[ALGO] Complexité des Algorithmes (6)

Par Rhaeven, relecture par Find3r

Écrire T(n) en fonction de T(n-1)

$T(n) = cn + \sum_{i=1}^{n-1} T(i)$
$T(n-1) = c(n-1) + \sum_{i=1}^{n-2} T(i)$

$T(n) - T(n-1) = c + T(n-1)$
$T(n) = 2T(n-1) + c$
$= 2(2T(n-2) + c) + c$
$= 4T(n-2) + 2c + c$
$= 8T(n-3) + 4c + 2c + c$
$= 2^{n-1}T(1) + 2^{n-2}c + 2^{n-3}c + \ldots + 2c + c$

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

QuickSort en moyenne

$T(n) = \Theta(n) + T(l) + T(n-l)$

> Ou T(l) = rec à gauche et T(n-l) = rec à droite.

- Les valeurs possibles de l : toutes les valeurs de $[1, n-1]$

En moyenne

$T(n) = \Theta(n) + \frac{1}{n-1} sum_{l=1}^{n-1}(T(l) + T(n-l))$

$T(n) = \Theta(n) + \frac{2}{n-1} \sum_{l=1}^{n-1} T(l)$

$(n-1)T(n) = cn(n-1) + 2\sum_{l=1}^{n-1} T(l)$
$(n-2)T(n) = c(n-1)(n-2) + 2\sum_{l=1}^{n-1} T(l)$

$(n-1)T(n) - (n-2)T(n-1) = 2c(n-1) + 2T(n-1)$
$(n-1)T(n) = nT(n-1) + 2c(n-1)$
On divise par (n(n-1)) : $\frac{T(n)}{n} = \frac{T(n-1)}{n-1} + \frac{2c}{n}$

ON pose $U(n) = \frac{T(n)}{n}$
$U(n) = U(n-1) + \frac{2c}{n}$
$= U(n-2) + \frac{2c}{n-1} + \frac{2c}{n}$
$= U(1) + \frac{2c}{2} + \frac{2c}{3} + \ldots + \frac{2c}{n}$
$= \Theta(1) + 2c \sum_{i=2}^{n} \frac{1}{i}$
$= \Theta(1) + 2c \times \Theta(logn)$

$U(n) = \Theta(logn)$

$T(n) = n \times U(n) = \Theta(nlogn)$

Comparaison

	QuickSort	MergeSort
Best	$\Theta(nlogn)$	$\Theta(nlogn)$
Any	$\Theta(nlogn)$	$\Theta(nlogn)$
Worst	$\Theta(n^2)$	$\Theta(nlogn)$

Partition de QuickSort

```
QuickSort(A,b,e)
    if e - b > 1:
        m = Partition(A,b,e)
        QuickSort(A,b,m)
        QuickSort(A,m,e)
```

```
Partition(A,b,e):
    i <- b - 1; j <- e; p = A[b]
    for ever:
        do ++i until A[i] >= p
        do --j until A[j] <= p
        if i < j
            A[i] <-> A[j]
        else
            return i + (b == i)
```

Recherche du pivot idéal pour Partition

Pivot idéal = Médiane du tableau

Récursion terminale : Le dernier appel récursif n'est suivi d'aucune autre opération avant le return.

```
int fac2(int n, int acc):
start:
    if (n <= 1)
        return acc;
    acc = n * acc;
    n = n - 1;
    goto start;
```

```
int fac2(int n, int acc)
    while (n <= 1) {
        acc *= n;
        n--;
    }
    return acc;
```

Optimisations de QuickSort

```
QS(A, b, e):
    while (e - b > 1)
        n <- Partition(A, b, e)
        QS(b, m)
        b <- m
```

Ne change pas la complexité mais on gagne en mémoire dans le cas d'un tableau trié à l'endroit : On passe de $\Theta(n)$ recursions à $\Theta(1)$

Optimisation 1 :
Faire l'appel récursif sur le plus petit côté, et faire une boucle sur l'autre côté

```
QS(A, b, e):
    while e - b > 1:
        m <- Partition(A, b, e)
        if m - b > e - m:
            QuickSort(A, m, e)
            e <- m
        else
            QuickSort(A, b, m)
            b <- m
```

- Nombre max d'appels récursifs empilés : $log_2 n$ au lieu de $n - 1$

⇒ On consomme moins de pile !

```
QuickSort'(A, b, e):
    while e - b > 4:
        m <- Partition(A, b, e)
        if m - b > e - m:
            QuickSort'(A, m, e)
            e <- m
        else
            QuickSort'(A, b, m)
            b <- m
    InsertionSort(A, b, e)

QuickSort(A, n):
    QuickSort'(A, 0, n)
    InsertionSort(A, n)
```

Andrei Alexandrescu - CPPCOM 2019