Méthode « Diviser pour Résoudre »

E. CHABBAR

DVR

- Les algorithmes sont regroupés en familles selon certains concepts t. q. Division pour Résoudre, Glouton, Programmation dynamique.
- \- L'aspect DVR consiste à diviser le problème initial (de taille n) en sous-problèmes similaires de tailles plus petites (en général de taille n/2, n/3,…).

  \- Ces sous-problèmes sont résolus récursivement.

  \- On peut combiner ces solutions récursives pour avoir la solution du problème initial.

DVR
Exemples

1. Calcul de x^n ($n \geq 1$)

- Le nombre de multiplications par la méthode classique ($x^n = x\, x^{n-1}$) est de l'ordre de n.
- La méthode DVR exploite la définition suivante:

$$x^n = \begin{cases} 1 & \text{si } n=1 \\ x^p \,.\, x^p & \text{si } n = 2p \\ x \,.\, x^p \,.\, x^p & \text{si } n = 2p+1 \end{cases}$$

- pour calculer x^n on fait appel (récursif) au calcul x^p ou $p = E(n/2)$. (algo. Slide suivant)

- si T(n) est le nombre de multiplications pour calculer x^n alors $T(n) = T(n/2) + 1$ si n est pair ou $T(n) = T(n/2) + 2$ si n est impair. On a, dans tous les cas, $T(n) = T(n/2) + O(1)$.

Comme il y a $\log_2 n$ divisions euclidiennes successives (avant d'avoir 1 comme quotient) , donc **$T(n) = O(\log(n))$**. (pour s'en convaincre prendre n une puissance de 2)

$(T(n)=O(1)+O(1)+\ldots+O(1)$ $\log_2 n$ fois)

DVR
Exemples

1. **Calcul de x^n ($n \geq 1$)**

```
 puissBin(x,n)
Début
        si n = 1 alors retourner x
        sinon
              si n mod 2 = 0 alors
                            y := puissBin(x,n/2)
                           retourner (y*y)
              sinon
                            y := puissBin(x,n/2)
                           retourner (x*y*y)
              fsi;
        fsi;
Fin.
```

DVR
Exemples

2. RECHERCHE DICHOTOMIQUE

- La recherche séquentielle d'un élément x dans un tableau A à n éléments est de l'ordre de n.
- La recherche dichotomique <u>exige</u> que le tableau soit **trié**. Elle consiste à:

  \- comparer x à l'élément du milieu

  \- si c'est différent, x peut se trouver soit dans la moitié gauche soit dans la moitié droite du tableau A, selon que x < A[milieu] ou x> A[milieu].

DVR
Exemples

2. **Recherche dichotomique**

```
 rechDicho(A,inf,sup,x)
début
        si inf ≤ sup alors
              m := (inf + sup)/2
              si x = A[m] retourner m;
                            si x < A[m] alors retourner rechDicho(A,inf,m-1,x);
                            sinon  retourner rechDicho(A,m+1,sup,x);
                            fsi;
              fsi;
        sinon retourner 0;
        fsi;
fin
```

DVR
Exemples

2. **Recherche dichotomique**

- Complexité de la recherche dichotomique

Dans le pire des cas (i.e. x n'est pas dans le tableau) le nombre de comparaisons T(n), pour chercher x dans un tableau A[1..n] à n éléments, vérifie:

$$T(1) = 1$$

$$T(n) = T(n/2) + 1 \quad , \quad n>1$$

- la solution de cette équation dépend du nombre d'itérations (nombre de divisions par 2)

DVR
Exemples

Itération	Nbre d'élts du ss tableau
0	n
1	$n/2$
2	$n/4$
3	$n/8$
.	.
.	.
p	$n/2^p$

L'algorithme utilise p itérations(et aussi p comparaisons) et s'arrête lorsque $n/2^p = 1$, c.à d. $p = \log_2 n$.

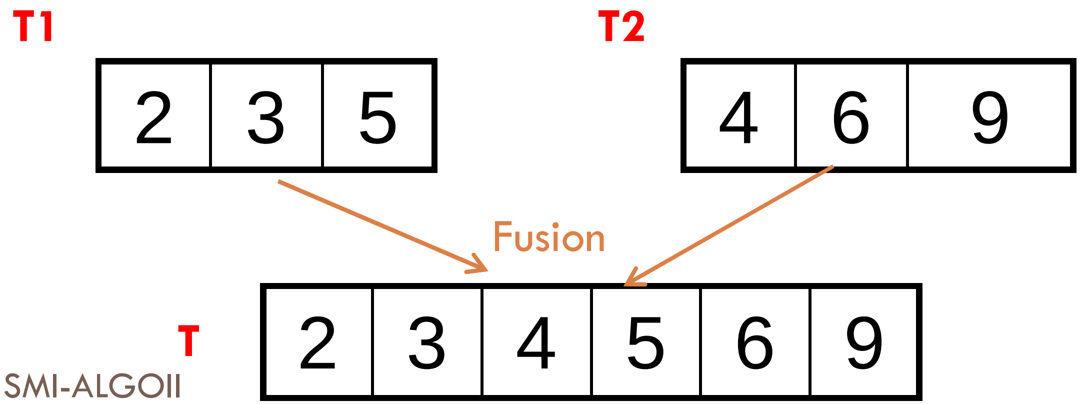
par conséquent, la recherche dichotomique est en $O(\log_2 n)$, i.e.

$$T(n) = O(\log_2 n)$$

3. Tri par fusion

- Fusion de deux tableaux triés.

  Etant donnés deux tableaux triés T1[1..n1] et T2[1..n2]. La fusion consiste à construire un tableau T[1..n1+n2] contenant tous les éléments de T1 et T2 dans l'ordre croissant.



Fusion


```
Fusion(T1,n1,T2,n2)
// T est un tableau qui contient le résultat de la fusion
  i1:=1; i2:= 1; k:=1;
  tantque (i1≤n1) et (i2≤n2) faire
       si T1[i1] ≤ T2[i2] alors
                    T[k] := T1[i1];
                     k:=k+1;  i1 := i1 +1;
       sinon
                    T[k] := T2[i2];
                    k:=k+1;  i2 := i2 + 1;
       fsi;
  ftantque;
  // on recopie les élts restants dans l'un des //tableaux Ti dans le tableau T
  si  i1 ≤ n1  alors  copier(T1,i1,n1,T,k)
  sinon          copier(T2,i2,n2,T,k)
  fsi;
retourner (T);
```

```
copier(A,d,f,B,j)
début
        pour i:=d à f  faire
                    B[j] := A[i]
                    j := j+1;
        fpour;
retourner(B);
fin
```


la complexité de la fusion est en $O(n1+n2)$

TriFusion

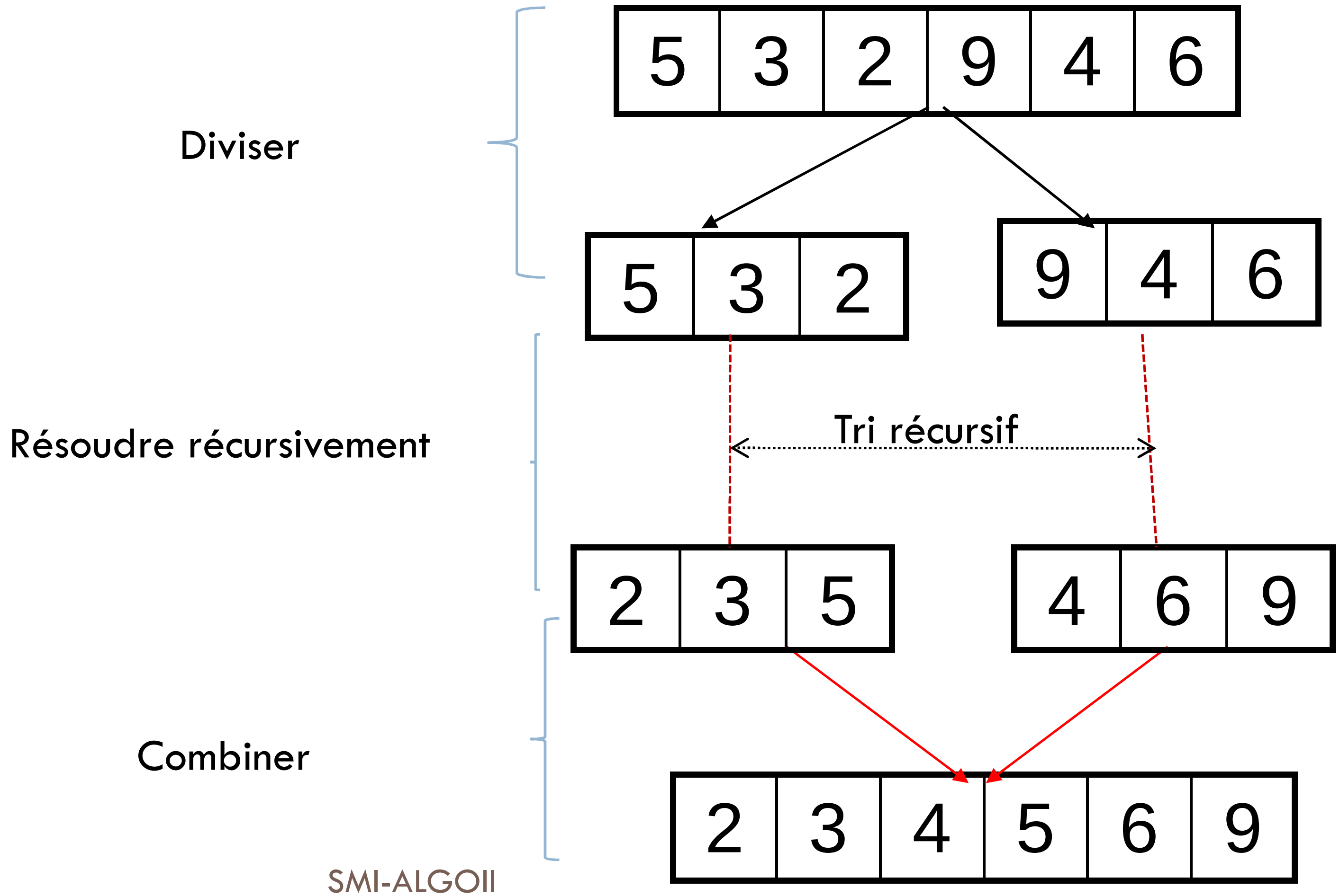
- Le tri par fusion est un exemple typique de la stratégie diviser pour résoudre, à savoir:

1. Diviser le tableau T[1..n] en deux sous-tableaux T[1..E(n/2)] et T[E(n/2)+1..n]
2. Trier (récursivement) les deux sous-tableaux (2 appels récursifs à la même fonction TriFusion).
3. Fusionner les deux sous-tableaux;

Ceci est schématisé par l'exemple suivant:

TriFusion

-


Diviser
Résoudre récursivement
Combiner
5 3 2 9 4 6
5 3 2
9 4 6
Tri récursif
2 3 5
4 6 9
2 3 4 5 6 9

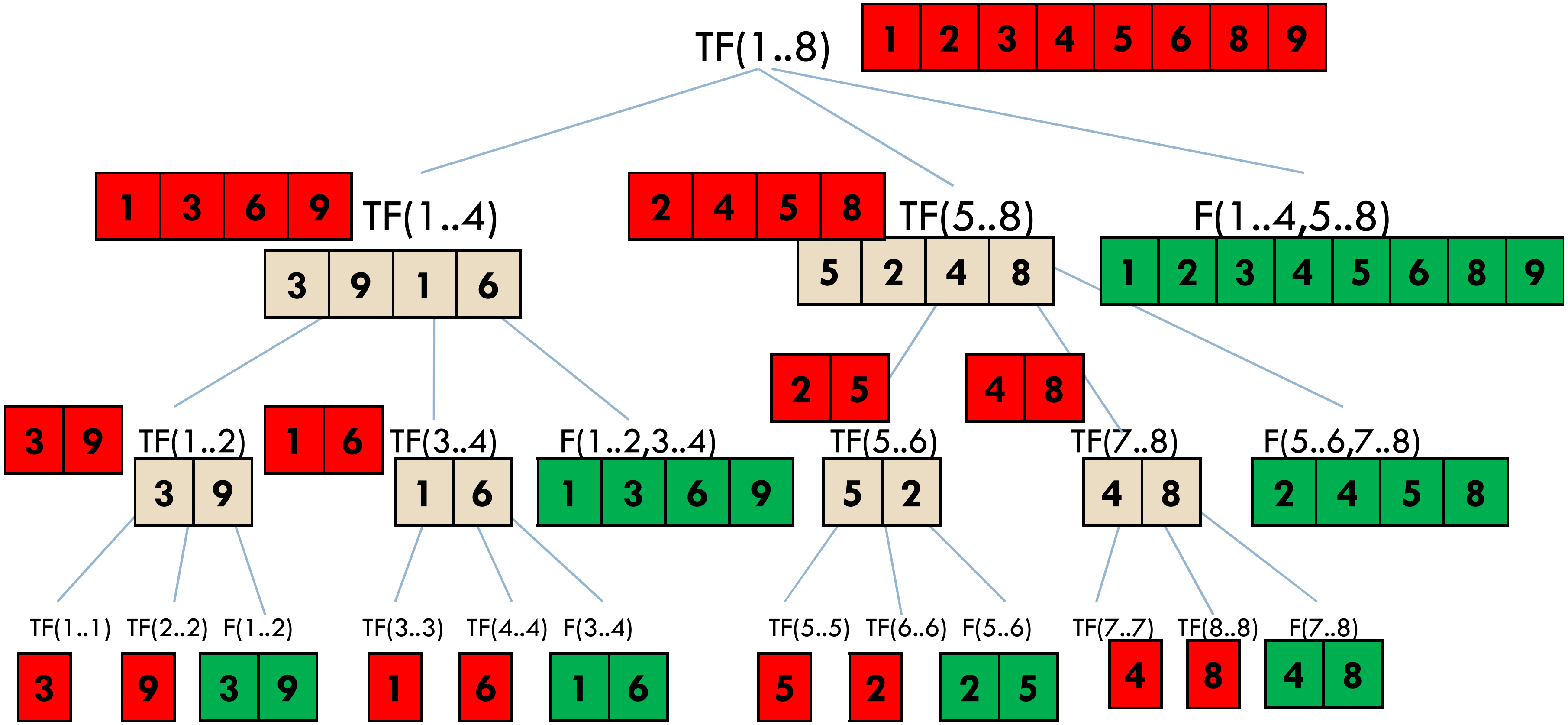
TriFusion


```
TriFusion(T, n)
// T1, T2 : des tableaux (locaux) de longueurs variables à chaque appel
début
    si n > 1 alors
            copier(T, 1, n/2, T1,1);
            copier(T, n/2 +1, n, T2, 1);
            T1 := TriFusion(T1, n/2);
            T2 := TriFusion(T2, n – n/2);
            T := Fusion(T1, n/2, T2, n – n/2);
    fsi;
retourner(T);
fin
```

Retourner(Fusion(TriFusion(T1,n/2),n/2,TriFusion(T2,n-n/2),n-n/2)

TriFusion: exemple d'exécution

Tableau à trier:	3	9	1	6	5	2	4	8



Recopie de ss-tableau

Tableau (vert)retourné à l'appel récursif

Fusion des 2 ss-tableaux(rouges) résultats des appels récursifs

DVR: Compléxité

- La complexité T(n) pour trier un tableau de n éléments par l'algorithme TriFusion vérifie:

$$\begin{cases} T(1) = 0 \\ T(n) = 2\,T(n/2) + O(n) \;\;,\;\; n > 1 \end{cases}$$

(Il y a 2 appels récursifs, chacun porte sur la moitié du tableau. O(n) pour recopier les 2 ss-tableaux en 2xO(n/2) + leur fusion en O(n))).

Equation de récurrence des alg.DVR

- La récurrence, utilisée par les algorithmes type DVR, est souvent de la forme:

$$T(n) = \begin{cases} \mathbf{O(1)} & \textbf{pour } \mathbf{n=1} \\ \mathbf{a\,T(n/b) + O(n^d)} & \mathbf{,\ n > 1} \end{cases}$$

a : est le nombre de divisions du problème initial en sous-problème (nombre d'appels récursifs)

n/b : la taille de chaque sous-problème ($b \geq 2$)

$O(n^d)$: le temps nécessaire pour décomposer le problème en sous-problème + le temps pour combiner les solutions des ss-problèmes pour avoir la solution du problème initial.

DVR: Résultat de Complexité

- **Théorème:**

Soit T : IN IR+ une fonction croissante à partir d'un certain rang, telle qu'il existe des entiers $n_0 \geq 1$, $b \geq 2$ et des réels $d \geq 0$, $a > 0$ pour lesquels

$$T(n_0) - k$$

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d \qquad n > n_0, \qquad \frac{n}{n_0} \; puissance\ de\ b$$

Alors on a :

$$T(n) = \begin{cases} O(n^d) & \text{si } a < b^d \\ O(n^d \log_b n) & \text{si } a = b^d \\ O(n^{\log_b a}) & \text{si } a > b^d \end{cases}$$

DVR: Résultat de Complexité

soit $T(n) = aT\left(\frac{n}{b}\right) + cn^d$.

On montre par récurrence sur p (p≥1), que : $T(n) = a^p\, T\left(\frac{n}{b^p}\right) + cn^d \sum_{j=0}^{p-1} (\frac{a}{b^d})^j$

prenons $\frac{n}{n_0} = b^p$*,et donc* $p = O(\log_b n)$*. La relation prédente devient :*

$$T(n) = ka^p + cn^d \sum_{j=0}^{p-1} (\frac{a}{b^d})^j = f(n) + cn^d g(n) \text{ , où :}$$

$$f(n) = ka^p = O\left(n^{\log_b a}\right) \text{ et } g(n) = \sum_{j=0}^{p-1} (\frac{a}{b^d})^j$$

du fait que : $a^p = e^{p \ln a} = e^{\log_b \frac{n}{n_0} \ln a} = e^{\ln\frac{n}{n_0}\frac{\ln a}{\ln b}} = (\frac{n}{n_0})^{\log_b a} = O(n^{\log_b a})$

$g(n)$ *est une suite géométrique de raison* $r = \frac{a}{b^d}$ *, par suite* $g(n) = \frac{r^p - 1}{r - 1}$ $(r \neq 1)$

premier cas: si $r = \frac{a}{b^d} = 1$*, alors* $g(n) = p = O(\log_b n)$ *et par conséquent* $T(n) = O\left(n^d \log_b n\right)$ *si* $a = b^d$

deuxième cas: $g(n) = \frac{1 - r^p}{1 - r} \leq \frac{1}{1 - r}$ *si* $r = \frac{a}{b^d} < 1$ *et* $g(n) = O(1)$ *dans ce cas ,* *par conséquent* $T(n) = O(\, n^d)$ *si* $a < b^d$

troisième cas: $g(n) = \frac{r^p - 1}{r - 1} \leq r^p$ *si* $r = \frac{a}{b^d} > 1$ *et* $g(n) = O\left(n^{-d}\, n^{\log_b a}\right)$ *du faite que:*

$$r^p = \frac{a^p}{b^{pd}} = O\left(n^{\log_b a}\right)(\frac{n}{n_0})^{-d} = O\left(n^{\log_b a}\right) O(n^{-d})$$

d'où $T(n) = O\left(\, n^{\log_b a}\right)$ *si* $a > b^d$

DVR: Résultat de Complexité

D'une manière générale. Si

$$\begin{cases} T(1) = O(1) \\ T(n) = a\,T\left(\frac{n}{b}\right) + O(n^d) \end{cases} \qquad (n>1,\ a > 0,\ b > 1,\ d \geq 0)$$

Alors

$$T(n) = \begin{cases} O(n^d) & \text{si } a < b^d \\ O(n^d \log_b n) & \text{si } a = b^d \\ O(n^{\log_b a}) & \text{si } a > b^d \end{cases}$$

DVR

- Applications

1. Pour la recherche dichotomique et le calcul de la puissance on a : $T(n) = T(n/2) + O(1)$

   donc $T(n) = O(\log n)$ $(a=1 , b=2, d=0)$

2. Pour le TriFusion on a : $T(n) = 2\,T(n/2) + O(n)$

   Alors $T(n) = O(n \log_2 n)$ $(a=2, b=2, d=1)$

3. $T(1) = 1$

   $T(n) = 2t(n/2) + O(n^2)$

   a pour solution $T(n) = O(n^2)$ $(a=b=d=2)$

DVR : tri rapide (quickSort)

- Soit à trier le tableau T[g..d] (au départ g=1 et d=n)
- Le principe de l'algorithme réside dans une procédure, appelée **partition**, qui réorganise les éléments de T autour d'un pivot (élément du tableau choisi au hasard) de sorte que :

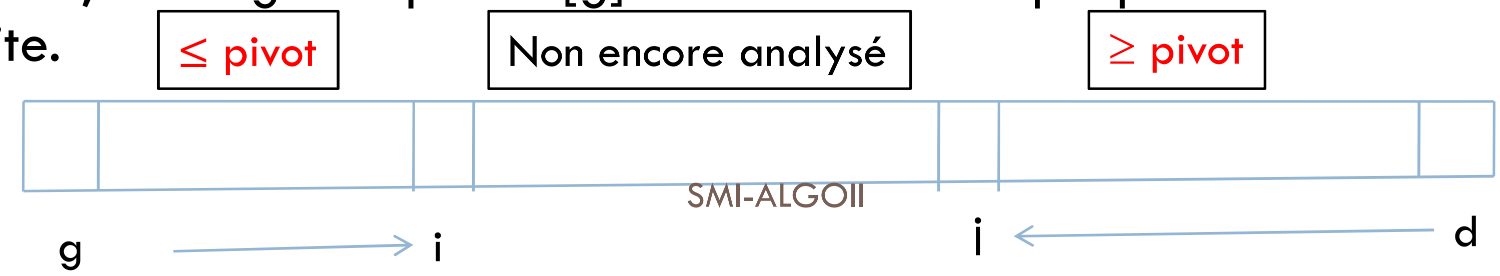
  1) Il existe un indice p ($g \leq p \leq d$) tel que p est la position définitive du pivot (T[p] = pivot).

  2) tous les éléments T[g], …, T[p-1] sont inférieurs ou égaux à T[p].

  3) tous les éléments T[p+1], …T[d] sont supérieurs ou égaux à T[p].

DVR : tri rapide

- Le travail de la fonction partition consiste à:
  - Choisir un élément du tableau comme pivot(par exemple le premier T[g])
  - Parcourir le tableau depuis la gauche(de gauche à droite) jusqu'à rencontrer un élément ≥ T[g]
  - Parcourir le tableau depuis la droite (de droite à gauche) jusqu'à rencontrer un élément ≤ T[g]
  - Echanger ces deux éléments dans le tableau
  - Continuer ce processus jusqu'à ce que les deux indices (de gauche et de droite) se croisent.
  - Finalement, échanger le pivot T[g] et l'élément indiqué par l'indice de droite.



DVR : Tri Rapide


```
//T[n+1] = + ∝
Partition(T, g, d)
début
      pivot := T[g];
      i := g;  j := d+1;
      tantque  i < j  faire
          i := i +1;
           tantque  T[i] < pivot  faire  i:=i+1;
           ftantque
          j := j -1;
           tantque  T[j] > pivot  faire  j:=j-1;
           ftantque
         si  i < j  alors  échanger(T, i, j); fsi
      ftantque
      échanger(T, g, j);
      retourner(j)
fin
```

- Exemple

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	i	j
3	1	4	1	5	9	2	6	5	3	5	8	9		
3	1	4	1	5	9	2	6	5	3	5	8	9	3	10
3	1	**3**	1	5	9	2	6	5	**4**	5	8	9	3	10
3	1	3	1	5	9	2	6	5	4	5	8	9	5	7
3	1	3	1	**2**	9	**5**	6	5	4	5	8	9	5	7
3	1	3	1	**2**	9	5	6	5	4	5	8	9	6	5
2	1	3	1	**3**	9	5	6	5	4	5	8	9		5

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

DVR : Tri Rapide

- La fonction partition, appliquée à un tableau T, produit trois sous-tableaux:
  - Un sous-tableau réduit à un seul élément T[p] qui garde sa place définitive dans le tri de T, et
  - Deux sous-tableaux T[g .. p-1], T[p+1 .. d].
- Pour trier T, il suffit d'appliquer récursivement le même algorithme sur les deux sous-tableaux.

DVR : Tri Rapide


```
quickSort(T, inf, sup)
début
   si  inf  < sup  alors
        p := partition(T, inf, sup);
        quickSort(T, inf, p-1);
        quickSort(T, p+1, sup);
   fsi
fin
```

Complexité du Tri rapide

La complexité de la fonction partition, appliquée à T[1..n], est en O(n).

- Cas le plus défavorable :

Cas où le pivot sort, à chaque fois, en premier (ou en dernier) élément (T: tableau trié).

La partition coupe le tableau en un morceau de un élément et un morceau de n-1 éléments, dans ce cas on a :

$$C(n) = C(n-1) + O(n)$$

(O(n) est le coût de la partition)

On en déduit que C(n) est en $O(n^2)$

Complexité du Tri rapide

- Cas le plus favorable :

cas où le pivot est l'élément médiane de T.

La partition coupe T en deux morceaux de taille n/2

$C(n) = 2\ C(n/2) + O(n)$

ce qui donne: $C(n) = O(n \log n)$

La complexité moyenne est aussi de l'ordre de n log n

Complexité du Tri rapide

- ## Complexité moyenne du tri rapide

La formule de récurrence donnant le nombre de comparaisons effectuées par le tri rapide pour une permutation aléatoire de n éléments vérifie :

$C_0 = C_1 = 0$ et

$$C_n = n - 1 + \frac{1}{n} \sum_{i=0}^{n-1} (C_i + C_{n-i-1}) \qquad \text{pour } n \geq 2$$

Le terme générique C_n peut s'écrire :

$$C_n = n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} C_i$$

Complexité moyenne du tri rapide

- $C_n = n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} C_i$

$$C_n = n - 1 + \frac{2}{n} C_{n-1} + \frac{2}{n} \sum_{i=0}^{n-2} C_i$$

$$C_n = n - 1 + \frac{2}{n} C_{n-1} + \frac{n-1}{n} \left(n - 2 + \frac{2}{n-1} \sum_{i=0}^{n-2} C_i\right) - \frac{(n-1)(n-2)}{n}$$

$$C_n = \frac{2}{n} C_{n-1} + \frac{n-1}{n} C_{n-1} + \frac{2n-2}{n}$$

$$C_n = \frac{n+1}{n} C_{n-1} + \frac{2(n-1)}{n}$$

En posant $D_n = \frac{C_n}{n+1}$ On aura la récurrence : $D_n = D_{n-1} + \frac{2}{n+1} - \frac{2}{n(n+1)}$

En négligeant le dernier terme de D_n on a: $D_n \simeq \log n$

Du fat que : $1 + \frac{1}{2} + \dots + \frac{1}{n} \simeq Ln(n)$ d'où C_n est en $O(n \log n)$