

**Module : algorithmique**

**Compte rendu**

Sujet :

**Les tries**

( à bulles/insertion/sélection/rapide/Extraction )

Filière d’ingénieur :

**I**ngénierie **L**ogicielle et **I**ntégration des **S**ystèmes **I**nformatiques

Encadrant :

* Pr. Abdelkrim BEKKHOUCHA

Réalisé par :

* SOUNEINI Salma
* ANEBA Meriem

*Année universitaire : 2023-2024*

**SOMMAIRE**

[SOMMAIRE 1](#_Toc153924344)

[INTRODUCTION 2](#_Toc153924345)

[PROBLEME 2](#_Toc153924346)

[I. Trie à bulles : 3](#_Toc153924347)

[1. Analyse : 3](#_Toc153924348)

[2. Exemple d’exécution "Trace" : 3](#_Toc153924349)

[3. Analyse fonctionnelle : 3](#_Toc153924350)

[II. Trie par insertion: 5](#_Toc153924351)

[1. Analyse : 5](#_Toc153924352)

[2. Exemple d’exécution "Trace" : 5](#_Toc153924353)

[3. Analyse fonctionnelle : 5](#_Toc153924354)

[III. Tri sélection-Échange : 6](#_Toc153924355)

[1. Analyse: 6](#_Toc153924356)

[2. Exemple d’exécution "Trace" : 6](#_Toc153924357)

[3. Analyse fonctionnelle : 7](#_Toc153924358)

[IV. Tri rapide : 7](#_Toc153924359)

[1. Analyse: 7](#_Toc153924360)

[2. Exemple d’exécution "Trace" : 7](#_Toc153924361)

[3. Analyse fonctionnelle : 8](#_Toc153924362)

[V. Tri par extraction: 9](#_Toc153924363)

[1. Analyse: 9](#_Toc153924364)

[2. Exemple d’exécution "Trace" : 9](#_Toc153924365)

[3. Analyse fonctionnelle : 9](#_Toc153924366)

[VI. DOSSIER DE PROGRAMMATION 10](#_Toc153924367)

[1. Liste statique : 10](#_Toc153924368)

[2. Liste dynamique : 11](#_Toc153924369)

**INTRODUCTION**

**PROBLEME**

# **Trie à bulles :**

## **Analyse :**

Le tri à bulles opère en traversant la liste séquentiellement, comparant chaque élément avec son successeur. Lorsqu'un élément est plus grand que le suivant, ils sont permutés, permettant aux éléments plus grands de "remonter" progressivement vers la fin de la liste. Ce processus itératif positionne le plus grand élément à la fin à chaque itération. Il se répète jusqu'à ce que tous les éléments soient placés dans leur ordre approprié, créant ainsi une liste entièrement triée. En effet, à chaque passage, les plus petits et les plus grands éléments trouvent leurs positions finales, contribuant à la formation d'une liste ordonnée.

## **Exemple d’exécution "Trace" :**

**Liste initiale :** 6, 3, 8, 2, 9

**Itérations :**

(6 3 8 2 9) : 6 > 3, échange (3 et 6).

(3 6 8 2 9) : 6 < 8, aucun échange.

(3 6 2 8 9) : 8 > 2, échange (2 et 8).

(3 6 2 8 9) : 8 < 9, aucun échange.

(3 6 2 8 9) : La liste est triée.

## **Analyse fonctionnelle :**

**1- Fonction "echange" :**

*1-a) Spécification des données et sorties :*

Entrée : Deux pointeurs vers des entiers (int \*a, int \*b).

Sortie : La fonction échange les valeurs des entiers pointés par a et b.

*1-b) Spécification fonctionnelle :*

La fonction "echange" prend en entrée deux pointeurs vers des entiers.

Elle utilise des opérations arithmétiques pour échanger les valeurs des entiers pointés par a et b sans utiliser de variable temporaire.

Elle réalise l'échange en trois étapes :

* Additionne la valeur de a et b et stocke la somme dans a.
* Soustrait la valeur de b de la nouvelle valeur de a pour obtenir la valeur originale de b, et stocke ce résultat dans b.
* Soustrait la valeur de b de la nouvelle valeur de a pour obtenir la valeur originale de a, et stocke ce résultat dans a.
* Après l'exécution de la fonction, les valeurs des entiers pointés par a et b sont échangées.

Le but de cette fonction est de réaliser une permutation sans avoir besoin d’utilisation de 3ème variable

**2- Fonction "trie\_bulle"**

*2-a) Spécification des données et sorties :*

Entrée : Un tableau matab contenant des éléments à trier.

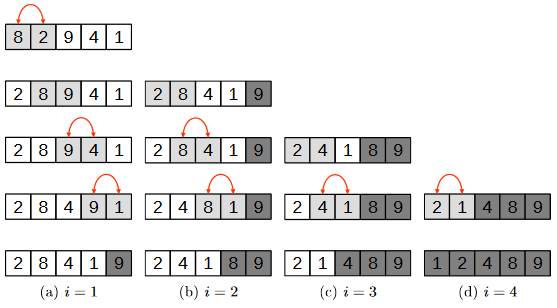
Sortie : La fonction trie le tableau matab par ordre croissant.

*2-b) Spécification fonctionnelle :*

* La fonction "trie\_bulle" utilise l'algorithme de tri à bulles pour trier les éléments du tableau matab.
* Elle parcourt le tableau en comparant chaque élément avec son successeur.
* Si un élément est plus grand que le suivant, elle effectue un échange.
* Elle répète ce processus jusqu'à ce que tous les éléments soient triés par ordre croissant.
* La variable "k" compte le nombre d'échanges effectués, représentant ainsi le nombre d'itérations nécessaires pour trier le tableau.

La fonction utilise la fonction "echange" pour échanger les éléments du tableau.

**Explication en utilisant une image qui illustre le principe de trie à bulles :**



# **Trie par insertion:**

## **Analyse :**

Le tri par insertion parcourt la liste en insérant chaque élément à sa place appropriée dans la section déjà triée de la liste. Initialement, considérant le premier élément comme une liste triée de taille 1, il insère ensuite les éléments suivants dans cette section triée. À chaque itération, il prend un élément non trié et le compare avec les éléments de la section triée, le plaçant à sa position correcte en décalant les éléments plus grands.

## **Exemple d’exécution "Trace" :**

**Liste initiale** : 5, 2, 4, 6, 1, 3

**Itérations :**

(2 5 4 6 1 3) : Insère 2 à sa place.

(2 4 5 6 1 3) : Insère 4 à sa place.

(2 4 5 6 1 3) : Insère 6 à sa place.

(1 2 4 5 6 3) : Insère 1 à sa place.

(1 2 3 4 5 6) : Insère 3 à sa place.

La liste est triée.

## **Analyse fonctionnelle :**

**1- Fonction "trie\_par\_insertion"**

*1-a) Spécification des données et sorties :*

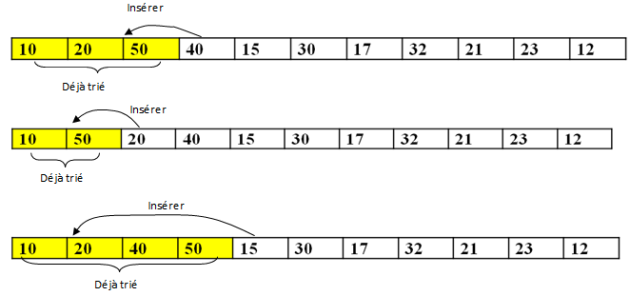
Entrée : Un tableau matab contenant des éléments à trier.

Sortie : La fonction trie le tableau matab par ordre croissant.

*1-b) Spécification fonctionnelle :*

* La fonction "trie\_par\_insertion" utilise l'algorithme de tri par insertion pour trier les éléments du tableau matab.
* Elle commence par considérer le premier élément du tableau comme une liste triée de taille 1.
* Elle itère sur les éléments restants, prenant chaque élément comme élément courant (crt).
* Elle insère l'élément courant dans la portion triée du tableau, en le comparant à ses prédécesseurs et en décalant les éléments plus grands d'une position vers la droite.
* Elle répète ce processus jusqu'à ce que tous les éléments soient insérés dans la portion triée, formant ainsi un tableau trié.
* La variable "k" compte le nombre de déplacements d'éléments effectués lors de l'insertion, représentant le nombre d'itérations nécessaires pour trier le tableau.
* La fonction utilise une boucle for pour parcourir les éléments du tableau et les insérer dans la portion triée.

**Explication en utilisant une image qui illustre le principe de trie par insertion :**



# **Tri sélection-Échange :**

## **Analyse:**

Le Tri sélection-Échange parcourt la liste en cherchant l'élément le plus petit dans la portion non triée et l'échange avec l'élément à la première position de cette portion. Il étend ainsi progressivement la section triée de la liste en plaçant les éléments les plus petits au début de la liste. À chaque itération, l'algorithme identifie le plus petit élément restant dans la portion non triée et le place dans la section triée.

## **Exemple d’exécution "Trace" :**

**Liste initiale :** 9, 7, 5, 11, 12, 2, 14, 3, 10, 6

**Itérations :**

(2 7 5 11 12 9 14 3 10 6) : Échange 9 et 2.

(2 3 5 11 12 9 14 7 10 6) : Échange 7 et 3.

(2 3 5 6 12 9 14 7 10 11) : Échange 11 et 6.

(2 3 5 6 7 9 14 12 10 11) : Échange 12 et 7.

(2 3 5 6 7 9 10 12 14 11) : Échange 14 et 10.

La liste est triée.

## **Analyse fonctionnelle :**

**2- Fonction "Tri\_Selection\_Tab"**

*2-a) Spécification des données et sorties :*

Entrée : Une liste T contenant des éléments à trier.

Sortie : La fonction trie les éléments de la liste T par ordre croissant.

*2-b) Spécification fonctionnelle :*

* La fonction "Tri\_Selection\_Tab" utilise l'algorithme de tri par sélection pour trier les éléments de la liste T.
* Elle parcourt la liste en sélectionnant à chaque itération la valeur minimale dans la portion non triée du tableau.
* Elle échange cette valeur minimale avec l'élément à la première position de la portion non triée, plaçant ainsi progressivement les éléments les plus petits au début du tableau.
* La variable "k" compte le nombre d'échanges d'éléments effectués pendant le tri, représentant le nombre d'itérations nécessaires pour trier le tableau.
* Elle utilise une fonction "EchangeT" pour échanger les éléments du tableau.

# **Tri rapide :**

## **Analyse:**

Le Tri rapide est un algorithme de tri récursif qui utilise la méthode de partitionnement pour trier la liste. Il choisit un élément appelé "pivot" dans la liste, puis partitionne la liste autour de ce pivot, plaçant tous les éléments plus petits que le pivot à sa gauche et tous les éléments plus grands à sa droite. Ce processus est récursivement appliqué aux sous-listes gauche et droite du pivot jusqu'à ce que toute la liste soit triée.

## **Exemple d’exécution "Trace" :**

**Liste initiale** : 12, 11, 13, 5, 6, 7

**Itérations :**

Étape 1 : Pivot = 7, (5 6 7) à gauche, (13 11 12) à droite.

Étape 2 : Pivot = 6, (5 6) à gauche, (7) à droite.

Étape 3 : Pivot = 12, (11) à gauche, (13) à droite.

La liste est triée.

## **Analyse fonctionnelle :**

**1- Fonction partition1**

*1-a) Spécification des données et sorties :*

Entrée : Une liste maTliste avec des éléments, et les indices i et j délimitant la portion de la liste à considérer.

Sortie : La fonction effectue une partition du tableau et retourne l'indice final du pivot.

*1-b) Spécification fonctionnelle :*

* La fonction partition1 prend en entrée une liste maTliste et deux indices i et j délimitant une portion du tableau.
* Elle sélectionne le premier élément comme pivot.
* Elle parcourt la portion de la liste entre les indices i et j.
* Les éléments plus petits que le pivot sont placés à gauche de celui-ci, en utilisant l'indice compt pour suivre les éléments inférieurs.
* Elle échange les éléments pour placer ceux plus petits à gauche du pivot.
* Elle replace le pivot à sa position finale.
* La fonction renvoie l'indice final du pivot.

**2- Fonction Tri**

*2-a) Spécification des données et sorties :*

Entrée : Une liste maTliste, l'indice gauche et l'indice droite délimitant la portion du tableau à trier.

Sortie : La fonction trie la portion délimitée de la liste.

*2-b) Spécification fonctionnelle :*

* La fonction Tri réalise le tri rapide (quicksort) récursivement sur la portion de la liste délimitée par les indices gauche et droite.
* Elle appelle la fonction partition1 pour obtenir l'indice du pivot et diviser la liste en deux parties.
* Elle trie récursivement les sous-sections gauche et droite du pivot jusqu'à ce que toute la liste soit triée

**3- Fonction TriRapide\_Tab**

*3-a) Spécification des données et sorties :*

Entrée : Une liste maTliste à trier.

Sortie : La fonction trie la liste maTliste en utilisant le tri rapide (quicksort).

*3-b) Spécification fonctionnelle :*

* La fonction TriRapide\_Tab initialise le tri rapide en appelant la fonction Tri avec les indices gauche et droite appropriés.
* Elle mesure le temps d'exécution du tri en utilisant la fonction temps.

# **Tri par extraction:**

## **Analyse:**

Le Tri par extraction, également connu sous le nom de HeapSort, utilise une structure de données appelée tas (heap) pour trier la liste. Il commence par construire un tas à partir des éléments de la liste, puis extrait successivement l'élément maximum (dans le cas d'un tri croissant) du tas, réduisant ainsi la taille du tas à chaque extraction. Cette extraction répétée place les éléments dans l'ordre souhaité

## **Exemple d’exécution "Trace" :**

**Liste initiale :** 4, 10, 3, 5, 1

**Itérations :**

(10 5 3 4 1) : Construit le tas initial.

(5 4 3 1) : Extraie 10 du tas.

(4 1 3) : Extraie 5 du tas.

(3 1) : Extraie 4 du tas.

(1) : Extraie 3 du tas.

La liste est triée.

## **Analyse fonctionnelle :**

**1- Fonction Descente\_tab**

*1-a) Spécification des données et sorties :*

Entrée : Une liste T avec des éléments, la taille de la liste taille, et l'indice du noeud à comparer noeud.

Sortie : La fonction réalise le processus de descente dans un arbre binaire en échangeant les valeurs maximales entre les noeuds.

*1-b) Spécification fonctionnelle :*

* La fonction Descente\_tab implémente le processus de descente dans un arbre binaire représenté par un tableau.
* Elle prend en compte un tableau T, sa taille taille et l'indice noeud pour suivre les valeurs maximales dans l'arbre.
* Elle compare les valeurs des fils gauche et droit par rapport au noeud courant et remplace le noeud par la valeur maximale parmi les noeuds fils s'ils sont plus grands.
* Si la valeur maximale n'est pas le noeud lui-même, elle échange les valeurs et appelle récursivement Descente\_tab pour le nouveau noeud maximal

**2- Fonction Tri\_extraction\_tab**

*2-a) Spécification des données et sorties :*

Entrée : Une liste T à trier en utilisant le tri par extraction.

Sortie : La fonction trie le tableau en utilisant le tri par extraction avec le processus de descente.

*2-b) Spécification fonctionnelle :*

* La fonction Tri\_extraction\_tab implémente le tri par extraction en utilisant un tableau représenté par la structure Tliste.
* Elle construit d'abord un arbre tel que le noeud père est plus grand que ses fils gauche et droit en utilisant la fonction Descente\_tab.
* Elle échange la plus grande valeur avec la dernière position du tableau à chaque itération et diminue la taille du tableau.
* Elle appelle Descente\_tab à chaque échange pour réorganiser le tableau jusqu'à ce que le tableau soit entièrement trié.

# **DOSSIER DE PROGRAMMATION**

## **Liste statique :**

## **Liste dynamique :**

L’objectif du programme est de trier la liste ou bien un tableau en utilisant les diffèrents type de tries .

**Programmation des fonctions de trie pour liste statique :**

1. Librairies :

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

1. Constantes :

#define MaxElem 100

1. Variables principales :

typedef struct TB

{

int TabInt[MaxElem]; /\* Tbleau des données \*/

int NbElem; /\* Nombre d'élemnts dans le tableau \*/

}Tab;

1. Programmation des fonctions de trie pour liste statique:

//fonction qui permet de creer la liste

Tab \*Creer\_Tab()

{

Tab \*matab;

/\*\*allocation dynamique du pointeur\*/

matab = (Tab\*)malloc(sizeof(Tab));

/\*\* Echec d'allocation de mémoire \*/

if (!matab)

{

/\*\* afficher un message d'erreur \*/

printf("\nErreur d allocation de mémoire!!!");

exit(-1);

}

/\*\* si l'allocation est bien fait on retourne le pointeur \*/

return ((Tab\*)matab);

}

/// Fonction du temps d'execution

void temps(clock\_t begin ,clock\_t end)

{

unsigned long tmp;

tmp = (end - begin) \* 1000 / CLOCKS\_PER\_SEC; /// temps en ms

printf("\n le programme a pris comme temps d'execution : %ld ms \n ",tmp);

}

//fonction qui permet d'nitialiser la liste

int Init\_Tab(Tab \*matab)

{

/\*\* Vérification de l'existence du tableau \*/

if (!matab) /\*\* cas où le tab n'existe pas \*/

{

printf("\nErreur d allocation memoire!!!");

exit(-1);/\*\* quitter le programme \*/

}//Fin if

matab->NbElem = 0; /\*\* Initialiser le nombre d'élement à 0 \*/

return ((int)1);

}

//fonction qui permet de tester si la liste est saturer on non

int Est\_Saturee\_Tab(Tab matab)

{

return ((int)(matab.NbElem == MaxElem));

}

//fonction qui permet de tester si la liste est vide on non

int Est\_Vide\_Tab(Tab matab)

{

return ((int)(matab.NbElem == 0));

}

/\* Affiche les éléments du tableau Tab \*/

void afficher\_Tab(Tab matab)

{

int i;

// printf("\nLes elements actuels du tableau:");

if (Est\_Vide\_Tab(matab))/\*\* si le tableau est vide \*/

printf("\nLa table est vide.");/\*\* message d'erreur \*/

/\*\* boucle pour parcourir le tableau case par case \*/

for (i = 0; i < matab.NbElem; i++)

printf("| %d |", matab.TabInt[i]);/\*\* afficher l'element \*/

}

//fonction qui permet d'inserer un element a la fin de la liste "la fin du tableau"

int Inserer\_Fin\_Tab(Tab \*matab, int val)

{

/\*\* vérification de l'existance de tableau \*/

if (!matab)/\*\* s'il n'existe pas \*/

{

printf("\nErreur d'allocation!!!");//afficher message d'erreur

return ((int)-1);

}

if (Est\_Saturee\_Tab(\*matab)) /\*Si La table est saturée \*/

{

printf("\nLe tableau est saturé !!!");//message d'erreur

return ((int)0);

}

/\* Insertion réussie (il y a de la place) \*/

matab->TabInt[matab->NbElem++] = val;

/\* Insertion réussie \*/

return 1;

}

//fonction qui permet de faire l’échange entre deux éléments

void echange(int \*a,int \*b)

{

\*a=\*a+\*b;

\*b=\*a-\*b;

\*a=\*a-\*b;

}

//fonction qui permet de trier les éléments su tableau par trie a bulles.

void trie\_bulle(Tab \*matab)

{

int i;

int n=matab->NbElem;

int a,k=0;

clock\_t begin = clock();

while(n>0)

{

for(i=0;i<matab->NbElem-1;i++)

{

if(matab->TabInt[i]>matab->TabInt[i+1])

{

k++;

echange(&matab->TabInt[i],&matab->TabInt[i+1]);

}

}

n--;

}

printf(" \n le nombre d'iterations : %d ",k);

clock\_t end = clock();

temps(begin,end);

}

//foction qui permet de trier le tableau par trie par insertion

int trie\_par\_insertion(Tab \*matab)

{

int i,j,crt,k=0;

if (!matab)/\*\* s'il n'existe pas \*/

{

printf("\nErreur d'allocation!! la liste n'existe pas !!");//afficher message d'erreur

return ((int)0);

}

clock\_t begin = clock();

for(i=1;i<matab->NbElem;i++)

{

crt=matab->TabInt[i];

for(j=i-1;j>=0 && matab->TabInt[j]>crt;j--)

{

matab->TabInt[j+1]=matab->TabInt[j];

k++;

}

matab->TabInt[j+1]=crt;

}

printf(" \n le nombre d'iterations : %d ",k);

clock\_t end = clock();

temps(begin,end);

return ((int)1);

}

La fonction principale main :

int main()

{

Tab \*matab=Creer\_Tab();

Init\_Tab(matab);

// Insertion d'éléments dans le premier tableau

Inserer\_Fin\_Tab(matab, 1);

Inserer\_Fin\_Tab(matab, 0);

Inserer\_Fin\_Tab(matab, 30);

Inserer\_Fin\_Tab(matab, 2);

Inserer\_Fin\_Tab(matab, 17);

Inserer\_Fin\_Tab(matab, 6);

// Affichage du premier tableau

printf("Affichage du tableau: ");

afficher\_Tab(\*matab);

printf("\ntrier le tableau: ");

trie\_bulle(matab);

// printf("\ntrier le tableau: ");

// trie\_par\_insertion(matab);

printf("\nAffichage du tableau apres le tri: ");

afficher\_Tab(\*matab);

return 0;

}

**Programmation des fonctions de trie pour liste dynamique :**

1. Librairies :

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

1. Variables principales :

/\*Definition d'une structure de liste dynamique\*/

typedef struct lst

{

int Entier;/\* l'élement de la liste \*/

struct lst \*svt; /\* poiteur sur l'élement au suivant \*/

}PtListe;//Cellule

/// Fonction du temps d'execution

void temps(clock\_t begin ,clock\_t end)

{

unsigned long tmp;

tmp = (end - begin) \* 1000 / CLOCKS\_PER\_SEC; /// temps en ms

printf("\n le programme a pris comme temps d'execution : %ld ms \n ",tmp);

}

//fonction qui permet de creer la liste "Cellule"

PtListe \*Creer\_Cellule(int val)

{

PtListe \*NE;

/\* Allouer la mémoire pour NE \*/

NE = (PtListe \*)malloc(sizeof(PtListe));

if (!NE) /\* L'allocation est mal passée \*/

{

/\* Afficher un message d'erreur \*/

printf("\nErreur d allocation du memoire.");

/\* Sortir du programme \*/

exit(-1);

}

/\* Indiquer que le nœud créé n'a pas de successeur \*/

NE->svt = NULL;

/\* Ajouter l'élément \*/

NE->Entier = val;

return ((PtListe \*)NE); /\* Recupérer le nœud créé \*/

}

//fonction qui permet de tester si la liste est vide on non

int Est\_Vide\_PtListe(PtListe \*Liste)

{

return ((int)(Liste == NULL));

}

//fonction qui permet d'afficher la liste

void afficher\_PtListe(PtListe \*Liste)

{

PtListe \*pcrt;

printf("\n");

if (Est\_Vide\_PtListe(Liste))

printf("\nLa liste est vide");

pcrt = Liste;

while (pcrt)

{

/\* Afficher élément par élément de la liste \*/

printf("| %d |", pcrt->Entier);

/\* Passer à l'élément au suivant \*/

pcrt = pcrt->svt;

}

}

//fonction qui permet d'inserer un element a la fin de la liste

PtListe \*InsererFin(PtListe \*maliste, PtListe \*macel)

{

PtListe \*pcrt;

/\* Vérification que la liste est vide \*/

if (Est\_Vide\_PtListe(maliste))

return ((PtListe \*)macel);

pcrt = maliste; /\* Initialisation de pcrt pour parcourir la liste \*/

/\* Parcours de la liste jusqu'au dernier nœud \*/

while (pcrt->svt)

pcrt = pcrt->svt;

pcrt->svt = macel; /\* Mise à jour du dernier nœud pour pointer vers macel \*/

return ((PtListe \*)maliste); /\* Renvoi de la liste mise à jour \*/

}

//fonction qui permet de calcule la taille d la liste

int taille\_liste(PtListe \*maliste)

{

PtListe \*tmp;

tmp=maliste;

int taille=0;

while(tmp)

{

taille++;

tmp=tmp->svt;

}

return ((int)taille);

}

//fonction qui permet de trier la liste par trie par insertion

//fonction qui permet d'inserer un element a la fin de la liste

PtListe \*trier\_liste\_bulle(PtListe \*maliste)

{

PtListe \*tmp,\*crt;

int k=0;

clock\_t begin = clock();

int taille=taille\_liste(maliste);

while(taille>0)

{

crt=maliste;

tmp=crt->svt;

while(crt->svt)

{

if(tmp->Entier<crt->Entier)

{

//permutation

k++;

tmp->Entier=tmp->Entier+crt->Entier;

crt->Entier=tmp->Entier-crt->Entier;

tmp->Entier=tmp->Entier-crt->Entier;

}

crt=crt->svt;

tmp=tmp->svt;

}

taille--;

}

clock\_t end = clock();

temps(begin,end);

printf(" \n le nombre d'iterations : %d ",k);

return((PtListe\*)maliste);

}

//fonction qui permet de faire la permutatione entre 2 entiers

void Echange\_P(int \*L1, int \*L2)

{

int echange;

echange = \*L1;

\*L1 = \*L2;

\*L2 = echange;

}

//fonction qui permet de trier la liste par insertion

PtListe \*trie\_par\_insertion(PtListe \*L)

{

int k=0;

clock\_t begin = clock();

PtListe \*Ptr1;//pointeur sur les elements triee

PtListe \*Ptr2;//pointeur sur l'element e inserer dans la partie gauche.

PtListe \*courant;//pointeur pour parcourir la laiste

courant = L->svt;

while(courant)

{

Ptr1=L;

/\*Tant queon nea pas parcouru toute la partie gauche\*/

while(Ptr1!= courant)

{

Ptr2=courant;

/\*insertion de l'element pointe par Ptr2 e la

bonne position\*/

if(Ptr1->Entier > Ptr2->Entier)

{

k++;

Echange\_P(&Ptr1->Entier,&Ptr2->Entier);

afficher\_PtListe(L);

}

Ptr1=Ptr1->svt;

}

courant = courant->svt;

} /// fin while(courant)

clock\_t end = clock();

temps(begin,end);

printf(" \n le nombre d'iterations : %d ",k);

return((PtListe\*)L);

}

La fonction principale main :

int main()

{

PtListe \*liste = NULL; // Initialisation de la liste

// Insertion d'éléments à la fin de la liste

PtListe \*element1 = Creer\_Cellule(17);

PtListe \*element2 = Creer\_Cellule(8);

PtListe \*element3 = Creer\_Cellule(0);

PtListe \*element4 = Creer\_Cellule(3);

liste = InsererFin(liste, element1);

liste = InsererFin(liste, element2);

liste = InsererFin(liste, element3);

liste = InsererFin(liste, element4);

// Affichage de la liste

printf("Liste initiale : ");

afficher\_PtListe(liste);

printf("\nla taile de la Liste est : %d ",taille\_liste(liste));

printf("\n\nla Liste avant le trie : ");

afficher\_PtListe(liste);

printf("\n\nla Liste apres le trie : ");

liste=trier\_liste\_bulle(liste);

afficher\_PtListe(liste);

// printf("\n\nla Liste apres le trie : ");

// liste=trie\_par\_insertion(liste);

// afficher\_PtListe(liste);

return ((int)0);

}