Méthode - Diviser pour Résoudre

Algorithmique II

SMI - S3

Diviser pour Résoudre

- 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.

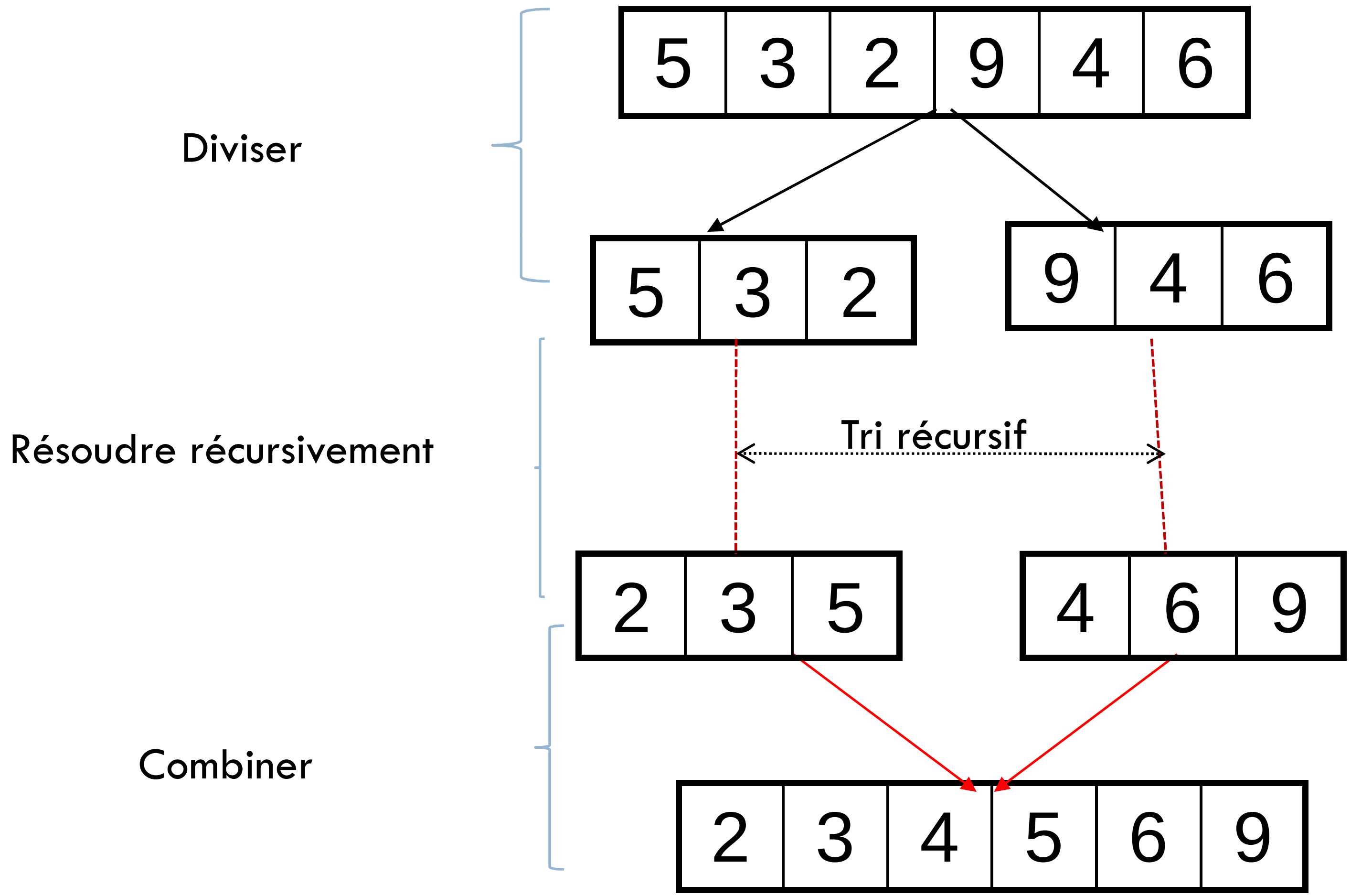
Example: TriFusion

- 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:

Example: TriFusion



TriFusion

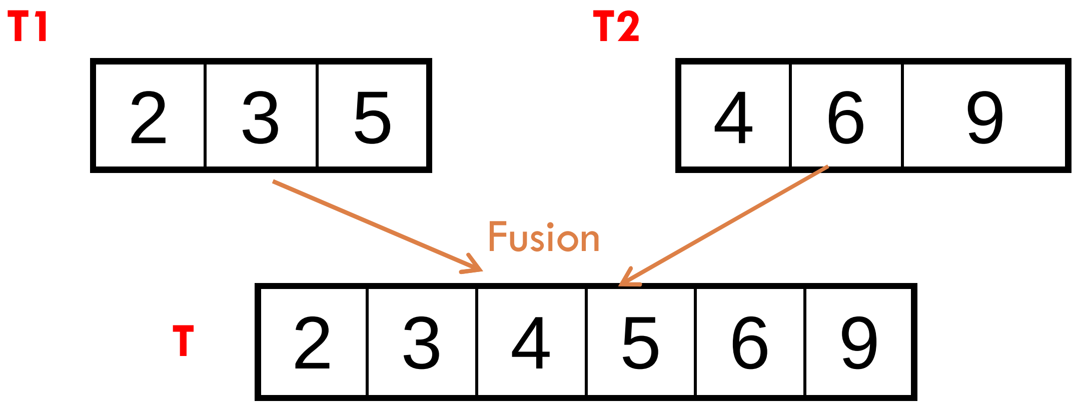

```
TriFusion(T, n)
// T1, T2 : des tableaux (locaux) de longueurs variables à chaque appel
début
    si n > 1 alors
            copier ( T ,      0      ,  n/2 - 1   , T1 , 0 )
            copier ( T ,   n/2       ,  n - 1     , T2  , 0 )
             T1  <--  TriFusion ( T1  ,  n/2 )
             T2  <--  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)

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.



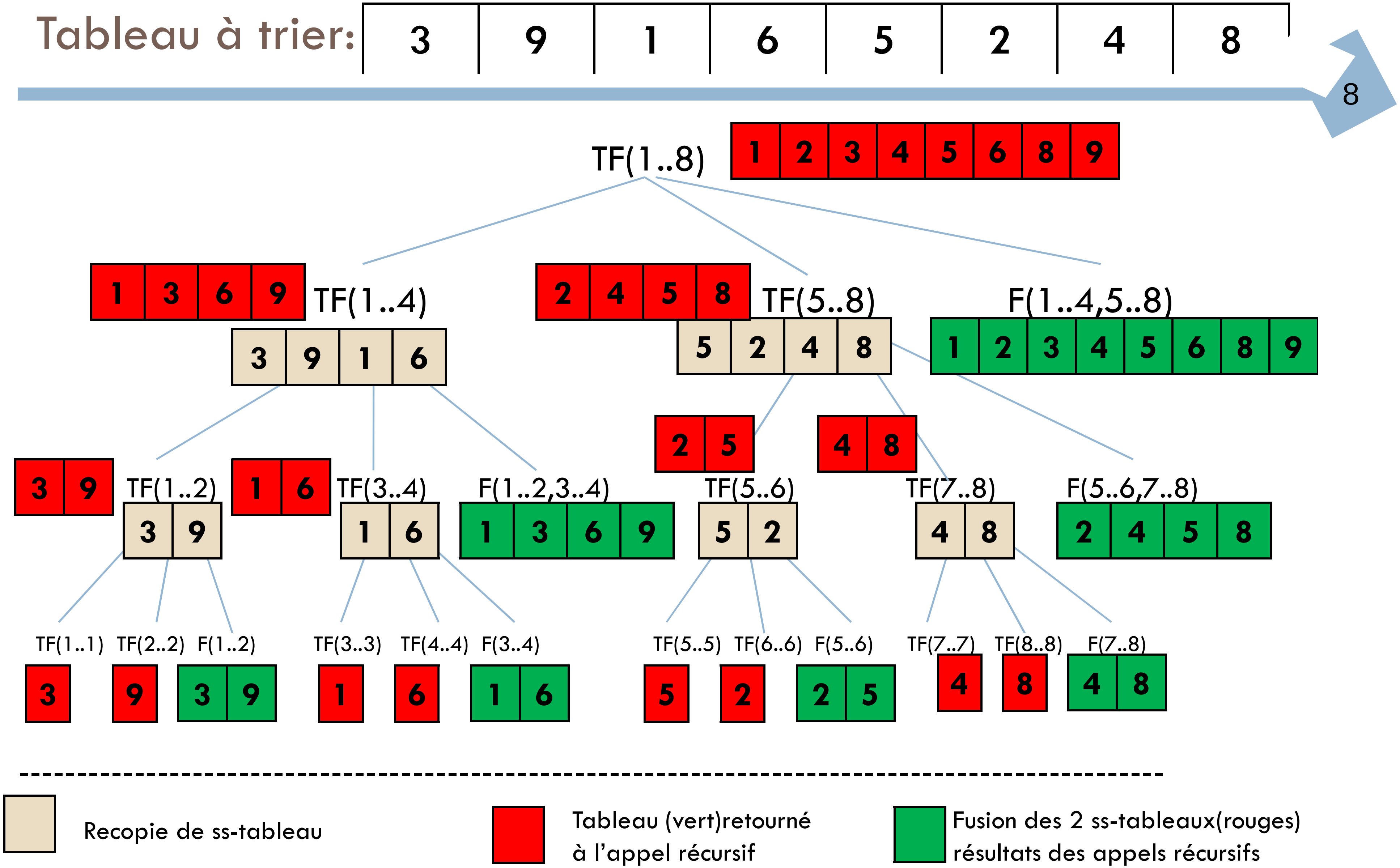
Fusion


```
Fusion(T1,n1,T2,n2)
// T est un tableau qui contient le résultat de la fusion
  i1:=0; i2:= 0; k:=0;
  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,k)
début
          pour i:=d à f  faire
                         B[k] := A[i]
                         k := k+1;
          fpour;
retourner(B);
fin
```

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

TriFusion: exemple d'exécution



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 $d \geq 0$, $a > 0$, $c > 0$ et $e > 0$ pour lesquels

$$T(n_0) - k$$

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

Alors on a :

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

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

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

2. $T(1) = 1$

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

   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 :

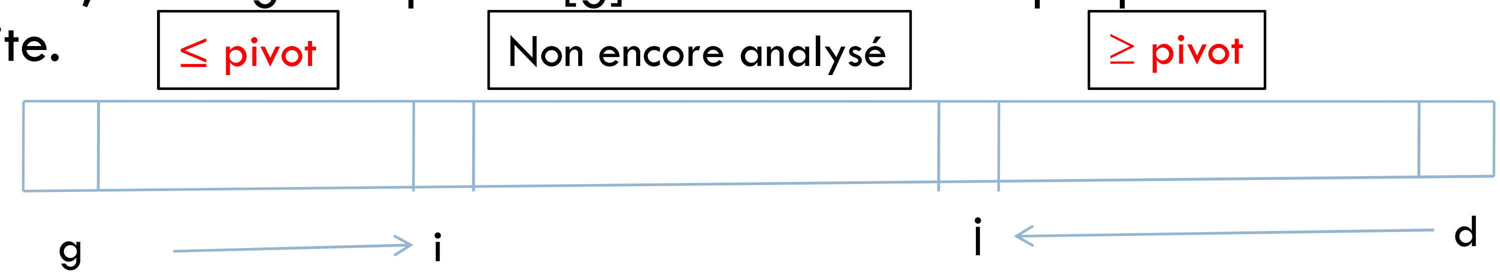
  1) Il existe un indice p ($g \leq p \leq d$) tel que p est la position définitive du pivot (T[p] = pivot).

  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	i	j
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 :

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

Complexité moyenne du tri rapide

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

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

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

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

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

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 $O(n \log n)$