

Tri rapide



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

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;}  
    }
```



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;}
    }
```

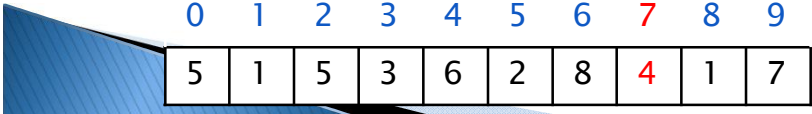
Elle place les 0 au début du tableau et les 1 à la fin

Fonction mystère

Idée :

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

1. Choisir x parmi les éléments de t : donc il existe $i \in [0, n-1]$ tel que $t[i] == x$.



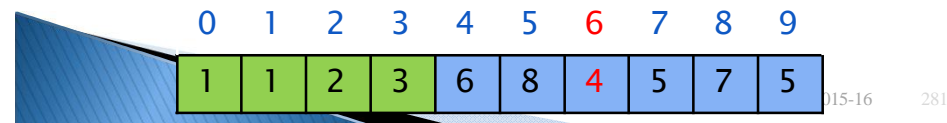
0	1	2	3	4	5	6	7	8	9
5	1	5	3	6	2	8	4	1	7

015-16 280

Fonction mystère

Idée :

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



015-16

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

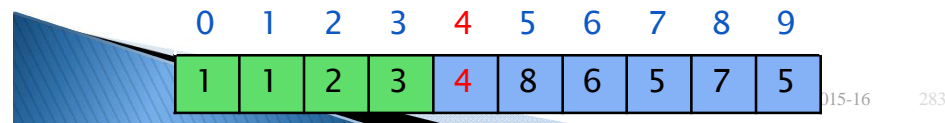


Fonction mystère

Idée :

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

Que peut-on dire de x ?

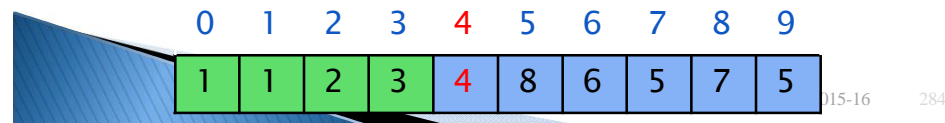


Fonction mystère

Idée :

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

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



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

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



015-16

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

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



015-16

286

Fonction mystère

Idée :

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

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

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



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 .

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

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

4. 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



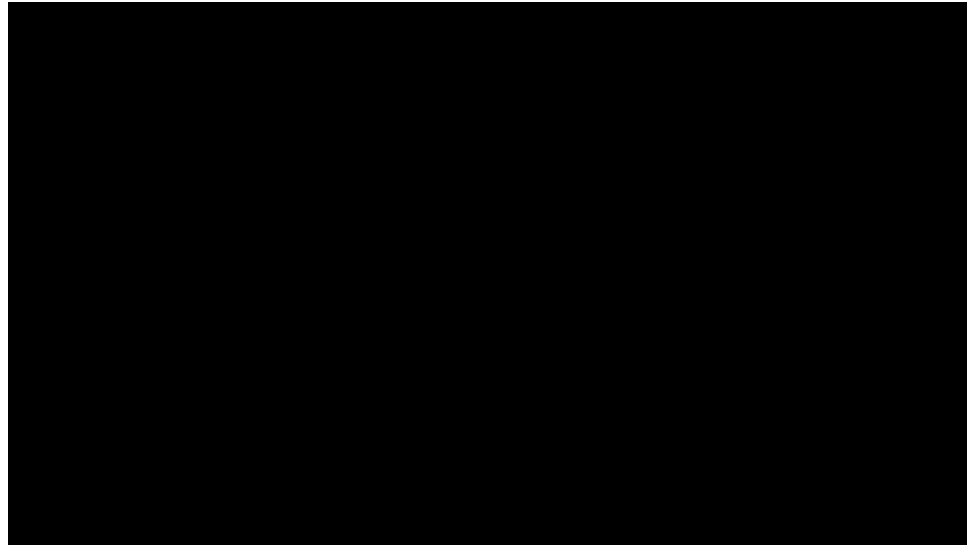
015-16

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

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

i

j



Exemple

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

ⁱ

^j

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



Exemple

8	3	9	2	7	4	1	6	5	5
									p
i								j	

Exemple

5	3	9	2	7	4	1	6	8	5
									p
i							j		

Exemple

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

i

j

p

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



Exemple

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



Exemple

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

i

j

p

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



Exemple

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

i

j

p

$t[i] < t[p]$ donc pas d'échange



Exemple

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



Exemple

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

i

j

p

$t[i] < t[p]$ donc pas d'échange



Exemple

1	3	9	2	7	4	6	5	8	5
		i		j					p



Exemple

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

p

i

j

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



Exemple

1	3	4	2	7	9	6	5	8	5
		i		j					p



Exemple

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

p

i

j

$t[i] < t[p]$ donc pas d'échange



Exemple

1	3	4	2	7	9	6	5	8	5
			i	j					p



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
---	---	---	---	---	---	---	---	---	---

p

j

i



Exemple

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

p

j

i

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



Exemple

1	3	4	2	7	9	6	5	8	5
			j	i					p



Exemple

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

p

j i

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

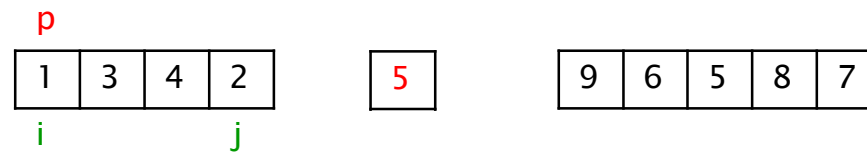
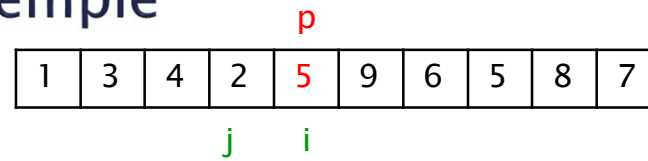


Exemple

1	3	4	2	5	9	6	5	8	7
			j	i					
				p					



Exemple



...



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 : $O(1)$

Séparation autour du pivot

Il existe de nombreuses variantes de cette fonction :

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



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



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*

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

