INF220 Algorithme de tri

Jean-François Berdjugin

IUT1, Département SRC, Grenoble

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

=>

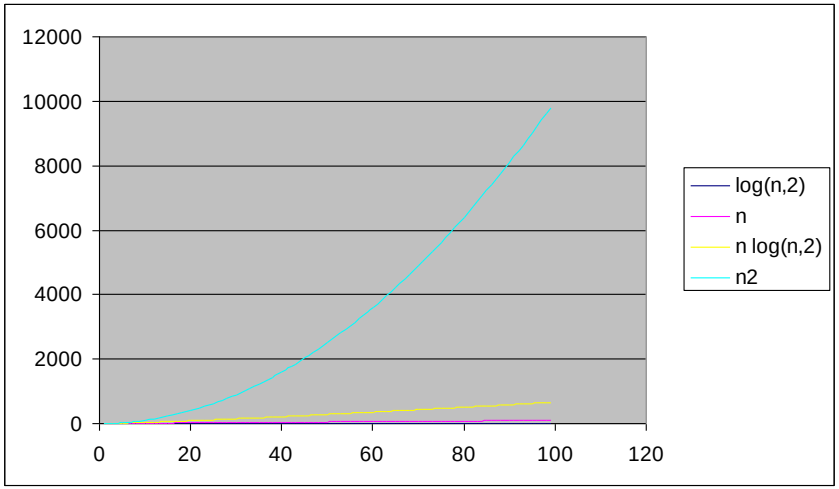
N=15 min = 900 s = 900 000 ms

$\text{Log}_2 900\,000 = 19$ 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, …

Algorithme lents

Les algorithmes que nous allons étudier sont dit lents, leur plus mauvais cas d'exécution a une complexité asymptotique en $O(n^2)$.

Pour aller plus vite ($O(n \ln n)$), il nous faut

- soit des arbres et la récurrence,
- soit des tris externes
- soit du parallélisme

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

```
public static void triInsertion(int tableau[])
    {
      int i,j, m;
      int l = tableau.length -1;
      for (i=1; i <= l; i++)
      {
              m = tableau[i];
              j=i;
              while (j>0 && tableau[j-1]>m)
              {
                      tableau[j]=tableau[j-1];
                      j=j-1;
              }
              tableau[j]=m;
      }
    }
```

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

```
public static void triSelection(int tableau[])
{
    int i,j,m, min;
    int l = tableau.length - 1;
    for (i=0; i< l; i++)
    {
        min = i;
        for (j=i+1; j<=l; j++ )
        {
            if (tableau[j]< tableau[min])
            min=j;
        }
        m=tableau[i];
        tableau[i] = tableau[min];
        tableau[min] = m;
    }
}
```

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

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

=>

Une boucle pour sélectionner les éléments

Une boucle pour les permutations

Tri à bulles

```
public static void triBulle(int tableau[])
{
    int i,j,m;
    int l = tableau.length -1;
    for ( i=l; i>=0; i--)
        for (j=1; j<=i; j++)
            if (tableau[j-1] > tableau[j])
            {
                m = tableau[j-1];
                tableau[j-1] = tableau[j];
                tableau[j]=m;
            }
}
```

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 :

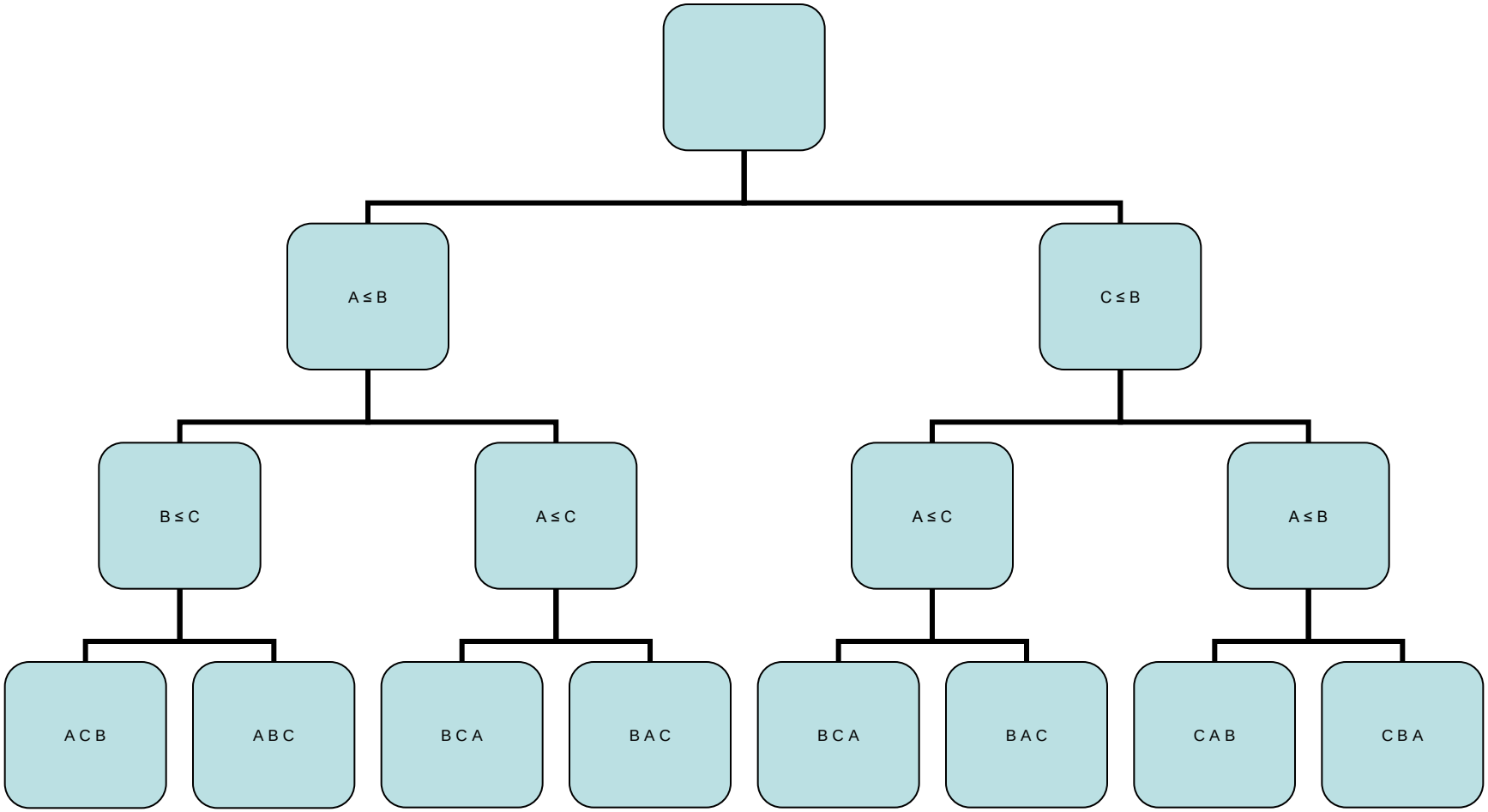
- Tri par casier : en N, c'est un algorithme de tri externe
- Tri par tas (tri par comparaison) : en N Log N
- …

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 $n! \leq 2^h$ $(6 \leq 8)$

<=>

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

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

$\Rightarrow$ h est la limite inférieure de $n \log_2 n$

$\Rightarrow$ Il faudra au moins $n \log_2 n$ comparaison

$\Rightarrow$ Bref dans le pire des cas on ne pas dépasser un certain niveau de performance

$\Rightarrow$ Les algorithmes sont adaptés à des circonstances (des cas, des sous-problèmes).

Tri par casier

La plage des données est limitée :

- On créer un tableau temporaire (les casiers) dont la taille est celle de la plage des données
- On range dans les casiers le nombre d’occurrence des valeurs du tableau initial
- On parcours les casiers dans l’ordre pour créer le tableau trié.

Tri casier

```
public static void triCasier(int tableau[]){
	int longueur=tableau.length;
	int mini=tableau[0]; int maxi=tableau[0];

	for(int i=1;i<longueur;i++){
		if(mini>tableau[i]) {mini=tableau[i];}
		if(maxi<tableau[i]) {maxi=tableau[i];}}


	int tmpLong=maxi-mini+1; int tmpTableau[]=new int[tmpLong];

	for(int i=0;i<tmpLong;i++){ tmpTableau[i]=0; }
	for(int i=0;i<longueur;i++) {tmpTableau[tableau[i]-mini]++; }

	int compt=0;
	for(int i=0;i<tmpLong;i++)
	{
		while(tmpTableau[i]>0)
		{
		tableau[compt]=mini+i;
		tmpTableau[i]--;
		compt++;
		}} }
```

Tri casier

tableau

4	3	10	4	3	2

mini : 2
maxi : 10
tmpLong: 9

1	2	2	0	0	0	0	0	1
0	1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9	10

casiers

tableau

2	3	3	4	4	10

La suite

Les objets : un moyen de lier les données (les variables d’instance) et les mécanismes portant sur ces données (les méthodes).