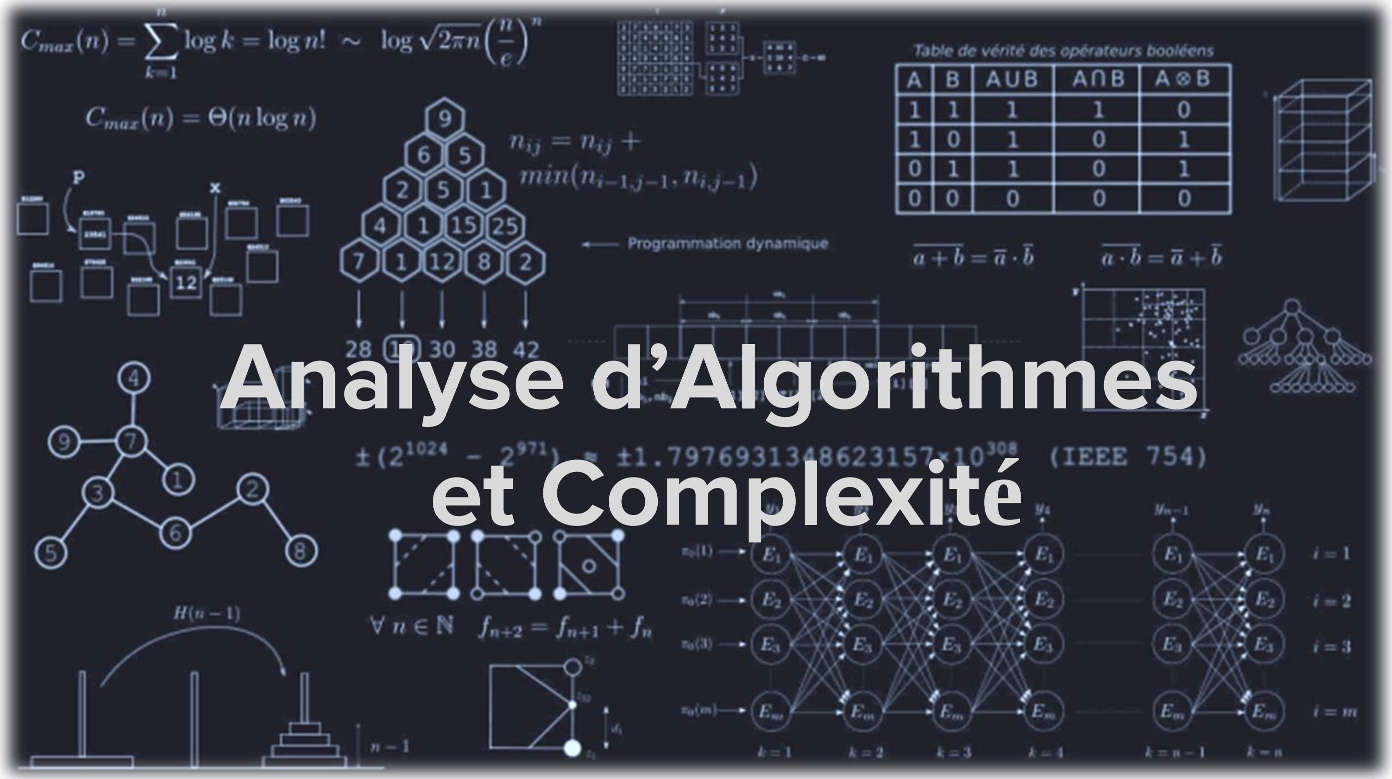

Analyse d’Algorithmes et Complexité

Pr Bekkari

Année universitaire 2021 - 2022

Chapitre 2 : Complexité

Exercice

- Déterminer la complexité de l'algorithme suivant (par rapport au nombre d'itérations effectuées), où m et n sont deux entiers positives.

$i \leftarrow 1\,;\, j \leftarrow 1$

tant que $j \leq n$ **faire**

  si $i \leq m$ **alors** $i \leftarrow i + 1$

  sinon $j \leftarrow j + 1 \quad i \leftarrow 1$

  fin si

fin tant que

O(n*m)

Exercice

- Déterminer la complexité de l'algorithme suivant où n est un entier positif.

```
void algo(int n, int tab[]){
int i = 0, j = 0;
for(; i < n; ++i)
        while(j < n && tab[i] < tab[j])
                j++;
}
```

O(n)

Exercice

- Déterminer la complexité de l'algorithme suivant où n est un entier positif.

```
int compteur (int n){
int c = 0;
for(int i = n; i > 0; i/= 2)
    for(int j = 0; j < i; j++)
        c += 1;
return c;
}
```

Comme « n » est un nombre entier en entrée, la déclaration de compteur() est exécutée comme suite:

n + n / 2 + n / 4 + … 1

Donc, la complexité temporelle T(n) peut être écrite comme suite:

T (n) = O (n + n / 2 + n / 4 + … 1) = O (n)

$=\sum_{i=0}^{logn-1} \frac{n}{2^i}$ $=n(\sum_{i=0}^{logn-1}(\frac{1}{2})^i)$ $=n(\frac{\left(\frac{1}{2}\right)^{logn}-1}{\frac{1}{2}-1})$ $=n(2(1-1/2^{logn})$

Or $2^{logn}=n$

$=2n-2$

O(n)

Exercice

- Déterminer la complexité de l'algorithme suivant où n est un entier positif.

```
int a = 0;
for (i = 0; i < n; i++) {
    for (j = n; j > i; j--) {
        a = a + i + j;
    }
}
```

= n + (n – 1) + (n – 2) + … 1 + 0
= n * (n + 1) / 2
= 1/2 * n^2 + 1/2 * n
O(n^2)

Exercice

- Déterminer la complexité de l'algorithme suivant où où m et n sont deux entiers positives

```
int a = 0, b = 0;
for (i = 0; i < n; i++) {
        a = a + rand();
}
for (j = 0; j < m; j++) {
        b = b + rand();
}
```

O(n+m)

Exercice

- Déterminer la complexité de l'algorithme suivant où n est un entier positif.

```
void function(int n)
{
        int i = 1, s =1;
        while (s <= n)
        {
                i++;
                s += i;
                printf("*");
        }
}
```

On peut définir les termes s selon la relation $s_i = s_{i-1} + i$.
La valeur de « i » augmente de un à chaque itération.
La valeur contenue dans s à la $i^{ème}$ itération est la somme des premiers 'i' entiers positifs.
Si k est le nombre total d'itérations effectuées par le programme, alors la boucle while se termine si :
$1 + 2 + 3 \ldots + k = [k(k+1)/2] > n$
D'où $k \in O(\sqrt{n})$.
Complexité temporelle de la fonction est $O(\sqrt{n})$.

Exercice

- Déterminer la complexité de la fonction T(n) suivante où n est un entier positif.

$T(n)=3T(n-1)$ si $n>0$,
$T(n) = 1$ sinon

$$
\begin{aligned}
T(n) &= 3T(n-1) \\
&= 3(3T(n-2)) \\
&= 3^2T(n-2) \\
&= 3^3T(n-3) \\
&\;\;\ldots \\
&\ldots \\
&= 3^nT(n-n) \\
&= 3^nT(0) \\
&= 3^n
\end{aligned}
$$

$T(n) \in O(3^n)$.

Exercice

- Déterminer la complexité de la fonction T(n) suivante où n est un entier positif.

T(n)=2T(n-1) - 1 si n>0,
T(n) = 1 sinon

$$
\begin{aligned}
T(n) &= 2T(n-1) - 1 \\
&= 2(2T(n-2)-1)-1 \\
&= 2^2(T(n-2)) - 2 - 1 \\
&= 2^2(2T(n-3)-1) - 2 - 1 \\
&= 2^3T(n-3) - 2^2 - 2^1 - 2^0 \\
&\ldots\ldots \\
&\ldots\ldots \\
&= 2^nT(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots\ldots 2^2 - 2^1 - 2^0 \\
&= 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots\ldots 2^2 - 2^1 - 2^0 \\
&= 2^n - (2^n-1)
\end{aligned}
$$

[**Note:** $2^{n-1} + 2^{n-2} + \ldots\ldots + 2^0 = 2^n - 1$]

D’où $T(n) = 1$

$T(n) \in O(1)$.

Exercice

- Déterminer la complexité de la fonction T(n) suivante où n est un entier positif.

T(n) = 2*T(n-2) +1

$T(n) = 2T(n-2) + 1$

➔ $r^2 - 2 = 0$

➔ $r = \pm\sqrt{2}$

La solution est donc de forme exponentielle

$T(n) = c_1 * (\sqrt{2})^n + c_2 * (-\sqrt{2})^n$

$T(n) = c_1 * 2^{n/2} + c_2 * (-1)^n * 2^{n/2}$

$T(n) \in O(2^{n/2})$.

Exercice

- Déterminer la complexité des fonctions T(n) suivantes où n est un entier positif.

$$T(n) = a \times T(n/b) + f(n)$$

avec $a \geq 1$, $b > 1$ et $f(n)$ est positive asymptotiquement.

$$T(n) = \begin{cases} O(n^d) & \text{si } d > \log_b a \\ O(n^d \log n) & \text{si } d = \log_b a \\ O(n^{\log_b a}) & \text{si } d < \log_b a \end{cases}$$

$$T(n) = 3T(n/2) + n^2$$

$$T(n) = 16T(n/4) + n$$

$$T(n) = 2T(n/4) + n^{0.51}$$

$$T(n) = 3T(n/3) + n/2$$

$$T(n) = 3T(n/2) + n$$