**travail pratique 1**

**La Complexité De Tri Rapide**

**Enseignant:**

**Pr. MOHAMED MEKNASSI**

**Réalisé par:**

**BERRAG AYOUB**

**EL BAGHDADI MOHAMED**

**SOMMAIRE**

[**Le Tri Rapide** 2](#_Toc527629790)

[**1.** **Principe de l’algorithme** 2](#_Toc527629791)

[**2.** **Algorithme** 2](#_Toc527629792)

[**3.** **Complexité** 2](#_Toc527629793)

[**4.** **Jeu d’essai** 3](#_Toc527629794)

[**Le Tri Par Sélection** 3](#_Toc527629795)

[**1.** **Principe de l’algorithme** 3](#_Toc527629796)

[**2.** **Algorithme** 4](#_Toc527629797)

[**3.** **Complexité** 4](#_Toc527629798)

[**4.** **Jeu d’essai** 4](#_Toc527629799)

[**1.** **Environnement de travail** 5](#_Toc527629800)

[**2.** **Expérience proprement dite** 5](#_Toc527629801)

[**3.** **Interprétation** 6](#_Toc527629802)

[**Annexe** 7](#_Toc527629803)

[**1.** **Tri rapide en C** 7](#_Toc527629804)

[**2.** **Tri par sélection en C** 8](#_Toc527629805)

[**3.** **Comparaison entre les deux en C** 8](#_Toc527629806)

# **Le Tri Rapide**

## **Principe de l’algorithme**

Le principe de ce tri est d’ordonner le vecteur T.(0)..T.(n) en cherchant dans celui-ci une clé pivot autour de laquelle réorganiser ses éléments. Il est souhaitable que le pivot soit aussi proche que possible de la clé relative à l’enregistrement central du vecteur, afin qu’il y ait à peu près autant d’éléments le précédant que le suivant, soit environ la moitié des éléments du tableau. On prend souvent le dernier élément du tableau.

On permute ceux-ci de façon à ce que pour un indice j particulier tous les éléments dont la clé est inférieure à pivot se trouvent dans T.(0)...T.(j) et tous ceux dont la clé est supérieure se trouvent dans T.(j+1)...T.(n). On place ensuite le pivot à la position j.

On applique ensuite le tri récursivement sur la partie dont les éléments sont inférieurs au pivot et sur la partie dont les éléments sont supérieurs au pivot. le code se trouve dans la partie Annexe.

## **Algorithme**

|  |
| --- |
| **Procedure tri\_rapide (tableau [1:n], gauche, droit )**  **DEBUT**  *// mur marque la separation entre les elements plus petits et ceux plus grands que pivot*  **mur ← gauche;**  *// On prend comme pivot l element le plus a droite*  **pivot ← tableau[droit];**  **placer a gauche de mur tout les elements plus petits**  **placer a droite de mur tout les element plus grands**  *// On place correctement le pivot*  **placer le pivot a la place de mur**  *// On poursuit par recursivite*  **SI (gauche<mur-1)**  **ALORS tri\_rapide(tableau,gauche,mur-1);**  **SI (mur+1<droit)**  **ALORS tri\_rapide(tableau,mur,droit);**  **FIN;** |

## **Complexité**

Dans le pire des cas, c'est à dire si, à chaque niveau de la récursivité le découpage conduit à trier un sous-tableau de 1 élément et un sous-tableau contenant tout le reste, la complexité du tri rapide est en O (n2).

Par contre, dans le cas moyen, cet algorithme a une complexité en O (n log n).

## **Jeu d’essai**

On remplit un tableau par 10 éléments qui ne sont pas tries.

tab[10]={1,2,3,432,13,12,6,31,43,0} ;

Puis on applique le tri rapide. Le résultat est ci-dessous.

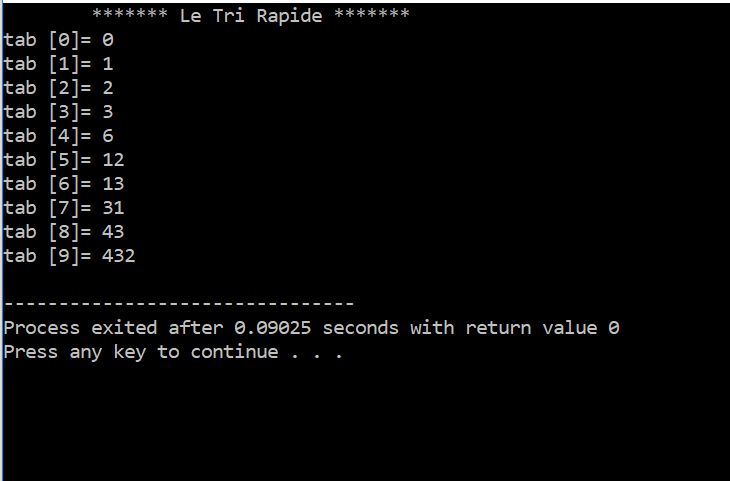


Figure : tri rapide

# **Le Tri Par Sélection**

## **Principe de l’algorithme**

Le principe du tri par sélection/échange (ou tri par extraction) est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second élément et d'aller chercher le plus petit élément du vecteur pour le mettre en second, etc...

Le code se trouve dans la partie Annexe.

## **Algorithme**

|  |
| --- |
| **procédure tri\_selection(tableau t, entier n)**  **pour i de 1 à n - 1**  **min ← i**  *//trouver j le plus petit élément de [i + 1:n]*  **pour j de i + 1 à n**  **si t[j] < t[min], alors min ← j**  **fin pour**  **si min ≠ i, alors échanger t[i] et t[min]**  **fin pour**  **fin procédure** |

## **Complexité**

Dans tous les cas l'algorithme effectuera n (n-1)/2 comparaisons. Sa complexité est donc en O (n2).

## **Jeu d’essai**

On remplit un tableau par 10 éléments qui ne sont pas tries.

tab[10]= {12,0,32,32,3,1,6,701,4,10} ;

Puis on applique le tri par sélection. Le résultat est ci-dessous.

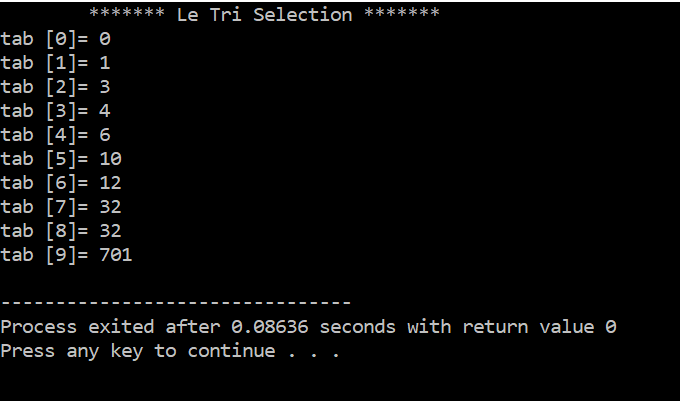


Figure : tri par sélection

**Expérimentation**

## **Environnement de travail**

On utilise une machine qui contient les caractéristiques suivantes :

**Processor :** Intel(R) Core (TM) M-5Y51 CPU @1.10 GHz 1.20 GHz.

**Memory :** Ram 8.00 GB.

## 

## **Expérience proprement dite**

On utilise la fonction Clock () disponible sous la librairie time.h en C pour calculer le temps d’exécution des deux tris. A chaque on change la dimension du tableau et on génère des nombre aléatoires pour le remplir grâce à la fonction rand () disponible sous la librairie stdlib.h en C.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nombres d’éléments. | 100 | 500 | 1000 | 5000 | 10000 | 20000 | 30000 |
| Temps d’exécution du tri rapide en ms. | 0.000000 | 0.000000 | 0.000000 | 0.047000 | 0.187000 | 0.687000 | 1.465000 |
| Temps d’exécution du tri par sélection en ms. | 0.000000 | 0.000000 | 0.015000 | 0.031000 | 0.203000 | 0.722000 | 1.547000 |

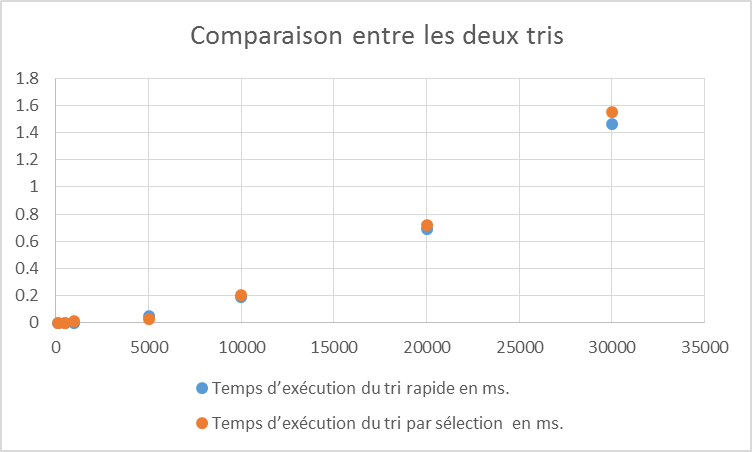


Figure : différence entre les deux tris

## **Interprétation**

L’expérience montre que le tri rapide (quick sort) est plus rapide que le tri par sélection, cela est visuel lorsqu’on augmente le nombre des éléments du tableau. On voit bien qu’après un tableau de 30000 éléments, le temps d’exécution du quick sort se distingue du l’autre. Le code utilisé dans l’expérience se trouve dans la partie Annexe.

# **Annexe**

## **Tri rapide en C**

|  |
| --- |
| #include "stdio.h"  void tri\_rapide (int \*tableau, int taille) {  int mur, courant, pivot, tmp;  if (taille < 2) return;  // On prend comme pivot l element le plus a droite  pivot = tableau[taille - 1];  mur = courant = 0;  while (courant<taille) {  if (tableau[courant] <= pivot) {  if (mur != courant) {  tmp=tableau[courant];  tableau[courant]=tableau[mur];  tableau[mur]=tmp;  }  mur ++;  }  courant ++;  }  tri\_rapide(tableau, mur - 1);  tri\_rapide(tableau + mur - 1, taille - mur + 1);  }  int main(){  int i;  int tab[10]={1,2,3,432,13,12,6,31,43,0};  tri\_rapide(tab,10);  printf("\t\*\*\*\*\*\*\* Le Tri Rapide \*\*\*\*\*\*\*\n");  for(i=0;i<10;i++)  printf("tab [%d]= %d\n",i,tab[i]);  return 0;  } |

## **Tri par sélection en C**

|  |
| --- |
| #include "stdio.h"  void tri\_selection(int \*tableau, int taille)  {  int en\_cours, plus\_petit, j, temp;  for (en\_cours = 0; en\_cours < taille - 1; en\_cours++)  {  plus\_petit = en\_cours;  for (j = en\_cours; j < taille; j++)  if (tableau[j] < tableau[plus\_petit])  plus\_petit = j;  temp = tableau[en\_cours];  tableau[en\_cours] = tableau[plus\_petit];  tableau[plus\_petit] = temp;  }  }  int main(){  int i;  int tab[10]={12,0,32,32,3,1,6,701,4,10};  tri\_selection(tab,10);  printf("\t\*\*\*\*\*\*\* Le Tri Selection \*\*\*\*\*\*\*\n");  for(i=0;i<10;i++)  printf("tab [%d]= %d\n",i,tab[i]);  return 0;  } |

## **Comparaison entre les deux en C**

|  |
| --- |
| #include "stdio.h"  #include"time.h"  #include"stdlib.h"  void tri\_rapide (int \*tableau, int taille) {  int mur, courant, pivot, tmp;  if (taille < 2) return;  // On prend comme pivot l element le plus a droite  pivot = tableau[taille - 1];  mur = courant = 0;  while (courant<taille) {  if (tableau[courant] <= pivot) {  if (mur != courant) {  tmp=tableau[courant];  tableau[courant]=tableau[mur];  tableau[mur]=tmp;  }  mur ++;  }  courant ++;  }  tri\_rapide(tableau, mur - 1);  tri\_rapide(tableau + mur - 1, taille - mur + 1);  }  void tri\_selection(int \*tableau, int taille)  {  int en\_cours, plus\_petit, j, temp;    for (en\_cours = 0; en\_cours < taille - 1; en\_cours++)  {  plus\_petit = en\_cours;  for (j = en\_cours; j < taille; j++)  if (tableau[j] < tableau[plus\_petit])  plus\_petit = j;  temp = tableau[en\_cours];  tableau[en\_cours] = tableau[plus\_petit];  tableau[plus\_petit] = temp;  }  }  int main(){  int i,j=35000;  int tab[j];  for(i=0;i<j;i++)  tab[i]=rand();  clock\_t beginSel = clock();  tri\_selection(tab,j);  clock\_t endSel = clock();  double time\_spent\_selection = (double)(endSel - beginSel) / CLOCKS\_PER\_SEC;  clock\_t beginRid = clock();  tri\_rapide(tab,j);  clock\_t endRid = clock();  double time\_spent\_rapide = (double)(endRid - beginRid) / CLOCKS\_PER\_SEC;  printf("le tempte d'excution pour selecation est %f \n",time\_spent\_selection);  printf("le tempte d'excution pour rapide est %f \n",time\_spent\_rapide);    return 0;  } |