QuickSort

1) Principe

Trier le tableau en l' "**arrangeant**" **petit à petit**.
On le **découpe en 2 "sous tableaux"**, avec comme **milieu** une valeur prise du tableau qu'on appel **pivot**, et on met les valeur **plus petite** que le pivot **à gauche** et les **plus grand à droite**. Puis on **répète** ce fonctionnement sur les **2 sous tableaux**, ainsi de suite.
C'est donc un **algorithme récursif**.

Il existe **plusieurs variante** du processus, surtout par rapport au **choix du pivot** et ce qu'on en fait.

2) Choix du pivot

a) Pivot arbitraire

Choix **constant** : toujours le premier, le dernier, le milieu ou une autre **position fixe**.
Cette technique est généralement **moins efficace** que de choisir le pivot aléatoirement.

b) Pivot optimal

Choix **optimisé** : **valeur médiane** du sous-tableau prise. Mais le calcule **prend du temps** et n'est pas suffisamment efficace dans la pratique, cette variante est **peu utilisée**.

c) Pivot aléatoire

Choix **aléatoire** : toujours calculé aléatoirement.
L'avantage est que l'algorithme s'écarte peu du temps d'**exécution moyen** grâce au côté aléatoire.
Avec ce choix, cet algorithme devient **probabiliste**, il utilise une source de hasard. C'est ce choix qui sera **retenu pour la suite**.
Avec un choix de pivot aléatoire, le pivot est donc aléatoirement placé dans le tableau, nous sommes donc obligé de le placer au début ou à la fin du tableau pour ne pas être gêné et ensuite le replacer au milieu.

3) Processus et implémentation en langage C

a) La base du quicksort

1) On **partitionne** le tableau en **2 tableaux** avec le **pivot**

2) On rappel **quicksort sur les 2 tableaux**

3) tout ça **uniquement si le tableau n'est pas fini** (plus d'une case)

```
void quick_sort(int array[], int start, int end)
{
   int pivot;
   if (start < end)
   {
      pivot = partion(array, start, end);
      quick_sort(array, start, pivot-1);
      quick_sort(array, pivot+1, end);
   }
}
```

b) Partitionnement

1) On choisit Pivot **aléatoirement**

2) On met **Pivot à la fin**

3) On boucle du début

    a) Si **Actuel < Pivot** alors on met **Actuel à gauche**

4) On met **Pivot** juste **à droite du sous tableau de gauche**

5) On retourne Pivot

```
int partion(int array[], int start, int end)
{
   int pivotIndex = start + rand()%(end - start + 1);
   int pivot;
   int left = start - 1;
   int temp;
   pivot = array[pivotIndex];
   swap(&array[pivotIndex], &array[end]);
   for (temp = start; temp < end; temp++)
   {
      if (array[temp] < pivot)
      {
         left++;
         swap(&array[left], &array[temp]);
      }
```

```
    }
    swap(&array[left+1], &array[end]);
    return left + 1;
}
```

c) Méthode swap (facultatif)

Une simple méthode qui prend **deux pointeur d'entier** (qui sont à l'intérieur du tableau) et **échange leur valeur**.

```
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

d) Méthodes d'initialisation/randomisation/affichage d'un tableau (facultatif)

1) **Initialisation** d'un tableau

```
for (i = 0; i < MAX; i++)
  array[i] = i;
```

2) **Randomisation** d'un tableau

```
    srand(time(NULL));
    int i, j, temp;
    for (i = MAX - 1; i > 0; i--)
    {
        j = rand()%(i + 1);
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
```

3) **Affichage** d'un tableau

```
    for (i = 0; i < MAX; i++)
        printf("%d ", array[i]);
```

e) Code complet

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

#define MAX 20


void random_shuffle(int array[])
{
   srand(time(NULL));
   int i, j, temp;
   for (i = MAX - 1; i > 0; i--)
   {
      j = rand()%(i + 1);
      temp = array[i];
      array[i] = array[j];
      array[j] = temp;
   }
}


void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}


int partion(int array[], int start, int end)
{
   int pivotIndex = start + rand()%(end - start + 1); //generates a random number as a pivot
   int pivot;
   int left = start - 1;
   int temp;
```

```
   pivot = array[pivotIndex];
   swap(&array[pivotIndex], &array[end]);
   for (temp = start; temp < end; temp++)
   {
      if (array[temp] < pivot)
      {
         left++;
         swap(&array[left], &array[temp]);
      }


   }
   swap(&array[left+1], &array[end]);
   return left + 1;
}


void quick_sort(int array[], int start, int end)
{
   int pivot;
   if (start < end)
   {
      pivot = partion(array, start, end);
      quick_sort(array, start, pivot-1);
      quick_sort(array, pivot+1, end);
   }
}


int main()
{
   int i;
   int array[MAX];


   for (i = 0; i < MAX; i++)
      array[i] = i;


   random_shuffle(array);


   for (i = 0; i < MAX; i++)
       printf("%d ", array[i]);
```

```
    printf("\n");


    quick_sort(array, 0, MAX-1);


    for (i = 0; i < MAX; i++)
        printf("%d ", array[i]);
    printf("\n");


    return 0;
}
```