Algorithme de tri

Jean-François Berdjugin
IUT1, Département SRC, L'Isle d'Abeau

Tri

- Nous passons notre vie à organiser, trier des données par ordre de mérite, par ordre alphabétique, par distance (Z-sorting), …
- Un ordinateur si il calcul mal, trie bien.
- Un algorithme juste est-il performant, suivant quels critères (mémoire, temps d'éxecution, accès disques, …) ?
- Un problème important qui à donné naissance aux méthodes formelle (C.A.R.Hoare)

Tri

- Algorithme
  - **séquentiel**
  - parallèle
- Algorithme
  - **itératifs**
  - Récursifs
- Tri :
  - **Interne**
  - externe
- Performance en temps (fonction des données)
  - Meilleur cas
  - Plus mauvais cas

Temps d'éxecution

- Temps d'exécution en $\log_2 n$, n, $n \log_2 n$, n^2

=>

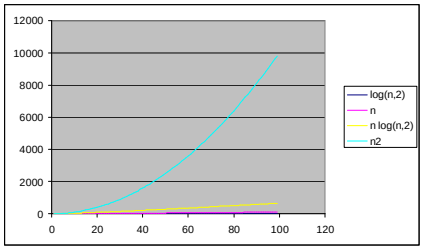
N=15 min = 900 s = 900 000 ms

$\text{Log}_2 900\,000$ =19 ms

$n \log_2 900\,000$ =17 801 609 ms = 17 801 s = 296 s = 5 min

$900\,000^2 = 8{,}1\ 10^{11}$ms

= 9 jours



=> Ça vaut le coup d'étudier des algorithme de tri.

Trier

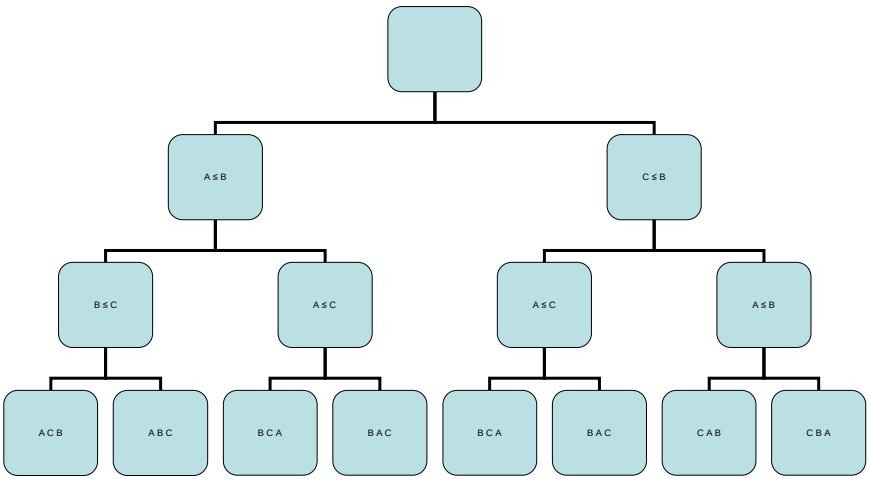
- Trier c'est ordonner suivant une relation d'ordre (relation binaire, réflexive, antisymétrique, transitive)
- Exemples : ≤, ordre alphabétique, grade, …

Tri par comparaison

- Principe comparer deux cases
- Si l'on doit trier n éléments alors ces n éléments peuvent former n! combinaisons.
- Si l'on utilise un arbre de résolution (représentant l'ensemble des comparaisons) de hauteur h (chemin de la racine vers la feuille) alors le nombre maximal de feuille est de 2^h

On a donc $n! \leq 2^h$

Tri par comparaison



Tri par comparaison

On a donc n! ≤ 2^h (6 ≤ 8)

<=>

$Log_2(n!) \leq log_2(2^h)$

Or la limite de n $log_2 n$ = log_2 n!

⇒ h est la limite inférieure de n $log_2 n$

⇒ Il faudra au moins n $log_2 n$ comparaison

⇒ Bref dans le pire des cas on ne pas dépasser un certain niveau de performance

⇒ Les algorithmes sont adaptés à des circonstances (des cas, des sous-problèmes).

Tri par insertion

- Idée : on prend les éléments les uns après les autres et on les insère parmi les éléments déjà triés

=>

Une boucle pour parcourir le tableau

Une boucle pour insérer à la bonne place

Tri par insertion

```
Fonction triInsertion(tab: tableau<entier>):vide
Var i,j,m: entier
Pour i de 1 à tab.length-1 faire
m←tab[i]                        //élément à insérer
j ←i
   tant que j>0 et tab[j-1]>m faire  //éléments plus grands
        tab[j] ←tab[j-1]
        j ←j -1
   fin tant que
   tab[j] ←m                    //insertion
Fin pour
Fin fonction
```

Tri par insertion

10	8	12	6	8	0	11

Tri par insertion

10	8	12	6	8	0	11
8	10	12	6	8	0	11
8	10	12	6	8	0	11
6	8	10	12	8	0	11
6	8	8	10	12	0	11
0	6	8	8	10	12	11
0	6	8	8	10	11	12

Tri par sélection

- Idée : trouver le plus petit et le placer en premier puis trouver le plus petit parmi les éléments non placés et le placer en second, etc.

=>

Une boucle de parcourt du tableau

Une boucle pour trouver le min

Tri par sélection

```
Fonction triSelection(tab: tableau<entier>):vide
Var i,j,m, min: entier
Pour i de 0 à tab.length -2 faire
min←i                          //élément à insérer
j ←i
    pour j de i+1 à tab.length-1 faire  //recherche min
        si tab[j] < tab[min] alors
                min ←j
        fin si
    fin pour
    m ← tab[i]                 //echange
    tab[i] ← tab[min]
    tab[min] ←m
Fin pour
Fin fonction
```

Tri par selection

10	8	12	6	8	0	11

Tri par selection

10	8	12	6	8	0	11
0	8	12	6	8	10	11
0	6	12	8	8	10	11
0	6	8	12	8	10	11
0	6	8	8	12	10	11
0	6	8	8	10	12	11
0	6	8	8	10	11	12

Tri a bulle

- Idee : permuter les éléments adjacents mal placé et parcourt autant de fois que nécessaire du tableau

=>

Une boucle pour selectioner les éléments

Une boucle pour les permutations

Tri à bulles

```
Fonction triBulles(tab: tableau<entier>):vide
Var i,j,m: entier
Pour i de tab.longueur()-1 à 0 pas -1 faire//élément à insérer
    pour j de 1 à i faire  //decaler les plus grands
        si tab[j-1] > tab[j] alors
                temp ←t[j-1]
                tab[j-1] ← tab[j]
                tab[j] ← tmp
        fin si
    fin pour
Fin pour
Fin fonction
```

Tri à bulle

10	8	12	6	8	0	11

Tri à bulle

10	8	12	6	8	0	11
8	10	12	6	8	0	11
8	10	12	6	8	0	11
8	10	6	12	8	0	11
8	10	6	8	12	0	11
8	10	6	8	0	12	11
8	10	6	8	0	11	12

Tri à bulle

8	10	6	8	0	11	12
8	10	6	8	0	11	12
8	6	10	8	0	11	12
8	6	8	10	0	11	12
8	6	8	0	10	11	12
8	6	8	0	10	11	12

Tri à bulle

8	6	8	0	10	11	12
6	8	8	0	10	11	12
6	8	8	0	10	11	12
6	8	0	8	10	11	12
6	8	0	8	10	11	12
6	8	0	8	10	11	12
6	0	8	8	10	11	12
6	0	8	8	10	11	12

Tri à bulle

6	0	8	8	10	11	12
0	6	8	8	10	11	12
0	6	8	8	10	11	12
0	6	8	8	10	11	12
….						
….						
0	6	8	8	10	11	12

Comparaison

Complexité :

- Tri par insertion : en N^2 (double imbriquées)
- Tri par sélection : en N^2
- Tri a bulle : en N^2

Mais d'autres algorithme donc beaucoup reposent sur le principe de diviser pour régner existent :

- Tri par casier : en N ce n'est pas un alogo par comparaison
- Tri par tas : en N Ln N
- …