AKLI Yassamine && MEDDAS NESRINE |

Mini projet d’Algo

Implémentation et analyse de quelques algorithmes de Tri

**USTHB** Module ALGO (L2Acad)

**Faculté d’électronique et d’informatique** Année 2017/2018

**Département d’informatique Section A groupe 2**

Chargé de cours et de TD : Mme S.BOUKHADOUMA

Chargé de TP : Mme BOUAKKAZ

#### Introduction 1

TABLE DE MATIERE

#### Objectif de travail 1

### Organigramme général de la solution 1

### Description des différents algorithmes

* 4 1 Procédures et fonctions concernant la partie vecteur 2

Tri fusion 4

* 4 2 Procédures et fonctions concernant la partie Matrice 4
* 4 2 1 Tri pour une matrice d’entier 4

1. Tri ligne par ligne 4
2. Tri d’une matrice d’entier entièrement 7

* 4 2 2 Tri pour une matrice de caractères
* 4 3 Procédures et fonctions concernant la partie liste

### Analyse des methodes programméés (ANALYSE THEORIQUE) 13

### Jeux d’essai (ANALYSE PRATIQUE)

1. Capture d’écran 14
2. Hypothèse  16
3. Rapport des temps Tableau : Rapport des temps d’exécutions des différents algorithmes sur Machine 16
4. Critique 17
5. Conclusion 17

### Conclusion

**1-Introduction :**

Ce projet nous aidera à comparer 3 différent type de tri (bulle \_ insertion \_ sélection) sous plusieurs structure, Et analyse leur performances en comptant la complexité, et de là on analysant leurs complexité en tirant les idées essentiel de chaque tri.

**2-Objective de travail :**

Dans ce projet « implémentation et analyse de quelques algorithme de tri » on a essayé d’implémenter ces algorithmes pour chaque structure (vecteur , matrice , liste chainée) afin de les analysé et comparé les un entre les autres pour déduire les problématique et les performances de chaque tri.

**3-Organigramme géneral de la solution:**

**4-1 Description des différentes algorithmes :**

* + 1. Procédures et fonctions concernant la partie vecteur

// Procedure de saisis

**procedure saisir(E/s tab :^entier ,E\taille:entier)**

Var i=entier;

**debut** printf("\n Remplissage des elements\n");

pour i 0 a taille

Faire printf("\t");printf("t[%d] : ",i+1);scanf("%d",&tab[i]); fait

**fin;**

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// Procedure d'affichage

**Procedure affiche(E\tab:^entier,E\ taille:entier)**

Var int i;

**debut** printf("\n\t\t \t\n\t");

pour i <-- 0 a taille

faire printf ("%c ",179); printf(" %d ",tab[i]);fait;

**fin;**

//-------------------------------------------------------------------------------------------

//TRI BULLE PAR ORDRE CROISSANT

//fonction pour t a bulle d'un tableau a une dimension de taille variable par ordre //croissant

**fonction tribullec(E\tabT: ^entier,E\taille:entier):^entier**

vartabp : tabp^entier ;

a, r, i, etape, n, nbCompare, tab\_en\_ordre : entie;

//etape pour compter le nombre de permutation nbCompare pour compter le //nombre de comparaison//variable de travail

debut

a=0 ;etape=0 ;n,nbCompare=0 ;tab\_en\_ordre = 0 ;

tabp= allouer(taille \*taillede(entier));

n=taille ;

//remplissage (qu'on travaillera dessus)

pour i<- 0 a n-1

faire

tabp[i]=tabT[i];

fait ;

tant que (!tab\_en\_ordre)//tant qu'il reste des elements a permuter on continue

faire tab\_en\_ordre = 1;

pour i=0 a taille-2 //parcourir le tableau et conparer l'element tab[i] a tab[i+1

faire

si (tabp[i] > tabp[i+1])

alors a=tabp[i] ;//permuter

tabp[i]=tabp[i+1] ;//permuter

tabp[i+1]=a ;//permuter

affiche(tabp,n);

printf(" // On compare %d et %d :On inverse Etape %d:\n",tabp[i],tabp[i+1],etape);//afficher

etape++;//incrementation \_ compter le nombre de permutation

tab\_en\_ordre = 0;

sinon affiche(tabp,n);

printf(" // On compare %d et %d :On laisse\n",tabp[i],tabp[i+1]);

fsi ;

nbCompare++;//incrementation \_ compter le nombre de comparaison

fait ;//pour

taille--;

fait //while

printf("\n Apres le tri on a :\n le nombre de permutation: %d\nle nombre de comparaison: %d\nVecteur apres trii\n", etape, nbCompare);

affiche(tabp, n); return tabp;

fin; cc //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//TRI INSERTION

//tris par insertion PAR ORDRE CROISSANT

**fonction Tri\_insertion\_C(E\tab:^entier, E\ N:entier): ^entier**

var

elem ,i, j, nbComp , nbPerm :entier;

A:^entier;

debut nbComp=0 ; nbPerm=0 ;

A= allouer(N \*tailled'entier);

//Remplissage

pour i <- 0 a N-1

faire

A[i] <-tab[i];

fait

//tri

pour i <- 1 < N-1

faire

elem <- A[i];

j<-i;

tan que (j > 0 && A[j-1] > elem)

faire

A[j] = A[j-1]; nbComp++; affiche(A, N);

printf("//On compare %d et %d : On decale a gauche\n",A[j],A[j-1]);

j--;

fait

si (A[j-1]<=elem && j != 0)alors

nbPerm++; affiche(A, N);

printf("//On compare %d et %d : On insere %d a l'indice %d'\n",A[j],A[j-1], elem, j);

sinon

affiche(A, N);

printf("//On compare indice j = 0: On laisse le premier et on passe a l'élément suivant %d++\n",i);

fsi; A[j] <- elem ; nbComp++;

fait ; //boucle pour

printf("\n Apres le tri on a :\nle nombre de permutation: %d\nle nombre de comparaison: %d\nVecteur apres tri:\n", nbPerm, nbComp);

affiche(A, N);//Affichage apres le tri

return A;

fin;

**//--------------tri par selection par ordre croissant----------------------------------**

**fonction tri\_selection1(E\tab:^entier,E\n:entier):^entier**

{

var I,J,AIDE,PMIN,nbPerm,nbComp :entier; A:^entier;

debut

A <- allouer (N \*taille d'entier);

nbPerm <- 0; nbComp <- 0;

//remplissage

pour I <- 0 a N-1

faire

A[I]<-tab[I];

fait;

//tri

pour I <- 0 a N-2

faire

PMIN <- I;

pour J<- I+1 a N-1 // liste non-triée : (Ai+1, A2, ... , An)

faire si (A[J] < A[PMIN])

alors // Aj est le nouveau minimum partiel

PMIN<-J;

//on a change les positions de Ai et de Aj

AIDE<-A[I]; A[I]<-A[PMIN];

A[PMIN]<-AIDE; PMIN<-I;

affiche(A, N);

printf("//On compare %d avec %d ",A[J],A[PMIN]);

printf("dans la partie non trie [%d,%d] ",I+1,N);

printf("indice J=%d>nouveau max->On permute",J);

nbPerm++;

si non

affiche(A, N);

printf("//On compare %d avec %d ",A[J],A[PMIN]);

printf("dans la partie non trie [%d,%d] ",I+1,N);

printf("indice J=%d>nouveau max->On laisse",J);

fin si

fait;

nbComp++;//incrementer le nombre de comparaison (condition))

fait;

printf("\n Apres le tri on a :\n le nombre de permutation: %d\n le nombre de comparaison: %d\n Vecteur apres tri:\n", nbPerm, nbComp);

affiche(A, N);//Affihage apres le tri

retourner A;

}

//Tri par selection par ordre decroissant

**fonction tri\_selection2(tab:^entier ,E\ N:entier) :^entier**

var I, J, AIDE, PMAX,nbPerm, nbComp : Entier;A ^Entier;

/\*c'est le même principe .. sauf qu'on cherche le max pas le min

la condition si (A[J]<A[PMIN]) //si1 change et devient :si (A[J]>A[PMAX]) \*/

//-------------------------------------------------------------------------------------------

**//TRI FUSION PAR ORDRE CROISSANT**

**//fonction pour t a FUSION d'un tableau a une dimension de taille variable par ordre //croissant**

**procedure Tri\_F\_C(ES\ T^Entier, E\n1, n2 :Entier) //n1 indice debut d’un sous tableau de T, n2 indice fin d’un sous //tableau //de T**

**/\*Resultat : Les éléments du sous tableau T[n1] jusqu’à T[n2] sont triés par ordre croissant\*/**

Var q :Entier ;

Debut :

Si ( n1 >= n2) //alors le sous tableau T de n1 a n2 a au plus un élément et il est déjà trié, retourner //simplement sans //traitement

Sinon q = (p+r) / 2 ;

Tri\_F\_C(T, n1, q) ;Tri\_F\_C(T, q+1, n2) ; Fusionner(T, n1, q, n2) ;//appeler récursivement Tri fusion puis Fusionner

Fsi ;

Fin ;

Procedure Fusionner(E/T :^Entier, n1 : Entier, q : Entier, n2 : Entier)/\*on suppose que les sous tableau T de n1 a q et T de q+1 a n2 sont déjà trié **/**Résultat : Le sous tableau T de n1 a n2 contient les éléments qui se trouvaient à l’origine dans T de n1 a q et T de q+1 a n2, \*/

Var : R, P, i, j Entier ;tabP, tabR : ^ENTIER ;

**P = q- n1 +1 ; R= n2 – q ;**

**//créer les tabkeau tabP de 1 a P+1 tabR de 1 a R+1**

**tabP=allouer(P +1\* taille d’entier),tabR=allouer(R +1\* taille d’entier) ;**

//copier T de n1 a q dans P et T de q+1 a n2 dans R

J=0 ;

pour I <- n1 a q

faire

P[j] = T[i] ;j=j+1 ;

fait;

J=0 ;

pour I <- q+1 a n2

faire

R[j] = T[i] ;j=j+1 ;

fait;

//fixer j et i a 1

J=1 ; i =1 ;// i indice du plus petit élément restant dans P De même pour j dans R

Pour k<- n1 a n2

Faire :

Si P[i] <= R[ j] alors T[ k] = P[ i ] ; i = i + 1 ;//chaque itération recopie le plus petit l’élément restant a l’emplacement //suivant dans T , elle se ermine lorsqu’elle a recopié tous les éléments de R et de P

Sinon T[ k] = R[ j] ; j = j + 1 ;

Fait ;Fin ;

**//TRI FUSION PAR ORDRE DECROISSANT**

**//fonction pour t a FUSION d'un tableau a une dimension de taille variable par ordre //croissant**

**procedure Tri\_F\_C(ES\ T^Entier, E\n1, n2 :Entier) //n1 indice debut d’un sous tableau de T, n2 indice fin d’un sous //tableau //de T**

**/\*Resultat : Les éléments du sous tableau T[n1] jusqu’à T[n2] sont triés par ordre decroissant\*/**

/\*Rien ne chane sauf dans la procedure Fusionner la partie //fixer j et i a 1

J=1 ; i =1 ;// i indice du plus petit élément restant dans P De même pour j dans R

Pour k<- n1 a n2

Faire :

Si P[i] <= R[ j] alors T[ k] = P[ i ] ; i = i + 1 ;//chaque itération recopie le plus petit l’élément restant a l’emplacement //suivant dans T , elle se termine lorsqu’elle a recopié tous les éléments de R et de P

Sinon T[ k] = R[ j] ; j = j + 1 ;

Devient

//fixer j et i a 1

J=1 ; i =1 ;// i indice du plus grand élément restant dans P De même pour j dans R

Pour k<- n1 a n2

Faire :

Si P[i] >= R[ j] alors T[ k] = P[ i ] ; i = i + 1 ;//chaque itération recopie le plus petit l’élément restant a l’emplacement //suivant dans T , elle se termine lorsqu’elle a recopié tous les éléments de R et de P

Sinon T[ k] = R[ j] ; j = j + 1 ;

\*/

* + 1. Procédures et fonctions concernant la partie matrice

4-1-2-1 Tri pour une matrice d’entier

4-1-2-1-I. Tri ligne par ligne

**//---------------fonction qui convertit une matrice en un tableau------------------------**

**procedure convertMatToTab(E\ mat:^^entier,E\s tab:^entier,E\n:entier,E\m:entier )**

{ var i,j; k=(n\*m)-1:entier;

debut

pour i <- n-1 a 0 pas(i--)

faire

pour j<-m-1 a 0 pas (j--)

faire tab[k]<-mat[i][j]

k--;

fait ;

fait;

fin;

}

**//----------------fonction qui convertit un tableau en une matrice-----------------------**

**procedure convertTabToMat(E\smat:^^entier, E\t:^entier ,E\n:entier,E\m:entier)**

{

var i,j,k:entier;

debut k<-(n\*m)-1 ;

pour i <- n-1 0 pas(i--)

faire

pour j <-m-1 a 0 pas (j--)

faire

mat[i][j]=t[k]; k--;

fait;

fait;

fin;

]}

**//-----------------fonction qui affiche une matrice d'entier-----------------------------**

**procedure afficheMatINT(E\Mat:^^entier,E\n:entier,E\m:entier)**

{

var i:entier;

debut

printf("\n\t\t Affichage\n");

pour <- i=0 n-1 faire

affiche(Mat[i], m);

printf("\n");

fait;

fin;

}

**//-----------------fonction pour saisir une matrice d'entier-----------------------------**

**procedure saisirMatINT(E\S Mat:^^entier,E\n:entier,E\m:entier)**

{ var i,j:entier ;

debut

printf("\n \t\t\t\t veuillez remplir la matrice\n");

pour i <- 0 n-1 faire

pour <- j=0 a m-1 faire

printf(" Matrice[%d][%d] : ", i, j);

scanf("%d",&Mat[i][j]);

fait;

fait;

fin;}

**//-----------------TRI INSERTION MATRICE Croissant--------------------------------------------**

**procedure Tri\_insertion\_C\_mat1(matt:^^entier,E\n:entier,E\m:entier)**

{

var i:entier; mat:^^entier;

debut

mat <- allouer( n \* taillede(^mat)); //allocation

printf("\n");

pour i <- 0 a n-1

faire

printf("\n--------------Ligne %d:\n",i+1);

mat[i] <- Tri\_insertion\_C(matt[i],m);

fait

printf("\n Affichage\n");

pour i <- 0 a n-1 faire

affiche(mat[i], m);

printf("\n");

fait;

//free

pour i <- de 0 a n-1 faire

liberer(mat[i]);

fait ;

liberer(mat);

fin;

}/\*TRI insertion decroissant

Procedure Tri\_insertion\_D\_mat1(E\matt ^^Entier, E\n : Entier,E\m : Entier)

Var i ::entier ; mat ^^entier ; // le meme principe du tri par insertion croissant d’une matrice d’entier (ligne par ligne) sauf lors de l’affectation (mat[i] <- Tri\_insertion\_C(matt[i],m);)devient (mat[i] <- Tri\_insertion\_D(matt[i],m);)ie : appel au Tri\_insertion\_D d’un vecteur // on tri ligne par ligne en considérant les lignes de la matrice des vecteurs\*/

**//-----------------TRI SELECTION MATRICE croissant-------------------------------------------**

**procedure tri\_selection1\_mat1(E\ matt:^^entier,E/ n:entier ,E/m:entier)**

{

var i:entier ; mat:^^entier;

debut //allocation

mat <- allouer( n \*taillede(^ mat));

printf("\n");

pour i<- 0 n-1

faire

printf("\n--------------Ligne %d:\n",i+1);

mat[i] <- tri\_selection1(matt[i], m);

fait

printf("\n Affichage\n");

pour i <- 0 n-1

faire

affiche(mat[i], m);

printf("\n");

fait;

//free

pour i <- 0 a n-1

faire

liberer(mat[i]);

fait;

liberer(mat);

fin;}/\*TRI selection decroissant

Procedure tri\_selection\_mat1(E\matt ^^Entier, E\n : Entier,E\m : Entier)

Var i ::entier ; mat ^^entier ; // le meme principe du tri par selection croissant d’une matric d’entier (ligne par ligne) saus lors de l’affectation (mat[i] <- tri\_selection1(matt[i], m);) devient (mat[i] <- tri\_selection2(matt[i], m);) ie : appel au tri\_selection2 d’un vecteur // on tri ligne par ligne en considérant les lignes de la matrice des vacteurs\*/

**//-----------------TRI Bulle MATRICE croissant-----------------------------------------------**

**procedure tribulled\_mat1(E\matt:^^entier,E\n:entier,E\m:entier)**

{

var i:entier ; mat:^^entier;

debut //allocation

mat <- allouer( n \*taillede(^ mat));

printf("\n");

pour i<- 0 n-1

faire

printf("\n--------------Ligne %d:\n",i+1);

mat[i] <- tri\_selection1(matt[i], m);

fait

printf("\n Affichage\n");

pour i <- 0 n-1

faire

affiche(mat[i], m);

printf("\n");

fait;

//free

pour i <- 0 a n-1

faire

liberer(mat[i]);

fait;

liberer(mat);

fin;

}/\*TRI bulle décroissant

Procedure tribulled\_mat1(E\matt ^^Entier, E\n : Entier,E\m : Entier)

Var i ::entier ; mat ^^entier ; // le même principe du tri par bulle croissant d’une matric d’entier (ligne par ligne) sauf lors de l’affectation (mat[i] <- tribullec(matt[i], m);) devient (mat[i] <- tribulled(matt[i], m);) ie : appel au tri a bulle decroissant d’un vecteur // on tri ligne par ligne en considérant les lignes de la matrice des vecteurs\*/

4-1-2-1-I. Tri d’une matrice d’entier entièrement

**//-----------------TRI SELECTION MATRICE croissant-------------------------------------------**

**procedure tri\_selection1\_Mat2(E\matt:^^entier,E\ n:entier,E\ m:entier)**

{

var i:entier; mat:^^entier; t:^entier;

debut //allocation int \*t;

t<- allouer ( n\*m\* taillede(entier));

mat <- allouer( n \*taillede(^mat));

printf("\n");

pour i <- 0 n-1

faire

mat[i]<- allouer( m \*taillede(entier));

fait;

convertMatToTab(matt, t, n , m);

converTabToMat(mat,tri\_selection1(t, n\*m), n, m );

printf(" Apres le tri ");

afficheMatINT(mat, n, m);

//free

pour i <- 0 a n-1

faire

liberer(mat[i]);

fait;

liberer(mat);

fin;

}/\*TRI entier par selection ordre décroissant d’une matrice d’entier

**proceduure Tri\_insertion\_C\_Mat2(E\matt:^^entier,E\n:entier,E\m:entier)**

var i:entier; mat:^^entier; t:entier; // le même principe du tri par sélection croissant d’une matrice d’entier (entière) sauf lors de converTabToMat (mat,tri\_selection1(t, n\*m), n, m ); devient converTabToMat (mat,tri\_selection2(t, n\*m), n, m ) ;ie : appel au tri a sélection décroissant d’un vecteur de travail t et le mettre dans matrice mat de travail aussi // on tri la matrice en considérant la convertissant en un vecteur t puis on l’utilise pour le tri de vecteurs\*/

**//-----------------TRI Bulle MATRICE croissant-------------------------------------------**

**procedure tribullec\_Mat2(E\matt:^^entier,E\n:entier,E\m:entier)**

{

var i:entier; mat:^^entier; t:entier;

debut //allocation int \*t;

t<- allouer ( n\*m\* taillede(entier));

mat <- allouer( n \*taillede(^mat));

printf("\n");

pour i <- 0 n-1

faire

mat[i]<-malloc( m \* taillede(entier));

fait;

convertMatToTab(matt, t, n , m);

converTabToMat (mat,tribullec(t, n\*m), n, m );

printf(" Apres le tri ");

afficheMatINT(mat, n, m);

//free

pour i <- 0 a n-1

faire

liberer(mat[i]);

fait;

liberer(mat);

fin;

}/\*TRI entier par bulle ordre décroissant d’une matrice d’entier

**proceduure tribulled\_Mat2(E\matt:^^entier,E\n:entier,E\m:entier)**

var i:entier; mat:^^entier; t:entier; // le même principe du tri par bulle croissant d’une matrice d’entier (entière) sauf lors de converTabToMat (mat,tribullec(t, n\*m), n, m ); devient converTabToMat (mat,tribulled(t, n\*m), n, m ); ie : appel au tri a bulle décroissant d’un vecteur de travail t et le mettre dans matrice mat de travail aussi // on tri la matrice en considérant la convertissant en un vecteur t puis on l’utilise pour le tri de vecteurs\*/

**//-----------------TRI Insertion MATRICE croissant-------------------------------------------**

**proceduure Tri\_insertion\_C\_Mat2(E\matt:^^entier,E\n:entier,E\m:entier)**

{

var i:entier; mat:^^entier; t:entier;

debut //allocation int \*t;

t<- allouer ( n\*m\* taillede(entier));

mat <- allouer( n \*taillede(^mat));

printf("\n");

pour i <- 0 n-1

faire

mat[i]<-malloc( m \* taillede(entier));

fait;

convertMatToTab(matt, t, n , m);

converTabToMat (mat,Tri\_insertion\_C(t, n\*m), n, m );

printf(" Apres le tri ");

afficheMatINT(mat, n, m);

//free

pour i <- 0 a n-1

faire

liberer(mat[i]);

fait;

liberer(mat);

fin;

}/\*TRI entier par insertion ordre décroissant d’une matrice d’entier

**proceduure Tri\_insertion\_C\_Mat2(E\matt:^^entier,E\n:entier,E\m:entier)**

var i:entier; mat:^^entier; t:entier; // le même principe du tri par insertion croissant d’une matrice d’entier (entière) sauf lors de converTabToMat (mat,Tri\_insertion\_C(t, n\*m), n, m ); devient converTabToMat (mat,Tri\_insertion\_D(t, n\*m), n, m ); ie : appel au tri a insertion décroissant d’un vecteur de travail t et le mettre dans matrice mat de travail aussi // on tri la matrice en considérant la convertissant en un vecteur t puis on l’utilise pour le tri de vecteurs\*/

4-1-2-2 Tri pour une matrice de caractères

/\*Pour les méthodes de tris de chaines de caractères le principe reste le même que celui du tri d’une matrice ligne par ligne, seulement la comparaison et affectation qui changent :

Pour une affectation on utilise à la place strcpy( const dest \*char, const source \*char) qui copy le contenu de source vers dest

Pour une comparaison on utilise le stcmp(const ch \*char, const ch2 \*char) rend :

stcmp(const ch \*char, const ch2 \*char )=0 si ils sont identique

stcmp(const ch \*char, const ch2 \*char)>0 si la première vient après la deuxième en code ascii

sinon stcmp(const ch \*char, const ch2 \*char)<0 si la première chaine vient avant la deuxième

\*/

* + 1. Procédures et fonctions concernant la partie liste

**//declaration de la liste chainnee**

**typedef struct elementListe{**

char \*donnee;

struct elementListe \*suivant;

}element;

**typedef struct listeRepere{**

element \*debut;

element \*fin;

int taille;}liste;

//---------------------TRI SELECTION LISTE CHAINEE croissant ----------------------------------

Procedure trier\_liste\_selection\_A\_Z (E\ma\_liste:^liste)

{

Var parcour, tete :^element; valeur: ^caractere;

debut

si (ma\_liste->debut == NULL) // si 1

printf ("\n\n la liste est vide \n");

retourner 0;

sinon //allocation

parcour<- (^element) allouer (taillede (element));

parcour.donnee<- (^caractere) allouer (50\* taillede (caractere));

tete< -(^element) allouer (taillede (element));

tete.donnee<- (^caractere) allouer (50\*taillede (caractere));

valeur<- (^caractere) allouer (50\*taillede (caractere));

tete<- ma\_liste.debut;

tant que (tete != ma\_liste.fin) //tantque 1 la tete sera fixÃ© au dÃ©but de la sous liste non //trier

Faire

parcour<- tete.suivant

tant que (parcour != NULL) //tantque 2 // on avance avec "parcour" pour //chercher le min

faire

si (strcmp (tete.donnee, parcour.donnee) < 0) Alors

printf ("\n\t la tete est avantâ€¦ rien a faire!\n\n");

si non

printf ("\n\tl'element dans la tete est aprÃ¨s..on doit Permuter\n\n");

// si l'element dans parcour est avant ...on doit permuter

Strcpy (valeur, tete.donnee);

Strcpy (tete.donnee, parcour->donnee);

Strcpy (parcour.donnee, valeur);

Affichage (ma\_liste);

fin.si

parcour<- parcour.suivant ;

fait ; //tantque2

tete<- tete.suivant;

fait ; //tantque 1

printf ("\n\n\t ~~~~ Apres le tri selection ordre croissant~~~~~~~~~~~~~~~~~\n\n");

affichage (ma\_liste);

Fin.si; fin;}

**//---------------------TRI Bulle LISTE CHAINEE croissant ---------------------------------------**

**Procedure trier\_liste\_bulle\_A\_Z(E\ma\_liste:^liste)**

{

Var parcour, tete :^element; valeur: ^caractere;

debut

si (ma\_liste.debut == NULL) // si 1

printf ("\n\n la liste est vide \n");

retourner 0;

sinon //allocation

parcour<- (^element) allouer (taillede (element));

parcour.donnee<- (^caractere) allouer (50\* taillede (caractere));

tete< -(^element) allouer (taillede (element));

tete.donnee<- (^caractere) allouer (50\*taillede (caractere));

valeur<- (^caractere) allouer (50\*taillede (caractere));

tete<- ma\_liste.debut;

tant que (tete!=NULL) //tantque 1â€¦â€¦. on compare chaque foi avec l'element suivant !

faire

parcour<- tete.suivant ;

tant que (parcour!=NULL) //tant que 2

faire

// tete=ma\_liste->debut;

si (strcmp (tete.donnee, parcour.donnee) <0) //if

printf ("\n\t la tete est avant... rien Ã  faire!\n\n");

sinon

printf ("\n\t on doit Permuter\n\n");

strcpy (valeur, tete.don nee);

strcpy (tete.donnee, parcour.donnee);

strcpy (parcour.donnee, valeur);

affichage (ma\_liste);

fin si ; //si 2

parcour<- parcour.suivant ;

fait; //tantque 2

tete<- tete.suivant;

fait; //tantque1

fin Si ; //si 1

printf ("\n\n\t ~~~~~~~~~ Apres le tri bulle ordre croissant~~~~~~~~~~~~~~~~~\n\n");

affichage (ma\_liste);

fin;

}

**//----------------------fonction pour afficher la liste de chaine de caracteres--------------------**

**procedure affichage(E\L:^liste)**

{

var i:entier; temp:^element;

debut

i<-1;

temp<-(^element)allouer(taillede(element ));

temp <- L.debut;

tant que ( temp!=NULL)

faire

printf("\t%d : %s",i,temp.donnee);

i++;

temp<-temp.suivant

fait;

fin;}

//----------------------fonction pour initialiser le chainage de la liste--------------------------

procedure initialiser\_liste(E\S L:^liste)

{

debut

liste.debut <- NULL;

liste.fin <- NULL;

liste.taille <- 0;

printf("initialisation ... \n\n ");

fin; }

**//----------------------Creation de la tete de liste------------------------------------------------**

**fonction creation\_tete(E\SL:^liste,donnee:^caractere):entier**

{

var Premier\_element:^element;

debut

si (liste.taille=0) return 0;

// l'allocation

Premier\_element<- (^element)allouer(taillede(element));

Premier\_element.donnee <- (^caractere)allouer(50\*taillede(caractere));

//copier la chaine dans la tete

strcpy(Premier\_element.donnee,donnee);

Premier\_element.suivant <- NULL;

liste.debut <- Premier\_element;

liste.fin <- Premier\_element;

liste.taille <- 1;

// printf(" l'element numero 1 est creer avec succÃ©e : %s !\n\n\n",liste->debut->donnee);

fin.si;

return 0;fin ;**}**

**//-----------------------fonction inserer une liste retourn------------------------------------------**

**fonction inser\_liste(liste:^liste):entier**

{

var parcour ,nouveau\_element:^element; donnee:^ caractere

debut

nouveau\_element <- (element\*)allouer (taiilede(element));

nouveau\_element.donnee <- (^caractere)alloue(50\*taille de (caractere));

parcour <- (element\*)allouer (taille de(element));

parcour.donnee <- (^caractere)alloue(50\*taille de (caractere));

//initialiser parcour o debut de la liste

parcour <-liste.debut;

tantque (parcour.suivant != NULL) faire // on est pa arriver a la fin ! pour faire insertion queue

parcour=parcour->suivant;

fait;

//quand on arrive au dernier on copie lÃ  chaine faire les chainage et changer la queue de la liste

printf("donnez la chaine de l'lement N' %d :\t",liste.taille);

scanf("%s",donnee);

strcpy(nouveau\_element->donnee,donnee); // on copie la chain saisi dans le nouveau element creer

parcour.suivant <- nouveau\_element ; // chainer l'ele creer a la liste

nouveau\_element.suivant <- NULL; // indiquer que c est la queue ( pointe vers nil )

liste.fin <- nouveau\_element; // metre a jours l indicateur de la queue

liste.taille++; // augmenter la taille de la liste

//printf("l'element numero %d etait creer avec succes\n\n",liste->taille);

returner 0;

fin;

}

**5 Analyse Théorique :**

Théorie: Avant d'écrire un programme on doit s'interroger sur l'efficacité de ses algorithme de ce fait on calcule leurs complexité.

Algorithme de tri d’un vecteur d’entier

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithme | Dans le cas le plus défavorable | Dans le cas le plus favorable | Echange dans le cas le plus défavorable |
| Tri par sélection | O(n ^2) | O(n ^2) | O(n) |
| Tri par insertion | O(n ^2) | O(n) | O(n ^2) |
| Tri à bulle | O(n ^2) | O(n) | O(n ^2) |
| Tri par fusion | O(n logn) | O(n logn) | O(n logn) |

Explication :

Tri par fusion

Pour l’agorithme d fusion on a deux procedure la premiere Fusionner la copie des élément dans les tableaux P et R prend O( n ) et la recopie des éléments dans u tabeaul (mme dans une matrice ) se fait en un temps constant-Chaque appels récursifs sur les sous tableau prend un temps T( n / 2 ) leurs combinaison par le tri fusion (appel de la fonction Fusionner )se fait O( n )- donc la complexité de tri fusion : T(n) = 2T(n/2) +O( n)

T(n ) = aT( n /b) + f( n ) //par méthode général pour passer du récursive au non récursive ou a et b sont des constantes entière positive

D’où T( n) = O( n log n )

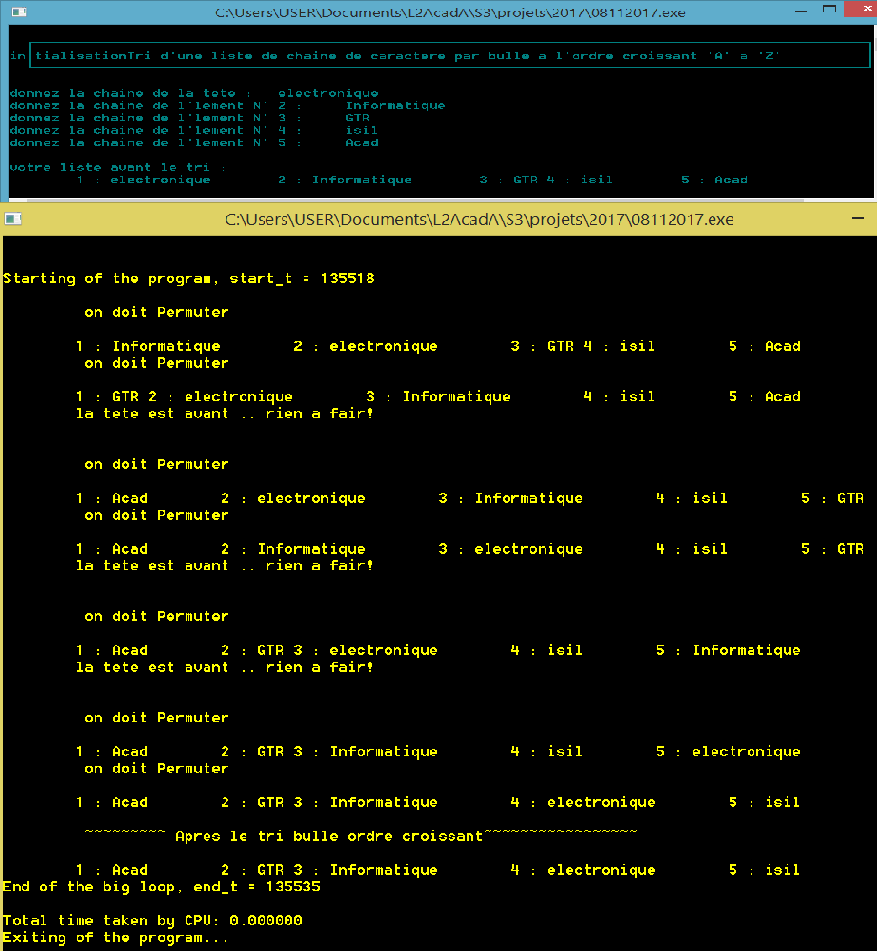
Conclusion : Le tri par fusion implémenté est beaucoup intéressant comparée aux autres.

**6 Jeux d’essai**:A) Capture d’écran :

**Dans cette partie on a utilisé une machine** :- Processeur ; Intel(R) Core(TM) i5\_5200U CPU @2.20GHZ –RAM ; 8.00 Go

Dans cet exemple on a pris un vecteur d’entier de n=15 éléments à des valeurs non rapproché mais sont presque trié, on observe après l’exécution de chacun différent temps d’exécution et parcours :

-Le tri par insertion prend le moins de temps et de nombre de comparaison ainsi de permutation que les deux autre parce que chaque élément se trouve initialement dans les positions ou il doit arriver dans la version triée.

****

Pour cet exemple on a rentré le nombre d’élément de notre liste chaîné, on la remplit mot par mot ensuite le programme exécute la partie du tri et affiche la liste trié dans l’ordre croissant (‘A’ ‘Z’), on remarque que le temps ici est beaucoup plus performant quand on manipule les adresses (chainages). Temps=70 ms

B)Hypothèse :

On suppose qu’on peut établir des correspondances Durée -> Valeur, car ici on a supposé que le temps est pseudo aléatoire c’est-à-dire qu’il est constant pour une valeur donnée (taille N=5 par exemple je prends plusieurs valeurs posé d’une manière ou le tri soit dans le cas le plus défavorable), et différent pour chaque valeur (N=5 ou N=500 ou N=1000).

c)Rapport de temps d’exécution de quelque algorithme de tri :

c) 1 pour une matrice d’enier :n=4 m=15 par unité de temps ms

Tri d’une matrice ligne par ligne

|  |  |  |
| --- | --- | --- |
| Algorithme | Dans le cas le plus défavorable | Dans le cas le plus favorable |
| Tri par sélection | 1359 | 1181 |
| Tri par insertion | 1609 | 625 |
| Tri à bulle | 1219 | 617 |

\*Commentaire1 : ayant le même principe de tri d’un vecteur la matrice d’enter trié ligne par ligne symbolise et confirme les résultats du calcul de leurs complexité.

Tri d’une matrice ligne par ligne

c) 2 pour une matrice d’enier :n>4 m>15 par unité de temps s :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| taille | [50][50] | [50][50] | [100][100] | [100][100] | [1000][1000] | [1000][1000] |
| Algo/Cas | défavorable | Favorable | défavorable | favorable | défavorable | favorable |
| Tri à bulle (ligne par ligne) | 379.9 s | 0.5523 s | 1519.6 s | 2.2092 s | 151960 s | 220.92 |

\*Commentaire2 : On a pris le tri à bulle seulement car sa complexité par rapport aux autres est la même (ceci n’est pas validé pour un temps réel, on remarque que à chaque Foix qu’on augmente le n (n m taille) le temps augmente.

D)Critique :

Dans la partie analyse du problème en plan pratique on s’est projeté sur le temps réels d’exécution ce qui n’est pas toujours fiables à cause de plusieurs critères ; tels que la performance de la machine et les processus en cours d’exécution (système, utilitaires …) qui peuvent interrompre l’exécution normal du programme.

F)Conclusion :

Comparant entre les tris Le nombre d'itération de la boucle interne

■pour le tri sélection il dépend du nombre d'indice i de la boucle externe et aucun des éléments eux même.

■ pour le tri insertion il dépend du nombre d'indice i de la boucle externe et des valeurs des éléments du tableau. -D'où Le tri par insertion est plus complexe que du tri par sélection.

D'où Le tri par insertion est plus complexe que du tri par sélection.

**Pour les différentes structures contiguës (vecteur d’entier, matrice (ligne par ligne, entière), matrice de caractère) les tris choisis (sélection insertion, bulle) deviennent moins intéressant en fonction de leurs tailles car ici on manipule des valeurs. Par contre les listes chainées prennent moins de temps car on ne fait pas d’échange ou de décalage de valeurs (des chainages).**

**7 Conclusion :**

Après avoir analysé les résultats de chaque algorithme, On peut conclure que ces trois tris ne sont pas performants et on estime qu’il y aura des améliorations pour avoir de meilleurs résultats au niveau de la complexité et du temps de réponse.