Algorithmes de tri

Algorithmique 1 - 2018-2019

Stéphane Grandcolas

Aix-Marseille Université

2018-2019

Tris.

- *Arbre de décision* : tri par insertion.
- Complexité des tris par comparaison dans le pire des cas : borne minimale.
- *Tri rapide*.
- *Tri par dénombrement*, *tri par base*.

Tris.

organiser un ensemble d'objets selon un ordre déterminé

relation d'ordre : comparaison de clés

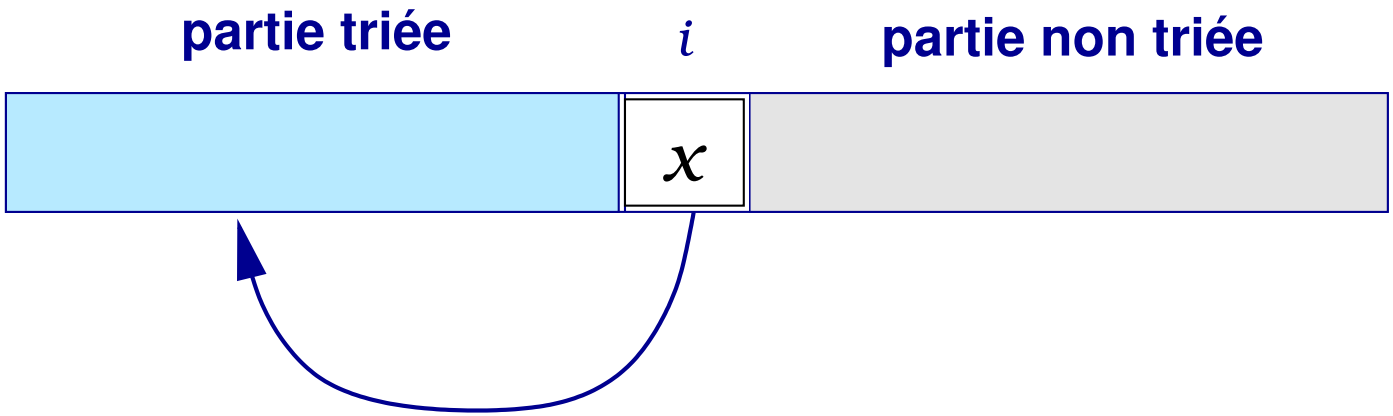
dans nos exemple nous confondrons les objets avec leurs clés

- tri par comparaison versus tri par indexation
- tri sur place : espace mémoire de taille constante
- tri stable : préserve l'ordre initial en cas d'égalité

Tris.

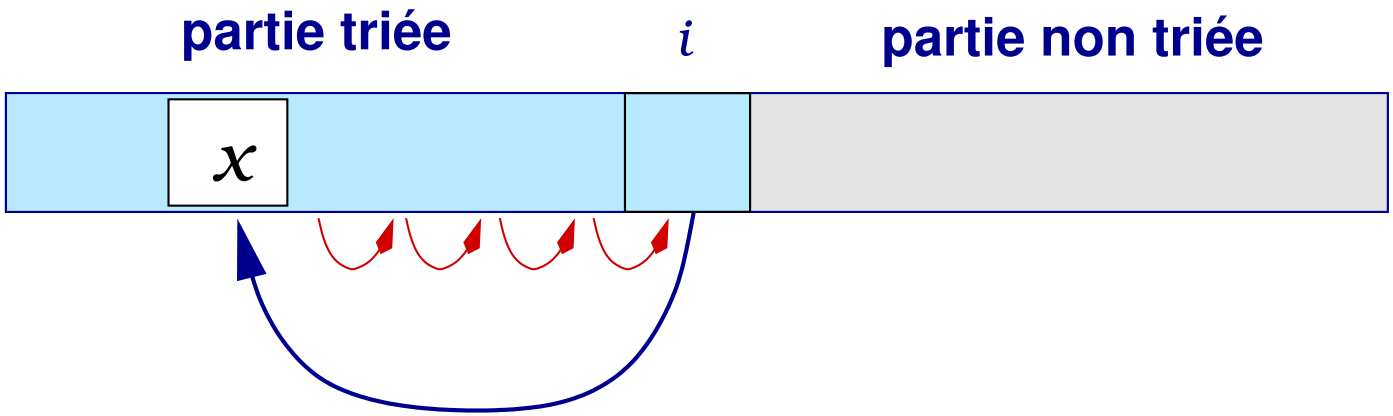
1. **Tris en $O(n^2)$.**
   - tri à bulles,
   - tri par insertion,
   - tri par sélection.
2. **Tris en $O(n \times \log\ n)$.**
   - tri par fusion,
   - tri par tas,
   - tri rapide (mais en $O(n^2)$ dans le pire des cas).
3. **Tris spéciaux.**
   - tri shell (probablement $O(n^{1.25})$),
   - tri par dénombrement ($O(n)$).

Tri par insertion



Principe : insérer les éléments les uns après les autres dans la partie triée

Tri par insertion



Principe : insérer les éléments les uns après les autres dans la partie triée

Insertion : recopie des éléments plus grands vers la droite

Tri par insertion

```
procédure TRI_PAR_INSERTION(T[1,...,n])
début
    pour i := 2 jusqu'à n faire
        j := i,
        tant que ((j > 1) et (T[j] < T[j − 1])) faire
            // la suite T[1,...,j − 1] est ordonnée
            PERMUTER(T, j, j − 1),
            j := j − 1,
        fin faire
    fin faire
fin procédure
```

Tri par insertion

Pire des cas : $i - 1$ permutations à chaque passage.

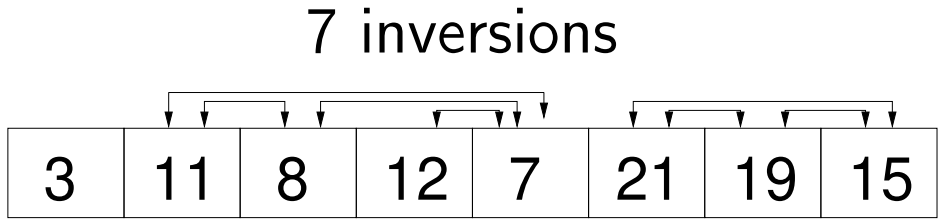
$$T(n) = \sum_{i=2}^{n} i = O(n^2)$$

Meilleur des cas : aucune permutation (mais une comparaison).

$$T(n) = \sum_{i=2}^{n} 1 = O(n)$$

Nombre d'inversions

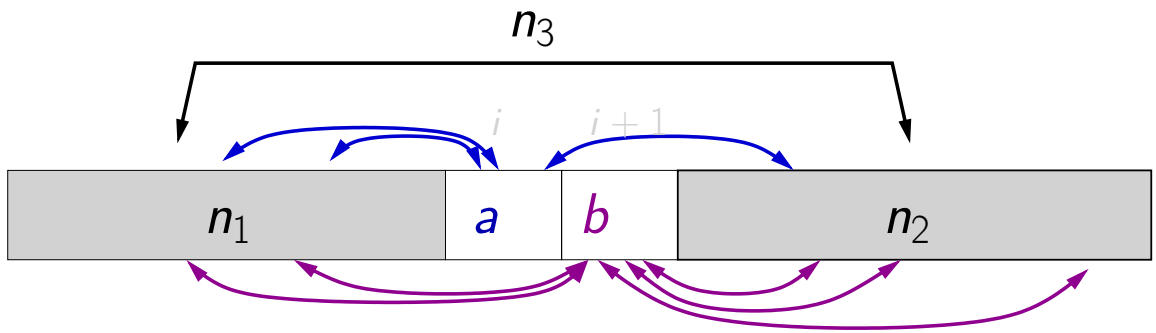
Inversion : i, j tels que $i < j$ et $T[i] > T[j]$



- suite triée : 0 inversions
- suite triée dans l'ordre décroissant : $n \times (n-1)/2$ inversions

Nombre d'inversions

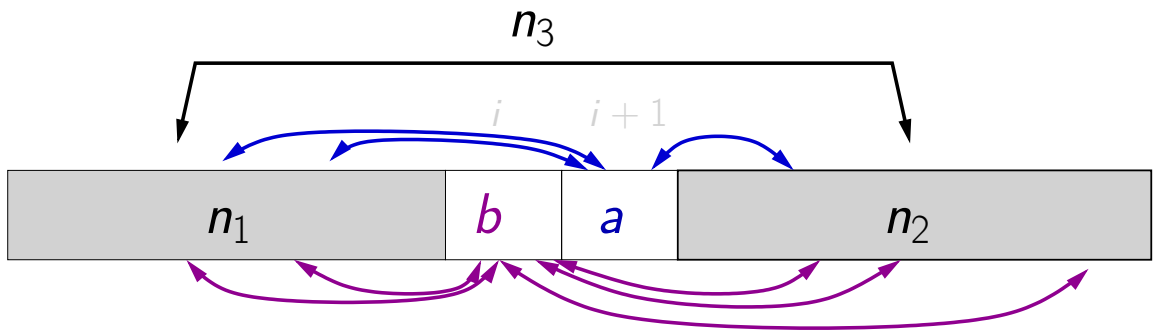
Permutation de deux éléments consécutifs inversés :



$$nb\ inversions = n_1 + n_2 + n_3 + nbInv(i) + nbInv(i+1) + 1$$

Nombre d'inversions

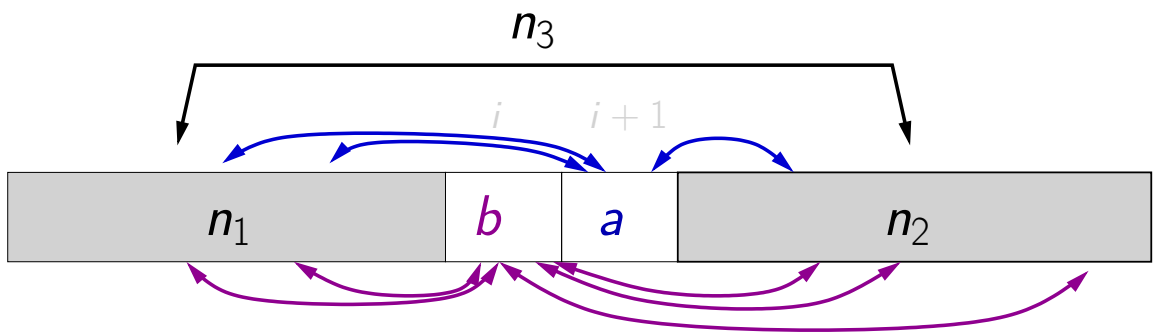
Permutation de deux éléments consécutifs inversés :



$$nb\ inversions = n_1 + n_2 + n_3 + nbInv(i) + nbInv(i+1)$$

Nombre d'inversions

Permutation de deux éléments consécutifs inversés :



$$nb\ inversions = n_1 + n_2 + n_3 + nbInv(i) + nbInv(i+1)$$

En permutant deux éléments successifs non ordonnés on diminue de 1 le nombre d'inversions

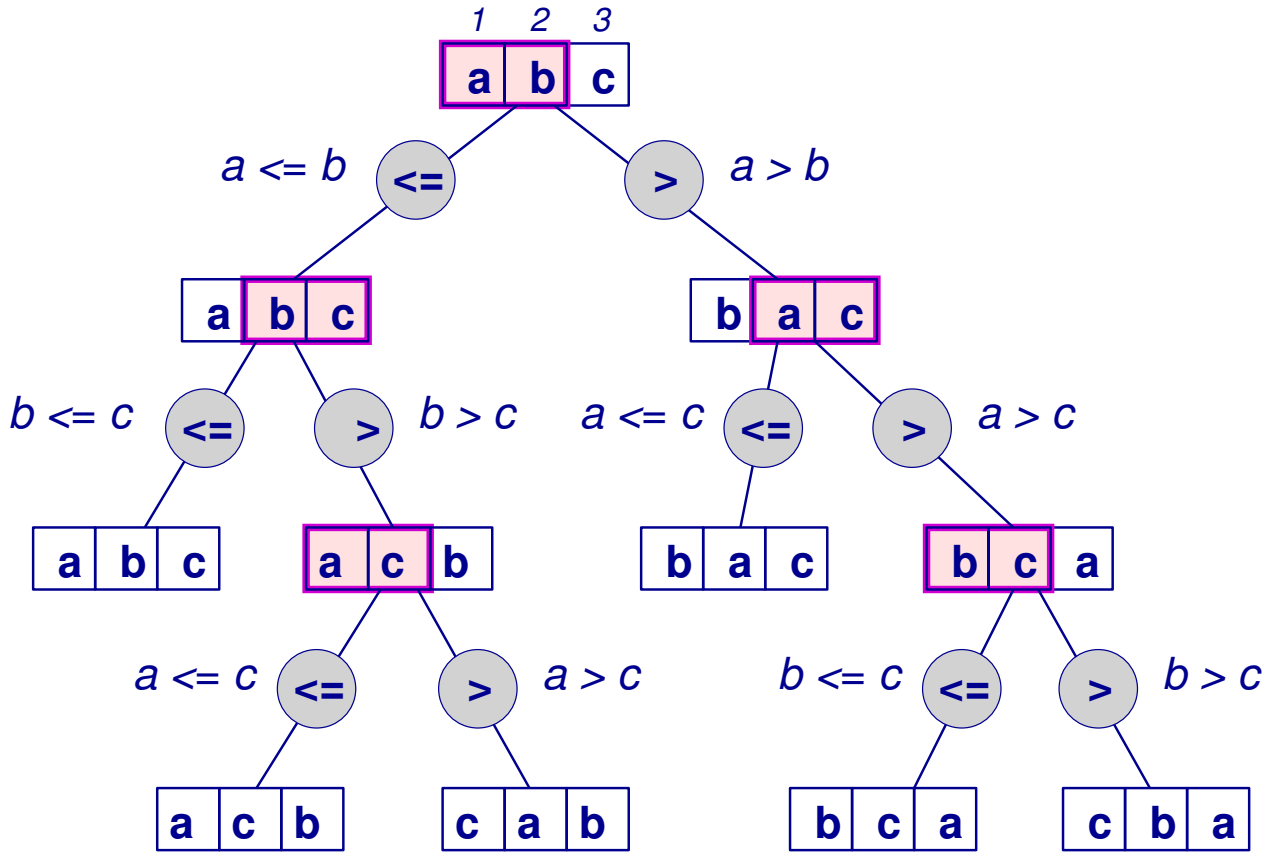
Arbre de décision d'un tri.

- tris par comparaison
- comparaison $\rightarrow$ **décision** de permuter ou non

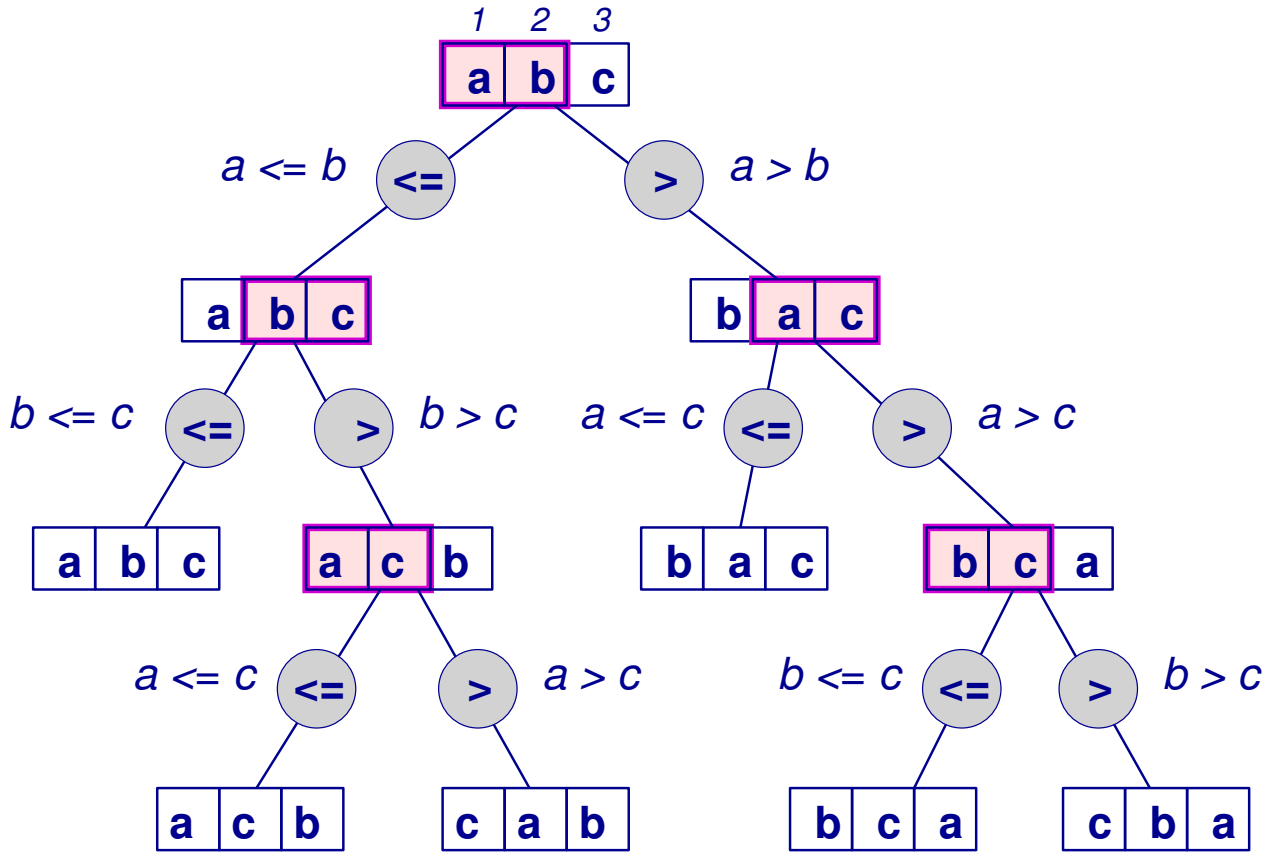
Arbre de décision : représente tous les scénarios possibles pour une suite de n valeurs.

Branche : la suite de permutations faites par le tri pour produire la séquence ordonnée

Arbre de décision du tri par insertion.



Arbre de décision du tri par insertion.



nombre de feuilles = nombre de branches = $n!$

(1) On peut composer $n!$ suites différentes à partir de n valeurs différentes.
(2) On ne peut pas trier deux suites u et v différentes avec la même séquence de permutations ρ.
En effet, $\exists i, j$ tels que $u_i < u_j$ et $v_i > v_j$, il faudrait donc $\rho(i) < \rho(j)$ et $\rho(i) > \rho(j)$.

Complexité des tris par comparaison

Nombre de feuilles : $n!$

hauteur de l'arbre de décision :

$$h(n) \geq \log(n!)$$

or (formule de Stirling)

$$\log(n!) \geq \log\left(\frac{n}{e}\right)^n = n \times \log\ n - n \times \log\ e$$

donc

$$h(n) \geq n \times (\log\ n - 1) \approx n \times \log\ n$$

Tri rapide (quick sort)

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors
2       | Choisir pivot dans T[g, ..., d],
3       | m :=PARTITIONNER(T, g, d, pivot),
4       | TRI_RAPIDE (T, g, m − 1),
5       | TRI_RAPIDE (T, m + 1, d),
```

- tri par comparaisons *en place*,
- tableau indexé,
- le plus rapide dans le cas général,
- très utilisé

Tri rapide (quick sort)

fonction TRI_RAPIDE (T, g, d)

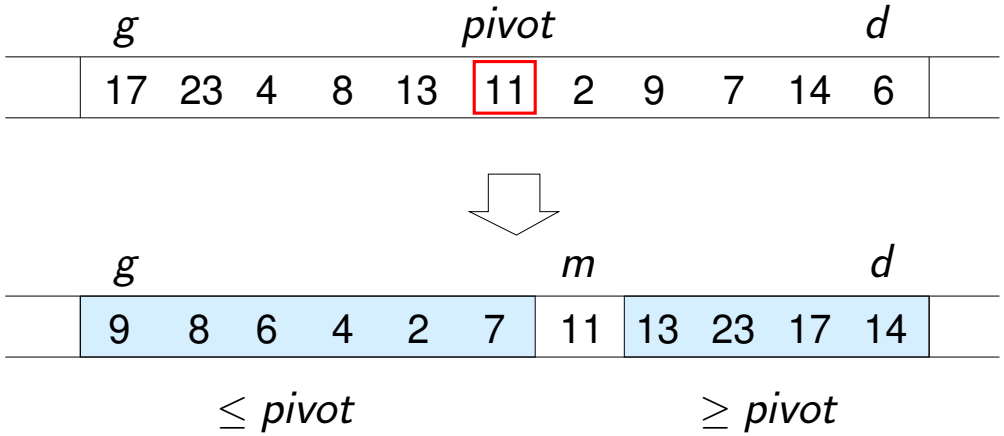
```
1   si (g < d) alors
2       | Choisir pivot dans T[g, ..., d],
3       | m :=PARTITIONNER(T, g, d, pivot),
4       | TRI_RAPIDE (T, g, m − 1),          { m − 1 < d}
5       | TRI_RAPIDE (T, m + 1, d),          { m + 1 > g}
```

Tri rapide (quick sort)

fonction TRI_RAPIDE (T, g, d)

```
1    si (g < d) alors
2        | Choisir pivot dans T[g, ..., d],
3        | m :=PARTITIONNER(T, g, d, pivot),
4        | TRI_RAPIDE (T, g, m − 1),          { m − 1 < d}
5        | TRI_RAPIDE (T, m + 1, d),          { m + 1 > g}
```

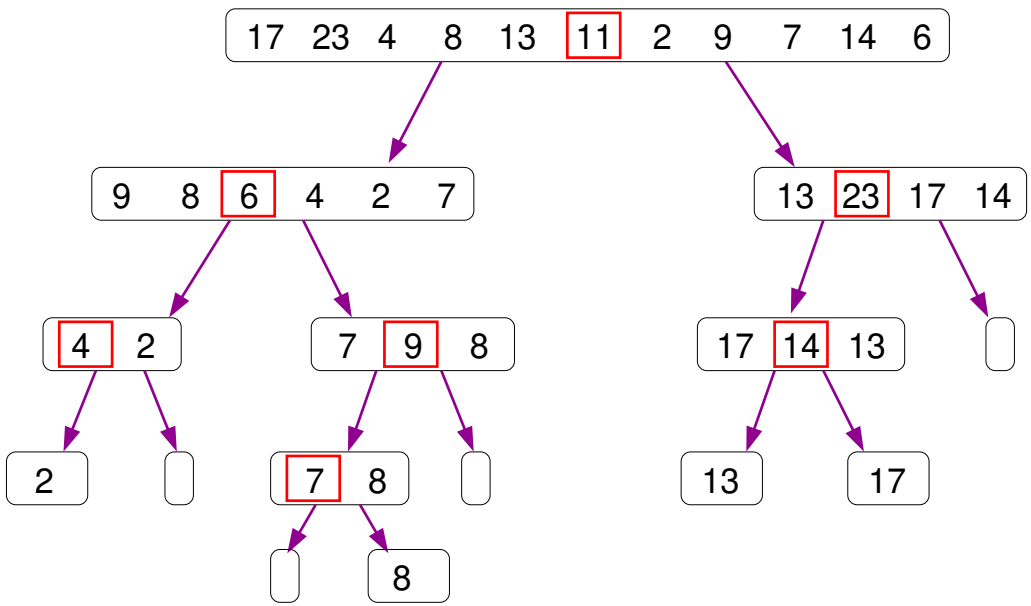
Partition :



Tri rapide (quick sort)

fonction TRI_RAPIDE (T, g, d)

```
1    si (g < d) alors
2        | Choisir pivot dans T[g, ..., d],
3        | m :=PARTITIONNER(T, g, d, pivot),
4        | TRI_RAPIDE (T, g, m − 1),
5        | TRI_RAPIDE (T, m + 1, d),
```



Tri rapide (quick sort)

fonction TRI_RAPIDE (T, g, d)

```
1    si (g < d) alors
2        | Choisir pivot dans T[g, ..., d],
3        | m :=PARTITIONNER(T, g, d, pivot),        O(n)
4        | TRI_RAPIDE (T, g, m − 1),                T(n/2)
5        | TRI_RAPIDE (T, m + 1, d),                T(n/2)
```

Dans le meilleur cas

$$T(n) = n + 1 + 2 \times T(n/2) = \mathcal{O}(n \times \log\ n)$$

Tri rapide (quick sort)

fonction TRI_RAPIDE (T, g, d)

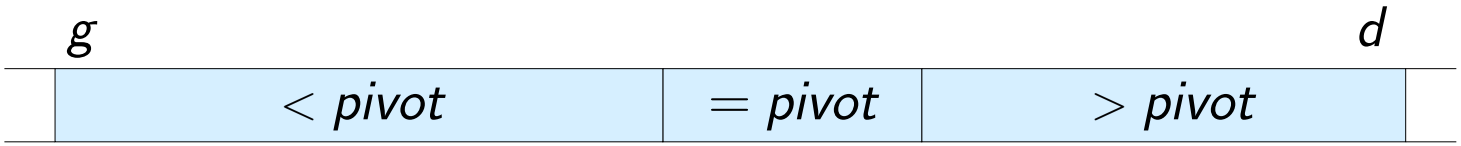
```
1   si (g < d) alors
2   |  Choisir pivot dans T[g, ..., d],
3   |  m :=PARTITIONNER(T, g, d, pivot),        O(n)
4   |  TRI_RAPIDE (T, g, m − 1),                T(0)
5   |  TRI_RAPIDE (T, m + 1, d),                T(n − 1)
```

Dans le pire des cas

$$T(n) = n + 1 + T(n-1) = \mathcal{O}(n^2)$$

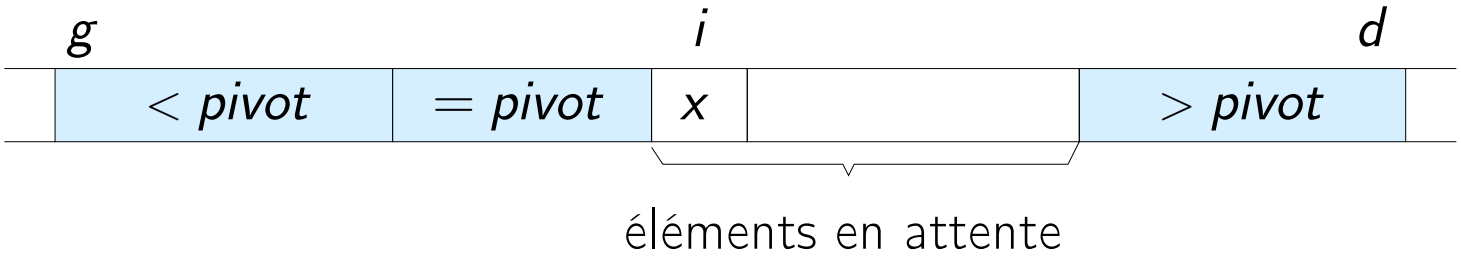
Partition type *drapeau*

Après la partition :



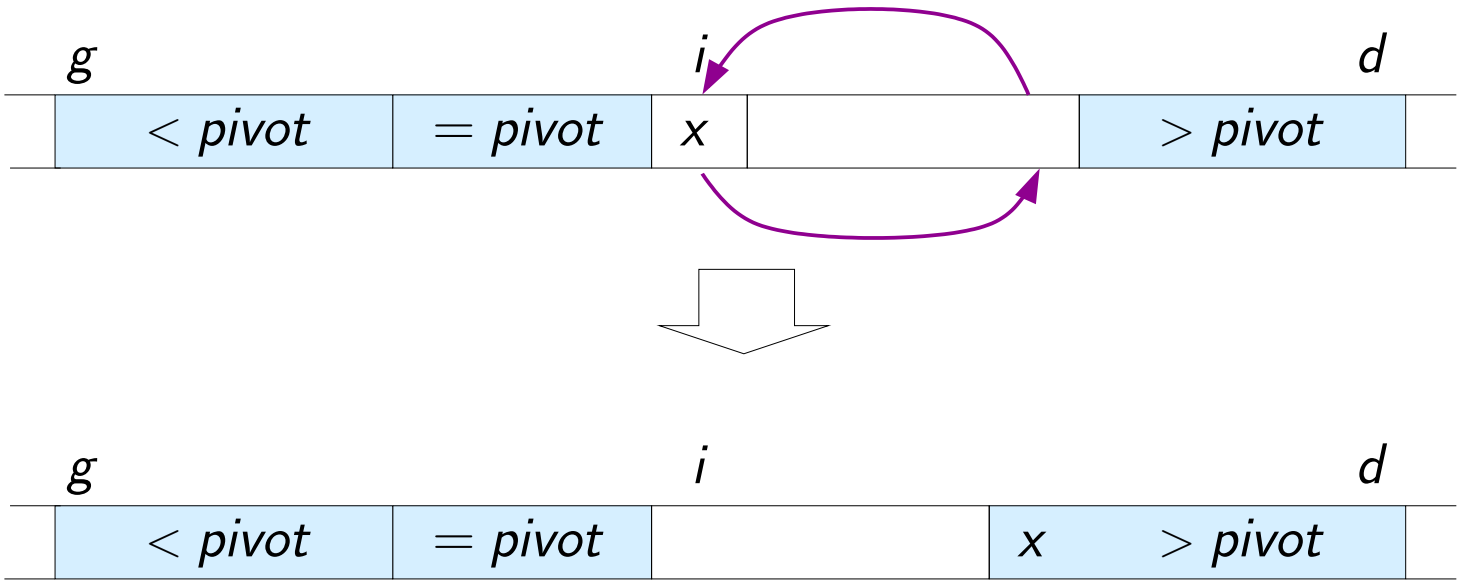
Partition type *drapeau*

Pendant la partition :



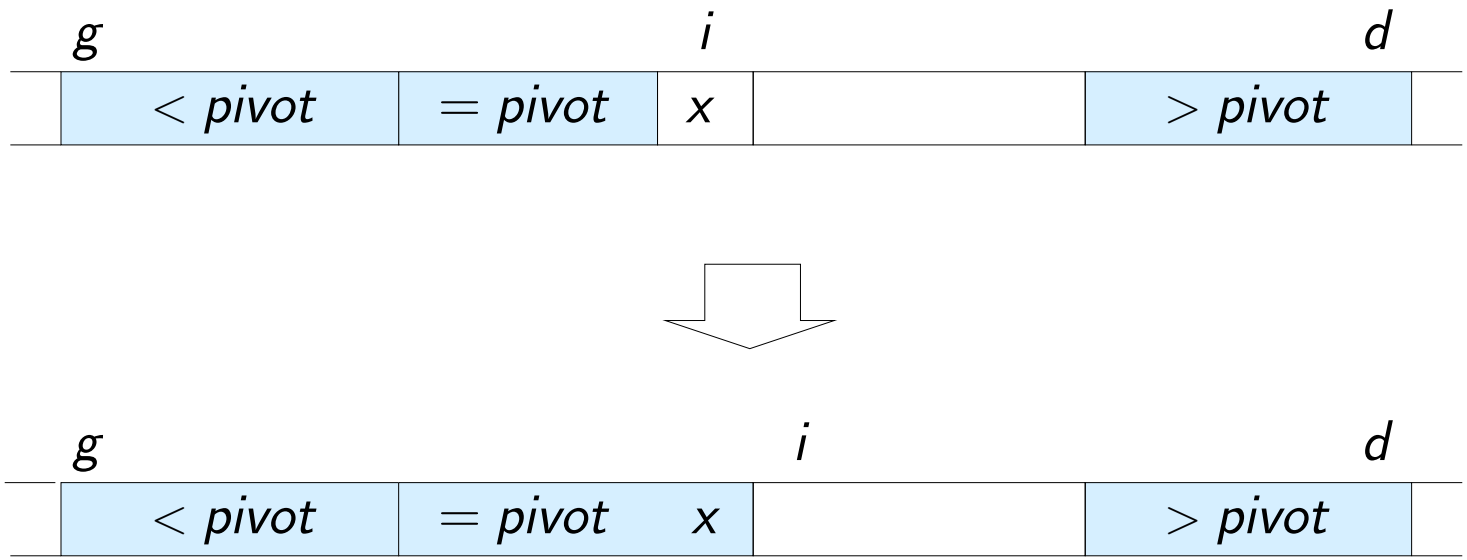
Partition type *drapeau*

cas 1 : $x > pivot$: permutation avec le dernier en attente



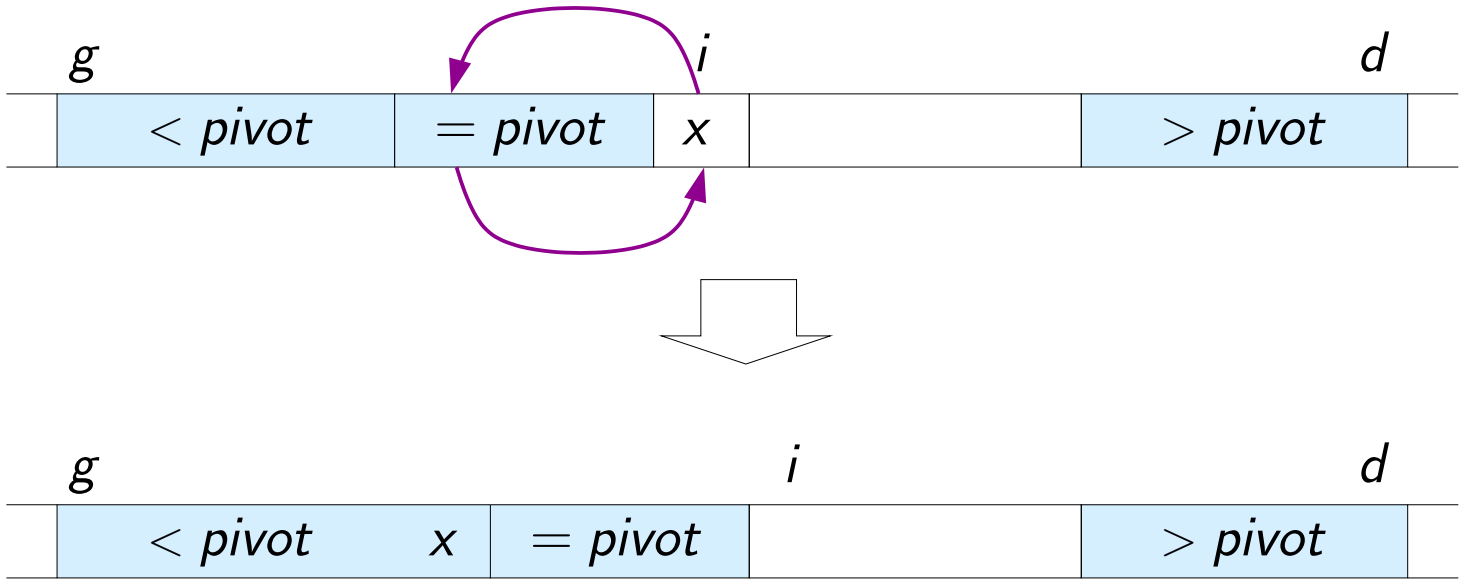
Partition type *drapeau*

cas 2 : $x = pivot$: extension de la zone



Partition type *drapeau*

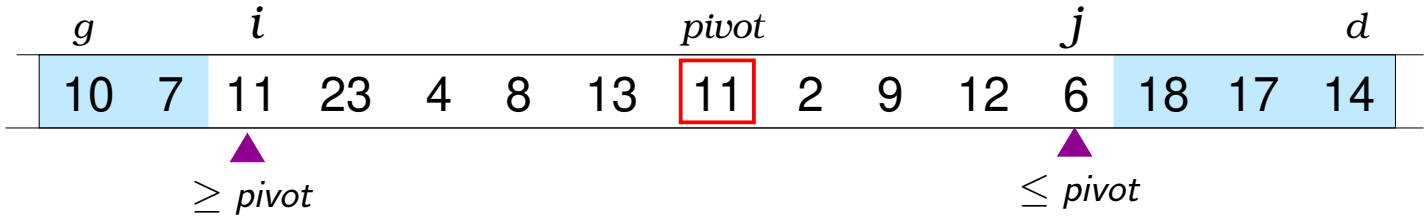
cas 3 : $x < pivot$: permutation avec le premier égal au pivot



Partition *rapide*

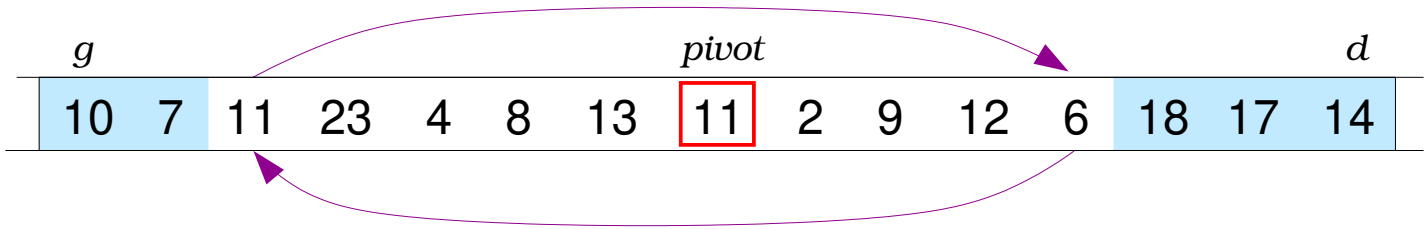
à gauche : stoppe avec $x_i \geq pivot$

à droite : stoppe avec $x_j \leq pivot$



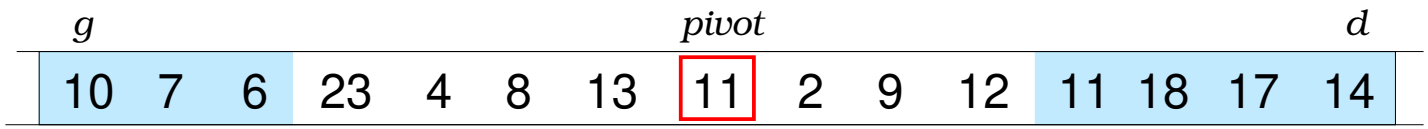
Partition *rapide*

Permutation



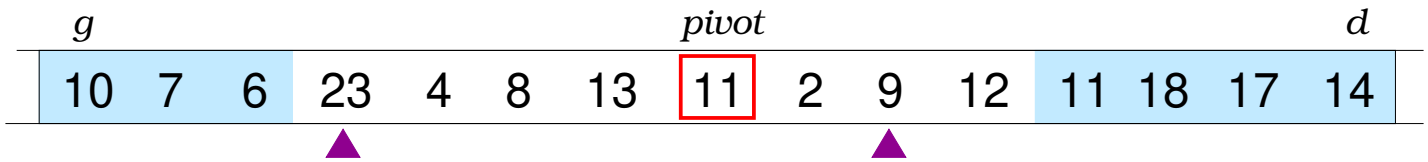
Partition *rapide*

Extension des zones



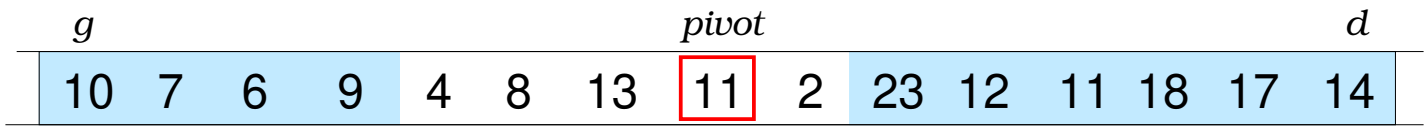
Partition *rapide*

Recherche d'un plus grand à gauche et d'un plus petit à droite



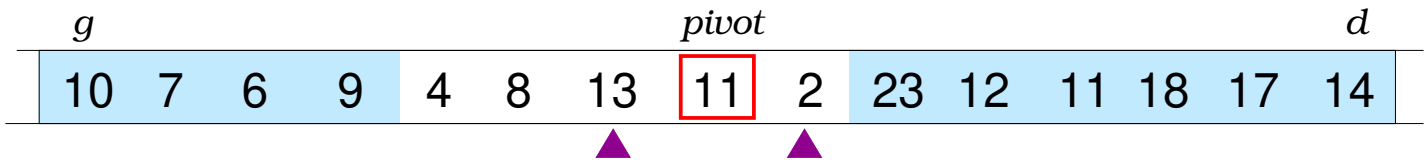
Partition *rapide*

Permutation et extension des zones



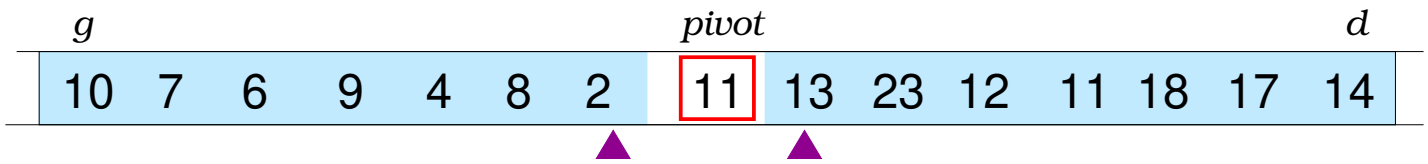
Partition *rapide*

Recherche d'un plus grand à gauche et d'un plus petit à droite



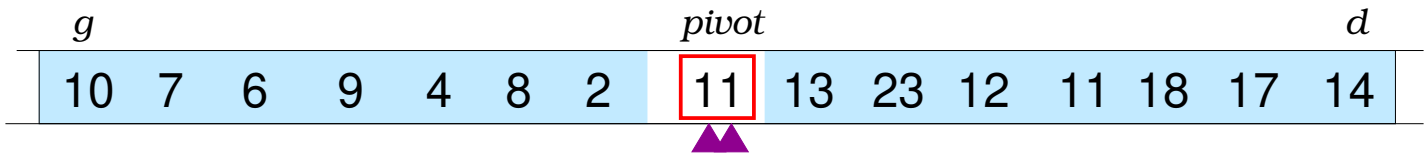
Partition *rapide*

Permutation et extension des zones



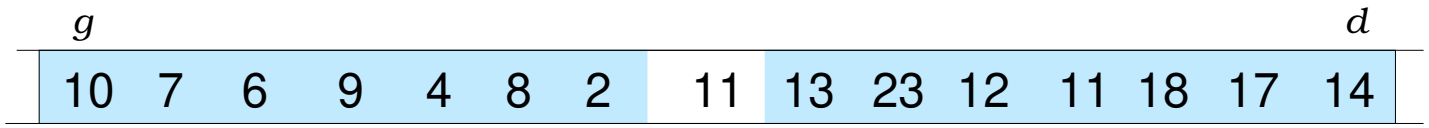
Partition *rapide*

Dernière permutation



Partition *rapide*

Produit deux zones strictement plus petites que $[g, d]$



La partition est en $\mathcal{O}(n)$

Quick sort avec partition rapide

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
5           tant que (T[a] < v) faire
6               a := a + 1,
7           tant que (T[b] > v) faire
8               b := b − 1,
9           si (a ≤ b) alors
10              PERMUTER(T, a, b),
11              a := a + 1,
12              b := b − 1,
13      TRI_RAPIDE (T, g, b),
14      TRI_RAPIDE (T, a, d),
15  fin fonction
```

Quick sort : terminaison

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors                                  | n = d − g + 1 > 1
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
5           tant que (T[a] < v) faire                 | la première fois stoppée par le pivot,
6               a := a + 1,                           | ensuite stoppée par T[b]
7           tant que (T[b] > v) faire
8               b := b − 1,
9           si (a ≤ b) alors
10              PERMUTER(T, a, b),
11              a := a + 1,
12              b := b − 1,
13      TRI_RAPIDE (T, g, b),
14      TRI_RAPIDE (T, a, d),
15  fin fonction
```

Quick sort : terminaison

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors                                  n = d − g + 1 > 1
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
5           tant que (T[a] < v) faire
6               a := a + 1,
7           tant que (T[b] > v) faire                 la première fois stoppée par le pivot,
8               b := b − 1,                           ensuite stoppée par T[a]
9           si (a ≤ b) alors
10              PERMUTER(T, a, b),
11              a := a + 1,
12              b := b − 1,
13      TRI_RAPIDE (T, g, b),
14      TRI_RAPIDE (T, a, d),
15  fin fonction
```

Quick sort : terminaison

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors                                   n = d − g + 1 > 1
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
5           tant que (T[a] < v) faire
6               a := a + 1,
7           tant que (T[b] > v) faire
8               b := b − 1,
9           si (a ≤ b) alors                           forcément vrai la première fois
10              PERMUTER(T, a, b),
11              a := a + 1,
12              b := b − 1,
13      TRI_RAPIDE (T, g, b),
14      TRI_RAPIDE (T, a, d),
15  fin fonction
```

Quick sort : terminaison

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors                                  n = d − g + 1 > 1
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
5           tant que (T[a] < v) faire
6               a := a + 1,
7           tant que (T[b] > v) faire
8               b := b − 1,
9           si (a ≤ b) alors                          forcément vrai la première fois
10              PERMUTER(T, a, b),
11              a := a + 1,                           on incrémente a au moins une fois
12              b := b − 1,                           on décrémente b au moins une fois
13      TRI_RAPIDE (T, g, b),
14      TRI_RAPIDE (T, a, d),
15  fin fonction
```

Quick sort : terminaison

fonction TRI_RAPIDE (T, g, d)

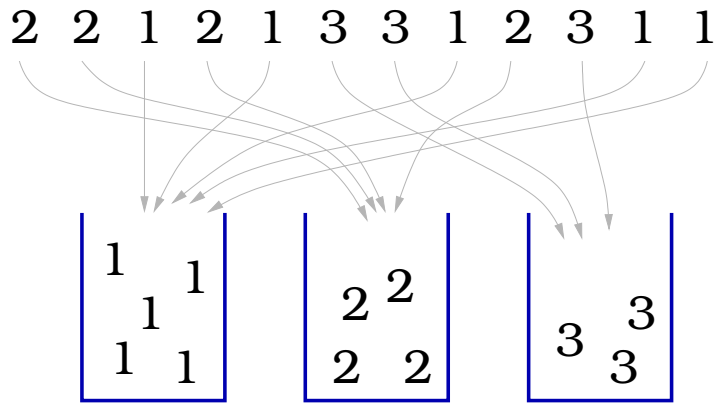
```
1   si (g < d) alors                                  | n = d − g + 1 > 1
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
5           tant que (T[a] < v) faire
6               a := a + 1,
7           tant que (T[b] > v) faire
8               b := b − 1,
9           si (a ≤ b) alors                          | forcément vrai la première fois
10              PERMUTER(T, a, b),
11              a := a + 1,                           | on incrémente a au moins une fois
12              b := b − 1,                           | on décrémente b au moins une fois
13      TRI_RAPIDE (T, g, b),                         | ici b < d donc b − g + 1 < n
14      TRI_RAPIDE (T, a, d),                         | ici a > g donc d − a + 1 < n
15  fin fonction
```

Quick sort : justesse

fonction TRI_RAPIDE (T, g, d)

```
1   si (g < d) alors
2       a := g, b := d,
3       v := T[(a + b)/2],
4       tant que (a ≤ b) faire
            {∀i, si g ≤ i < a alors T[i] ≤ v, et si b < i ≤ d alors T[i] ≥ v,}
5           tant que (T[a] < v) faire
6               a := a + 1,                          {T[a − 1] < v}
7           tant que (T[b] > v) faire
8               b := b − 1,                          {T[b + 1] > v}
9           si (a ≤ b) alors
10              PERMUTER(T, a, b),
                {T[a] ≤ v et T[b] ≥ v}
11              a := a + 1,
12              b := b − 1,
            {donc si i < a alors T[i] ≤ v et si i > b alors T[i] ≥ v,}
13      TRI_RAPIDE (T, g, b),                        {b < a et ∀i, i < a on a T[i] ≤ v}
14      TRI_RAPIDE (T, a, d),                        {a > b et ∀i, i > b on a T[i] ≥ v}
15  fin fonction
```

Tri par dénombrement.



Tri sans comparaison : suppose que l'on sait *indexer* les éléments à trier, i.e. affecter à chacun un rang

- qui dépends uniquement de sa valeur
- qui correspond à l'ordre défini sur les éléments

tri du facteur qui prépare sa tournée,
tri de valeurs entières assez proches et nombreuses.

Tri par dénombrement.

Calcul des nombres d'apparitions

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

Nombres d'apparitions

	0	1	2
nb	3	4	3

Tri par dénombrement.

Calcul des positions des premiers de chaque classe

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

Nombres d'apparitions

	0	1	2
nb	3	4	3

Indices des premiers

	0	1	2
pos	0	3	7

Tri par dénombrement.

Positionnement dans le tableau résultat

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

Nombres d'apparitions

	0	1	2
nb	3	4	3

Indices des premiers

	0	1	2
pos	0	3	7

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

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

Tri par dénombrement.

Positionnement dans le tableau résultat

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

Nombres d'apparitions

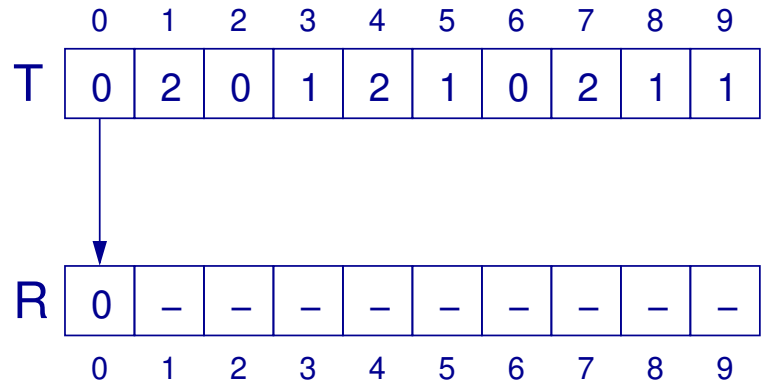
	0	1	2
nb	3	4	3

Indices des premiers

	0	1	2
pos	0	3	7

placement de T[0] en R[0]

pos	1	3	7



Tri par dénombrement.

Positionnement dans le tableau résultat

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

Nombres d'apparitions

	0	1	2
nb	3	4	3

Indices des premiers

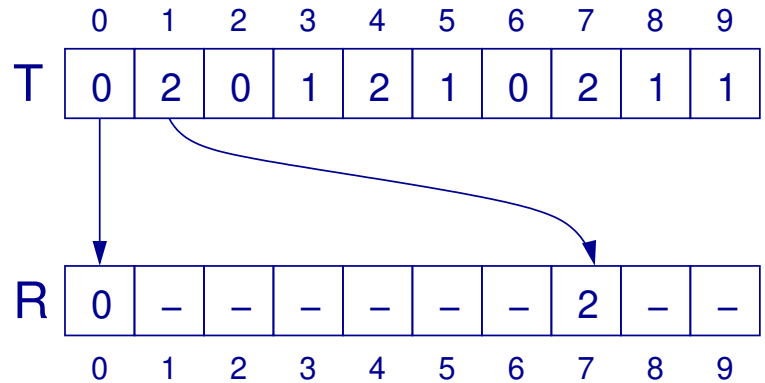
	0	1	2
pos	0	3	7

placement de T[0] en R[0]

pos	1	3	7

placement de T[1] en R[7]

pos	1	3	8



Tri par dénombrement.

Positionnement dans le tableau résultat

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

Nombres d'apparitions

	0	1	2
nb	3	4	3

Indices des premiers

	0	1	2
pos	0	3	7

placement de T[0] en R[0]

pos	1	3	7

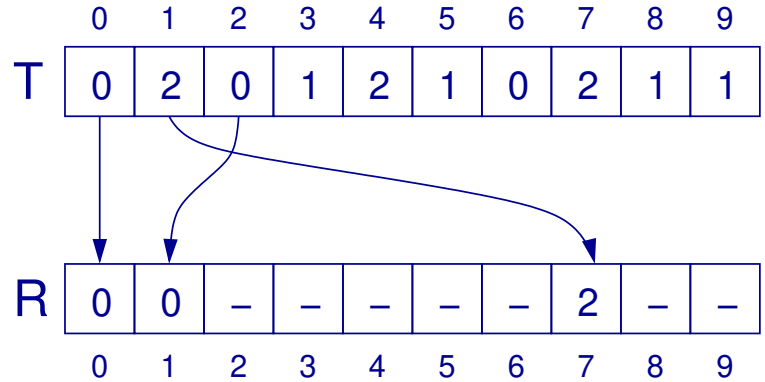
placement de T[1] en R[7]

pos	1	3	8

placement de T[2] en R[1]

pos	2	3	8

⋮



Tri par dénombrement.

```
fonction TRI_PAR_DENOMBREMENTS(T, n)
{In : T un tableau de n éléments}
{Out : R le tableau trié des éléments de T}
début
    pour i := 0 à k − 1 faire                    initialisations
        nb[i] := 0,
    pour i := 1 à n faire                        calcul des nombres d'apparitions
        nb[T[i]] := nb[T[i]] + 1,
    pos[0] := 0,                                 calcul des indices du premier
    pour i := 1 à k − 1 faire                    élément de chaque catégorie
        pos[i] := pos[i − 1] + nb[i − 1],
    pour i := 1 à n faire                        recopie des élément originaux
        R[pos[T[i]]] := T[i],                    du tableau T dans R
        pos[T[i]] := pos[T[i]] + 1,
    renvoyer R
fin procédure
```

Complexité : $\mathcal{O}(n+k)$.

Tri par base.

Utilise le tri par dénombrement en plusieurs passes

536
893
427
167
853
592
197
462

Tri par base.

Utilise le tri par dénombrement en plusieurs passes

536	592
893	462
427	893
167	853
853	536
592	427
197	167
462	197

Tri par base.

Utilise le tri par dénombrement en plusieurs passes

536	592	427
893	462	536
427	893	853
167	853	462
853	536	167
592	427	592
197	167	893
462	197	197

Tri par base.

Utilise le tri par dénombrement en plusieurs passes

536	592	427	167
893	462	536	197
427	893	853	427
167	853	462	462
853	536	167	536
592	427	592	592
197	167	893	853
462	197	197	893

n nombres à c chiffres en base k : $\mathcal{O}(c \times n + c \times k)$.

Si $k = \mathcal{O}(n)$ le tri par base est linéaire ($\mathcal{O}(n)$).