Exemples d'algorithmes de tri. Complexité.

Motivations : Les tris sont d'un usage constant en informatique, souvent en tant que sous-routines de programmes. Il faut donc un algorithme efficace pour traiter cette tâche.

Pb : On considère un tableau T de longueur n d'éléments comparables. On cherche à réorganiser le tableau par ordre croissant. Un algorithme de tri doit donc satisfaire :
- la sortie est triée
- la sortie est une permutation de l'entrée

I) Tris naïfs

Tri par insertion

C'est une méthode usuelle pour trier un jeu de cartes.
À l'instant j on considère que le tableau $T[1..j-1]$ est trié et l'on insère $T[j]$ dedans.

```
TRI-INSERTION(T):
pour j=2 à n faire
    x ← A[j], i ← j-1
    tant que i>0 et A[i]>x faire
        A[i+1] ← A[i], i ← i-1
    A[i+1] ← x
```

Terminaison : claire

Correction : au $j^{è}$ passage $T[1..j-1]$ est $T_0[1..j-1]$ trié et $T[j..n] = T_0[j..n]$.
↑ tableau initial

Complexité au pire :
Ici et dans toute la suite on évalue la complexité en comptant le nombre de comparaisons entre éléments du tableau.

* Ce tri est **en place** : on ne stocke qu'un nombre borné d'éléments en dehors de T.
* Ce tri est **stable** : si deux éléments du tableau ont la même clé, l'algo ne change pas leur ordre relatif.

D'autres exemples de tris naïfs :
tri par sélection, tri bulle, ...

II) Tris en $\Theta(n \log n)$

1/ Tri Fusion

C'est un algorithme qui utilise le paradigme "diviser pour régner".
On trie d'abord $T[1..\frac{n}{2}]$ et $T[\frac{n}{2}+1..n]$ puis on les combine.

```
TRI-FUSION(T,p,r):
  si p<r alors
    q ← ⌊(p+r)/2⌋
    TRI-FUSION(T,p,q)
    TRI-FUSION(T,q+1,r)
    FUSION(T,p,q,r)
```

```
FUSION(T,p,q,r):
  copier T[p..q] dans G
  copier T[q+1..r] dans D
  ajouter ∞ à la fin de G et D
  i←1, j←1
  pour k=p à r faire
    si G[i]≤D[j] alors
      T[k]←G[i]
      i←i+1
    sinon
      T[k]←D[j]
      j←j+1
```

Terminaison et correction : découlent de celles de FUSION (démontré avec un invariant de boucle).

Ce tri n'est pas en place (c'est son défaut majeur). Toutefois, il est stable.

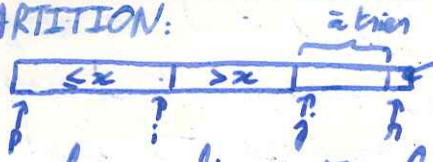
2/ Tri Rapide

Une autre utilisation de diviser pour régner :

```
TRI-RAPIDE(T,p,r):
  si p<r alors
    q ← PARTITION(T,p,r)
    TRI-RAPIDE(T,p,q-1)
    TRI-RAPIDE(T,q+1,r)
```

```
PARTITION(T,p,r):
  x←A[r], i←p-1
  pour j=p à r-1 faire
    si A[j]≤x alors
      i←i+1
      A[i]↔A[j]
  A[i+1]↔A[r]
  retourner i+1
```

Effet de PARTITION :



Terminaison et correction : découlent de celles de PARTITION.

Ce tri est en place et n'est pas stable.

Rq : cette version est stable...

Complexité :
Au pire, le tri fusion est en $O(n \log n)$ et le tri rapide en $O(n^2)$. En moyenne, ils sont en $O(n \log n)$.] dev 1

3/ Tri par tas

Ce tri est plus complexe : il utilise la structure de tas. Mais il combine les avantages des deux tris précédents : il est en $O(n \log n)$ au pire et il est en place.] dev 2

III Est-ce optimal ?

1/ Sans hypothèses supplémentaires : oui

Thm : Un tri par comparaisons utilise au moins $\Omega(n \log n)$ comparaisons en moyenne.

Preuve : avec les arbres de décision

)) Enoideaux

Rq: Il faut distinguer la théorie et la pratique: le tri rapide est souvent plus performant que le tri fusion ou le tri par tas, et pour trier moins de 30 éléments on utilise souvent un tri par insertion.

2/ Avec plus d'hypothèses : oui

Principe : plus on a d'hypothèses sur l'entrée plus on peut donner un algorithme performant.
Ex : si on sait que l'entrée est triée on a un algorithme en $O(1)$.

* Tri par dénombrement

On suppose que les éléments du tableau peuvent prendre k valeurs.
On parcourt une première fois le tableau pour compter combien de fois chaque valeur apparaît dans T.
Puis on reparcourt T et on copie les éléments dans T' (une copie de T) directement au bon endroit.
Ce tri n'est pas en place, il est stable, il s'exécute en $O(n+k)$.

* Tri par base

On suppose que les entrées sont des entiers donnés en base b avec au plus d chiffres.
On trie d'abord selon le chiffre de poids le plus faible, puis selon celui d'avant, ... (d passes) en utilisant un algorithme de tri stable. Par exemple en utilisant le tri par dénombrement on obtient un algo en $O(d(n+b))$.

* Tri par paquets

Cette fois on suppose que les entrées sont réparties uniformément dans $[0,1[$ et on subdivise régulièrement $[0,1[$ en n intervalles ($\sum n = |T|$) ("paquets").
Pour chaque paquet on crée une liste vide.
Puis on parcourt T en insérant chaque élément dans la liste du paquet correspondant (la $\lfloor x \cdot n \rfloor$-ième).
Ensuite on trie toutes les listes avec un tri par insertion (elles sont très courtes !).
Enfin on concatène les listes dans l'ordre.
On peut montrer que cet algorithme est en $\Theta(n)$ en moyenne.

Réf = Cormen !
↳ surtout tri fusion et rapide, pb sélection et tout III.2