Université de Paris

L2 informatique - Année 2021–2022

TD d'Éléments d'Algorithmique n° 3

Dans ce TD, "trié" signifie "trié par ordre croissant".
*** Les exercices marqués d'une étoile sont à faire à la maison.**

Exercice 1. *Algorithme du cours.*
Appliquez l'algorithme du tri par insertion sur les tableaux suivants :

8	4	10	-5	1
1	2	3	4	5
5	4	3	2	1

Comptez dans chaque cas le nombre de comparaisons effectuées, ainsi que le nombre d'affectations à des cases du tableau.

Exercice 2. *Tri insertion selon la parité.*
On veut trier un tableau d'entiers de telle manière que les entiers pairs apparaissent dans la première partie du résultat, triés, suivis des entiers impairs, triés. Par exemple :

2	1	6	8	5	-3

devient :

2	6	8	-3	1	5

Adaptez l'algorithme de tri par insertion à ce cas.

Exercice 3. *Trier des cartes.*
Une carte contient deux informations, une couleur qui est un entier compris entre 1 et 4 (1 pour pique, 2 pour cœur, 3 pour carreau et 4 pour trèfle) et un rang qui est un entier entre 1 et 13. Le dix de pique sera représenté par [1,10], l'as de carreau par [3,1]. Un ensemble de cartes sera donc représenté par un tableau de tableaux d'entiers.
Lorsqu'on trie un jeu de cartes, on met les cartes ayant la même couleur ensemble — pique avant cœur, cœur avant carreau, etc.
En vous inspirant des exercices précédents, donnez un algorithme qui trie un tableau de cartes.

Exercice 4. *Tri stable.*
En algorithmique, on trie habituellement des entiers. En programmation, on trie généralement des structures plus compliquées. Par exemple, en Java on pourrait avoir une structure `Etudiant` :

```
public class Etudiant {
    public String nom;
    public int note;
}
```

et utiliser l'ordre défini par : `e1` $\leq$ `e2` lorsque `e1.note` $\leq$ `e2.note`.

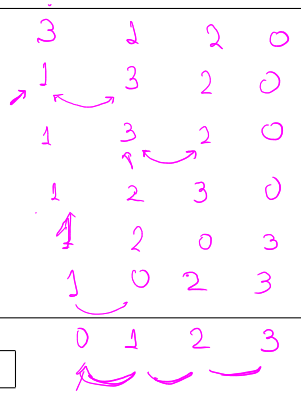
↳ deux étudiants qui ont les mêmes notes sont équivalants.

1. Montrez que l'ensemble des étudiants muni de la relation définie ci-dessus n'est pas un ordre. (Indication : exhibez deux éléments distincts `e1` et `e2` qui sont *équivalents* pour l'ordre, c'est-à-dire tels que `e1` $\leq$ `e2` et `e2` $\leq$ `e1`.) Un tel espace s'appelle un *préordre*.
2. On dit qu'un tri est *stable* lorsqu'il préserve l'ordre relatif des éléments équivalents : si `e1` était placé avant `e2` dans le tableau d'origine, et `e1` et `e2` sont équivalents, alors `e1` est encore avant `e2` dans le tableau trié. Le tri par insertion est-il stable ? Le tri par selection est-il stable ?
3. Quand cette propriété est-elle importante ?

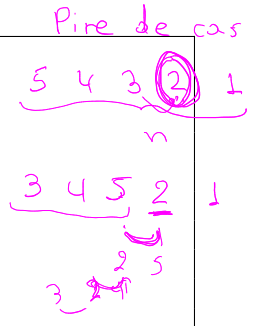
↳ stable
↳ par stable
↳ Quand on considère au même temps plus qu'un ordre (ex. par nom et par note)

Exercice 5. *Lecture d'algorithme.*
T est un tableau d'entiers de longueur n.

```
1  Fonc (T) :
2  i=0;
3  while i < n-1 {
4    if T[i] > T[i+1] {
5      exchange T[i] and T[i+1];
6      i=0;
7    } else {
8      i++;
9    }
10 }
```



Pire de cas :



1. Exécuter cette fonction sur

3	1	2	0

2. Que fait `Fonc(T)` ?
3. Combien de comparaisons sont effectuées ?

Alors, pour mettre element i dans la bonne position on fait :
$\sim i + (i-1) + (i-2) + \dots + 1 =$
comparaisons $= \frac{i(i+1)}{2}$

Exercice 6. *Fusion de tableaux triés *.*
On suppose que notre tableau est constitué de deux tableaux collés l'un à l'autre dont chacun est lui-même trié. Par exemple :

T :

2	9	4	6	7	10

Pour tous les element
$\sum_{i=1}^{n} \frac{i(i+1)}{2} \sim \sum_{i=1}^{n} i^2 \sim$
$O(n^3)$

1. Donnez un algorithme qui prend en entrée un tableau qui a cette propriété et retourne un nouveau tableau trié, qui contient les mêmes éléments. (Remarquez que la frontière entre les deux tableaux n'est pas donnée.)
2. Combien de comparaisons sont effectuées dans le pire des cas ? Indiquez votre réponse dans un commentaire que vous insèrerez dans le fichier source que vous avez utilisé à la question précédente.

Ex 2 :

```
fonction comparer(e1, e2)
    ret ←false
    si e2 est pair et e1 est impair
        ret ←true
    si (e1 est pair et e2 est pair) ou
       (e1 est impair et e2 est impair)
       si e2 < e1
           ret ←true
       // sinon ret ←false //pas besoin, ret est déjà false
    //si e2 est impair et e1 est pair //pas besoin, ret est déjà false
    retourne ret
```

```
fonction triPartiel(T, i )
    j ←i
    tant que j > 0 et comparer(T[j], T[j-1]) faire
    échanger T[j −1] et T[j ]
    j ←j −1

fonction triParInsertion(T)
    n ← longueur de T
    pour i ←1 à n −1 faire
    triPartiel(T,i)
```

Exo 3 : Il faut changer la fonction comparer.

tableaux T_1, T_2 ont chacun deux elements

fonction comparer (T_1 , T_2) :

ret ← false

si $T_1[0] > T_2[0]$

ret ← true

si $T_1[0] == T_2[0]$

si $T_1[1] > T_2[1]$

ret ← true //else false ✓

(si $T_1[0] < T_2[0]$ } déjà bon !

ret ← false)

T :

[1 | 3] [2 | 10] [4 | 13] ...

↓ couleur ↓ rang

TriPartiel et TriParInsertion

tel quel.