cié sur la ligne $n + 1$ (le paquet n'a pas de fils), on le place sur le paquet correspondant (son père) de la ligne $n - 1$ s'il s'agit d'un fils gauche, et en dessous s'il s'agit d'un fils droit.

La complication de ce tri inventé en 1962 par C. Hoare en vaut vraiment la peine, car le nombre de comparaisons qu'on doit réaliser pour classer n cartes (ou données) est cette fois en moyenne à peu près $n.\log_2(n)$. Dans cette formule, il faut arrondir le logarithme en base 2 de n à l'entier supérieur (rappelons que $\log_2(2) = 1$, $\log_2(4) = 2$, $\log_2(8)=3$, etc.) En définitive, pour 10 données, il faut environ 40 comparaisons, pour 100 données, environ 700, et pour 1 000, environ 10 000, ce qui est sensiblement moins qu'avec le tri par insertion.

La justification de ce $n.\log_2(n)$ est la suivante. D'une part, pour construire chaque ligne, il faut comparer chaque carte restante à une carte de la ligne précédente. Il faut donc faire environ n comparaisons à chaque construction de ligne. D'autre part, le nombre de lignes est environ $\log_2(n)$, car il y a 1 carte au niveau 1, 2 cartes au niveau 2, 4 au niveau 3, 2^{n-1} au niveau n. Ce calcul n'est qu'une évaluation, car certains paquets s'épuisent plus vite que d'autres, mais l'idée expliquée peut être rendue parfaitement rigoureuse.

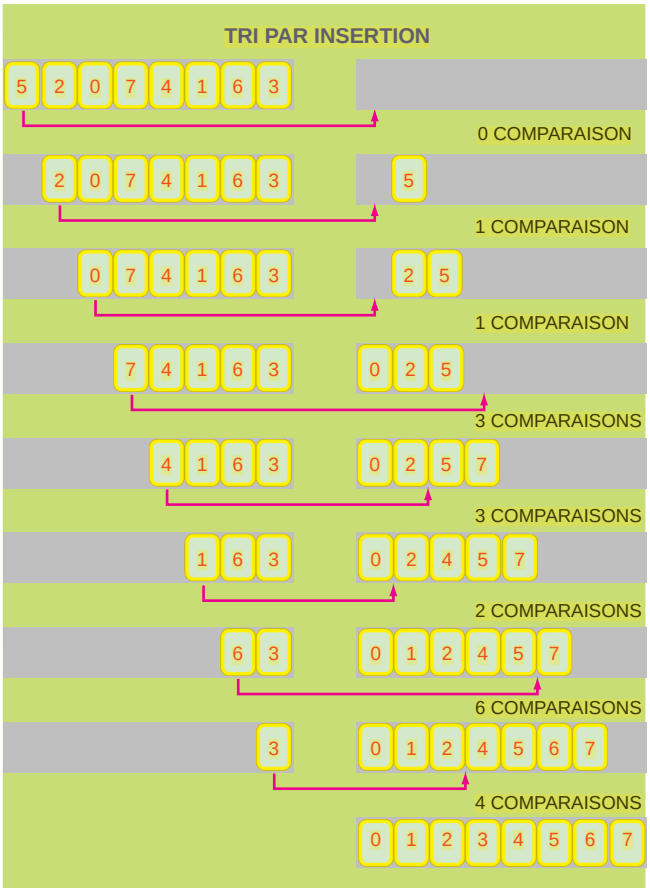
Dans les mauvais cas (quand la séparation par le pivot est systématiquement déséquilibrée), le tri-rapide a exactement une complexité de $n\,(n-1)/2$ comparaisons, c'est-à-dire la complexité du tri par insertion : ce qui distingue les deux tris, c'est le comportement en moyenne, qui est toujours quadratique pour le tri par insertion, alors qu'il est en $n.\log_2(n)$ pour le tri-rapide.

Assez paradoxalement, les cas où le tri-rapide est le plus lent se présentent quand le paquet initial est déjà totalement ou partiellement trié. Statistiquement, ces mauvais cas se présentent très rarement (d'où la moyenne en $n.\log_2(n)$). Cependant il se trouve que les données qu'on doit trier ne sont bien souvent pas statistiquement quelconques et qu'en pratique elles sont au contraire souvent partiellement ou totalement triées : se reposer sur la rareté statistique (très différente de la rareté pratique) des mauvais cas est donc un mauvais calcul (que certains programmeurs mal informés font pourtant).

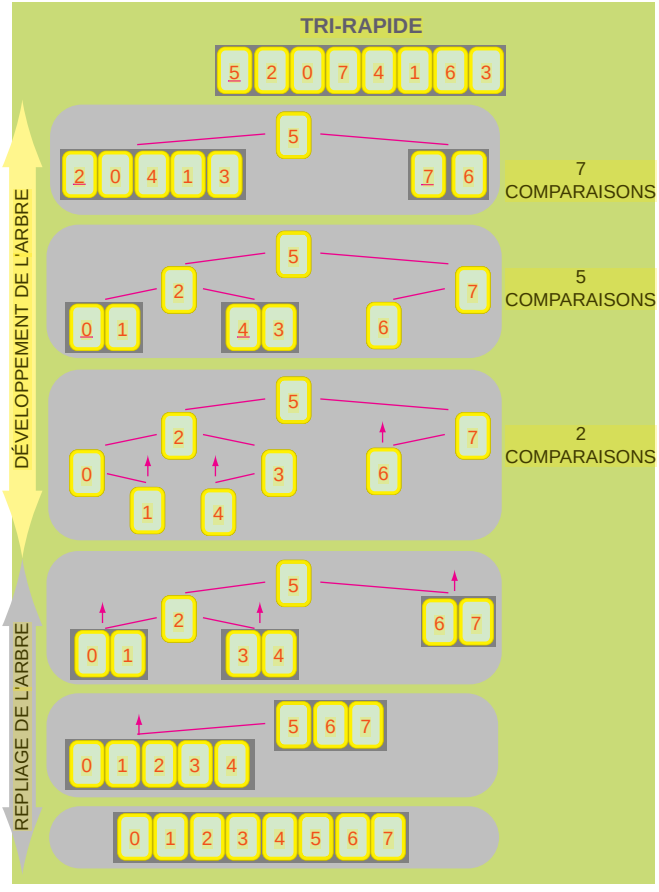
Il existe heureusement un moyen simple d'arranger les choses et qui permet de considérer en définitive que le tri-rapide est un bon algorithme : il faut mélanger les données avant de les trier ! C'est aujourd'hui ce qu'on recommande à ceux qui, en informatique, veulent utiliser le tri-rapide. Avec cette précaution, le tri-rapide est alors très efficace, et bien meilleur que le tri par insertion.

LE MIEUX QUE L'ON PUISSE FAIRE

C'est même un algorithme qu'on ne peut pas espérer améliorer, car $n.\log_2(n)$ est la borne la meilleure qu'on puisse atteindre : aucun algorithme de tri de données n'utilisant que des comparaisons ne peut en faire moins de $n.\log_2(n)$ en moyenne. La justification de ce résultat est intéressante. Notons qu'il y a $n!$ classements différents des cartes et que pour les trier il faut nécessairement les discriminer les uns des autres. Avec une comparaison, on ne peut discriminer que 2 classements ; avec 2 comparaisons, on ne peut discriminer que $4 = 2^2$ classements, etc. Donc, pour discriminer les $n!$ classements possibles



1. Le tri par insertion est naturel et simple à programmer. Malheureusement, ce n'est pas un tri efficace, car, pour trier n données, il lui faut en moyenne effectuer $n^2/4$ comparaisons.



2. Le tri-rapide est beaucoup plus efficace, même s'il est plus complexe à programmer : pour trier n données, il lui faut, en moyenne, effectuer $n.\log_2 n$ comparaisons.

de n cartes, il faut un nombre de comparaisons m tel que $2^m \geq n!$ Le résultat annoncé s'obtient maintenant en utilisant la célèbre formule de Stirling, $n! \approx \sqrt{(2\pi n)}\,(n/e)^n$, qui donne $\log_2(n!) \approx n \log_2(n)$. Pour discriminer tous les classements possibles de cartes (ce qui est nécessaire pour trier les cartes), il faut donc au moins effectuer $n.\log_2(n)$ comparaisons.

TRI DE CARTES PERFORÉES

Pour les cartes, le tri-rapide n'est pas agréable, il est encombrant. De plus, le tri-rapide n'utilise pas l'information que, dans un jeu normal, il y a une carte de chaque sorte ni plus, ni moins.

En informatique, bien souvent, on ne sait pas si les données qu'on doit trier appartiennent ainsi à un ensemble de valeurs réduit, mais si c'est le cas il existe d'autres algorithmes qui peuvent être préférés au tri-rapide.

Lorsqu'on doit trier des numéros dont on sait qu'ils forment une suite complète (ou seulement contenue dans un intervalle assez petit), il existe un algorithme amusant qui fut utilisé à l'époque des cartes perforées, car il pouvait s'effectuer mécaniquement avec une grande rapidité.

Je vais présenter l'idée de ce tri appelé *tri par base* sur un exemple. Imaginons que nous voulions trier les nombres suivants, tous inférieurs à 16 :
2 ; 7 ; 4 ; 9 ; 1 ; 15 ; 3 ; 13 ; 6 ; 12 ; 5.
On les écrit tous en base 2, ce qui donne cette liste de nombres binaires de quatre chiffres binaires au plus chacun :
10 ; 111 ; 100 ; 1001 ; 1 ; 1111 ; 11 ; 1101 ; 110 ; 1100 ; 101.

On les classe selon le dernier chiffre binaire : ceux dont ce dernier chiffre binaire est 0 sont amenés en tête :
10 ; 100 ; 110 ; 1100 ; 111 ; 1001 ; 1 ; 1111 ; 11 ; 1101 ; 101.

On les classe selon l'avant-dernier chiffre binaire : ceux dont cet avant-dernier chiffre binaire est 0 sont amenés en tête en respectant l'ordre qu'ils ont les uns par rapport aux autres (c'est important) :
100 ; 1100 ; 1001 ; 1 ; 1101 ; 101 ; 10 ; 110 ; 111 ; 1111 ; 11.

On continue ainsi en les classant selon l'avant-avant-dernier chiffre binaire, puis l'avant-avant-avant-dernier chiffre binaire.
1001 ; 1 ; 10 ; 11 ; 100 ; 1100 ; 1101 ; 101 ; 110 ; 111 ; 1111.

1 ; 10 ; 11 ; 100 ; 101 ; 110 ; 111 ; 1001 1100 ; 1101 ; 1111.

Cette fois, c'est bon :
1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 9 ; 12 ; 13 ; 15

Avec des cartes perforées, il suffisait d'écrire sur chaque carte son numéro en notation binaire, de sélectionner mécaniquement toutes les cartes dont le dernier chiffre est un zéro, de les extraire d'un coup et de les faire passer devant, etc. Une méthode d'encoche permet *(voir la figure 3)* de faire l'opération de sélection instantanément avec une tige métallique.

Ce tri appartient à l'histoire de l'informatique, et d'ailleurs sa première description est due à L. J. Comrie qui, en 1929, l'a introduite en présentant des équipements pour cartes perforées. Du matériel de bureau dans les années 1960 était fondé sur cette méthode.

Le classement de fiches étant maintenant fait dans la mémoire des ordinateurs, ce tri est moins pratiqué. Cependant Thomas Cormen fait remarquer que, pour trier un million de nombres de 64 chiffres binaires, un tel tri (programmé et non plus bien sûr effectué avec des cartes perforées) supporte bien la comparaison en temps de calcul avec le tri-rapide. Il ne faut donc pas l'oublier.

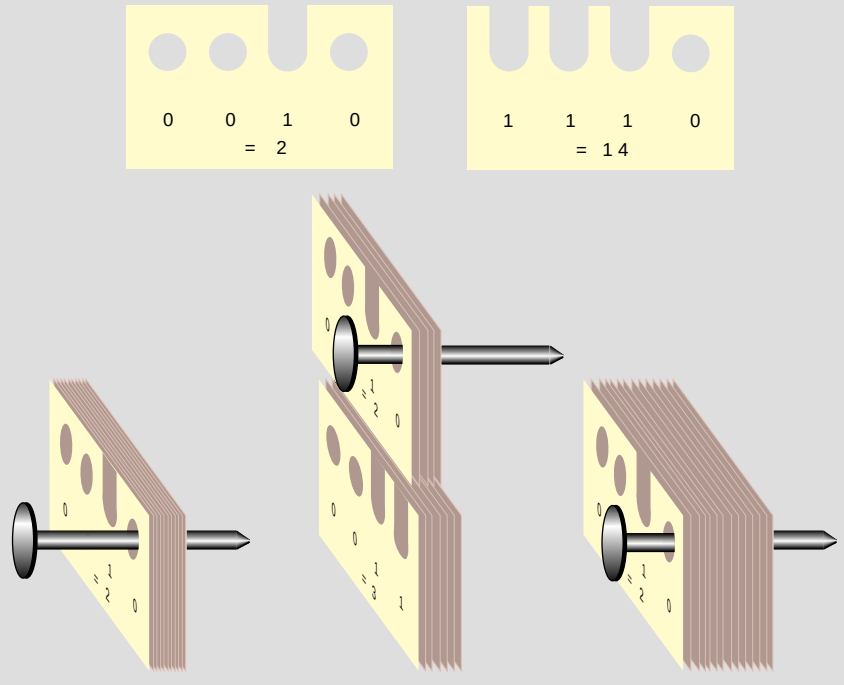
La comparaison sur critère mathématique (comme nous l'avons faite entre le tri par insertion et le tri-rapide) n'est pas facile, car le tri par base utilise la décomposition en base 2 des nombres (l'obtenir est bien sûr un travail). De plus, on ne compare que des chiffres binaires et non plus des nombres. On ne peut donc pas évaluer le tri par base par la méthode utilisée précédemment, il faut mener des analyses plus fines. On montre cependant qu'une fois que la taille maximale des nombres à trier est fixée, le temps de calcul du tri par base est proportionnel au nombre de données à trier : il faut kn étapes de calcul pour trier n données, k étant une constante qui dépend de la taille maximale des données à trier.

En apparence, cela semble signifier qu'il faut toujours préférer cet algorithme aux deux précédents. Mais la constante de proportionnalité k étant assez grande, cette conclusion n'est vraie que pour des grandes valeurs de n.

En définitive, si vous devez réaliser un tri de données, choisir le bon algorithme n'est pas simple et demande une analyse préalable. Néanmoins la règle suivante devrait suffire dans presque toutes les situations :

– (a) s'il y a peu de données à trier (moins de 100), utilisez le tri par inser-

3. TRI PAR BASE AVEC DES CARTES PERFORÉES



INSÉRER LA TIGE DANS LES TROUS ET ENCOCHES EN DERNIÈRE POSITION

SOULEVER LES CARTES PRISES (CELLES QUI SE TERMINENT PAR 0)

PLACER CE PAQUET DE CARTES DEVANT LES AUTRES

FAIRE ENSUITE LA MÊME OPÉRATION AVEC LES TROUS ET ENCOCHES EN POSITION 3, PUIS 2, PUIS 1. LE PAQUET SERA ALORS CLASSÉ.