

## TD Invariants n°1

Calcul de complexité

## I. Avec des boucles

Dans cet exercice, on demande de compter le nombre d'appels à la fonction  $f$  dans les algorithmes ci-dessous. On l'exprimera sous forme *exacte*, puis on donnera un grand O plus simple.

On notera  $F(n)$  le nombre d'appels à la fonction  $f$ , en fonction de  $n$ .

```

f()
i ← 0
Tant que i ≤ n faire
  | f()
  | i ← i + 1
Fin tant que

```

**RÉPONSE.** Pour cet algorithme, le nombre d'appels à  $f$  s'écrit :

$$F(n) = 1 + \sum_{i=1}^n 1 = n + 2 = O(n)$$

▷ Résolu par Raminia

**Fig. 1.** Premier algorithme

```

i ← 0
Tant que i2 ≤ n faire
  | f()
  | f()
  | Pour j ∈ [1, n] faire
  | | f()
  | Fin pour
  | i ← i + 1
Fin tant que

```

**RÉPONSE.** Pour cet algorithme, le nombre d'appels à  $f$  s'écrit :

$$F(n) = \sum_{i=0}^m (2 + n) = (2 + n)m = O(n\sqrt{n})$$

où  $m = \lfloor \sqrt{n} \rfloor$ . En effet,  $i^2 \leq n$  ssi  $i \leq \sqrt{n}$  ssi  $i \leq \lfloor \sqrt{n} \rfloor$  car  $i$  et  $n$  sont des entiers.

▷ Résolu par Simon P.

**Fig. 2.** Second algorithme

```

i ← 0
Tant que 2i < n faire
  | j ← i
  | Tant que j < n faire
  | | f() j ← j + 1
  | Fin tant que
  | i ← i + 1
Fin tant que

```

**RÉPONSE.** Pour cet algorithme, le nombre d'appels à  $f$  s'écrit :

$$\begin{aligned}
 F(n) &= \sum_{i=0}^{m-1} \sum_{j=i}^{n-1} 1 \\
 &= \sum_{i=0}^{m-1} (n - i) \\
 &= mn - \frac{m(m-1)}{2} \\
 &= O(n \log n)
 \end{aligned}$$

**Fig. 3.** Troisième algorithme

où  $m = \lfloor \log_2 n \rfloor$ . En effet,  $2^i < n$  ssi  $i < \log_2 n$  ssi  $i \leq \lfloor \log_2 n \rfloor + 1$  car  $i$  et  $n$  sont des entiers.

▷ Résolu par Raminia

```

i ← n
Tant que i > 0 faire
  j ← 0
  f()
  Tant que j < n faire
    f()
    k ← j
    Tant que k < n faire
      f()
      k ← k + 1
    Fin tant que
    j ← j + 1
  Fin tant que
  i ← ⌊i/2⌋
Fin tant que

```

**RÉPONSE.** Pour cet algorithme, le nombre d'appels à  $f$  s'écrit :

$$\begin{aligned}
F(n) &= \sum_{i=1}^m \left[ 1 + \sum_{j=0}^{n-1} \left( 1 + \sum_{k=j}^{n-1} 1 \right) \right] \\
&= \sum_{i=1}^m \left[ 1 + \sum_{j=0}^{n-1} (1 + n - j) \right] \\
&= m + \sum_{j=0}^{n-1} m(1 + n - j) \\
&= m + mn + m \frac{n(n+1)}{2} \\
&= O(n^2 \log n)
\end{aligned}$$

où  $m = \lceil \log_2 n \rceil$ . En effet, on peut réaliser le changement de variables  $i = 2^p$ , et l'opération  $i \leftarrow \lfloor i/2 \rfloor$  devient  $p \leftarrow p - 1$ . De plus,  $\lfloor i \rfloor > 0$  ssi  $p > 0$ . Initialement,  $i = n$  et donc  $p \leq \lceil \log_2 n \rceil$ .

**Fig. 4.** Quatrième algorithme

▷ Résolu par Ewen

## II. Avec de la récursivité

Dans cet exercice, on demande de compter le nombre d'appels à la fonction  $f$  dans les algorithmes ci-dessous, pour *chacune* des fonctions ( $g$ ,  $h$ , et  $k$ ). On l'exprimera uniquement sous forme d'un grand O simple.

**Remarques.** Ces algorithmes ne terminent pas forcément mais, passé un certain instant, les appels à la fonction  $f$  s'arrêtent. Attention, la variable  $n$  n'est pas forcément un entier.

On notera  $G(n)$ ,  $H(n)$  et  $K(n)$  le nombre d'appels de la fonction  $f$ , lors de l'appel de  $g(n)$ ,  $h(n)$  et  $k(n)$ .

```

Procédure g(n)
  i ← 1
  Tant que i < n faire
    f()
    i ← i + 2
  Fin tant que
  g(n/2)
  g(n/2)
  g(n/2)

```

**RÉPONSE.** On détermine la relation de récurrence :

$$G(n) = 3 \cdot G\left(\frac{n}{2}\right) + \left\lfloor \frac{n}{2} \right\rfloor$$

Ainsi,  $G(2^p) = 3G(2^{p-1}) + 2(p-1)$ . On crée donc la suite  $(v_p)_{p \in \mathbb{N}}$ , où  $v_p = G(2^p)$  quel que soit  $p$ . On a donc  $v_p = 3v_{p-1} + 2^{p-1}$ . On en déduit que  $v_p = O(3^p)$ . Ceci nous permet de conclure que  $G(n) = O(n^{\log_2 3})$  car  $3^{\log_2 n} = e^{(\ln 3)(\ln n)/(\ln 2)} = n^{\log_2 3}$ .

▷ Résolu par Ewen & Carla

**Fig. 5.** Premier algorithme

**Procédure**  $h(n)$  $i \leftarrow 1$ **Tant que**  $i < n$  **faire** $f()$  $i \leftarrow i + 1$ **Fin tant que** $k(n)$  $k(n)$ **Procédure**  $k(n)$  $i \leftarrow 2$ **Tant que**  $i < n$  **faire** $f()$  $i \leftarrow i^2$ **Fin tant que** $h(n/2)$ **RÉPONSE.** On obtient les relations de récurrence :

$$\begin{cases} H(n) = n - 1 + 2 \cdot K(n) \\ K(n) = H\left(\frac{n}{2}\right) + \lceil \log_2 \log_2(n-1) \rceil + 1 \end{cases}$$

En effet, dans la procédure  $k$ , on peut réaliser un changement de variables  $i = 2^{(2^p)}$ . L'opération  $i \leftarrow i^2$  devient  $p \leftarrow p + 1$ , et on a  $i < n$  ssi  $i \leq n - 1$  ssi  $2^p \leq \log_2(n - 1)$  ssi  $p \leq \log_2 \log_2(n - 1)$ .

Ensuite, on calcule

$$\begin{aligned} H(n) &= n + 2H\left(\frac{n}{2}\right) + 2\lceil \log_2 \log_2(n-1) \rceil \\ &\leq 3n + H\left(\frac{n}{2}\right). \end{aligned}$$

Comme pour l'exercice précédent, on pose  $v_p = H(2^p)$ , on trouve  $v_p = v_{p-1} + 3 \cdot 2^{p-1}$ . Ainsi,  $v_p = 3 \sum_{i=0}^{p-1} 2^i = O(p2^p)$ . On en déduit que  $G(n) = O(n \log n)$ , par croissance de  $H(n)$ .

De plus,  $K(n) \leq H(n/2) + O(\log \log n)$ , et donc  $K(n) = O(n \log n)$ .**Fig. 6.** Second algorithme

▷ Résolu par Simon P. &amp; Raminia