pour les algorithmes de tri rapide, tri par tas et tri fusion représente le moyen cas chez les algorithmes de tri par selection, tri par insertion et tri par bulles.

On ne distingue pas de moyen cas pour les algorithmes de tri rapide et tri fusion car en effet l'augmentation du temps d'execution avec en entrée une séquence de donnée en ordre inverse (pire cas) et aléatoire (moyen cas) est similaire.

Les algorithmes de tri fusion et de tri rapide sont plus efficaces avec des instances de données non triés. Le nombre de comparaison effectué par un algorithme de tri influe directement sur sa complexité. Plus le nombre de comparaison est élevé (respect. bas) plus la complexité est grande (respect. faible) il en est de même pour la taille de la séquence de données et aussi la taille du nombre lui-même.

Répartition des taches :

BOUADI Nassima :

> Ecriture des pseudos code des fonctions .

> La rédaction du rapport et son organisation sous LATEX(sommaire, table de figure ....).

> Dessin des graphes.

FERKOUS Sarah :

> Ecriture du code source du main() .

> Faire les tests d'executions.

> Remarques et critiques sur le rapport.

Guerbas Tinhinane :

> Ecriture du code des fonctions .

> Analyses des graphes et conclusions.

> Remaques et critiques sur le rapport.

> Fait des tests de la question 05.