Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique
Département IA et SD

Rapport travaux pratiques n° 2

Module : Conception et Complexité des Algorithmes

Les Algorithmes de Tri

Travail présenté par :

- **BENKOUITEN Aymen————-191931046409**
- **MALKI Omar Chouaab————-191931081333**
- **KENAI Imad Eddine————-191932017671**
- **MEKKAOUI Mohamed————-191931081338**

2022/2023

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Lorsqu'on veut traiter un tableau à très grand nombre d'éléments ce serait bien de les présenter dans un ordre précis pour les bien gérer ou bien on peut dire "trie ces éléments", car sur un tableau de petite taille, la différence de temps mis par l'algorithme est quasiment invisible selon sa complexité. Cependant, si le tableau a plusieurs millions d'éléments, la durée de son tri peut varier de plusieurs jours selon l'algorithme utilisé. Donc la classification des algorithmes de tri est primordiale, car elle permet de choisir l'algorithme le plus adapté au problème traité, tout en tenant compte des contraintes imposées par celui-ci... Mais la question qui se pose est **comment différencier les algorithmes de tri ?**[1]

L'objectif dans ce TP est de mettre en pratique et tester les algorithmes de tri et calculer leur complexité et comparer les différentes méthodes d'un point de vue théorique et expérimental, cela permet donc de différencier ces algorithmes de tri.

Première partie

Développement de l'algorithme et du programme correspondant

0.1 Question 01

Implémenter les algorithmes de tri suivants en langage C :

Réponse 01 :

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

La réponse a cette question ce fait dans la partie annexe.

Deuxième partie

Mesure du temps d'exécution.

0.2 Question 01

Les pseudos code des algorithmes de tri :

Réponse 01 :

Tri par sélection : L'idée du tri par selection consiste à chaque étape à rechercher le plus petit élément non encore trié et à le placer à la suite des éléments déjà triés.[3]

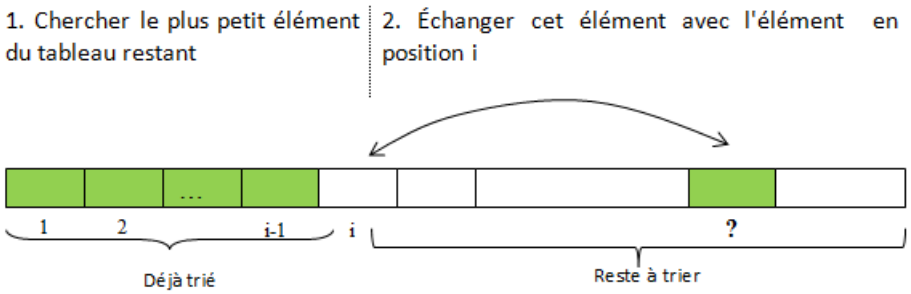

FIGURE 1 – schéma de tri par selection

Pseudo code :

```
Procédure TriSelection( T :tableau d'entiers , n : entier)
Var
  I,j,min,tmp : entiers ;
Debut
   Pour i :=0  à n-1  faire   n fois
     min :=i ;
      pour j :=i+1 à n faire   (n-(i+1)+1) fois
          Si (t[j] < t[min])  alors
            min = j;
            tmp = t[i];
            t[i] = t[min];
            t[min] = tmp;
            min=i;
          Finsi
      Fait
   Fait
Fin
```

FIGURE 2 – pseudo code du tri par selection.

Complexité de l'algorithme : Le pire cas qui nécessite le plus long temps, et le temps pris par cet algorithme pour trier n éléments dépend de nombre de comparaisons effectuées entre

éléments du tableau. Dans ce cas, l'algorithme exécute entièrement les deux boucle imbriquées dans toutes les cas. La complexité est donc en $O(n^2)$.

Tri par insertion : L'idée du tri par insertion et de trier le tableau de gauche à droite en insérant à chaque fois l'élément i+1 dans le tableau (déjà trié) des premiers éléments.[2]

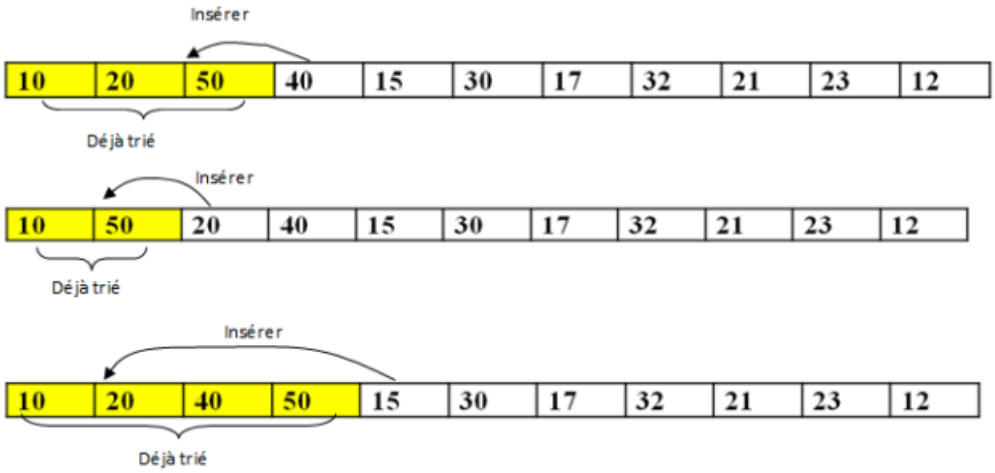

FIGURE 3 – Schéma de Tri par insertion.

Pseudo code :

```
Procédure 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 ;
    Fintq
   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 4 – pseudo code d'algorithme du tri par insertion

Complexité de l'algorithme :

Dans le pire des cas, avec des données triées à l'envers, les parcours successifs du tableau imposent d'effectuer (n-1)+(n-2)+(n-3)..+1 comparaisons et échanges, soit (n2-n)/2. On a donc une complexité dans le pire des cas du tri par insertion en (n^2).

Tri par bulles : L'idée du trie à bulle est de parcourir le tableau T de taille N du dernier au premier élément, avec un indice i. A chaque étape, la partie du tableau située à droite de i est considérée comme triée. On parcourt alors la partie de gauche (partie non triée) avec un indice j. Pour chaque j, si T[j 1] > T[j], on les permute.[5]

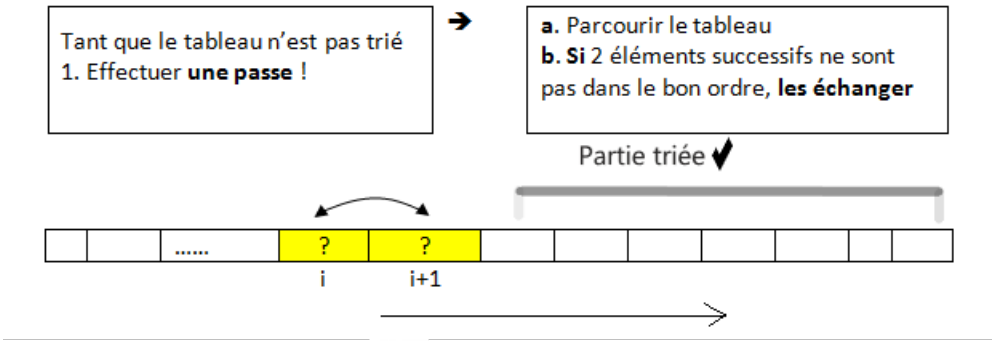

FIGURE 5 – Schéma de Tri par bulles.

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 6 – pseudo code d'algorithme du tri a bulle

Complexité de l'algorithme : Dans le pire des cas, avec des données triées à l'envers, les parcours successifs du tableau imposent d'effectuer (n2-n)/2 comparaisons et échanges. On a donc une complexité dans le pire des cas du tri bulle en $O(n^2)$.

Tri rapide : L'idée du trie rapide est de 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 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.[4]

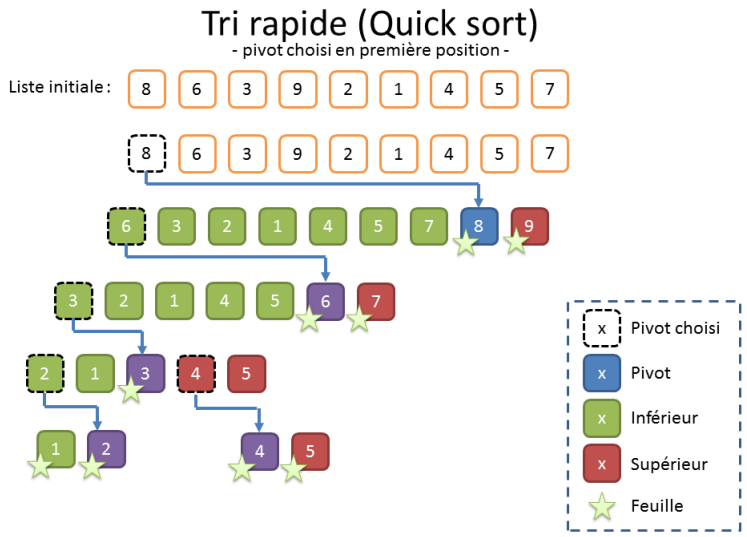

FIGURE 7 – Schéma de tri rapide

Pseudo code :

```
procédure TriRapide (E/S t : Tableau[1..MAX] d'Entier; gauche, droit : entier)
var  i, j ,pivot, x : Entiers
Début
        i ← gauche;
        j←droit;
       pivot←choixpivot()// il existe plusieurs méthode pour choisir le pivot
répéter
            tant que t[i] <pivot  faire i ←i+1 fintant que
            tant que t[j] >pivot faire j←j-1 fintant que
            si (i<=j) alors  permutation(t[i] ,t[j]);
                    i← i+1 ;
                    j←j-1 ;
            finsi
jusqu'à ce que i> j ;

          Si  gauche < j alors TriRapide(t, gauche, j)   finsi ;
          Si  i<droit alors TriRapide(t, i, droit) finsi ;
Fin
```

FIGURE 8 – pseudo code du tri rapide

Complexité de l'algorithme :

Le meilleur cas : le meilleur cas est atteint lorsque le processus de division choisit toujours la médiane comme pivot. Donc à chaque fois la taille des sous-tableaux se divise sur deux jusqu'à arriver à des sous-tableaux avec taille qui égale à 1 élément. On précise que chaque étape nécessite n opérations. Et donc la complexité devient O (n log n).

Le pire cas : c'est lorsque le tableau est trié dans le sens inverse et le pivot obtenu correspond au plus grand élément ou au plus petit élément et donc dans ce cas, le tri d'une liste de n éléments se divise en un tri d'une liste de 0 éléments et une liste de n 1 éléments. Alors trier une liste de n 1 se divise en une liste de taille 0 et une liste de taille n 2, etc. et donc la complexité théorique de ce algorithme devient $O(n^2)$.

Tri fusion : Il s'agit à nouveau d'un tri suivant le paradigme diviser pour régner.[6] Le principe du tri par fusion (ou tri par interclassement) en est le suivant :

- On divise en deux moitiés la liste à trier (en prenant par exemple, un élément sur deux pour chacune des listes).
- On trie chacune d'entre elles.
- On fusionne les deux moitiés obtenues pour reconstituer la liste triée.

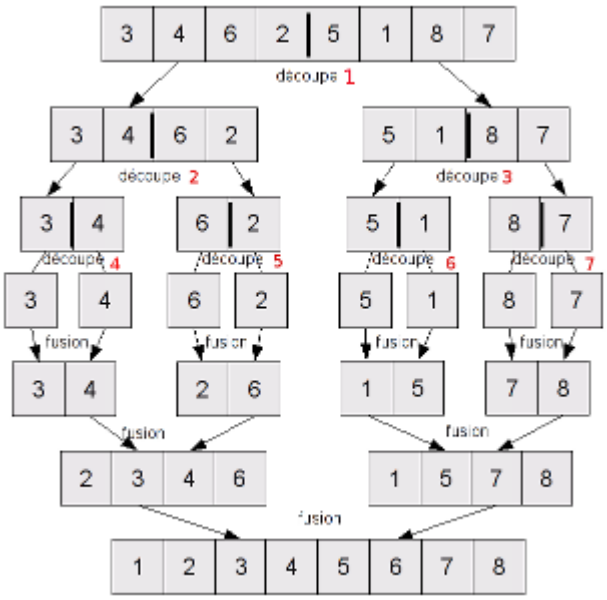

FIGURE 9 – Schéma de tri fusion

Pseudo code :

```
Procedure  triFusion(E/S t:Tableau[1..MAX] d'Entier,deb: entier, fin:entier )
Var   m:entier ;
Début
            Si (deb < fin ) alors
                    m ←(deb+fin) / 2 ;
                triFusion( t, deb, m ) // trier partie gauche
                triFusion( t, m+1, fin) // trier partie droite
                fusionner (t,deb,m,fin) ;
            Finsi ;
Fin
```

FIGURE 10 – pseudo code du tri fusion

Complexité de l'algorithme : La complexité dans le pire des cas du tri fusion est en O(n log n) car si n>1 on a 2 appels récursifs —> 2* T (n/2) et aussi le fusion en O(n) donc on a 2*T(n/2) + O(n) qui donne O(n log n).

Tri par tas : Le tri par tas se base sur une structure de données particulière : le tas. Il s'agit d'une représentation d'un arbre binaire sous forme de tableau. L'arbre est presque complet : il est rempli à tous les niveaux, sauf potentiellement le dernier. Le parcours de l'arbre se fait par un calcul d'indice sur le tableau. Pour un noeud d'indice i, on a le père à l'indice , le fils gauche à l'indice 2i et le fils droit à l'indice 2i+1.

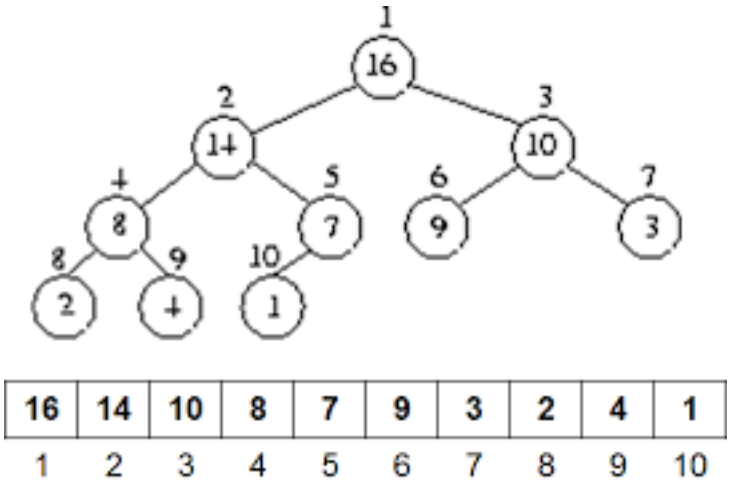

FIGURE 11 – Schéma de tri par tas

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

```
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
     Else  imax :=2*i;   //guauche
        Si  (T[imax]>T[i]) alors
          echanger (T, imax, i);
          redescendre (T, n, imax);
        Finsi ;
   Finsi ;
Fin
```

FIGURE 12 – 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 13 – pseudo code du tri par tas suite

Complexité de l'algorithme : Le pire cas à la même complexité qu'au meilleur cas, car on fait l'appel récursive N fois, est dans cette dernière le parcours du tableau se fait deux éléments à la fois, chaque élément à deux sous tableau est donc la complexité devient O (n log(n)).

0.3 Question 02

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

Réponse 02 :

Algorithmes	Pire cas	Moyen cas	meilleur cas
Tri par Sélection	Le pire cas qui nécessite le plus long temps, et le temps pris par cet algo pour trier n éléments dépend de nombre de comparaisons effectuées entre éléments du tableau. Dans ce cas, l'algorithme exécute entièrement les deux boucle imbriquées dans toutes les cas. La complexité est donc en $O(n^2)$.	Dans toutes les cas l'algorithme exécute entièrement les deux boucle imbriquées. La complexité est donc en $O(n^2)$.	soit en pire, moyen et meilleur cas l'algorithme exécute entièrement les deux boucle imbriquées. La complexité est aussi en $O(n^2)$.
Tri par insertion	Dans le pire des cas, avec des données triées à l'envers, les parcours successifs du tableau imposent d'effectuer (n-1)+(n-2)+(n-3)..+1 comparaisons et échanges, soit (n*n-n)/2. On a donc une complexité dans le pire des cas du tri par insertion en $O(n^2)$.	Si tous les éléments de la série à trier sont distincts et que toutes leurs permutations sont équiprobables, la complexité en moyenne de l'algorithme est de l'ordre de (n*n-n)/4 comparaisons et échanges. La complexité en moyenne du tri par insertion est donc également en (n^2).	Dans le meilleur des cas, avec des données déjà triées, l'algorithme effectura seulement n comparaisons. Sa complexité dans le meilleur des cas est donc en (n).
Tri a bulles	Dans le pire des cas, avec des données triées à l'envers, les parcours successifs du tableau imposent d'effectuer (n*n-n)/2 comparaisons et échanges. On a donc une complexité dans le pire des cas du tri bulle en (n^2).	Si tous les éléments de la série à trier sont distincts et que toutes leurs permutations sont équiprobables, la complexité en moyenne de l'algorithme est de l'ordre de (n*n-n)/4 comparaisons et échanges. La complexité en moyenne du tri bulle est donc également en (n^2)	Dans le meilleur des cas, avec des données déjà triées, l'algorithme effectura seulement n - 1 comparaisons. Sa complexité dans le meilleur des cas est donc en O(n).

TABLE 1 – Etude de meilleur, moyen et pire cas pour chaque méthode de tri (selection, insertion, a bulles)

Algorithmes	Pire cas	Moyen cas	meilleur cas
Tri rapide	Dans le pire des cas, c'est à dire si, à chaque niveau de la récursivité le découpage conduit à trier un sous-tableau de 1 élement et un sous-tableau contenant tout le reste, la complexité du tri rapide est en $O(n^2)$.	Par contre, dans le cas moyen, cet algorithme a une complexité en O(nlog n)	le meilleur cas est atteint lorsque le processus de division choisit toujours la médiane comme pivot. Donc à chaque fois la taille des sous-tableaux se divise sur deux jusqu'à arriver à des sous-tableaux avec taille qui égale à 1 élément. On précise que chaque étape nécessite n opérations. Et donc la complexité devient O(n log n).
Tri fusion	La complexité dans le pire des cas du tri fusion est en O(n log n) car si n>1 on a 2 appels récursifs —> 2* T (n/2) et aussi le fusion en O(n) donc on a 2*T(n/2) + O(n) qui donne O(n log n).	La complexité dans le cas moyen du tri fusion est en O(n log n) car le processus récursif ne dépend pas de l'instance mais uniquement de sa taille n.	soit en pire, moyen et meilleur cas le processus récursif ne dépend pas de l'instance mais uniquement de sa taille n et donc La complexité est aussi en O(n log n).
Tri par tas	La complexité du tri par tas dans le pire cas est égale à la complexité de la construction d'un tas qui est de l'ordre n log(n) , donc la complexité égale à O(n log(n)).	La complexité du tri par tas dans le moyen cas est égale à la complexité de la construction d'un tas qui est de l'ordre n log(n) , donc la complexité égale à O(n log(n)).	Le meilleur cas à la même complexité qu'au pire cas, car on fait l'appel récursive N fois, est dans cette dernière le parcours du tableau se fait deux éléments à la fois, chaque élément à deux sous tableau est donc la complexité devient O(n log(n)).

TABLE 2 – Etude de meilleur, moyen et pire cas pour chaque méthode de tri (rapide, fusion, par tas)

0.4 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).

Réponse 03 :

0.4.1 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	2203.999	56558.998	228468.994	long	long	long	long

TABLE 3 – Temps d'éxécution du tri par selection bon ordre en "ms".

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	0.00	0.00	0.00	0.00	0.00	5.00	0.00	0.00	0.00

TABLE 4 – Temps d'éxécution du tri par insertion bon ordre en "ms".

Tri a 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	12.00	70.999	1802.00	31016.00	202733.99	long	long	long	long

TABLE 5 – Temps d'éxécution du tri a bulles bon ordre en "ms".

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	0.00	0.00	0.00	3.00	7.00	50.00	78.00	long	long

TABLE 6 – Temps d'éxécution du tri rapide bon ordre en "ms".

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	0.00	5.00	17.99	50.00	126.00	637.00	1271.99	long	long

TABLE 7 – Temps d'éxécution du tri par fusion bon ordre en "ms".

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	0.00	3.00	5.00	24.00	52.00	326.00	661.00	long	long

TABLE 8 – Temps d'éxécution du tri par tas bon ordre en "ms".

0.4.2 Ordre inversé :

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	83.99	2184.99	8579.00	220391.01	8807501.40	long	long	long	long

TABLE 9 – Temps d'éxécution du tri par selection ordre inversé en "ms".

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	2512.00	9887.00	253740.99	10178621.00	long	long	long	long

TABLE 10 – Temps d'éxécution du tri par insertion ordre inversé en "ms".

Tri a 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	165.99	309.00	17399.00	443002.99	long	long	long	long	long

TABLE 11 – Temps d'éxécution du tri a bulles ordre inversé en "ms".

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	0.00	2.00	3.00	17.00	32.00	239.99	397.00	2259.00	long

TABLE 12 – Temps d'éxécution du tri rapide ordre inversé en "ms".

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	2.00	9.00	18.99	82.00	182.99	981.00	1899.00	10142.00	long

TABLE 13 – Temps d'éxécution du tri par fusion ordre inversé en "ms".

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	1.00	6.00	14.00	71.99	164.00	1019.00	2137.00	12373.00	long

TABLE 14 – Temps d'éxécution du tri par tas ordre inversé en "ms".

0.4.3 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	104.99	509.00	9519.00	55201.01	975940.02	long	long	long	long

TABLE 15 – Temps d'éxécution du tri par selection ordre Aléatoire en "ms".

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	54.00	294.99	5202.00	31066.99	509996.01	long	long	long	long

TABLE 16 – Temps d'éxécution du tri par insertion ordre Aléatoire en "ms".

Tri a 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 165.00	2720.00	24430.00	285277.00	long	long	long	long	long	

TABLE 17 – Temps d'éxécution du tri a bulles ordre Aléatoire en "ms".

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	0.00	1.00	7.00	21.00	78.99	211.99	455.00	2535.00	long

TABLE 18 – Temps d'éxécution du tri rapide ordre Aléatoire en "ms".

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	2.00	9.00	18.99	82.00	182.99	981.00	1899.00	10142.00	long

TABLE 19 – Temps d'éxécution du tri par fusion ordre Aléatoire en "ms".

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	1.00	0.00	20.00	55.99	219.99	783.99	1843.99	13454.00	long

TABLE 20 – Temps d'éxécution du tri par tas ordre Aléatoire en "ms".

0.5 Question 04 :

Représenter ces mesures dans un tableau puis avec un graphe. Que pouvez-vous conclure ? Les representation dans un tableaux ont les a effectuer dans la question précédente,passons a la representation graphique :

Représentation graphique :

0.5.1 Bon ordre :

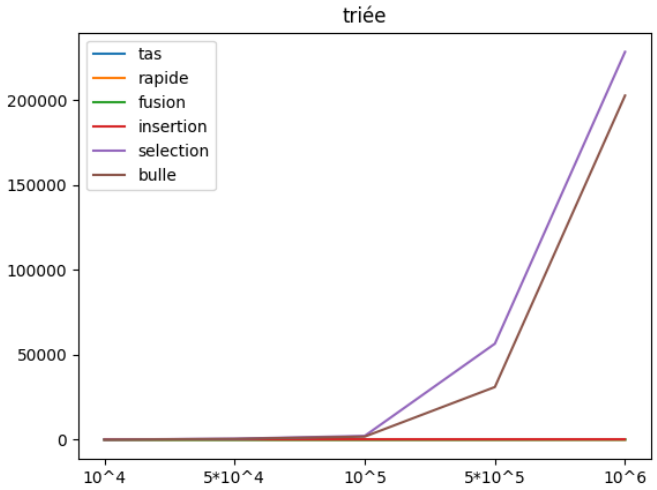

FIGURE 14 – Représentation graphique des résultats d'exécution des 6 algo et les elements triées en bon ordre

Analyse :

D'abord, on remarque à travers ce dernier graphe [Figure 2.15] que les courbes des algorithmes de (tri rapide et fusion et par tas et par insertion) sont totalement identiques rapproche

au t=0.00s. Et même pour les algorithmes de (tri par selection et a bulle) sont presque similaires.

Ensuite, on remarque que pour les algorithmes (tri par selection et a bulle)... plus les séquences de données augmentent plus les temps d'exécution augmentent et donc on peut dire qu'ils sont dans une relation de corrélation directe avec la longeur des nombres, mais par rapport aux algorithmes de (tri rapide et fusion et par tas et par insertion) leur temps d'exécution se rapproche de 0.00000 s.

L'analyse a révélé que les Algorithmes de (tri rapide et fusion et par tas et par insertion) vue à leur temps d'exécution sont très rapides par rapport les autres algorithmes et surtout algorithme de tri par selection qui est le plus long a cause du grands nombres d'itérations et comparisons exécutées. Et donc **les Algorithmes de tri par insertion et selection et le tri par tas et le tri rapide sont beaucoup plus optimaux et efficaces dans le cas au les éléments sont triées en bon ordre.**

0.5.2 Ordre inversé :

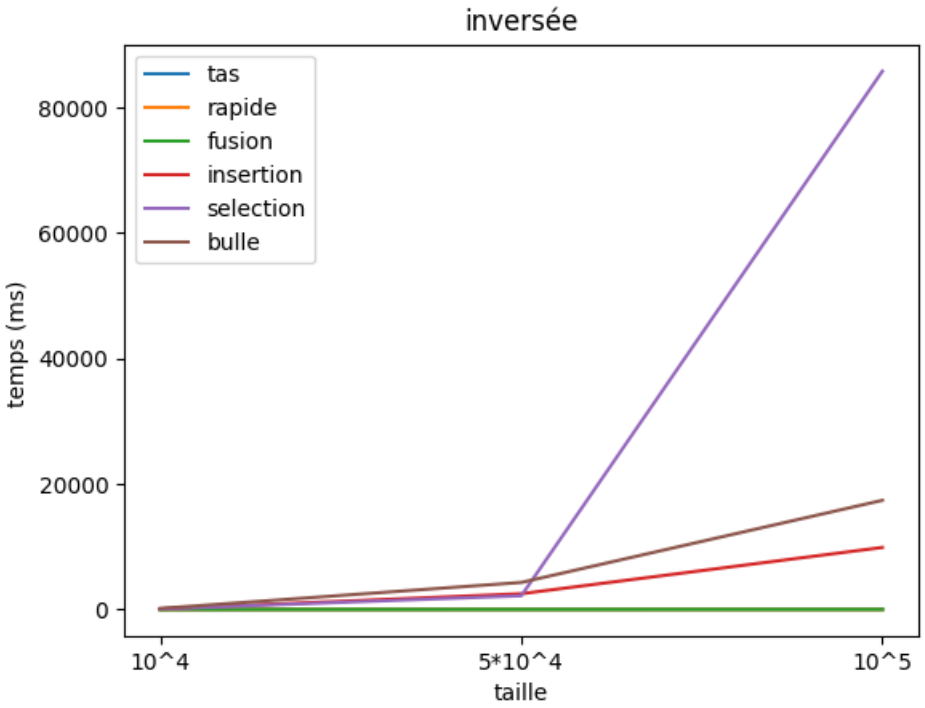

FIGURE 15 – Représentation graphique des résultats d'exécution des 6 algo et les elements triées en ordre inversé

Analyse :

D'abord, on remarque à travers ce dernier graphe [Figure 2.14] que les courbes des algorithmes de (tri rapide et fusion et par tas) sont totalement identiques rapproche au t=0.00s. Et même pour les algorithmes de (tri insertion et selection et a bulle) sont presque similaires.

Ensuite, on remarque que pour les algorithmes (tri insertion et selection et a bulle)... plus les séquences de données augmentent plus les temps d'exécution augmentent et donc on peut dire qu'ils sont dans une relation de corrélation directe avec la longeur des nombres, mais par rapport aux algorithmes de (tri rapide et fusion et par tas) d leur temps d'exécution se rapproche de 0.00000 s.

L'analyse a révélé que les Algorithmes de (tri rapide et fusion et par tas) vue à leur temps d'exécution sont très rapides par rapport les autres algorithmes et surtout algorithme de tri par selection qui est le plus long a cause du grands nombres d'itérations et comparisons exécutées. Et donc **les Algorithmes de tri par insertion et selection et le tri a bulle sont beaucoup plus optimaux et efficaces dans le cas au les éléments sont triées en bon ordre.**

0.5.3 Ordre aléatoire :

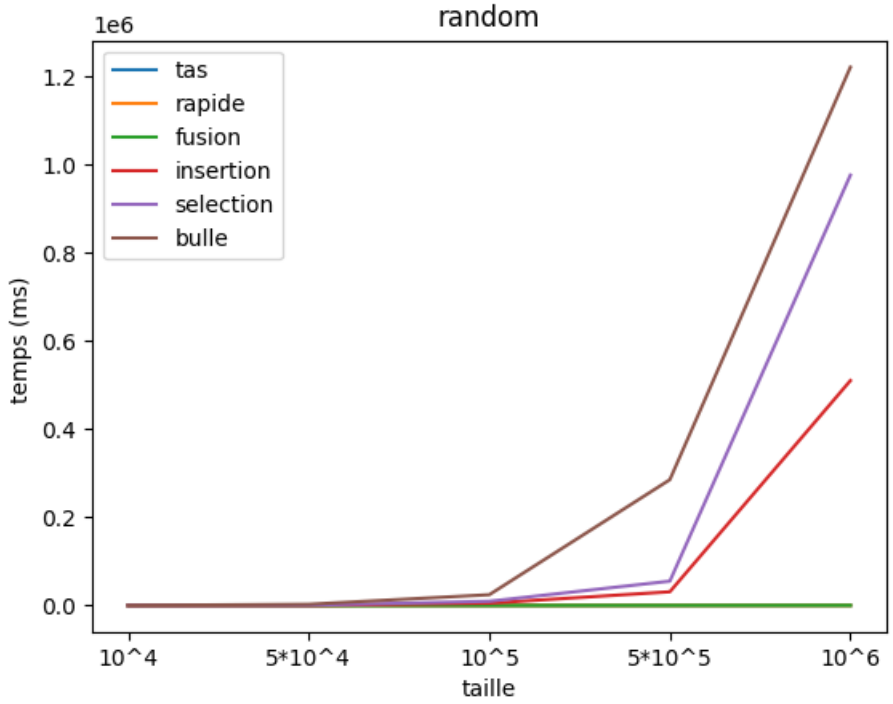

FIGURE 16 – Représentation graphique des résultats d'exécution des 6 algo et les elements aléatoire

Analyse :

D'abord, on remarque à travers ce dernier graphe [Figure 2.14] que les courbes des algorithmes de (tri rapide et fusion et par tas) sont totalement identiques rapproche au t=0.00s. Et même pour les algorithmes de (tri insertion et selection et a bulle) sont presque similaires.

Ensuite, on remarque que pour les algorithmes (tri insertion et selection et a bulle)... plus les séquences de données augmentent plus les temps d'exécution augmentent et donc on peut dire qu'ils sont dans une relation de corrélation directe avec la longeur des nombres, mais par rapport aux algorithmes de (tri rapide et fusion et par tas) d leur temps d'exécution se rapproche de 0.00000 s.

L'analyse a révélé que les Algorithmes de (tri rapide et fusion et par tas) vue à leur temps d'exécution sont très rapides par rapport les autres algorithmes et surtout algorithme de tri a bulles qui est le plus long a cause du grands nombres d'itérations et comparisons exécutées. Et donc **les Algorithmes de tri par insertion et selection et le tri a bulle sont beaucoup plus optimaux et efficaces dans le cas au les éléments sont triées en bon ordre.**

0.6 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 ?

0.6.1 Bon ordre :

On a choisit un fichier random de taille 10^4

Taille du tableau	nombre de comparaisons
Tri par sélection	49995000
Tri par insertion	9999
Tri a bulles	49995000
Tri rapide	49995000
Tri par fusion	69008
Tri par tas	180584

TABLE 21 – nombre de comparaisons des 6 algos de tri et les valeur sont trier.

Représentation graphique :

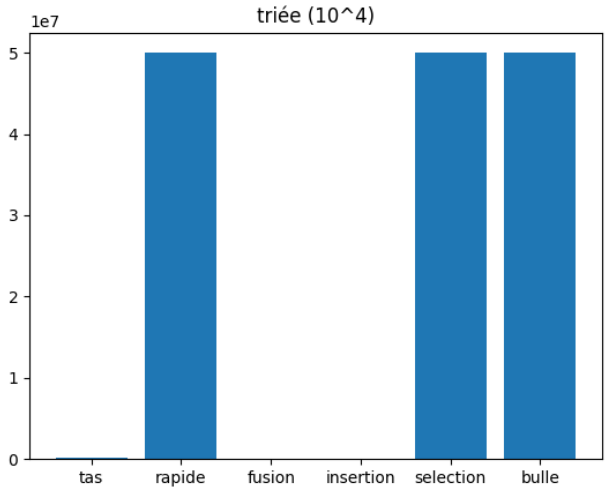

FIGURE 17 – Représentation graphique des résultats d'exécution des 6 algo et les elements triées en bon ordre

0.6.2 Ordre inversé :

On a choisit un fichier random de taille 10^4

Taille du tableau	nombre de comparaisons
Tri par sélection	49995000
Tri par insertion	49995000
Tri a bulles	49995000
Tri rapide	24995000
Tri par fusion	64608
Tri par tas	153619

TABLE 22 – nombre de comparaisons des 6 algos de tri et les valeur sont inversé.

Représentation graphique :

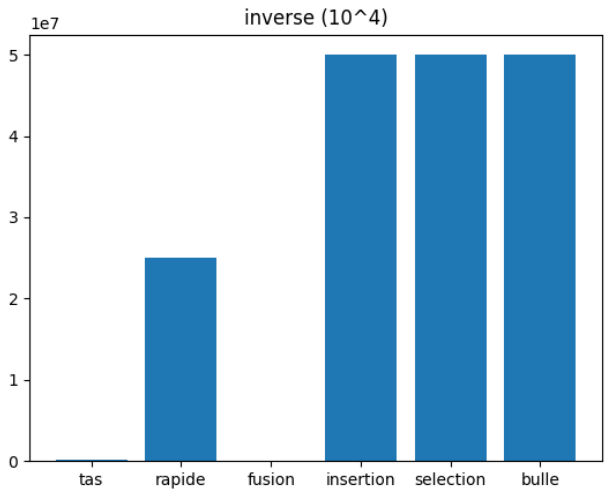

FIGURE 18 – Représentation graphique des résultats d'exécution des 6 algo et les elements triées en ordre inversé

0.6.3 Ordre Aléatoire :

On a choisit un fichier random de taille 10^4

Taille du tableau	nombre de comparaisons
Tri par sélection	49995000
Tri par insertion	25351212
Tri a bulles	49995000
Tri rapide	81273
Tri par fusion	120428
Tri par tas	166422

TABLE 23 – nombre de comparaisons des 6 algos de tri et les valeur sont aléatoires.

Représentation graphique :

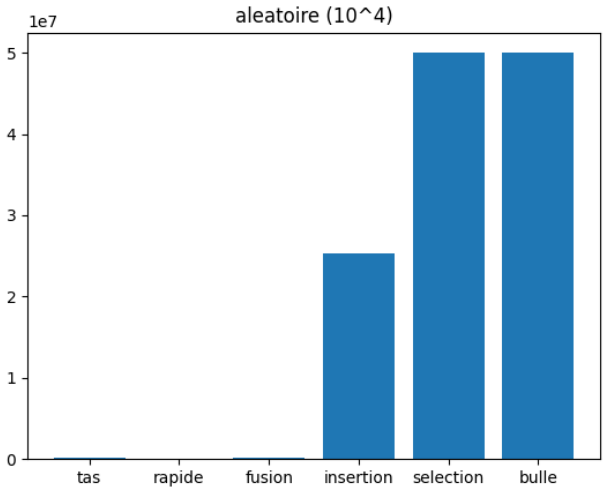

FIGURE 19 – Représentation graphique des résultats d'exécution des 6 algo et les elements aléatoire

Analyse globale :

On remarque à partir de ces derniers graphes qu'on peut dire qu'il y a des algorithmes qui sont les plus optimales tels que (par tas ,par fusion) qui offrent du meilleur performance dans tous les cas (trié ,inversé,random). Et il y a des algoritumes tels que (bulle ,selection) qui peuvent pas etre un bon choix dans tout les cas. Et il y a ceux qui dependent sur la nature du tableau comme le tri rapide et le tri par insertion.

On déduit que le tri par tas est plus efficace que les autres algorithmes de tri car il effectue le moins de comparaison. Les algorithmes de tri fusion et de tri rapide sont plus efficaces avec des instances de données non trié alors que les algorithmes de tri par selection, bulle et insertion sont tres coûteuse grace a leurs nombre de comparison qui est intéressent.

Troisième partie

Environnement expérimental

Caractéristiques de la machine	Système d'exploitation	CPU	RAM	Version compilateur C
BENKOUITEN Aymen	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz	8,00 GO	codeblocks-20.03mingw
KENAI Imad Eddine	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz	16,00 GO	codeblocks-8.1.0mingw
MALKI Omar Chouaab	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz	8,00 GO	codeblocks-20.03mingw
MEKKAOUI Mohamed	Système d'exploitation 64 bits, processeur x64. Windows 11 Home V-22H2.	Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz	16,00 GO	codeblocks-20.03mingw

Table 24 – Environnement expérimental

0.7 Répartition des tâches :

- Écriture du code source : Tout le Team
- La rédaction du rapport et son organisation avec Latex : BENKOUITEN
- Exécution des taches : MEKKAOUI + MALKI + BENKOUITEN
- Représentation graphique : KENAI
- NB : Tous ce travail est fait durant un meet entre le team et toutes les réponses et les analyses effectuer sur les graphes sont bien été discuter avant la rédaction de rapport.

Conclusion Générale

Les résultats de notre analyse des résultats obtenus ainsi que le calcul théorique de la complexité temporelle des six algorithmes de tri, nous a permis de comparer l'efficacité des six algorithmes de tri et établir lequel d'entre eux est le optimal et le plus rapide.

On conclut que 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 avec une complexité $O(n^2)$ sont donc beaucoup moins utilisé.

Annexe

Code source des algorithmes :

Algorithme de tri par sélection :

```c
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;


    for (i = 0; i < n-1; i++)
    {
        // trouver le min
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // permuter le min
            if(min_idx != i)
             swap(&arr[min_idx], &arr[i]);
    }
}
```

FIGURE 20 – code source d'algorithme de tri par sélection

Algorithme de tri par insertion :

```c
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;


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

FIGURE 21 – code source d'algorithme de tri par insertion

Algorithme de tri a bulles :

```
// tri a bulle
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)


        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}
```

FIGURE 22 – code source d'algorithme de tri a bulles

Algorithme de tri rapide methode 1 :

```
void quickSort(int array[], int low, int high) {
  if (low < high) {
    //int pi = partition_pivot_first(array, low, high);
    //int pi = partition_pivot_last(array, low, high);
    //int pi = partition_pivot_random(array, low, high);
    int pi = partition_pivot_median(array, low, high);

    // Sort the elements on the left of pivot
    quickSort(array, low, pi - 1);

    // Sort the elements on the right of pivot
    quickSort(array, pi + 1, high);
  }
}
```

FIGURE 23 – code source d'algorithme de tri rapide méthode 1

Algorithme de tri rapide les 3 autres methodes :

```c
int partition_pivot_last(int array[], int low, int high) {
  int pivot = array[high];
  int i = (low - 1);

  for (int j = low; j < high; j++) {
    if (array[j] < pivot) {
      swap(&array[++i], &array[j]);
    }
  }
  swap(&array[i + 1], &array[high]);
  return (i + 1);
}

int partition_pivot_first(int array[], int low, int high) {
  int pivot = array[low];
  int i = (low + 1);

  for (int j = low + 1; j <= high; j++) {
    if (array[j] < pivot) {
      if (j != i) {
        swap(&array[i], &array[j]);
      }i++;
    }
  }

  swap(&array[i - 1], &array[low]);
  return (i - 1);
}

int partition_pivot_random(int array[], int low, int high) {
  int pivot;
  int n = rand();
  pivot = low + n % (high - low + 1); // Randomizing the pivot
  return partition_pivot_last(array, low, high);
}

int partition_pivot_median(int array[], int low, int high) {

  int pivot;
  int mid = (low + high) / 2;
  if (array[mid] < array[low])
    swap(&array[mid], &array[low]);
  if (array[high] < array[low])
    swap(&array[high], &array[low]);
  if (array[high] < array[mid])
    swap(&array[high], &array[mid]);
  swap(&array[mid], &array[high-1]);

  pivot = array[high-1];

  return partition_pivot_last(array, low, high);
}
```

FIGURE 24 – code source d'algorithme de tri rapide méthode 2 et 3 et 4

Algorithme de tri par fusion :

```
void mergeSort(int arr[], int l, int r){
  if (l < r) {


    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);


    merge(arr, l, m, r);
  }
}
```

FIGURE 25 – code source d'algorithme de tri par fusion "main"

```c
void merge(int arr[], int p, int q, int r) {


  int n1 = q - p + 1;
  int n2 = r - q;

  int *L=malloc(sizeof(int)*n1);
  int *M=malloc(sizeof(int)*n2);

  for (int i = 0; i < n1; i++)
    L[i] = arr[p + i];
  for (int j = 0; j < n2; j++)
    M[j] = arr[q + 1 + j];

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

  while (i < n1 && j < n2) {
    if (L[i] <= M[j]) {
      arr[k] = L[i];
      i++;
    } else {
      arr[k] = M[j];
      j++;
    }
    k++;
  }


  while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
  }

  while (j < n2) {
    arr[k] = M[j];
    j++;
    k++;
  }
  free(L);free(M);
}
```

FIGURE 26 – code source d'algorithme de tri par fusion "function"

Algorithme de tri par tas :

```
void heapSort(int arr[], int N) /*Tas max*/
{


    for (int i = N / 2 - 1; i >= 0; i--)

        heapify(arr, N, i);


    for (int i = N - 1; i >= 0; i--) {

        swap(&arr[0], &arr[i]);


        heapify(arr, i, 0);
    }
}
```

FIGURE 27 – code source d'algorithme de tri par tas "main"

```
void heapify(int arr[], int N, int i)
{

    int largest = i;


    int left = 2 * i + 1;


    int right = 2 * i + 2;

    if (left < N && arr[left] > arr[largest])

        largest = left;

    /
    if (right < N && arr[right] > arr[largest])

        largest = right;


    if (largest != i) {

        swap(&arr[i], &arr[largest]);

        |
        heapify(arr, N, largest);
    }
}
```

FIGURE 28 – code source d'algorithme de tri par tas "function"

Bibliographie

[1] definition de tri. https://fr.wikipedia.org/wiki/Algorithme_de_tri.

[2] Tri par insertion. http://lwh.free.fr/pages/algo/tri/tri_insertion.html.

[3] Tri par sélection. http://lwh.free.fr/pages/algo/tri/tri_selection.html.

[4] Tri rapide. http://lwh.free.fr/pages/algo/tri/tri_rapide.html.

[5] Tri à bulles. http://lwh.free.fr/pages/algo/tri/tri_bulle.html.

[6] Tri à fusion. http://lwh.free.fr/pages/algo/tri/tri_fusion.html.