# Fiche TD6 : Complexité

**Exercice 1** *Terme dominant*

Indiquer dans quelle classe de complexité asymptotique se trouve chacune des fonctions $T_i$ suivantes en utilisant la notation grand-O. Par exemple, si $T_0(n) = 3n$, on attend la réponse suivante : $T_0(n) = O(n)$ (remarquez que cette notation, bien que classique, est abusive).

1. $T_1 : n \mapsto 3n + 8$.
2. $T_2 : n \mapsto 6n^3 + n^2 + 5$.
3. $T_3 : n \mapsto 3\log_2(n) + 42$.
4. $T_4 : n \mapsto 6n^4 + 2^n + 7n$.
5. $T_5 : n \mapsto 7k + 4$ où $k$ est une constante positive.
6. $T_6 : n \mapsto 4\log_2(n) + 5n$
7. $T_7 : n \mapsto 2(\log_{10}(n))^8 + kn^2$ où $k$ est une constante positive.

**Exercice 2** *Ordres de grandeur*

On dispose d'une machine capable d'effectuer 2 milliards d'opérations en une seconde. On considère des algorithmes dont les complexités sont les suivantes : $\log(n)$, $\sqrt{n}$, $n + 1000000$, $20\,n$, $n\log(n)$, $n^2$, $n^3$ et $2^n$.

1. Calculer le temps nécessaire à l'exécution de l'algorithme sur des données de taille $10^3$, $10^6$ et $10^9$.
2. En supposant que la taille des données à traiter dans chaque cas est de l'ordre de $10^9$, lesquels de ces algorithmes vous semblent utilisables à chaque chargement d'une page web ? Et à chaque démarrage d'une machine ? Et à la mise en place de l'infrastructure informatique d'un lycée ?

**Exercice 3** *Relations de comparaison*

1. On dispose de trois algorithmes : l'un a une complexité en $\Theta(n^2\log(n))$, le deuxième en $\Theta(n(\log n)^4)$ et le dernier en $\Theta(1.5^n)$. Lequel préférer si on souhaite l'utiliser sur des données de grande taille ?
2. a) La complexité d'un algorithme peut-elle être à la fois en $O(n^2)$ et en $O(n\log(n))$ ?
   b) La complexité d'un algorithme peut-elle être à la fois en $\Omega(n^2)$ et en $O(n\log(n))$ ?
   c) La complexité d'un algorithme peut-elle être à la fois en $\Theta(n^2)$ et en $\Omega(n\log(n))$ ?
3. Soit $f, g, h$ des fonctions positives.
   a) Montrer que si $f \in O(g)$ et $g \in \Theta(h)$, alors $f \in O(h)$.
   b) Montrer que si $f \in \Theta(g)$ et $g \in O(h)$, alors $f \in O(h)$.
   c) Est-il vrai que si $f \in \Theta(g)$ et $g \in O(h)$, alors $f \in \Theta(h)$ ?

**Exercice 4** *Complexité non asymptotique*

On considère deux algorithmes, $A_1$ et $A_2$ dont les temps d'exécution maximaux sur une entrée de taille $n$ sont respectivement $T_1(n) = 9n^2$ et $T_2(n) = 100n + 96$.

1. Déterminer la complexité asymptotique de ces deux algorithmes en utilisant la notation grand-O.
2. Quel algorithme a la meilleure complexité asymptotique ?
3. Pour une entrée de taille 5, quel algorithme est-il préférable d'utiliser ?
4. Pour quelles tailles d'entrée l'algorithme $A_1$ s'exécute-t-il en moins de temps que l'algorithme $A_2$ ?

**Exercice 5** *Invariants ?*

On propose les quatre algorithmes suivants pour calculer la factorielle de l'entier $n$ passé en argument.

```
facto1(n) =
  res ← 1
  i ← 0
  Tant que i ≤ n
    i ← i + 1
    res ← res × i
  Renvoyer res
```

```
facto2(n) =
  res ← 1
  i ← 0
  Tant que i < n
    i ← i + 1
    res ← res × i
  Renvoyer res
```

```
facto3(n) =
  res ← 1
  i ← 1
  Tant que i ≤ n
    res ← res × i
    i ← i + 1
  Renvoyer res
```

```
facto4(n) =
  res ← 1
  i ← 0
  Tant que i < n
    res ← res × i
    i ← i + 1
  Renvoyer res
```

On note $P$ la propriété suivante : res = i! Dans chacun des quatre cas précédents, répondre aux questions suivantes :

1. La propriété $P$ est-elle un invariant pour la boucle tant que de l'algorithme ?
2. L'algorithme permet-il le calcul de la factorielle de $n$ ?
3. a) En estimant que les seules opérations pertinentes dans cet algorithme sont les multiplications, sommes et affectations, quelle est la complexité du deuxième algorithme ?
   b) En estimant que les seules opérations pertinentes dans cet algorithme sont les multiplications et les sommes, quelle est la complexité du deuxième algorithme ? Qu'en penser ?

**Exercice 6** *Itérations imbriquées*

On considère les algorithmes suivants, dont l'entrée est un entier $n$ :

```
1. S1(n) =
2.   Pour i = 1 à n
3.     Pour j = 1 à n
4.       Schtroumpfer()
```

```
1. S2(n) =
2.   Pour i = 1 à n
3.     Pour j = 1 à i
4.       Schtroumpfer()
```

```
1. S3(n) =
2.   Pour i = 5 à n-5
3.     Pour j = i-5 à i+5
4.       Schtroumpfer()
```

```
1. S4(n) =
2.   Pour i = 1 à n
3.     Pour j = 1 à i
4.       Pour k = 1 à j
5.         Schtroumpfer()
```

1. Pour chacun de ces algorithmes :
   a) Compter précisément le nombre d'opérations *Schtroumpfer* exécutées en fonction de $n$.
   b) Si l'opération *Schtroumpfer* est une opération élémentaire, en déduire la complexité asymptotique de l'algorithme en question.
2. Quelle est la complexité asymptotique des algorithmes suivants ?

```
If(n) =
  Si n est pair
    S1(n)
  Sinon
    S3(n)
```

```
Seq(n) =
  S1(n)
  S2(n)
  S3(n)
  S4(n)
```

**Exercice 7** *Pire cas, meilleur cas*

On cherche à construire un algorithme permettant de tester la présence de l'élément $x$ dans un tableau $T$.

1. Spécifier et écrire un tel algorithme.
2. Quelle est la taille de l'entrée de cet algorithme ?
3. Quelle est la complexité asymptotique au pire cas de cet algorithme ?
4. Déterminer les cas favorables et défavorables (en termes de temps d'exécution) pour cet algorithme et dans chaque cas le nombre de comparaisons effectuées.