Méthodologie de la programmation *–E2I.1-*

Les algorithmes de tri

Cyrille CHAVET

Vue d'ensemble

- Introduction aux algorithmes de tri
- Les tris séquentiels
- Les tris récursifs

Plan

Objectifs

- Tri:
  - Ordonner un ***ensemble d'éléments*** selon un ensemble de ***clés*** sur lesquelles est définie une ***relation d'ordre***

- Algorithmes fondamentaux en informatique:
  - Base de données, ordonnancement...
- Illustration de concepts vus précédemment:
  - Boucles, Récursivité
- Introduction de concepts clefs en informatique:
  - Complexité
  - Stratégie «Diviser pour régner»

Procédure d'échange

- Les algorithmes de tri utilisent une procédure permettant d'échanger (de permuter) deux variables.

```
Procédure Echanger (a,b : Entier)
    Déclaration Temp : Entier
Début
    Temp ←a
    a ← b
    b ← Temp
Fin
```


Concept d'invariant de boucle

- Prouver la validité d'un algorithme
- Définition:
  - Un invariant de boucle est une propriété qui reste vraie à chaque passage dans la boucle (en un point précis)
- Il faut donc:
  - Définir un invariant qui assure qu'à la fin la propriété voulue est satisfaite
  - S'assurer qu'il est vrai avant d'entrer dans la boucle
  - Vérifier par récurrence qu'il reste vrai après chaque exécution de la boucle

Exemple d'invariant de boucle

- Produit des chiffres d'un nombre

```
Fonction ProduitsChiffres(n : Entier) : Entier
    Déclaration x : Entier
Début
    x ← 1
    Répéter
        x ← x*(n mod 10)
        n ← n / 10
    Jusqu'à (n < 1)
Fin
```

Exemple:

*ProduitChiffres(1234) = 1*2*3*4*
= 24

Exemple d'invariant de boucle

- Produit des chiffres d'un nombre

```
Fonction ProduitsChiffres(n : Entier) : Entier
    Déclaration x : Entier
Début
    x ← 1
    Répéter
        x ← x*(n mod 10)
        n ← n / 10
    Jusqu'à (n < 1)
Fin
```

Précondition: *n>0, x=1*

Exemple d'invariant de boucle

- Produit des chiffres d'un nombre

```
Fonction ProduitsChiffres(n : Entier) : Entier
    Déclaration x : Entier
Début
    x ← 1
    Répéter
        x ← x*(n mod 10)
        n ← n / 10
    Jusqu'à (n < 1)
Fin
```

Précondition: *n>0, x=1*

Invariant:
- $n_i = n/10^i$
- x_i = *produit des chiffres de* $(n-10^i*n_i)$

Exemple d'invariant de boucle

- Produit des chiffres d'un nombre

```
Fonction ProduitsChiffres(n : Entier) : Entier
    Déclaration x : Entier
Début
    x ← 1
    Répéter
        x ← x*(n mod 10)
        n ← n / 10
    Jusqu'à (n < 1)
Fin
```

Précondition: *n>0, x=1*

Invariant:
- $n_i = n/10^i$
- x_i = *produit des chiffres de* $(n-10^i*n_i)$

Terminaison:
$i > log_{10}(n)$, $n_i = 0$

Correction:
x_i = *produit des chiffres de (n)*

Exemple d'invariant de boucle

- Produit des chiffres d'un nombre

```
Fonction ProduitsChiffres(n : Entier) : Entier
    Déclaration x : Entier
Début
    x ← 1
    Répéter
        x ← x*(n mod 10)
        n ← n / 10
    Jusqu'à (n < 1)
Fin
```

Précondition: *n>0, x=1*

Invariant:
- $n_i = n/10^i$
- x_i = *produit des chiffres de* $(n-10^i*n_i)$

Terminaison:
$i > log_{10}(n)$, $n_i = 0$

Correction:
x_i = *produit des chiffres de (n)*

Evaluer la complexité d'un algorithme

- Idée:
  - Compter le nombre de permutations, le nombre de tours dans une boucle…

```
Fonction Exemple(n : Entier) : Entier
    Déclaration i, j : Entier
Début
    …
    pour i de 0 à n-1
        pour j de 0 à n-1
            …
        fin pour
    fin pour
    …
Fin
```

Evaluer la complexité d'un algorithme

- Idée:
  - Compter le nombre de permutations, le nombre de tours dans les boucles…

```
Fonction Exemple(n : Entier) : Entier
    Déclaration i, j : Entier
Début
    …
    pour i de 0 à n-1
        pour j de 0 à n-1
            …
        fin pour
    fin pour
    …
Fin
```

n tours *n tours*

⇒ *Soient n*n = n^2 tours,*

La complexité de cet algorithme peut s'exprimer en $O(n^2)$

Evaluer la complexité d'un algorithme

- Idée:
  - Compter le nombre de permutations, le nombre de tours dans une boucle…

- Définition:
  - Déterminer une fonction $f: \mathbb{N} \to \mathbb{R}^+$ qui,
    à un paramètre ***n*** dépendant de la donnée soumise à l'algorithme (*n* est la taille de cette donnée, e.g. la taille du tableau a trier),
    associe le coût ***f(n)*** (exprimé en unités arbitraires de temps ou d'espace) de l'exécution de algorithme pour la donnée

Plan

Algorithme de Tri à bulles

- Idée de M. De Lapalisse :
  - Un tableau trié en ordre croissant, c'est un tableau dans lequel tout élément est **plus petit que celui qui le suit**…
- Principe de la méthode :
  - Sélectionner le minimum du tableau en parcourant le tableau de la fin au début, et en échangeant tout couple d'éléments consécutifs non ordonnés

Cet algorithme sera vu en TD …

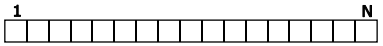
Algorithme de Tri par sélection

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

Algorithme de Tri par sélection

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

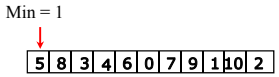
```
pour i de 1 à N-1:
    min = i
    pour j de i+1 à N:
        si T_j < T_min alors min = j
    Echanger(T_i, T_min)
```



Déroulement de l'algorithme

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

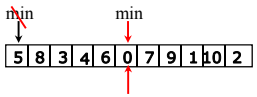
```
pour i de 1 à N-1:
    min = i
    pour j de i+1 à N:
        si T_j < T_min alors min = j
    Echanger(T_i, T_min)
```



Déroulement de l'algorithme

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

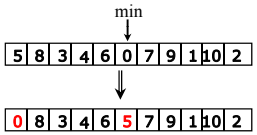
```
pour i de 1 à N-1:
    min = i
    pour j de i+1 à N:
        si T_j < T_min alors min = j
    Echanger(T_i, T_min)
```



Déroulement de l'algorithme

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

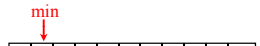
pour *i* ***de*** *1 à N-1:*
  min = i
  pour *j* ***de*** *i+1 à N:*
    si $T_j < T_{min}$ ***alors*** $min = j$
  Echanger(T_i, T_{min})



Déroulement de l'algorithme

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

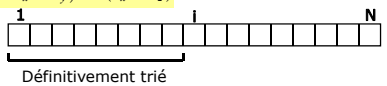
pour *i* ***de*** *1 à N-1:*
  min = i
  pour *j* ***de*** *i+1 à N:*
    si $T_j < T_{min}$ ***alors*** $min = j$
  Echanger(T_i, T_{min})



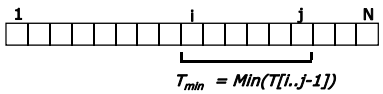
Invariants de boucles

- Idée:
  - A partir du 1er élément, on recherche le plus petit éléments dans le reste du tableau et on place ce dernier dans la première case. On poursuit ensuite avec le 2ième élément…

pour *i* ***de*** *1 à N-1:*
  $\{\forall x, y \in [1..i-1],\ \forall z \in [i..N]:\ (x < y \Rightarrow T_x \le T_y)\ et\ (T_x \le T_z)\}$



  min = i
  pour *j* ***de*** *i+1 à N:*
    $\{\forall x \in [i..j-1],\ T_{min} \le T_x\}$
    si $T_j < T_{min}$ ***alors*** $min = j$
  Echanger(T_i, T_{min})



Complexité de l'algorithme

- Idée:
  - Compter le nombre de test : $T_j < T_{min}$
  - Complexité du pire cas

$$\frac{n(n-1)}{2} \Leftrightarrow O(n^2)$$

Algorithme de Tri par insertion

- Idée:
  - A la $n^{ième}$ itération, le $n^{ième}$ élément est inséré à la bonne place dans le tableau trié…

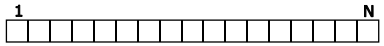
pour *i* ***de*** *2 à N:*

$val = T_i$
$j = i\text{-}1$

tant que $j \neq 0$ ***etpuis*** $T_j > val$

$T_{j+1} = T_j$
$j = j\text{-}1$

$T_{j+1} = val$

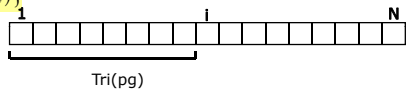


Invariants de boucles

- Idée:
  - A la $n^{ième}$ itération, le $n^{ième}$ élément est inséré à la bonne place dans le tableau trié…

pour *i* ***de*** *2 à N:*

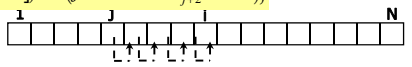
$\{\forall x, y \in [1..i-1]: (x < y \Rightarrow T_x \leq T_y)\}$



$val = T_i$
$j = i\text{-}1$

tant que $j \neq 0$ ***etpuis*** $T_j > val$

$\{(T[1..j] \& T[j+2] = T_0[1..i-1])\ et\ (j+2 \leq i \Rightarrow T_{j+2} > val)\}$



$T_{j+1} = T_j$
$j = j\text{-}1$

$T_{j+1} = val$

Complexité de l'algorithme

- Idée:
  - Compter le nombre de transfert : $T_{j+1} < T_j$
  - Complexité du pire cas

$$\frac{n(n-1)}{2} \Leftrightarrow O(n^2)$$

Plan

Algorithme de Tri par fusion

- Diviser pour régner
- Idée :
  - On divise le tableau en deux, les sous tableaux sont triés puis fusionnés

Algorithme de Tri par fusion

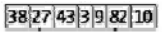
- Diviser pour régner
- Idée :
  - On divise le tableau en deux, les sous tableaux sont triés puis fusionnés

- Procédure de fusion de sous tableaux:
  - Interclasser(*T: tableau de [1..N]*; *i,m,s: entiers dans [1..N]*)
  - Sous tableaux: *T[i,m]* et *T[m+1,s]*
  - Résultat: *T[i,s]*

Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)

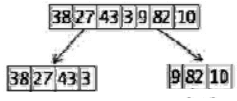


TriFusion (Inf, Sup : Entiers):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)

Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)



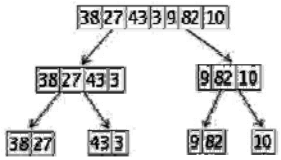
TriFusion (Inf, Sup : Entiers):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)

Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)

TriFusion (Inf, Sup : Entiers):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)

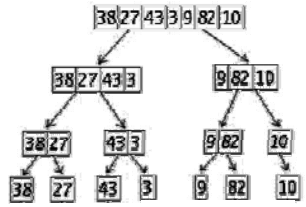


Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)

TriFusion (Inf, Sup: Entiers):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)

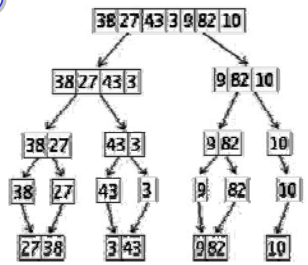


Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)

TriFusion (Inf, Sup):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)

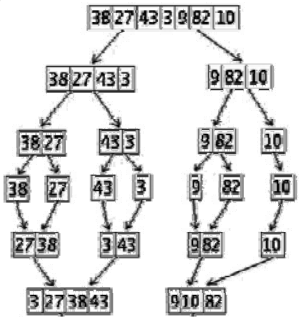


Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)

TriFusion (Inf, Sup):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)

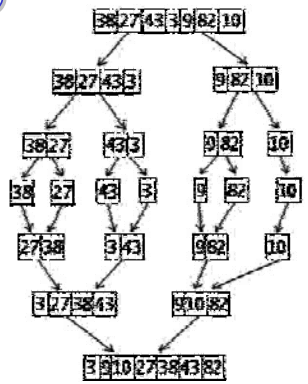


Algorithme de Tri par fusion

- Sous fonction: *TriFusion*
  - Tri du tableau *T* entre *inf* et *sup*

TRI:
TriFusion(1,N)

TriFusion (Inf, Sup):
mil = (inf + sup) div 2
TriFusion(inf, mil)
TriFusion(mil+1, sup)
Interclasser(T, inf, mil, sup)



Complexité de l'algorithme

- Idée:
  - Complexité du pire cas
  - Traitement d'un niveau en O(N)
  - Log(N) niveaux différents

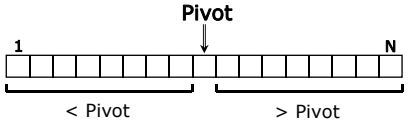
$$O(N\log(N))$$

Comparaison de deux algorithmes de tri

- Tri par sélection:
  - Lors du parcours de ***k*** éléments pour recherche du minimum, on ne mémorise que le minimum alors que ***k*** comparaisons ont étés effectuées
- Tri par fusion:
  - Toutes les comparaisons sont mémorisées en ordonnant les éléments comparés

Algorithme de Tri par segmentation (Quick sort)

- Idée :
  - Choisir un élément du tableau : ***pivot***
  - Ordonner les éléments du tableau par rapport au pivot
  - Appeler récursivement le tri sur les sous tableaux à gauche et à droite du pivot



Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup

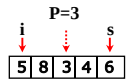
répéter
  tantque $T_j < p$
    i++
  tantque $T_s < p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



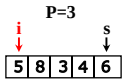
répéter
  tantque $T_i < p$
    i++
  tantque $T_s < p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



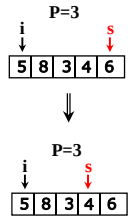
répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

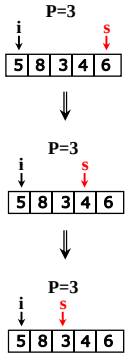
si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup
répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

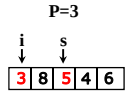


Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup
répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

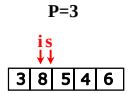


Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup
répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

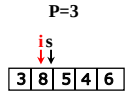


Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup
répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

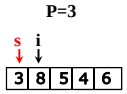
si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*



Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

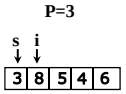
si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Introduction aux algorithmes de tri
Les tris séquentiels
Les tris récursifs
Tri par fusion
Tri par segmentation

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

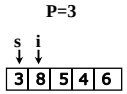
si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Introduction aux algorithmes de tri
Les tris séquentiels
Les tris récursifs
Tri par fusion
Tri par segmentation

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

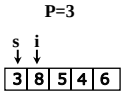
si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Introduction aux algorithmes de tri
Les tris séquentiels
Les tris récursifs
Tri par fusion
Tri par segmentation

Algorithme de Tri par segmentation (Quick sort)

TriSegmentation *(inf, sup: entiers dans [1..N])*:

P = ChoixPivot(T, inf, sup)
i = inf
s = sup



répéter
  tantque $T_i < p$
    i++
  tantque $T_s > p$
    s--
  si $i \leq s$ ***alors***
    Echanger(T_i, T_s)
    i++
    s--
jusqu'à $i > s$

.....

si $s > inf$ ***alors*** *TriSegmentation(inf, s)*
si $i < sup$ ***alors*** *TriSegmentation(i, sup)*

Invariant de boucle

TriSegmentation *(inf, sup: entiers dans [1..N])*:

```
P = ChoixPivot(T, inf, sup)
i = inf
s = sup
répéter
    tantque T_j < p
        i++
    tantque T_s < p
        s--
    si i ≤ s alors
        Echanger(T_i, T_s)
        i++
        s--
jusqu'à i > s
si s > inf alors TriSegmentation(inf, s)
si i < sup alors TriSegmentation(i, sup)
```

$$\{\inf \le k < i \Rightarrow T_k \le p \quad et \quad s < k \le \sup \Rightarrow T_k \ge p\}$$

Complexité

- Algorithme le plus efficace pour des données réparties aléatoirement
- Complexité moyenne

$$O(N \log(N))$$

- Complexité du pire cas: O(N²)
- Problème : choix d'un bon pivot

Conclusion

- Nombreux autres algorithmes de tri: tri par tas, tri de shell…
- Les meilleurs algorithmes de tris sont en: O(N.log N)
- Choisir un algorithme de tri différents selon la taille de l'ensemble à ordonner

Références

- Site
  - *http://deptinfo.cnam.fr/Enseignement/CycleA/SD/demonstration/Tri.html*
- Livres
  - N.Wirth (Algorithmes et structures de données, Prentice Hall 86 ou Eyrolles 87)
  - R.Sedgewick (Algorithmes en langage C, Addison-Wesley 90 ou InterEditions 91)
  - D.E.Kunth (The art of the computer programming, Vol.3, Addison-Wesley 74)