

## Tri rapide

AlgoProg 2015-16 276

## Fonction mystère

```
void mystere (int t[],int n){
    int i=0;
    int j= n-1;
    while(i < j){
        if (t[i] == 0)
            i= i+1;
        else {
            echanger(t,i,j);
            j= j-1;}
    }
}
```

Que fait-elle en supposant que le tableau T ne contient que des 0 et des 1

AlgoProg 2015-16 277

## Fonction mystère

```
void mystere (int t[],int n){
    int i=0;
    int j= n-1;
    while(i < j){
        if (t[i] == 0)
            i= i+1;
        else {
            echanger(t,i,j);
            j= j-1;}
    }
}
```

1 0 0 1 1 1 0 1 1 0

AlgoProg 2015-16 278

## Fonction mystère

```
void mystere (int t[],int n){
    int i=0;
    int j= n-1;
    while(i < j){
        if (t[i] == 0)
            i= i+1;
        else {
            echanger(t,i,j);
            j= j-1;}
    }
}
```

Elle place les 0 au début du tableau et les 1 à la fin

AlgoProg 2015-16 279

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont **inférieurs** ou **supérieur et égal** à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .

0	1	2	3	4	5	6	7	8	9
5	1	5	3	6	2	8	4	1	7

15-16 280

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont **inférieurs** ou **supérieur et égal** à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .

0	1	2	3	4	5	6	7	8	9
1	1	2	3	6	8	4	5	7	5

15-16 281

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont **inférieurs** ou **supérieur** à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .
  - Placer  $x$  entre les 2 zones.

0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	8	6	5	7	5

15-16 282

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont **inférieurs** ou **supérieur** à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .
  - Placer  $x$  entre les 2 zones.

Que peut-on dire de  $x$  ?

0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	8	6	5	7	5

15-16 283

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont inférieurs ou supérieur à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .
  - Placer  $x$  entre les 2 zones.

$x$  est à la place qu'il occupera dans le tableau trié



15-16 284

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont inférieurs ou supérieur à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .
  - Placer  $x$  entre les 2 zones.

Combien de fois chaque élément de  $t$  est-il examiné ?



15-16 285

## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont inférieurs ou supérieur à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .
  - Placer  $x$  entre les 2 zones.

Combien de fois chaque élément de  $t$  est-il examiné ?  
une seule fois



15-16 286

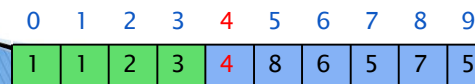
## Fonction mystère

Idée :

- Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont inférieurs ou supérieur à une valeur  $x$ .
- Choisir  $x$  parmi les éléments de  $t$  : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
  - Séparer les éléments de  $t$  en 2 zones par rapport à  $x$ .
  - Placer  $x$  entre les 2 zones.

$x$  est à la place qu'il occupera dans le tableau trié

- répéter 1, 2 et 3 sur chaque zone pour trier  $t$  récursivement



15-16 287

## Fonction mystère

Idée :

Utiliser la fonction `mystere` pour séparer les éléments d'un tableau selon qu'ils sont inférieurs ou supérieur à une valeur `x`.

- Choisir `x` parmi les éléments de `t` : donc il existe  $i \in [0, n-1]$  tel que  $t[i] = x$ .
- Séparer les éléments de `t` en 2 zones par rapport à `x`.
- Placer `x` entre les 2 zones.

`x` est à la place qu'il occupera dans le tableau trié

- répéter 1, 2 et 3 sur chaque zone pour trier `t` récursivement

Question :

Comment choisir `x` ?

`t[0]` ou `t[n-1]` ou heuristique

On appelle `x` le pivot



15-16 288

## Tri rapide

Exemple :

<https://www.youtube.com/user/AlgoRythmics/videos>

## Exemple

0	1	2	3	4	5	6	7	8	9
5	3	9	2	7	4	1	6	5	8

`p`

0	1	2	3	4	5	6	7	8	9
5	3	9	2	7	4	1	6	5	8

`i`

`j`

AlgoProg 2015-16 290

## Exemple

<code>p</code>									
5	3	9	2	7	4	1	6	5	8

`i`

`j`

`t[i] >= t[p]` donc `echanger(t, i, j)`

AlgoProg 2015-16 291

## Exemple

8	3	9	2	7	4	1	6	5	5
---	---	---	---	---	---	---	---	---	---

*i* *j* *p*

AlgoProg 2015-16 292

## Exemple

8	3	9	2	7	4	1	6	5	5
---	---	---	---	---	---	---	---	---	---

*i* *j* *p*

`t[i] >= t[p] donc echanger(t, i, j)`

AlgoProg 2015-16 293

## Exemple

5	3	9	2	7	4	1	6	8	5
---	---	---	---	---	---	---	---	---	---

*i* *j* *p*

AlgoProg 2015-16 294

## Exemple

5	3	9	2	7	4	1	6	8	5
---	---	---	---	---	---	---	---	---	---

*i* *j* *p*

`t[i] >= t[p] donc echanger(t, i, j)`

AlgoProg 2015-16 295



## Exemple

1	3	9	2	7	4	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

$t[i] < t[p]$  donc pas d'échange

AlgoProg 2015-16 300

## Exemple

1	3	9	2	7	4	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

AlgoProg 2015-16 301

## Exemple

1	3	9	2	7	4	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

$t[i] \geq t[p]$  donc `echanger(t,i,j)`

AlgoProg 2015-16 302

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

AlgoProg 2015-16 303

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

$t[i] < t[p]$  donc pas d'échange

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

i j

$t[i] < t[p]$  donc pas d'échange

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

j

i



## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

j

i

$t[i] \geq t[p]$  donc `echanger(t,i,j)`

AlgoProg 2015-16 308

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

j

i

AlgoProg 2015-16 309

## Exemple

1	3	4	2	7	9	6	5	8	5
---	---	---	---	---	---	---	---	---	---

j

i

$i > j$  donc placer  $t[p]$  à la place de  $t[i]$

AlgoProg 2015-16 310

## Exemple

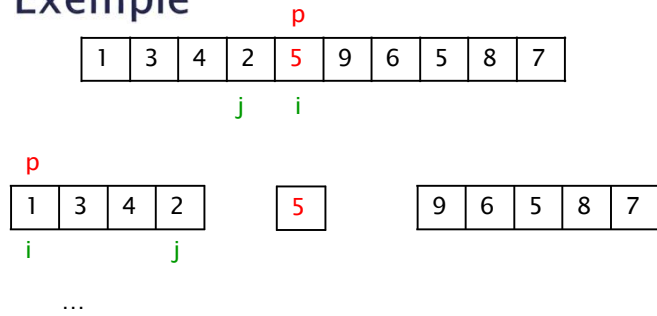
1	3	4	2	5	9	6	5	8	7
---	---	---	---	---	---	---	---	---	---

j

i

AlgoProg 2015-16 311

## Exemple



AlgoProg 2015-16 312

## Séparation autour du pivot

```
int separationPivot(int t[],int n, int d, int f) {
    int p = d;
    int i = d;
    int j = f;
    while(i<=j){
        if (t[i]<t[p])
            i=i+1;
        else {
            echanger(t,i,j);
            if (i==p)
                p=j;
            else {
                if (j==p)
                    p=i;
                j=j-1;
            }
        }
    }
    echanger (t, i, p);
    return (p);
}
```

En sortie le tableau t est réorganisé entre les indices d et f :

- >  $d \leq k < p$ ,  $t[k] < t[p]$
- >  $p \leq k \leq f$ ,  $t[k] \geq t[p]$
- >  $t[p]$  est à sa place.
- > Chaque élément a été examiné une seule fois
- > Complexité en temps :  $O(f-d)$
- > Complexité espace :  $O(1)$

AlgoProg 2015-16 313

## Séparation autour du pivot

Il existe de nombreuses variantes de cette fonction :

- > Heuristique pour le choix du pivot.
- > Sans placement du pivot.
- > Etc

AlgoProg 2015-16 314

## Tri rapide

```
void
triRapideRec(int t[], int n, int d, int f){
    int p;
    if (d<f) {
        p=separationPivot(t,n,d,f);
        triRapideRec(t,n,d,p-1);
        triRapideRec(t,n,p+1,f);
    }
}

void triRapide(int t[], int n){
    triRapideRec(t,n,0,n-1);
}
```

- > Complexité en temps :  $\Omega(n \log_2(n))$  et  $O(n^2)$
- > Complexité en espace :  $O(1)$
- > Stabilité : non

AlgoProg 2015-16 315

## Tris récursifs

Tri	Complexité en temps	Complexité espace	Stabilité
Fusion	$\Theta(n \log_2(n))$	$\Theta(\log_2(n))$ sur la pile $\Theta(n)$ en mémoire	Oui
Tri rapide	$\Omega(n \log_2(n))$ et $O(n^2)$	$\Theta(1)$	Non*

AlgoProg 2015-16 316

## Tris rapide

La méthode pour choisir le pivot influence directement l'efficacité du tri rapide.

En moyenne le tri rapide est l'algorithme de tri le plus efficace.

Quicksort

AlgoProg 2015-16 317