Main.c

```
/**
* @file main.c
* @author Durand Thomas
* @brief fonction main avec preuve de concept des differents tris
* @version 0.1
* @date 2019-11-12
*
* @copyright Copyright (c) 2019
*
*/

#include <stdio.h>
#include <stdlib.h>
#include "triDenombrement.h"
#include "triFusion.h"
#include "trisInsertion.h"
#include "operationTab.h"

/**
* @brief fonction main pour preuve de concept des tris
*
* @return int
*/
int main(){
int N = 20;
int tab[] = {2,3,7,9,8,6,4,1,2,5,8,9,6,3,4,8,9,6,5,45};

afficheTab(tab,N);
printf("tri insertion:\n");
tri_insertion(tab,N);
afficheTab(tab,N);

int tab2[] = {2,3,7,9,8,6,4,1,2,5,8,9,6,3,4,8,9,6,5,45};
printf("tri fusion:\n");
triFusion(tab2,N);
afficheTab(tab2,N);

int tab3[] = {2,3,7,9,8,6,4,1,2,5,8,9,6,3,4,8,9,6,5,45};
printf("tri denombrement:\n");
triDenombrement(tab3,N);
afficheTab(tab3,N);
}
```

operationtab.c

```c
/**
* @file operationTab.c
* @author Durand Thomas
* @brief ensemble des fonctions lie au tableau
* @version 0.1
* @date 2019-11-12
*
* @copyright Copyright (c) 2019
*
*/

#include <stdio.h>
#include <stdlib.h>

/**
* @brief affichage d'un tableau
*
* @param tab tableau a afficher
* @param len longueur du tableau
*/
void afficheTab(int tab[],int len){
for (int i = 0; i < len; i++){
printf(" %d ",tab[i]);
}
printf("\n");
}
```

triDenombrement.c

```c
/**
* @file triDenombrement.c
* @author Durand Thomas
* @brief ensemble des fonction lie au tri par denombrement
* @version 0.1
* @date 2019-11-12
*
* @copyright Copyright (c) 2019
*
*/
#include <stdio.h>
#include <stdlib.h>

/**
* @brief trouve le min et le max d'un tableau
*
* @param tab le tableau
* @param taille taille du tableau
* @param min pointeur pour le minimum
* @param max pointeur pour le maximum
*/
void minMaxTableau(int* tab, int taille, int* min, int* max){
for (int i = 1; i < taille; i++){
if(tab[i] > *max){
*max = tab[i];
}
if(tab[i] < *min){
*min = tab[i];
}
}
}

/**
* @brief creer l'histogramme d'un tableau
*
* @param tab le tableau
* @param taille la taille du tableau
* @param histo pointeur de l'histogramme a cree
* @param tailleH taile de l'histogramme
* @param min valeur min de tab
```

```
*/
void histogramme(int* tab, int taille, int* histo, int tailleH,
int min){
for (int i = 0; i < taille; i++){
histo[tab[i] - min]++;
}
int place=0;
for (int i = 0; i < tailleH;i++){
for (int j = 0; j < histo[i]; j++){
tab[place] = i+1;
place++;
}
}
}

/**
* @brief applique le tri par denombrement sur un tableau
*
* @param tab le tableau
* @param taille taille du tableau
*/
void triDenombrement(int* tab,int taille){
int min = tab[0];
int max = tab[0];

minMaxTableau(tab,taille,&min,&max);

int tailleH = (max - min + 1 );
int* histo = malloc(tailleH * sizeof(int));

for (int i = 0; i < tailleH; i++){
histo[i] = 0;
}

histogramme(tab,taille,histo,tailleH,min);

free(histo);
}
```

triFusion.c

```c
/**
* @file triFusion.c
* @author Durand Thomas
* @brief ensemble des foncions lie au tri Fusion
* @version 0.1
* @date 2019-11-12
*
* @copyright Copyright (c) 2019
*
*/
#include <stdio.h>
#include <stdlib.h>

/**
* @brief creer un sous tableau et copie les valeurs du tableau
dedans
*
* @param src tableau source
* @param debut debut des valeurs a copie
* @param fin fin des valeurs a copie
* @return int* le sous tableau cree
*/
int* copieSousTableau(int* src,int debut,int fin){
int* dest = malloc((fin - debut)*sizeof(int));
int j = 0;
for (int i = debut; i < fin; i++){
dest[j] = src[i];
j++;
}
return dest;
}

/**
* @brief fusionne 2 tableau en les triants
*
* @param tab1 premier tableau
* @param taille1 taille du premier tableau
* @param tab2 deuxieme tableau
* @param taille2 taille du deuxieme tableau
* @param tabRes tableau resultat qui vas accueillir la fusion trie
```

```
*/
void fusion(int* tab1, int taille1, int* tab2, int taille2,int*
tabRes){
int indice1 = 0;
int indice2 = 0;
int indiceRes = 0;
while (indice1 < taille1 && indice2 < taille2){
if (tab1[indice1] < tab2[indice2]){
tabRes[indiceRes] = tab1[indice1];
indice1 ++;
}else{
tabRes[indiceRes] = tab2[indice2];
indice2 ++;
}
indiceRes ++;
}
if (indice1 < taille1){
for (int i = indice1; i < taille1; i++){
tabRes[indiceRes] = tab1[i];
indiceRes++;
}
}else if (indice2 < taille2){
for (int i = indice2; i < taille2; i++){
tabRes[indiceRes] = tab2[i];
indiceRes++;
}
}

}
/**
* @brief execute le tri fusion sur un tableau
*
* @param tab le tableau a trie
* @param taille taille du tableau
*/
void triFusion(int* tab, int taille){
if (taille > 1){
int millieu = taille/2;
int* tab1 = copieSousTableau(tab,0,millieu);
int* tab2 = copieSousTableau(tab,millieu,taille);
triFusion(tab1,millieu);
triFusion(tab2,taille-millieu);
fusion(tab1,millieu,tab2,taille-millieu,tab);
free(tab1);
```

```
free(tab2);
}
}
```

triInsertion.c

```c
/**
* @file triInsertion.c
* @author Durand Thomas
* @brief ensemble des fonxtions lie au tri par insertion
* @version 0.1
* @date 2019-11-12
*
* @copyright Copyright (c) 2019
*
*/
#include <stdio.h>
#include <stdlib.h>


/**
* @brief execute le tri par insertion sur un tableau
*
* @param t le tableau
* @param len la longueur du tableau
*/
void tri_insertion(int* t,int len){
int i, j;
int temp;
for (i = 1; i < len; i++) {
temp = t[i];
for (j = i; j > 0 && t[j - 1] > temp; j--) {
t[j] = t[j - 1];
}
t[j] = temp;
}
}
```