

# Tony Hoare

Un bref aperçue  
Par Théo.H

*«There are two methods in  
software design. One is to make the pro-  
gram so simple,  
there are obviously no errors.  
The other is to make it so complicated,  
there are no obvious errors.»*



## Table des matières

<b>1</b>	<b>La logique de Hoare</b>	<b>1</b>
1.1	Presentation . . . . .	2
1.2	Méthode . . . . .	2
<b>2</b>	<b>Le Quicksort</b>	<b>2</b>
2.1	Presentation . . . . .	2
2.2	Principe et implementation . . . . .	3
2.3	Complexité . . . . .	4

## *Une bref introduction*

*Tony Hoare* ou *Charles Antony Richard Hoare* est un Informaticien influent du XX siecle. Il est principalement connue pour sont implication sur la synchronisation des processus avec les moniteur, La logique de Hoare pour des preuve formelle de justesse algorithmique et le *quicksort*.

## 1 La logique de Hoare

## 1.1 Présentation

Le but de la *Logique de Hoare* est de permettre un système formel de raisonnement sur la justesse d'un programme. Plus concrètement, comme chacun le sait, il existe des systèmes critiques pour lesquels l'incertitude n'est pas acceptable. Cette logique est basée sur les **triplet de Hoare**, ce triplet de la forme  $\{P\}S\{Q\}$  est respectivement composé de :

- $P$  Une precondition
- $S$  Le programme
- $Q$  Une postcondition

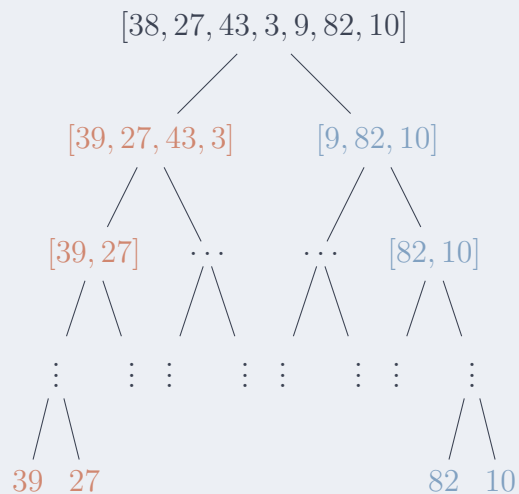
Les precondition et postcondition sont deux assertions appartenant à la **logique des prédicats**.

## 1.2 Méthode

# 2 Le Quicksort

## 2.1 Présentation

Le *Quicksort* ou tri rapide dans la langue de Molière, est un algorithme de tri qui est ... rapide. De manière plus concrète, il s'agit d'un tri qui se rapproche des limites possibles d'un tri comparatif sans propriété connue.



Ce tri repose sur le principe du diviser pour mieux régner. Si en politique cela consiste à augmenter le nombre de candidats d'opposition, en informatique cela consiste en trois étapes :

1. **Diviser**, on divise l'instance en plus petite instance, en generale en separant en deux ce qui explique le retour recurent du  $\log_2$  dans la complexiter de cette famille d'algorithme.
2. **Regner**, cette partie est surement la plus evidente car c'est la que l'ont le trie.
3. **Reunir**, Enfin reunie afin d'obtenir notre resultat final, c'est d'ailleur cette partie qui donne le caractere linaire des trie "*divide and conquere*"

Le trie dans l'exemple n'est pas le trie rapide, mais le trie fusion un trie baser lui aussi sur le *Divide and conquere*, il est a noter que la figure ne represente que la phase de division et pas la fusion elle meme, en effet celle ci est specifique au merge sort.

## 2.2 Principe et implementation

Le trie rapide, comme dit precedament, repose sur le *divide and conquere* plus particulierment autour d'un **pivot**. Ce pivot possede neanmoins une propriete interessante, a la suite d'un *partitionnement* tout les element a gauche lui sont inferieur et respectivment, ce que l'ont peut resumer par :

$$\forall (i, j) \in [p, q] \times [q + 1, r], \quad L[i] \leq L[j].$$

Cette propriete implique une infomation inportante, **tout les partition sont trier**, ainsi et de maniere naturelle. on peut juste ce contenter de reproduire cette procedure de maniere recursive.

```
int Partitionner(liste t, uint p, uint r){
    int x=t.t[p];
    int i=p;
    int j=r;
    while (t.t[j]>x)
    {
        j--;
    }
    while (i<j)
    {
        swap(t.t, i, j);
        do
        {
            j--;
        } while (t.t[j]>x);
        do
        {
            i++;
        } while (t.t[i]<x);
    }
    return j;
}

void TriRapide(liste L, uint p, uint r){
```

```

    if (p < r)
    {
        q = Partitionner(L, p, r);
        TriRapide(L, p, q);
        TriRapide(L, q + 1, r);
    }
}

```

## 2.3 Complexité

De manière étonnante pour un trie utiliser aussi fréquemment sont pire cas est en  $\Theta(n^2)$ , néanmoins il existe de nombreuses variantes de ce trie qui résolvent ce problème. Un tri rapide est par exemple le choix du pivot de manière aléatoire.

Concernant la complexité dans le meilleur cas l'algorithme **partitionner** est en  $\Theta(n)$ , en effet les deux indices  $i$  et  $j$  sont respectivement incrémenter et décrémenter, et ceux jusqu'au pivot donc ces opérations se feront donc  $n$  fois. On peut s'étonner de ces résultats mitigés et se demander pourquoi cette algorithme est si utilisé.