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.

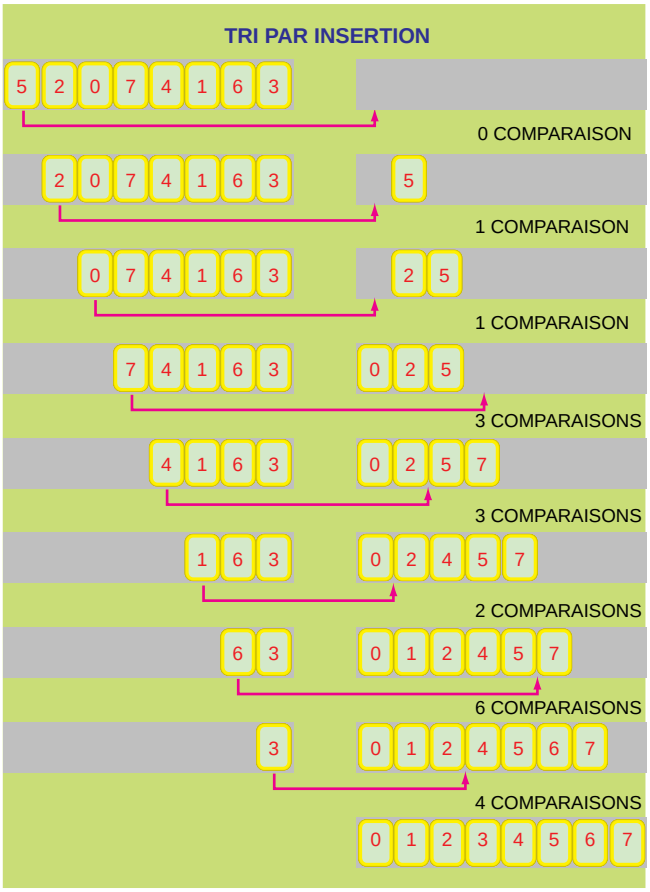
Dans les mauvais cas (quand la séparation par le pivot est systématiquement déséquilibrée), le tri-rapide a exactement une complexité de $n\,(n-1)/2$ comparaisons, c'est-à-dire la complexité du tri par insertion : ce qui distingue les deux tris, c'est le comportement en moyenne, qui est toujours quadratique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

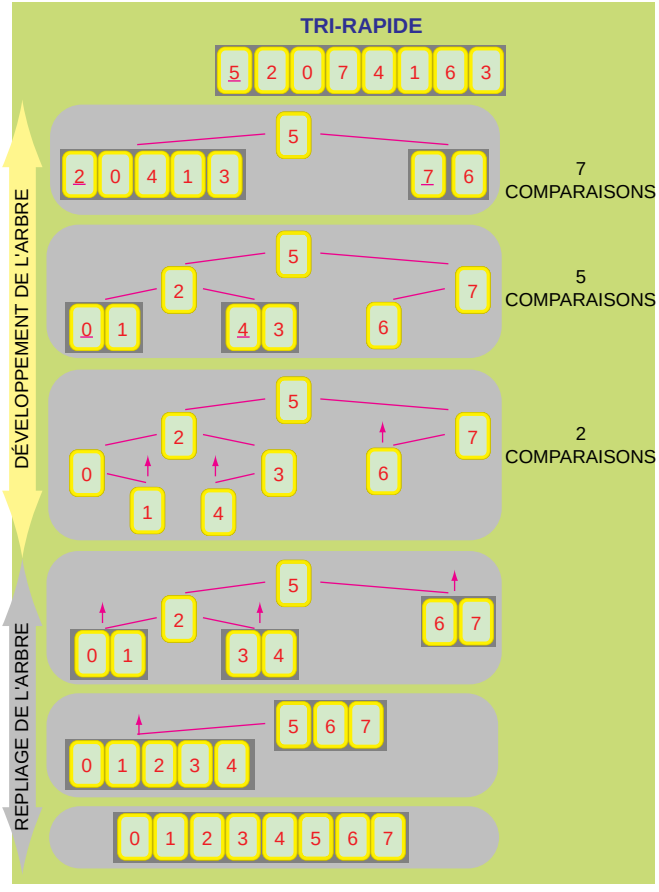
Il existe heureusement un moyen simple d'arranger les choses et qui permet de considérer en définitive que le tri-rapide est un bon algorithme : il faut mélanger les données avant de les trier ! C'est aujourd'hui ce qu'on recommande à ceux qui, en informatique, veulent utiliser le tri-rapide. Avec cette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