Algorithmique

**Tris**

**Arbres**

**Algorithmes Mathématiques**

**Listes**

**Piles**

Samuel CHICHA Antinéa LOISEAU--PINTAUX

Info 0401

**Tris**

1\_Tri à bulle………………...02

2\_Tri Couleur………….……04

3\_Tri Rapide………………..06

4\_Tri Topologique…………..09

**Tri à bulle**

**Algorithme : Tri à bulle**

**Données :**

**T tableau d’entiers ;**

**Entiers inf, taille, n, i ;**

**Début :**

**Lire(n) ;**

**Lire(T) ;**

**Taille <- n ;**

**Inf <- 1 ;**

**Tant que taille > 1 Faire :**

**Pour i allant de inf à taille-1 :**

**Si T[i] > T [i+1]**

**Aux <- T [i+1]**

**T [i+1] <- T [i]**

**T [i] <- aux**

**Fin Si**

**Fin Pour**

**Taille <- taille-1**

**Fin Tant que**

**Fin**

**Code :**

#include <stdio.h>

#include <stdlib.h>

void afficher(int \*tab, int size) {

int i;

for (i=0; i<size; i++)

printf("%d ", tab[i]);

printf("\n");

}

int main() {

int \*tab;

int size, n, aux, i;

do {

printf("Entrez la taille du tableau (max 100):");

scanf("%d", &n);

} while (n<0 || n>100);

tab=malloc(n\*sizeof(int));

size=n;

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

printf("tab[%d] = ", i);

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

}

while (size>1) {

for (i=0; i<size-1; i++) {

if (tab[i]>tab[i+1]) {

aux=tab[i+1];

tab[i+1]=tab[i];

tab[i]=aux;

}

}

size=size-1;

}

afficher(tab, n);

free(tab);

return 0;

}

**Tri Couleur**

**Algorithme : Tri Couleur**

**Données :**

**Tableau T**

**Bleu, rouge, blanc : entiers**

**Sup, n, i, j, k, aux : entiers**

**Début :**

**Lire (taille n)**

**Remplir le tableau**

**i <- 1 ;**

**j <- 1 ;**

**k <- sup ;**

**Tant que k != j Faire :**

**Si T [j+1] = bleu Alors :**

**Echange (T [i+1] , T [j+1]) ;**

**i++ ;**

**j++ ;**

**Sinon Si T [j+1] = rouge Alors :**

**Echange (T [j+1] , T [k]) ;**

**k-- ;**

**Sinon**

**j++ ;**

**Fin Si**

**Fin Tant que**

**Fin**

**Code :**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define bleu 1 // indice i

#define blanc 2 // indice j

#define rouge 3 // indice k

void afficher(int \*tab, int size) {

int i;

for (i=0; i<size; i++)

printf("%d ", tab[i]);

printf("\n");

}

int main() {

    int k, n, tmp, i, j;

    int \*tab;

    do {

printf("Entrez la taille du tableau (max 100):");

scanf("%d", &n);

} while (n<0 || n>100);

tab=malloc(n\*sizeof(int));

printf("Remplir le tableau avec les entiers correspondant a la couleur voulue: \n 1=bleu \n 2=blanc \n 3=rouge \n");

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

printf("tab[%d] = ", i);

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

}

    printf("\n");

printf("\nVoici le contenu du tableau avant tri :\n");

afficher(tab, n);

    printf("\n");

    k=n-1;

j=-1;

i=-1;

while (j!=k) {

if (tab[j+1]==bleu) {

tmp=tab[j+1];

tab[j+1]=tab[i+1];

tab[i+1]=tmp;

i++;

j++;

}

else if (tab[j+1]==rouge) {

tmp=tab[k];

tab[k]=tab[j+1];

tab[j+1]=tmp;

k--;

}

Else j++;

}

    printf("\nVoici le contenu du tableau apres tri :\n");

afficher(tab, n);

free(tab);

    return 0;

}

**Tri Rapide**

**Algorithme : Trirapide (T , inf, sup)**

**Données :**

**T tableau d’entiers**

**D, G, P, i, j : entiers**

**inf , sup : entiers**

**Début :**

**G <- inf <- 1**

**D <- sup ; (taille du tableau)**

**P <- T [sup] ;**

**i <- G ;**

**j <- sup-1 ;**

**Tant que i != j Faire :**

**Tant que T [i] < P &&( i != j ) Faire :**

**i++ ;**

**Fin Tant que**

**Tant que T [j] > P Faire :**

**j-- ;**

**Fin Tant que**

**Echange ( Ti, Tj ) ;**

**Fin Tant que**

**Si Ti > P Alors :**

**Echange ( Ti, P ) ;**

**Fin Si**

**Si G > i Alors :**

**Trirapide( T, G, i)**

**Fin si**

**Si D > i+1 Alors :**

**Trirapide( T, i+1, D)**

**Fin si**

**Fin**

**Code :**

#include <stdio.h>

#include <stdlib.h>

void afficher(int \*tab, int size) {

int i;

for (i=0; i<size; i++)

printf("%d ", tab[i]);

printf("\n");

}

void TriRapide(int\* tab, int inf, int sup) {

    int G,D,i,j,P,k;

    G=inf;

    D=sup;

    i=G;

    j=sup-1;

    P=tab[sup];

    printf("Le pivot courant est: %d\n",P);

    while (i != j ) {

        while ((tab[i]<P) && (i != j)) {

            i++;

        }

        while ((tab[j]>P) && (i != j)) {

            j--;

        }

        k=tab[i];

        tab[i]=tab[j];

        tab[j]=k;

    }

    if(tab[i]>P) {

        k=tab[i];

        tab[i]=tab[sup];

        tab[sup]=k;

    }

    if(G<i) {

        printf("Appel recursif gauche, inf=%d, sup=%d\n",G,i-1);

        TriRapide(tab,G,i);

    }

    if(D>i+1) {

        printf("Appel recursif droite, inf=%d, sup=%d\n",i+1,D);

        TriRapide(tab,i+1,D);

    }

}

int main() {

    int n;

    int i;

    int \*tab;

    do {

printf("Entrez la taille du tableau (max 100):");

scanf("%d", &n);

} while (n<0 || n>100);

tab=malloc(n\*sizeof(int));

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

printf("tab[%d] = ", i);

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

}

    printf("\n");

printf("\nVoici le contenu du tableau avant tri :\n");

afficher(tab, n);

    printf("\n");

    TriRapide(tab,0,n-1);

    printf("\nVoici le contenu du tableau apres tri :\n");

afficher(tab, n);

    free(tab);

    return 0;

}

**Tri Topologique**

**Algorithme : Tri topologique (matrice)**

**Données :**

**G : matrice**

**V- : vecteur d’arc entrant**

**Nbsommet, j, i, k : entiers**

**L : liste des sommets**

**Début :**

**Pour i allant de 1 à Nbsommet Faire :**

**j 🡨 1 ;**

**Tant que V [j]!= 0 Faire :**

**J🡨j+1 ;**

**Fin Tant que**

**Construction L[j] ;**

**V[j] 🡨 -1 ;**

**Pour k allant de 1 à Nbsommet Faire :**

**Si G [j, k]=1 Alors :**

**V[k] 🡨 K[k]-1 ;**

**Fin Si**

**Fin Pour**

**Fin Pour**

**Fin**

**Code :**

**Arbres**

1\_Structure Arbre……………………..……………...02

2\_Creer Arbre…………………………………...……04

3\_Insertion d’une valeur……………….……………..06

4\_Vider un Arbre……………………………………..09

5\_Chercher les val min et max d’un Arbre……….…..09

6\_Chercher si une val est présente dans un Arbre…....09

7\_Affichage des parcours d’un Arbre…………...…...09

8\_Affichage d’un Arbre……………………….……..09

**Structure Arbre**

**Algorithmes Mathématiques**

1\_Ackermann…………….…...02

2\_Suite de Fibonacci…….……04

3\_Nombre Parfait……………..06

4\_Majorite ………………..…..09

**Ackermann**

**Listes**

1\_Structure LISTE………………………..……...02

2\_Créer liste vide…………………………...……04

3\_Vérifier si la liste est vide……………….……..06

4\_Vider une liste……………………………..…..09

5\_Afficher une liste……………………….……..09

6\_Ajout d’un élément……….………...……..…..09

7\_Supprimer le dernier élément..………..…….....09

8\_Saisir une liste……………………………..…..09

9\_Compter le nombre d’éléments d’une liste……01

10\_Fusion de deux listes triées……………………..10

11\_Recherche d’un élément dans une liste………...10

12\_

**Structure LISTE**

**Piles**

1\_Structure PILE……………………………...02

2\_Créer pile……………………………………04

3\_Vérifier si la pile est vide …………………..06

4\_Initialiser une pile …………………………..09

5\_Empiler une valeur ……..……………….…..06

6\_Rajouter des valeurs sur une pile ……….…..06

7\_Dépiler la valeur du sommet …………....…..06

8\_Obtenir la valeur du sommet ………………..06

9\_Vider une pile ………………...............……..06

10\_Recherche et suppression d’une occurrence ....06

**Structure PILE**