Introduction

Dans les exercices précédents, vous avez découvert que compter le nombre exact d'opérations effectuées par un algorithme peut être très fastidieux (et quelque peu ennuyeux !). Il arrive qu'on souhaite réellement comptabiliser chaque opération mais, en général, on souhaite uniquement connaître une approximation suffisamment bonne pour pouvoir comparer des algorithmes entre eux. En effet si deux algorithmes calculent la même chose on veut savoir si l'un des deux est vraiment plus rapide que l'autre, pas qu'il fait quelques multiplications en moins !

On va donc calculer une approximation du nombre d'opérations effectuées par un algorithme, ce qui nous permettra de classifier les algorithmes en "catégories" selon ce qu'on appelle leur *complexité*. On pourra alors très rapidement comparer deux algorithmes entre eux mais également prédire comment va évoluer leur temps de calcul en fonction de la taille des données lues !

Regardons quelques exemples de la vie courante :

- Quand vous achetez un DVD le prix est le même quelque soit le nombre de fois où vous allez le regarder. On dit que le coût est *constant*.
- Quand vous achetez des bonbons au poids et que vous doublez le poids acheté, alors vous doublez aussi le prix. On dit que le coût est *linéaire* en fonction du poids acheté.
- Quand vous achetez une pizza carrée et que vous doublez la longueur de son côté (pizza de 50cm x 50cm au lieu de 25cm x 25 cm), alors vous avez multiplié par 4 la quantité de pizza disponible et donc multiplié par 4 le prix. On dit que le coût est *quadratique* en fonction de la longueur du côté de la pizza.

Pour les algorithmes, on va utiliser les mêmes catégories (constant, linéaire,...) pour expliquer comment le temps de calcul va augmenter si on augmente la taille des données. Pour des données de grande taille, un algorithme quadratique sera forcément plus lent qu'un algorithme linéaire, qui sera lui même plus lent qu'un algorithme constant.

Il existe bien sur d'autres catégories que celles-ci, vous les découvrirez au fur et à mesure des exercices.

Faire une approximation

Supposons qu'un algorithme lise la valeur d'un entier *nbEleves* et que le nombre d'opérations qu'il fasse soit *2nbEleves+5* : quand *nbEleves* devient très grand (exemple, nbEleves=10 000) le "+5" devient négligeable car le nombre d'opérations vaut déjà 20 000 donc le "+5" ne va pas changer grand chose ! Une bonne approximation du nombre d'opérations est donc simplement *2nbEleves*.

D'une manière générale le nombre d'opérations efectuées par un algorithme est constitué de différents termes (sur l'exemple ci-dessus, on a "2nbEleves" et "5") et on va conserver le terme le plus "gros" (quand *nbEleves* est grand) en oubliant les termes négligeables. Ainsi :

- Un terme constant (par exemple "5") est négligeable par rapport à un terme proportionnel à N (par exemple "2nbEleves").
- Un terme proportionnel à *nbEleves* (par exemple "50nbEleves") est négligeable par rapport à un terme proportionnel à $nbEleves^2$ (par exemple "2nbEleves^2").
- Un terme proportionnel à nbEleves^2 (par exemple "3nbEleves^2") est négligeable par rapport à un terme proportionnel à nbEleves^3 (par exemple "10nbEleves^3").
- ...

Exercice 1

Dans chacun des cas ci-dessous, éliminez les termes qui deviennent négligeables quand la (ou les) variables utilisées deviennent grandes. Attention si deux termes ne peuvent pas être comparés entre eux, il faut les garder tous les deux.

1. Nombre d'opérations de $2N + 5$ en fonction de la variable N.
2. Nombre d'opérations de $50N + N^2$ en fonction de la variable N.
3. Nombre d'opérations de $6N^3 + N + 100 + 3N^2$ en fonction de la variable N.
4. Nombre d'opérations de $2N^2 + 5N + 3N^2 + 8$ en fonction de la variable N.
5. Nombre d'opérations de $50N + N\sqrt{N}$ en fonction de la variable N.

Afficher/cacher la solution

Solution de l'exercice :

On compare les termes entre eux, en éliminant ceux qui sont négligeables par rapport à d'autres.

1. Le terme 5 est négligeable par rapport à $2N$. Il ne reste donc que $2N$.
2. Le terme $50N$ est négligeable par rapport à N^2. Il ne reste donc que N^2.
3. Les termes 100, N, et $3N^2$ sont négligeables par rapport à $6N^3$. Il ne reste donc que $6N^3$.
4. Les termes $5N$ et 8 sont négligeables par rapport à $2N^2$ donc on les enlève. Il reste donc les termes $2N^2$ et $3N^2$ soit $5N^2$ au total.
5. Le terme $50N$ est négligeable par rapport à $N\sqrt{N}$. Il ne reste donc que $N\sqrt{N}$.

Exercice 2

Même consigne que l'exercice précédent.

1. Nombre d'opérations de $N + P$ en fonction des variables N et P.
2. Nombre d'opérations de $Q^2 + Q + P^3$ en fonction des variables Q et P.
3. Nombre d'opérations de $QP + P + Q$ en fonction des variables P et Q.
4. Nombre d'opérations de $Q^2 + 2PQ + P^2$ en fonction des variables P et Q.

Afficher/cacher la solution

Solution de l'exercice :

1. Les deux termes N et P ne sont pas comparables, on les garde donc tous les deux. On obtient donc $N + P$.
2. Le terme Q est négligeable par rapport à Q^2. Les deux termes Q^2 et P^3 ne sont pas comparables, on les garde donc tous les deux. On obtient donc $Q^2 + P^3$.
3. Les termes P et Q sont négligeables par rapport à QP. Il ne reste donc que QP.
4. Aucun terme n'est négligeable par rapport aux autres, on les garde donc tous. On peut cependant écrire ce nombre d'opérations sous la forme simplifiée $(P + Q)^2$.

Déterminer la complexité

Une fois qu'on a conservé uniquement le terme (ou les termes) qui n'est pas négligeable, on regarde à quoi il est proportionnel, ce qui donne la complexité de l'algorithme. Par exemple pour un nombre d'opérations valant *2nbEleves+5*, on ne garde que *2nbEleves* ce qui est proportionnel à *nbEleves*. On dit que l'algorithme correspondant à une complexité proportionnelle à *nbEleves* : le nombre d'opérations double (avec l'approximation que nous avons fait) si on double la valeur de *nbEleves*.

Il existe certaines complexités pour lesquelles on a un nom, ce qui nous évite de toujours répéter "proportionnel à ...". Pour un algorithme dépendant d'une valeur N :

- si la complexité ne dépend pas de N, alors elle est dite *constante*,
- si la complexité est proportionnelle à N, alors elle est dite *linéaire* en N,
- si la complexité est proportionnelle à N^2, alors elle est dite *quadratique* en N,
- si la complexité est proportionnelle à N^3, alors elle est dite *cubique* en N.

Voilà les 4 complexités que vous rencontrerez le plus souvent dans les exercices. Il en existe d'autres mais qui n'ont pas forcément de "nom", on indiquera alors simplement par rapport à quoi la complexité est proportionnelle. Par exemple, une complexité peut être proportionnelle à N^6 !

Pour chaque exercice, lisez bien l'algorithme proposé et indiquez ce qu'il va faire/calculer/afficher. Indiquez alors sa complexité en fonction des variables pertinentes. Notez que vous n'avez pas besoin de calculer le nombre exact d'opérations pour déterminer la complexité.

Exercice 3

```
nbLivres <- LireEntier()
Si nbLivres < 10
   prix <- nbLivres * 10
Sinon
   prix <- nbLivres * 9
Afficher prix
```

Afficher/cacher la solution

Solution de l'exercice :

L'algorithme a déjà été vu précédemment. Il a une complexité constante car il effectue toujours un nombre constant d'opérations.

Exercice 4

```
total <- 0
i <- 1
Tant que i <= 100
   total <- total + i
   i <- i + 1
Afficher total
```

Afficher/cacher la solution

Solution de l'exercice :

Là encore, l'algorithme a déjà été vu précédemment et il a également une complexité constante car il effectue toujours un nombre constant d'opérations.

Exercice 5

```
iMax <- LireEntier()
total <- 0
i <- 1
Tant que i <= iMax
   total <- total + i
   i <- i + 1
Afficher total
```

Afficher/cacher la solution

Solution de l'exercice :

L'algorithme a déjà été vu précédemment. Le nombre d'opérations est proportionnel à *iMax* (une seule boucle avec *i* allant de 1 à *iMax*) donc l'algorithme a une complexité linéaire en *iMax*.

Exercice 6

```
iMax <- LireEntier()
total <- 0
Tant que iMax > 0
   total <- total + iMax
   iMax <- iMax - 1
Afficher total
```

Afficher/cacher la solution

Solution de l'exercice :

L'algorithme a déjà été vu précédemment. Le nombre d'opérations est proportionnel à *iMax* (une seule boucle avec *iMax* diminuant de 1 à chaque itération, jusqu'à valoir 0) donc l'algorithme a une complexité linéaire en *iMax*.

Exercice 7

```
taille <- LireEntier()
X <- 0
Tant que X < taille
   Y <- 0
   Tant que Y < taille
      Si X = Y
         Afficher "X"
      Sinon
         Afficher "."
      Y <- Y + 1
   Retour à la ligne
   X <- X + 1
```

Afficher/cacher la solution

Solution de l'exercice :

L'algorithme a déjà été vu précédemment. On a deux boucles imbriquées sur les variables *X* et *Y* qui vont prendre les valeurs allant de 0 à *taille-1*. On a donc $taille^2$ couples de valeurs *(X, Y)* qui vont être testés à la ligne 6. Le nombre d'opérations est donc proportionnel à $taille^2$ donc l'algorithme a une complexité quadratique en *taille*.

Exercice 8

```
nbBonbons <- LireEntier()
nbBoites <- 0
Tant que nbBoites * 100 < nbBonbons
   nbBoites <- nbBoites + 1
Afficher nbBoites
```

Afficher/cacher la solution

Solution de l'exercice :

On a un gros paquet de bonbons qu'on souhaite ranger dans des boîtes et chaque boîte peut contenir 100 bonbons. Alors cet algorithme calcule le nombre de boîtes nécessaires pour un nombre de bonbons qu'on lui indique.

Le nombre d'opérations va être proportionnel au nombre de fois que la boucle "Tant que" est effectuée, c'est-à-dire au nombre de boîtes qu'il va afficher à la fin, c'est-à-dire à environ *nbBonbons/100*, ce qui est donc proportionnel à *nbBonbons*. L'algorithme a une complexité linéaire en *nbBonbons*.

Qu'on ait eu un facteur de "1/100" (comme ici) ou de "1/1000" ou de "1/1000000" dans tous les cas la complexité reste linéaire car le nombre d'opérations reste toujours proportionnel à *nbBonbons*.

Exercice 9

```
taille <- LireEntier()
Répéter taille fois
   Afficher "*"
Afficher un retour à la ligne
Répéter taille fois
   Répéter taille fois
      Répéter taille fois
         Afficher "X"
      Afficher un retour à la ligne
   Répéter taille fois
      Afficher "*"
   Afficher un retour à la ligne
```

Afficher/cacher la solution

Solution de l'exercice :

Essayons de regarder ce que cela donne pour une petite valeur de *taille*, disons 3 :

```
***
XXX
XXX
XXX
***
XXX
XXX
XXX
***
XXX
XXX
XXX
***
```

L'algorithme affiche donc *taille* carrés de "X", encadrés par des lignes de *taille* "*".

Pour la complexité, les affichages de "*" (et les retours à la ligne) sont négligeables par rapport aux affichages de "X" car si *taille* devient grand il y aura environ *taille* fois plus de "X" que de "*" (et que de retours à la ligne). En terme de complexité, on peut donc se ramener à la structure de programme suivante :

```
Répéter taille fois
   Répéter taille fois
      Répéter taille fois
```

```
        Afficher "X"
```

On a donc 3 boucles imbriquées ce qui donne un nombre total d'opérations proportionnel à *taille*taille*taille* soit $taille^3$. L'algorithme a donc une complexité cubique en *taille*.