Tri lexicographique

Optimalité des tris par comparaisons : O(*n* log*n*)

Classements linéaires

Tris lexicographiques

Tri de 4 éléments

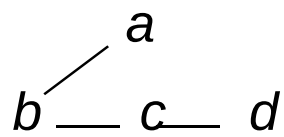
Suite (*a, b, c, d*) à classer

Méthode optimale : comparaisons

1. classer *a* et *b* 1
2. classer *c* et *d* 1
3. comparer les 2 plus petits 1
4. insérer le dernier 1 ou 2
   au sein des trois éléments classés

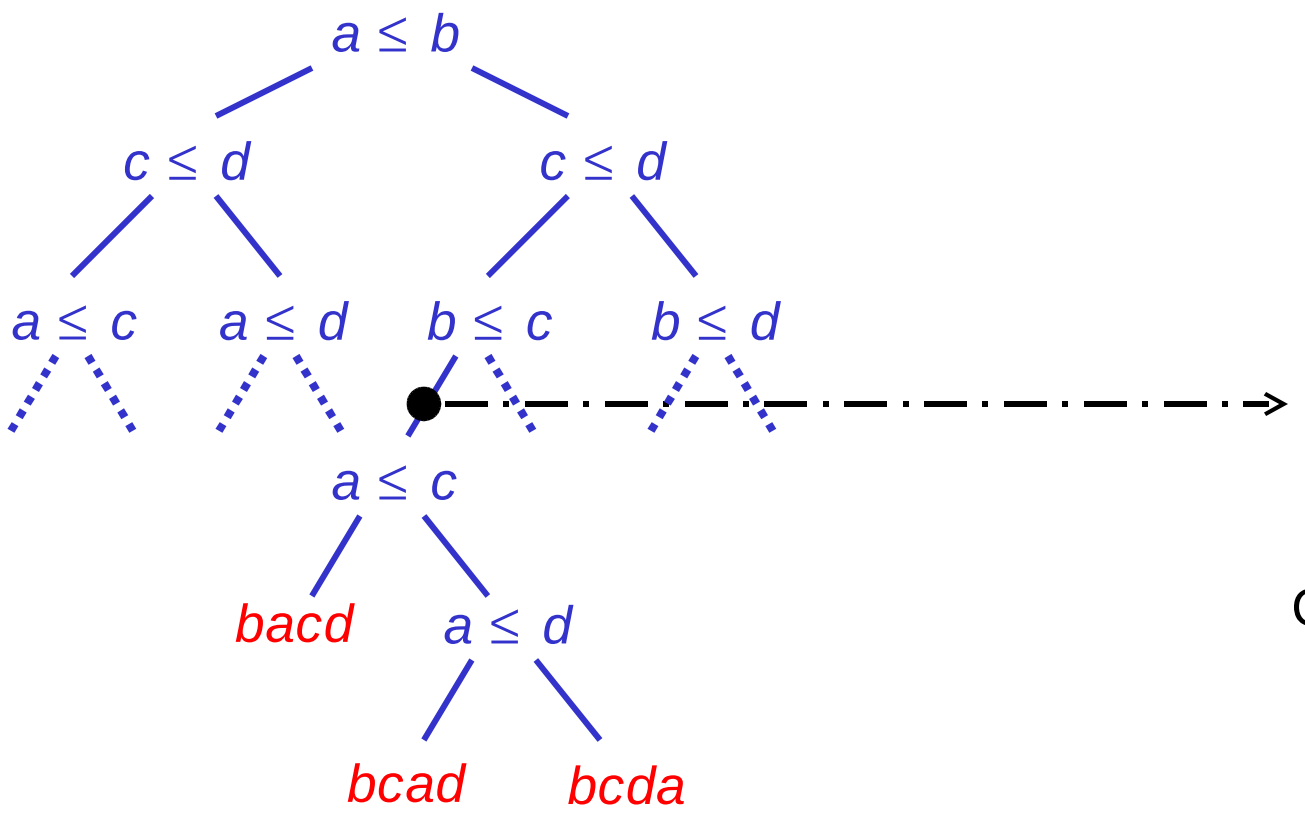
Exemple : classer (9, 1, 5, 17)

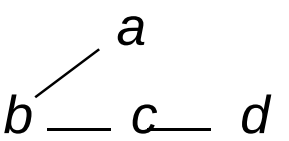
1. *b*=1 < *a*=9
2. *c*=5 < *d*=17
3. *b*=1 < *c*=5 on a *b*=1 < *c*=5 < *d*=17



4. Insertion de *a*=9 après comparaisons avec *c* puis *d*

Arbre de décision





Ordres possibles :

bacd

bcad

bcda

plus longue branche (hauteur) : 5

plus courte branche : 4

4! = 24 feuilles

Tris par comparaisons

algorithme de tri par comparaisons sur $a_1, \ldots, a_n$

début

- - - - $a_k \leq a_i$

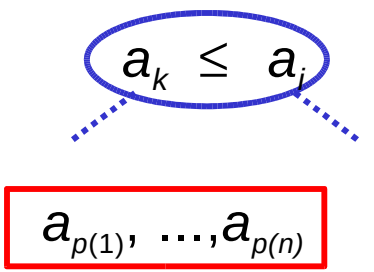
si ($a_k \leq a_i$) **alors**

- - - -

fin

arbre de décision

feuilles : permutations de $a_1, \ldots, a_n$



Théorème : a tri par comparaisons

Alors $T_{MAX}(a,n) = \Omega(n \log n)$

Preuve :

$T_{MAX}(a,n)$ = hauteur de l ' arbre de décision à $n!$ feuilles

hauteur $\geq \log_2 n! = \Omega(n \log n)$ [car $n! \geq (n/2)^{n/2}$]

Analyse en moyenne

Théorème a tri par comparaisons

permutations des entrées équiprobables

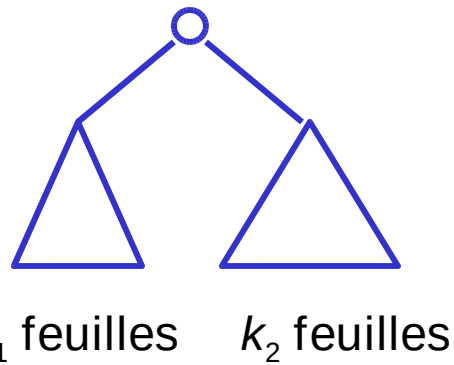
Alors $T_{MOY}(a,n) = \Omega(n \log n)$

Preuve

pour un arbre binaire à k feuilles (nœuds externes)

soit $l(k)$ la longueur moyenne des branches

$l(k) \geq \log_2 k$



$k = k_1 + k_2 \qquad k_1 < k, \;\; k_2 < k$

$$l(k) = 1 + \frac{k_1}{k} l(k_1) + \frac{k_2}{k} l(k_2)$$

$$\geq 1 + \frac{k_1}{k} \log k_1 + \frac{k_2}{k} \log k_2 \text{ minimum pour } k_1 = k_2$$

$$\geq 1 + \frac{1}{2} \log \frac{k}{2} + \frac{1}{2} \log \frac{k}{2} \geq \log_2 k$$

Tris par comparaisons optimaux

Optimaux au pire

Tri par le tas
Tri par fusion
Tri par arbre AVL

} $O(n \log n)$

Optimaux en moyenne

précédents, plus
Tri rapide
Tri par arbre binaire
de recherche

} $O(n \log n)$

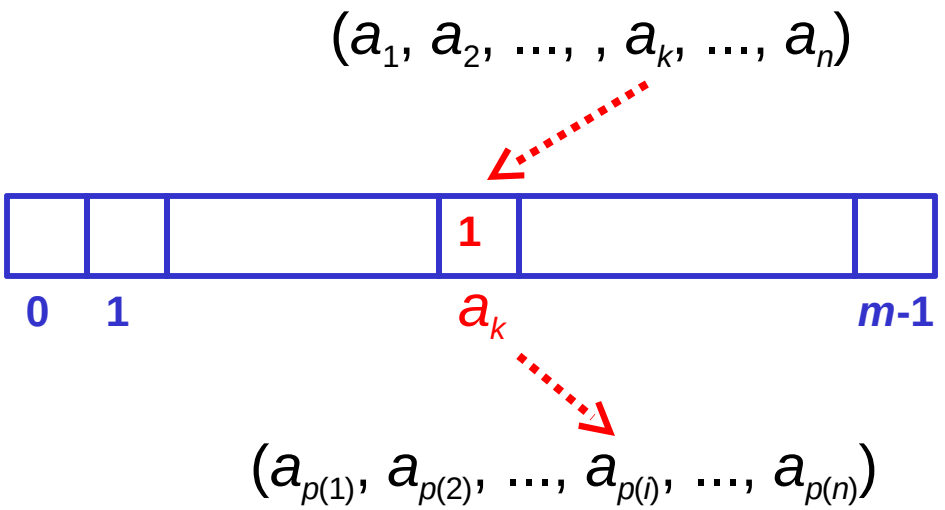
Tri par distribution

Classer $L = (a_1, a_2, ..., a_n)$ constituée de n entiers $\in (0, 1, ..., m\text{-}1)$

distribution

vecteur booléen ou entier

collecte



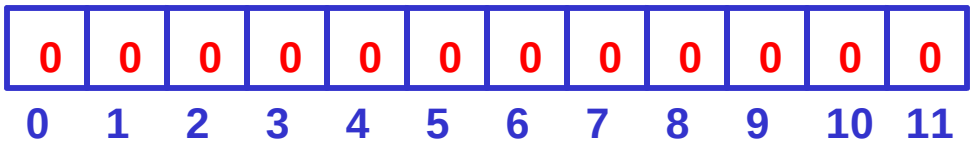
Temps : O(n+m) (pas de comparaisons)

Espace supplémentaire : O(m)

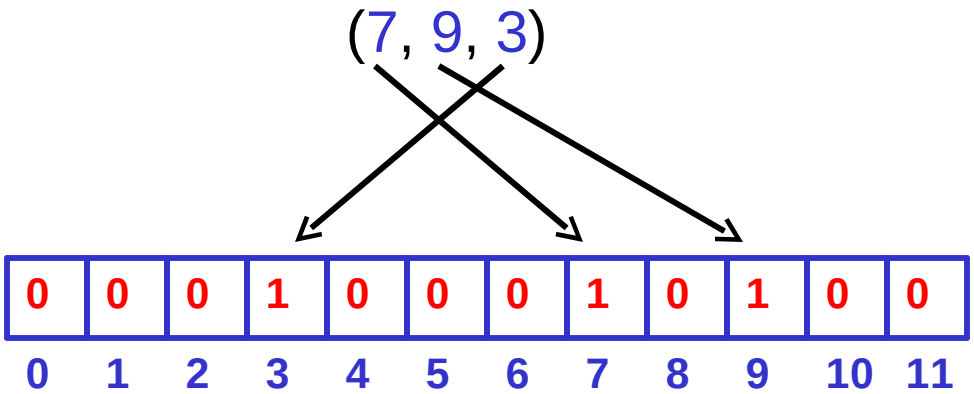
Exemple

L = (7, 9, 3) constituée d' entiers $\in$ (0, 1, ..., 11)

initialisation



distribution



collecte

(3, 7, 9)

Tri « sur place »

Suite (a_1, ..., a_n) dans une table ; clés = entiers 1,2, ..., n

Classement (permutation) :

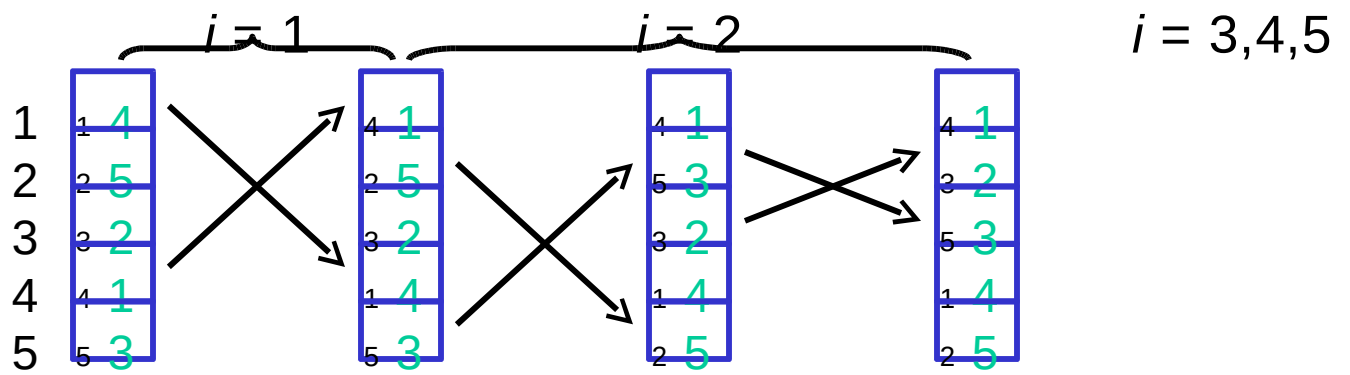
pour $i \leftarrow$ 1 **à** n **faire**

tant que clé(a_i) $\neq i$ **faire**

échanger (a_i, $a_{\text{clé}(a_i)}$)

Temps : O(n)

Espace supplémentaire constant.



Tri par bacs
(sélection de place)

Classer $L = (a_1, a_2, ..., a_n)$ clés $\in (0, 1, ..., m\text{-}1)$

Idée : utiliser m bacs et ranger a_i dans bac[clé(a_i)]

```
fonction tri-par-bacs (L liste) : liste ;
début
1       pour i ← 0 à m-1 faire vider bac[ i ] ;
2       pour a ← premier au dernier élément de L faire
                ajouter a à la fin de bac[clé(a)] ;
        L' ← liste-vide ;
3       pour i ← 0 à m-1 faire
                L' ← L' . bac[ i ] ;
    retour ( L' ) ;
fin
```

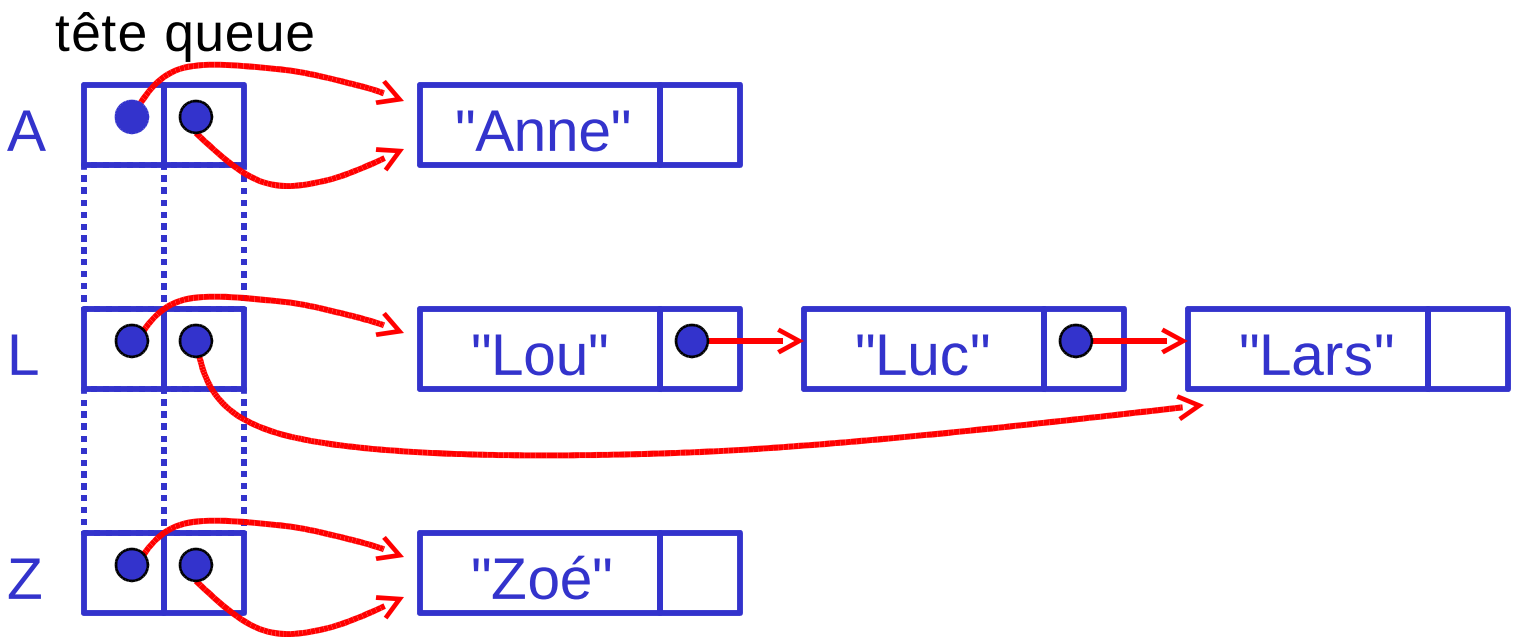
Temps : O(n+m) si bonne gestion des bacs [O(n) pour 2, O(m) pour 1 et 3]

Espace supplémentaire : O(m)

Distribution

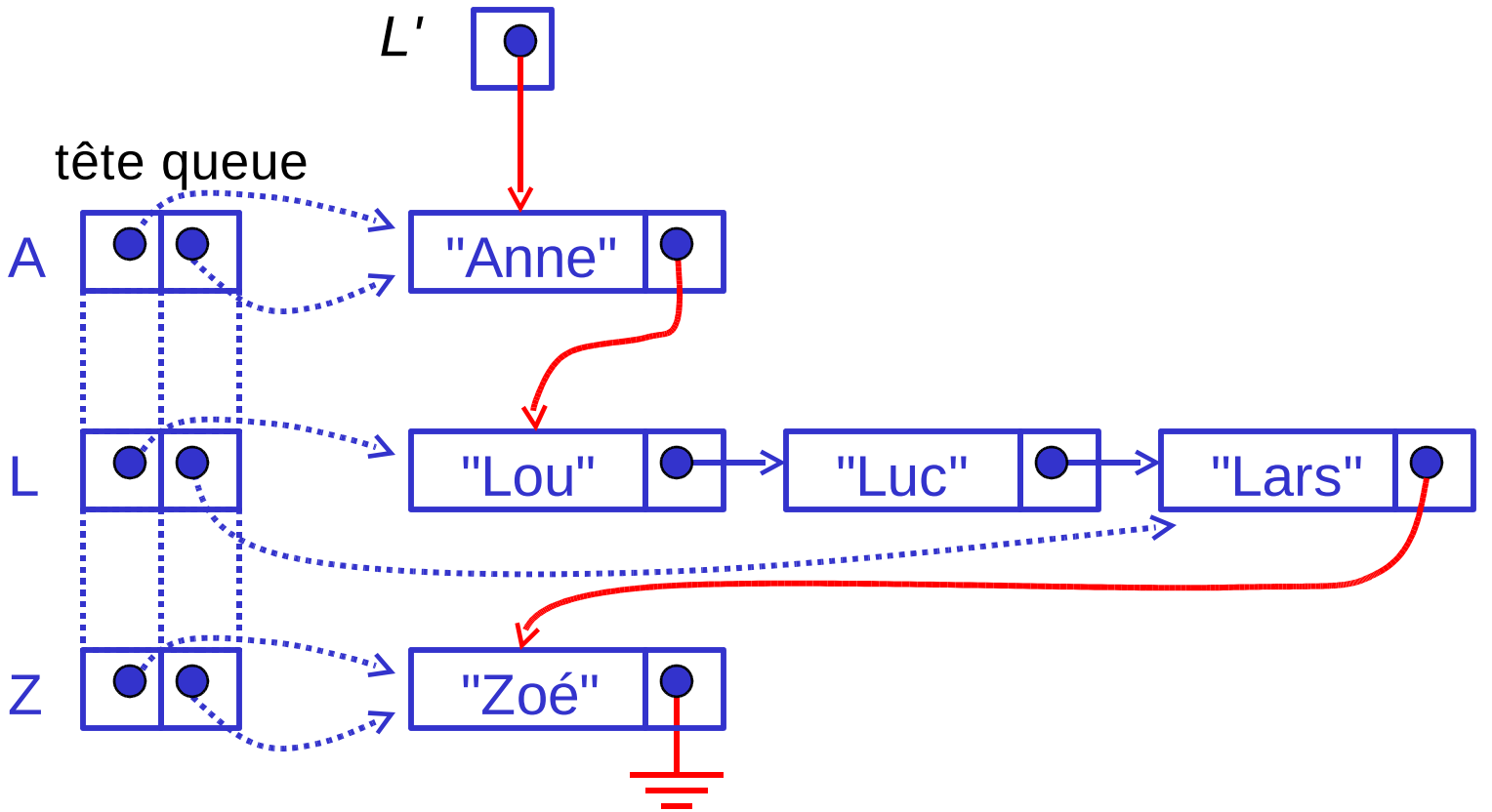
Classer ("Zoé", "Lou", "Anne", "Luc", "Lars")
clé = première lettre

Bacs (listes chaînées) : temps constant par élément



Collecte

Concaténation des bacs : temps constant par bac
(tri stable)



Exemple

Classer $L = (a_1, .., a_n)$ clés $\in (0, 1, 2, ..., n^2 - 1)$

Tri en O(*n*) :

1. tri par bacs avec clé : clé(a_i) mod n
2. tri par bacs avec clé : $\lfloor$clé(a_i) / $n\rfloor$

Équivalent à un tri lexicographique
avec la clé (q, r) telle que clé(a_i) = $q.n + r$

n=10 L = (26, 9, 0, 25, 1, 29, 64, 16, 81, 4)
après 1
(0, 1, 81 ,64 ,4 ,25, 26, 16, 9, 29)
après 2
(0, 1, 4, 9, 16, 25, 26, 29, 64, 81)

Ordre lexicographique

Suites = mots de A^*

Fortement inférieur, <<

$(a_1, .., a_k) << (b_1, .., b_l)$ ssi

il existe i, $1 \leq i \leq \min\{k,l\}$, $a_1=b_1, .., a_{i-1}=b_{i-1}, a_i<b_i$

information → **informatique**

Ordre $\leq$

$x, y \in A^*$ $\quad x \leq y$ ssi x préfixe de y ou $x << y$

Classement

oubli	**bague**
orgue	**baguette**
baguette	**orgue**
oublier	**oubli**
bague	**oublier**

Tri lexicographique

L liste de mots de même longueur *k*

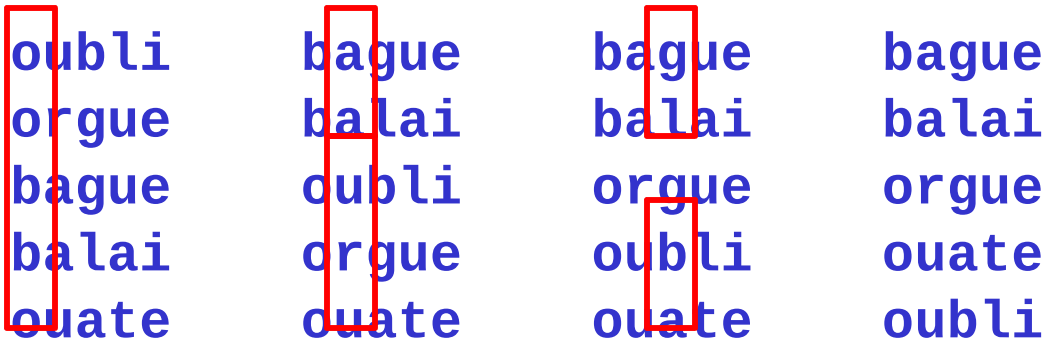
fonction tri-lexico (*L* liste) : liste ;
début
    pour *i* ← *k* à 1 **pas** -1 **faire**
        L ← tri-par-bacs(*L*) avec clé = *i*-ème composante ;
    retour (*L*) ;
fin

Preuve : car "tri-par-bacs" stable
Temps : O(*k* (*n* + *m*)) si composantes ∈ (0, 1, ..., *m*-1)

	↓	↓	↓	↓	↓
oubli	orgue	balai	ouate	bague	bague
orgue	bague	oubli	oubli	balai	balai
bague	ouate	ouate	orgue	orgue	orgue
balai	oubli	orgue	bague	ouate	ouate
ouate	balai	bague	balai	oubli	oubli

Tri lexicographique (suite)

Méthode ordinaire

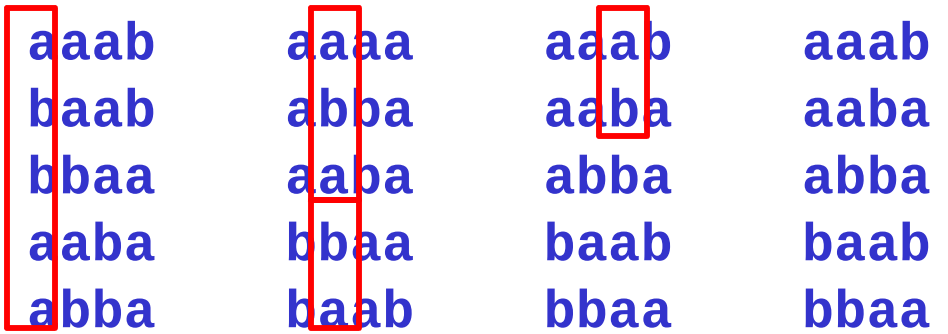


Temps maximal : O(k (n + m)) si composantes $\in$ (0, 1, ..., m-1)
Temps moyen : O((n + m) log n)
si composantes équiprobables et indépendantes
Alphabet fixe : « m disparaît »

Théorème
On peut classer une suite de chaînes de caractères en temps O(l) où l est la somme de leurs longueurs.

Tri lexico par échanges

Alphabet binaire *A* = {a, b} (ou {0,1})



Tri analogue au tri rapide (*i* indice de composante)

fonction tri-lexico-par-échanges (*L* liste) : liste ;
début
    g ← 1; *d* ← *n* ; *i* ← 1 ;
    TLE (*L*, *g*, *d*, *i*) ;
    retour (*L*) ;
fin

TLE

L[*p,i*] = *i*-ème lettre du mot d' indice *p* de *L*

```
procédure TLE ( L, g, d, i ) ;
début
        si g < d et i ≤ k alors {
                p ← g ;   q ← d ;
                répéter
                        tant que ( L[ p,i ] ) = a et p < q faire p ← p+1 ;
                        tant que ( L[ q,i ] ) = b et p < q faire q ← q-1 ;
                        échanger (L[ p,i ], L[ q,i ] ) ;
                jusqu'à ce que p = q ;
                si ( L[ q,i ] ) = a alors q ← q+1 ;
                TLE ( L, g, q-1, i+1 ) ;
                TLE ( L, q, d, i+1 ) ;
        }
fin
```