

IFT 2010 HIVER 2006

Structures de données

Miklós Csűrös

André-Aisenstadt 3149

csuros@iro.umontreal.ca

[http ://www.iro.umontreal.ca/~csuros/IFT2010/](http://www.iro.umontreal.ca/~csuros/IFT2010/)

Plan de cours

Répertoire des cours

IFT2010 Structures de données

- Crédits: 4 dont 1 de travaux pratiques - labo
- Durée: 1 trimestre(s)
- Généralement offert à l'automne et à l'hiver
- Période: le jour

Responsables

Faculté des arts et des sciences - Département ou école: Informatique et recherche opérationnelle

Description Types abstraits pour les structures de données, arbres, dictionnaires, files avec priorités, graphes, méthodes externes.

Préalables [IFT1020](#) et [IFT1063](#)

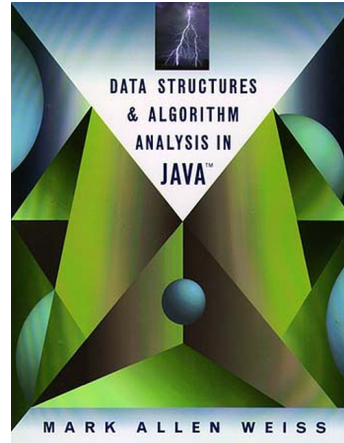
Horaire :

mardi 8 :30–10 :30 AA 1360, **jeudi** 10 :30–11 :30 AA 1360

mercredi démo 15 :30–17 :30 AA 1411

Matériel

Livre principal : [**Weiss**]



D'autres livres en réserve à Math-Info :

[**AHU**] Aho, Hopcroft, Ullman *Data Structures and Algorithms* («Structures de données et algorithmes»)

[**Drozdek**] *Data Structures and Algorithms in Java*

notes de cours affichés avant ou juste après le cours

devoirs : $4 \times 10\%$ intra : 30% final : 30%

Sujets

- Analyse d'algorithmes, notation *big-Oh* [**Ch. 2**]
- Listes et piles [**Ch. 3**]
- Arbres — binaires, balancés et *splay* [**Ch. 4**]
- Hachage [**Ch. 5**]
- Tas — *heap* ou *priority queue* [**Ch. 6**]
- Exemple d'algos — tris [**Ch. 7**]
- Exemple de SD : ensemble [**Ch. 8**]
- Graphes — SD et algo [**Ch. 9**]
- Dessert

Exemple

Problème : triage de nombres entiers

Entrée : un ensemble de nombres **IN**

Sortie : tableau `out` des nombres de **IN** en ordre croissant

P.e. : **IN** = {3, 5, 8, 1, 7}, `int[]` `out` = {1, 3, 5, 7, 8}

Noter : je n'ai pas défini la représentation de **IN** (par contre, `out` est un tableau d'entiers `int` en Java)

Tri A

on utilise une liste **OUT** qui préserve l'ordre d'addition des éléments

```
A1 initialiser OUT  $\leftarrow \emptyset$            // OUT est vide au début
A2 tandis que IN  $\neq \emptyset$ 
A3      $m \leftarrow \min \text{IN}$ 
A4     enlever  $m$  de IN
A5     ajouter  $m$  à la fin de OUT
A6 retourner OUT transformé en int[]
```

Questions

1. Implantation de **IN** : doit supporter

- (initialisation lors de la lecture de l'entrée)
- vérification si vide
- choix de **min**
- suppression d'un élément

2. Implantation de **OUT** : doit supporter

- ajout d'un élément
- transformation en `int []`

Solution simple

Supposons que **IN** est implanté par `int [] in`

- `vide? : in.length == 0`
- choix de **min** : `for (i=0; i<in.length; i++) { ... }`
- suppression d'un élément : (copier dans un autre tableau)

Solution de l'informaticien/ne

Recursion !

- on peut trier **IN** avec < 2 éléments sans problème
(exercice à la maison ...)
- récursion : trier deux moitiés séparément + fusionner les listes triées

Tri B

implantation de **IN** : `int []`

```
B1 si in.length < 2 alors retourner in
B2 int [] in1 ← {in[0],in[2],in[4],...}
B3 int [] in2 ← {in[1],in[3],in[5],...}
B4 int [] out1 ← triB(in1); int [] out2 ← triB(in2)
B5 int [] out ← fusion(out1,out2)
B6 retourner out
```

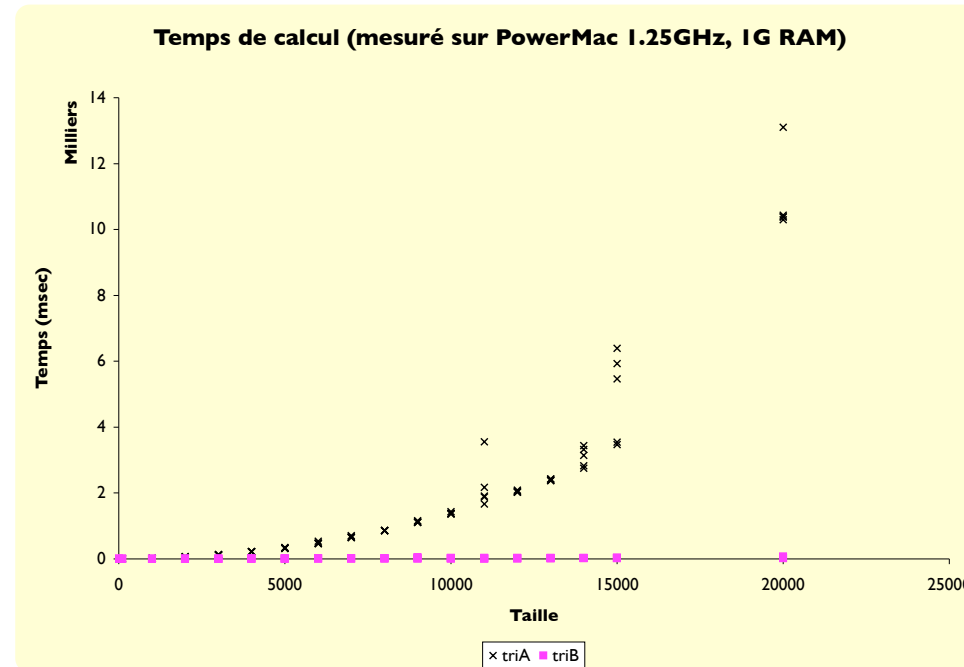
`int [] fusion(int [] arr1, int [] arr2)` prend deux tableaux triés et calcule leur fusion triée

Vitesse d'exécution

On peut mesurer le temps d'exécution des deux approches

```
...  
long T0 = System.currentTimeMillis(); // temps de début  
...  
long dT = System.currentTimeMillis()-T0; // temps (ms) dépassé
```

Vitesse d'exécution 2



triB a l'air d'être meilleur ...

Vitesse d'exécution 3

Le temps d'exécution dépend de beaucoup de conditions :

- entrée
- ordinateur
- environnement *run-time*
- température
- phase de la lune
- [...]

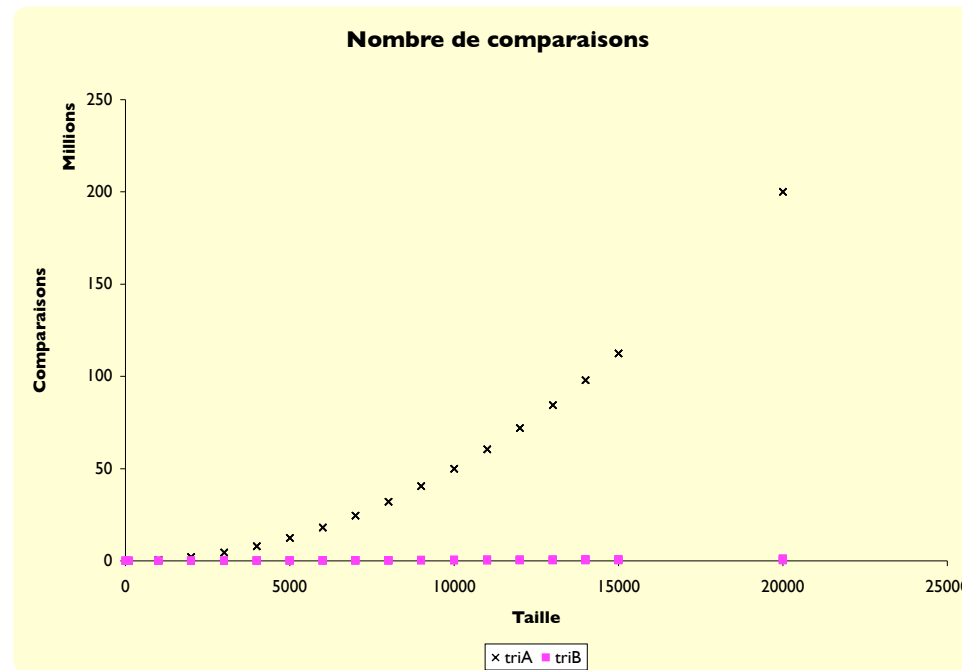
Ce qui nous intéresse, c'est la dépendance de l'entrée

⇒ abstraction de l'implantation

p.e., calculer nombre d'opérations «primitives»

Temps de calcul

mesuré ici en nombre de comparaisons entre éléments de IN



triB a toujours l'air d'être meilleur . . . indépendamment du plate-forme

Temps de calcul 2

But : preuve formelle pour comparer les nombre d'opérations

Thm. Nombre de comparaisons performés par tri A est $A(n) = n(n - 1)/2$ si $|\text{IN}| = n$ et on utilise l'implantation par `int []`.

Preuve. $A(n)$ est déterminé par le nombre de comparaisons utilisés pour `min`.

```
int min_pos = 0;
for (int i=1; i<in.length; i++)
    if (in[i]<in[min_pos]) min_pos=i;
// maintenant in[min_pos] est le min.
```

Nombre de comparaisons : `in.length - 1`

Preuve (cont.) Nombre total de comparaisons :

$$(n - 1) + (n - 2) + (n - 3) + \cdots + (2 - 1) + 0 = \frac{n(n - 1)}{2}.$$

