Université Hassan II
Mohammedia-Casablanca

Ecole Normale Supérieure de
l'Enseignement Technique de
Mohammedia

Différentes Méthodes de tri

DAFLI Youness

GLSID1

INTRODUCTION :

Un tri est une opération de classement d'éléments d'**une liste selon un ordre total** défini.

Il existe différentes méthodes de tri comme le tri a bulle, insertion, sélection, rapide , shell, fusion …

Le but de ce mini projet est de comparer les différentes méthodes de tri selon une représentation graphique et savoir la complexité de chaque tri.

Pour ce faire , j'ai travaillé avec la bibliothèque winbgim pour tous ce qu'est graphique et avec un compilateur Dev++

Mon projet est composé de trois fichiers :

-tri.h : contient les prototypes des fonctions de tri.

-tris.cpp : contient les définitions des fonctions de tri.

-main.cpp : contient le programme principal.

1-Le contenu de fichier tri.h :

```
// prototypes des fonctions de tri :

void triSelect(int t[], int max);

void triShell(int tab[], int maxSaisi);

void triABulle(int t[], int max);

void triInsertion(int t[],int max);

void triRapide(int *t,int max);
```

Différentes méthodes de tri que j'ai utilisé : (contenu de fichier tris.cpp)

Algorithme	Code en c

Tri par insertion
 	
Le tri par insertion consiste à parcourir la liste : on prend les éléments dans l'ordre. Ensuite, on les compare avec les éléments précédents jusqu'à trouver la place de l'élément qu'on considère. Il ne reste plus qu'à décaler les éléments du tableau pour insérer l'élément considéré à sa place dans la partie déjà triée.
	`// le tri par insertion`
	
	`void triInsertion(int t[],int max)`
	`{`
	
	`int i,pos ,tmp ;`
	
	`for(i=1;i < max ; i++)`
	`{`
	`tmp = t[i];`
	`pos = i;`
	`while ( ( pos > 0 ) && ( tmp < t[pos-1] ) ){`
	
	`t[pos] = t[pos-1];`
	`pos = pos-1;`
	`}`
	`t[pos] = tmp;`
	`}`
	
	`}`

Tri par Sélection :
 	
A chaque étape i :
1 .On recherche parmi les t[i],.. t[n] le plus petit élément qui doit être positionné à la place i.
 	
2. Supposons cet élément à ind_min t[ind_min] et t[i] sont échangés.
 	
3. i = i+1, retourner en 1
	`//tri Selection`
	`void triSelect(int t[], int max)`
	`{`
	
	`int i, j;`
	
	`for(i = 0; i<max; i++) {`
	`int posmin = i;`
	
	`for(j = i + 1; j < max; j++)`
	`if(t[j] < t[posmin])`
	`posmin = j;`
	
	`int tmp = t[i];`
	`t[i] = t[posmin];`
	`t[posmin] = tmp;`
	`}`
	
	`}`

Le tri à bulle
 	
L'algorithme parcourt la liste ,et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet de la liste,
	`void triABulle(int t[],int max){`
	
	`int i;`
	`while(max >0){`
	
	`for(i=0;i<max-1 ;i++)`
	
	`if(t[i]>t[i+1])`
	`{`
	`int tmp=t[i];`

l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que la liste est triée : l'algorithme peut s'arrêter.	t[i]=t[i+1];
	t[i+1]=tmp;
	}
	max--; }
	
	}

Le tri Shell
 	
On pratique donc un tri insertion en déplaçant les valeurs d'un pas donné. On raffine ce pas jusqu'à la valeur 1.
	// triShell
	
	void triShell(int tab[], int maxSaisi)
	{ int pas=0, i, j, temp;
	while (pas < maxSaisi) // Calcul du pas
	pas = 3*pas+1;
	while (pas != 0) // tant que le pas est > 0
	{ pas = pas/3;
	for(i= pas; i < maxSaisi; i++)
	{ temp = tab[i]; // valeur à décaler éventuellement
	j = i;
	while((j>(pas-1)) && (tab[j-pas]> temp))
	{ tab[j] = tab[j-pas]; // échange des valeurs
	j = j-pas;
	}
	tab[j] = temp;
	}
	}
	}

Le tri rapide
 	
La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui lui sont inférieurs soient à sa gauche et que tous ceux qui lui sont supérieurs soient à sa
 	
droite. Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.
	int partitionner(int tableau[], int p, int r) {
	int pivot = tableau[p], i = p-1, j = r+1,temp;
	while (1) { do j--;
	while (tableau[j] > pivot);
	do i++;
	while (tableau[i] < pivot);
	if (i < j) {
	temp = tableau[i];
	tableau[i] = tableau[j];
	tableau[j] = temp;
	} else return j; } }
	void triRapide_(int *tableau, int p, int r) {
	
	int q;
	if (p < r) {
	q = partitionner(tableau, p, r);
	triRapide_(tableau, p, q);
	triRapide_(tableau, q+1, r); }
	
	}

Le contenu de fichier main.cpp :

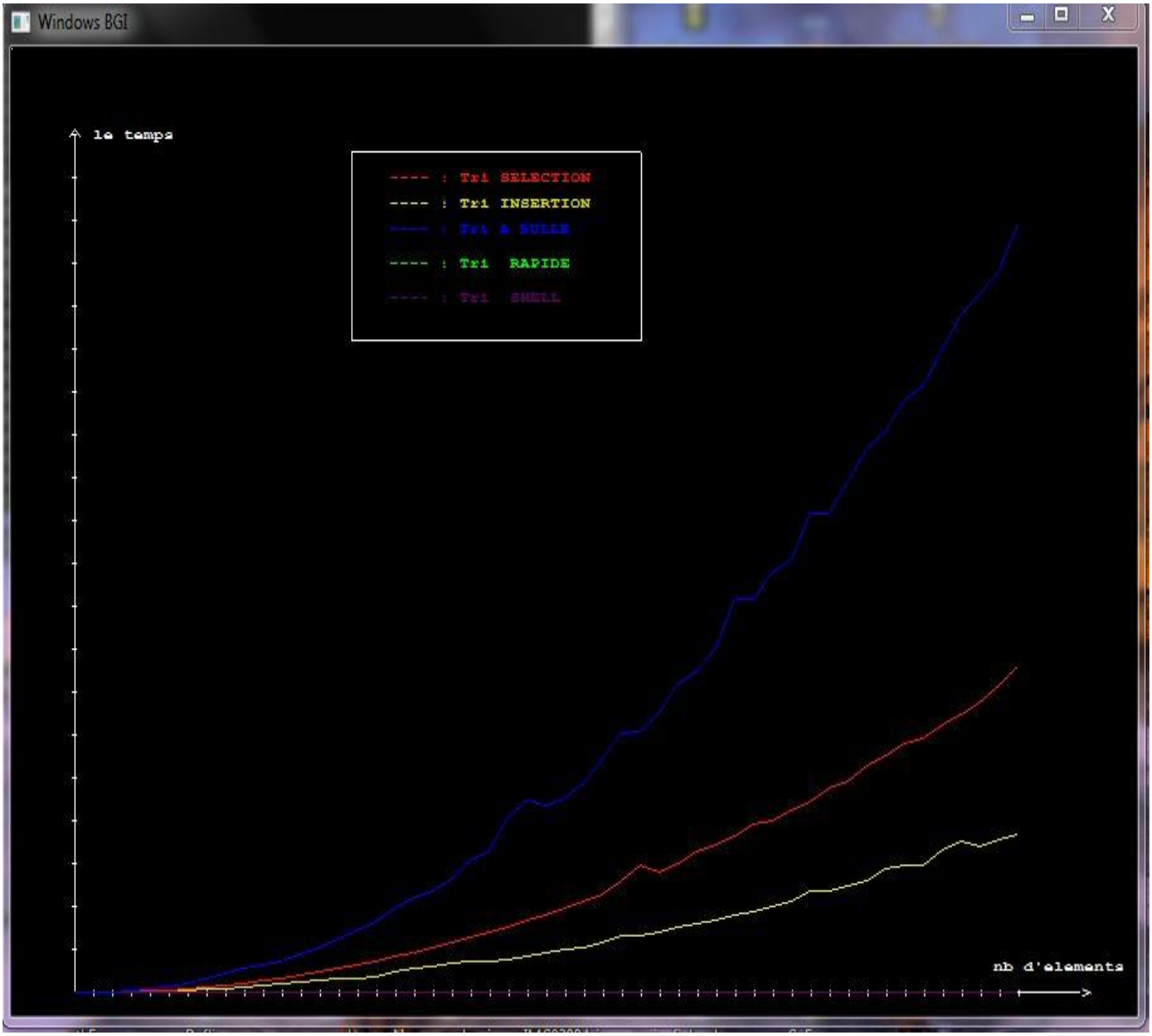
```
// fonction qui  permet de remplir le tableau  d'une manière aléatoire
void remplireTableau(int tab[], int taille)
{
   int i;
   for (i = 0; i < taille; i++)
      tab[i] = rand()%10; // rand() permet de générer une suite d'éléments d'une manière aléatoire.
                          // selon la taille
}
// fonction qui calcule  le temps après chaque appel d'une  fonction de tri
// Elle prend comme paramètre un pointeur de fonction de tri
double profileFunction(void (*f)(int*, int), int *tab, int max)
{
   clock_t debut = clock(); // récupération de temps avant l'appel  de fonction de tri
   f(tab, max); // appel d'une fonction de tri
   clock_t fin = clock(); //récupération de temps après  l'appel  de fonction de tri
   return (double) (fin - debut) / CLOCKS_PER_SEC;   // en divise sur le nombre de cycle  d'horloge
pour
}
```

```
// cette fonction permet : d'allouer un espace mémoire dynamique  à chaque appel d'une fonction

// la fonction permet l'appel de la fonction qui permet de remplir le tableau

// la fonction  permet de retourner le temps de tri en appel   la fonction profileFunction qui permet de le  // calculer


double faireTri(int nombreElements, void (*tri)(int[], int))

{

      int *tableau = (int*) malloc(nombreElements * sizeof(int)); // allocation mémoire

      remplireTableau(tableau, nombreElements);// remplissage  du tableau

      return profileFunction(tri, tableau, nombreElements); // récupération du  temps


}


// Transformation de repère original vers le nouveau repère

int _x(int x) {

   return x +50  ;

    }


int _y(int y) {


   return 550 - y;

}
```

/* cette fonction permet de tracer chaque point trier en fonction de type de tri attribué à la fonction en appelant les différentes fonctions définies en dessus */

```
void Tracer(void (*tri)(int[], int))
{
    int i;
     for(i = 0; i < nombreTests; ++i) {
     int nombreElements = 1000*i+1000 ; // à chaque itération j'augmente le nombre d'élément
                                        // jusqu'à 50000, car le nombreTests est fixé en 50 .
     double sec = faireTri(nombreElements, tri);
     lineto(_x((int)(nombreElements*0.015) ), _y((int) (sec*15)));     // le traçage du points
    printf("%d : lineto(%d, %d) \n",nombreElements, _x(nombreElements / 1000), _y((int) sec *15));
    }
    }
```

Représentation graphique comme celle-ci :



Conclusion :

La représentation graphique montre que le tri a bulle est le plus lent parmi les tris, après vient le tri sélection qui est plus rapide que le tri a bulle , après vient le tri insertion qui est rapide par apport au tri sélection, après vient les tris rapide et shell qui sont les plus rapide parmi les tri et on voit pas la différence entre eux car ils sont superposer