Algorithmes de tri

- **Entrée** : une séquence d'éléments ordonnables (entiers, chaînes de caractères, ...)
- **Sortie** : une séquence de ces mêmes objets, triés par ordre

- Quels algorithmes ?
- Quelle complexité ?
- Peut-on faire mieux ?

Tri par insertion (tri du joueur de cartes)

Entrée : Tableau de n éléments T

For $i \leftarrow 1$ **to** $n-1$

- Insérer $T[i]$ parmi les éléments entre $T[0]$ et $T[i-1]$ en respectant l'ordre (en faisant une boucle $j \leftarrow i-1$ to 0 et en échangeant $T[j]$ et $T[j+1]$ si $T[j] > T[j+1]$)

For $j \leftarrow i-1$ to 0
  If $T[j] \leq T[j+1]$
    Break
  Échange $T[j]$ et $T[j+1]$

Complexité

Meilleur cas (tableau déjà trié) : $\Theta(n)$

Pire cas (tableau dans l'ordre inverse) :

$$0 + 1 + 2 + \dots + n-1 \text{ échanges}$$

$$= \frac{n(n-1)}{2} = \boxed{\Theta(n^2)}$$

Cas moyen (non prouvé) : $\boxed{\Theta(n^2)}$

Tri fusion (diviser pour régner)

cas de base : (Si le tableau est de taille 1, déjà trié $O(1)$

cas récursif : Sinon :

* Divise le tableau en deux tableaux de tailles quasi-identiques $O(1)$ ou $\Theta(n)$ suivant la structure de données
* On applique récursivement le tri fusion aux deux tableaux $2 \times T(n/2)$
* On fusionne les deux tableaux triés en les parcourant dans l'ordre et en prenant à chaque fois le plus petit des deux $\Theta(n)$

Exemple :

$(1\ 4\ 5\ 9\ 12)$ et $(2\ 3\ 7\ 8\ 10)$ $\hookrightarrow$ $1\ 2\ 3\ 4\ 5\ 7\ 8\ 9\ 10\ 12$

$$\begin{cases} T(n) = 2 \times T\left(\frac{n}{2}\right) + \Theta(n) \\ T(1) = O(1) \end{cases}$$

Thm maître

$a = 2$

$b = 2$

$c = \log_2 2 = 1$

$f(n) = \Theta(n^1)$

$2^{ème}$ cas du thm maître $\longrightarrow$ $\boxed{T(n) = \Theta(n \times \log n)}$

T_1, T_2 2 tableaux triés
$i_1 \leftarrow 0, i_2 \leftarrow 0$
tant que $i_1 < \ell(T_1)$ et $i_2 < \ell(T_2)$
  Si $T_1[i_1] < T_2[i_2]$
    $T[i_1 + i_2] \leftarrow T_1[i_1]$
    $i_1 \leftarrow i_1 + 1$
  Sinon
    $T[i_1 + i_2] \leftarrow T_2[i_2]$
    $i_2 \leftarrow i_2 + 1$
Si $i_1 < \ell(T_1)$ Ajoute $T_1[i_1] \ldots$ à T
Si $i_2 < \ell(T_2)$ Ajoute $T_2[i_2]$ à T

Tri rapide (quicksort)

(diviser pour régner)

Cas de base : Si un seul élément → déjà trié $O(1)$

Cas récursif :

* Choisir un pivot x (par exemple, le 1er élément du tableau) $O(1)$
* Construire deux sous-tableaux
  - tous les éléments $\leq x$
  - tous les éléments $> x$

  $\Theta(n)$
* Tri rapide récursif sur ces deux sous-tableaux (sans le pivot, mis entre les deux sous-tableaux)

  $T(n_1) + T(n_2)$ avec n_1 taille n_2 des ss-tableaux

Complexité

Pire cas : pivot est tjrs le premier ou dernier élément du tableau dans l'ordre (c'est ce qui se passe si le tableau est déjà trié et que le pivot est choisi comme le premier élément) : $\Theta(n^2)$

Meilleur cas

à chaque fois, on divise le tableau en 2 ss-tableaux de taille égale

$$T(n) \approx 2 \times T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\longrightarrow \boxed{T(n) = \Theta(n \log n)}$$

Optimisation : On choisit le pivot aléatoirement parmi tous les éléments.

En pratique, $\Theta(n \log n)$ et très efficace.

Borne inférieure

Peut-on mieux faire que $\Theta(n \log n)$ pour trier un tableau de n éléments <u>en comparant ses éléments 2 à 2</u>

Combien y a-t-il de manières d'ordonner n éléments ?

$n!$

1er él. n choix

2 $n-1$ —

3 $n-2$ —

$n^{ième}$ él. 1 choix

Algorithme de tri

→ fait des comparaisons

→ fait des échanges, des insertions...

Formule de Stirling

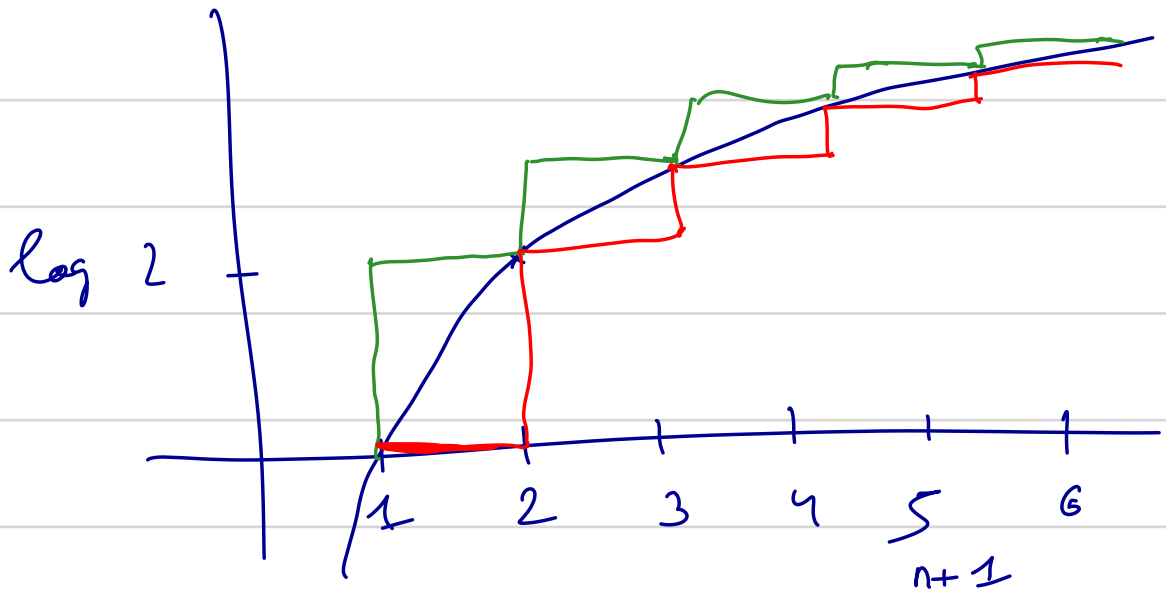
$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

$$\ln n! \sim n \ln n$$

$$\log_2 n! = \Theta(n \log n).$$

Autre preuve que $\log n! = \Theta(n \log n)$

$$\log n! = \sum_{k=2}^{n} \log k \geq \int_1^n \log x \, dx$$



$$\log n! \leq \int_1^{n+1} \log x \, dx$$

$$\int_1^N \log x \, dx = \Big[x \log x - x \Big]_1^N$$

$$= N \log N - N + 1$$

$$= \Theta(N \log N)$$

D'où $\log n! = \Theta(n \log n)$.

En conséquence, un algorithme de tri se basant sur des comparaisons ne peut avoir une complexité asymptotique meilleure qu'un $\Omega(n \log n)$.

Cette borne inf. n'est valable que pour des algos basés sur des comparaisons. Par exemple, si on nous dit que les n éléments à trier sont [les n premiers entiers], on peut les trier en $\Theta(n)$.

↳ cas dégénéré mais qui s'étend à d'autre cas.

Tri comptage : si j'ai une borne m sur les valeurs possibles (de 0 à m)

Tableau de stockage

1	2	3	4	5	6	7	8	9	10	J	Q	K
1	1	11	11	1		1	1	1	1	1	1	1

Entrée : 3 4 5 10 J K 4 8

7 Q 2 3 9

Sortie : 2 3 3 4 4 5 7 8 9 10 J Q K

Complexité : $\textcircled{LI}(n + m) = \textcircled{H}(\max(n, m))$

n : nb d'objets à trier

m : nombre de valeurs possibles

$\textcircled{H}(m)$ → Construire un tableau T des m éléments possibles (initialisé à 0)

$\textcircled{H}(n)$ → Pour chacun des n objets

Incrémenter la case de T correspondant à l'objet

$\textcircled{LI}(m+n)$ → Pour chacune des cases de T

Construire T[i] de fois l'objet i

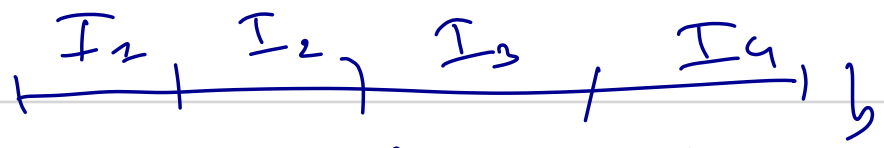
Autre ex. de tri sans comparaison : tri par paquets

$B = \alpha n$ avec $\alpha \leq n$

B : # de paquets (buckets)

On suppose que les valeurs à trier sont uniformément réparties dans un intervalle [a;b]

* On construit B paquets de même taille dans [a;b]



B = 4

* On met chaque élément à trier dans l'intervalle I_j correspondant
* On trie (par insertion) chaque I_j

Si α est une constante (et si les valeurs sont vraiment uniformément réparties) alors tri en $\Theta(n)$.