Etude d'algorithmes

1. Les concepts

(a) Terminaison

Il s'agit de justifier que l'algorithme se termine bien. La question est véritablement posée lors de l'utilisation d'une boucle **`while`** pour laquelle il y a un risque de boucle infinie.

(b) Complexité

Il s'agit de donner un ordre de grandeur du nombre d'opérations effectuées.

(c) Correction

Il s'agit de prouver que l'algorithme retourne bien le résultat attendu.
Pour cela, on utilise un invariant de boucle. C'est une propriété qui
— est vraie avant la boucle,
— est vraie après chaque itération.

2. Maximum d'une liste + indice

(a) Algorithme

```
Entrée : L  #liste contenant au moins 2 nombres
indice ⟵ 0
maximum ⟵ L[0]
n ⟵ taille de L
POUR i allant de 1 à n-1 FAIRE
   SI L[i]>maximum ALORS
      indice ⟵ i
      maximum ⟵ L[i]
Sortie : indice et maximum
```

(b) Terminaison

La boucle est une boucle **for**, il y a exactement $n-1$ itérations. L'algorithme se termine bien.

(c) Complexité

```
indice ⟵ 0          [3 opérations]
maximum ⟵ L[0]
n ⟵ taille de L
POUR i allant de 1 à n-1 FAIRE   [n-1 itérations]
   SI L[i]>maximum ALORS      [1 à 3 opérations]
      indice ⟵ i
      maximum ⟵ L[i]
```

3 opérations avant la boucle **for** (3 affectations) La boucle effectue $n-1$ itérations. Pour chacune d'elle, 1 à 3 opérations sont effectuées : un test + éventuellement 2 affectations.

Donc

$$3 + (n-1) \times 1 \leq c(n) \leq 3 + (n-1) \times 3$$

donc

$$n + 2 \leq c(n) \leq 3n$$

Finalement,

$$\boxed{c(n) = O(n)}$$

(d) Invariant de boucle

Notons $\mathcal{P}(k)$ la propriété suivante :
Après la k-ième itération,

i. la valeur de **`maximum`** est égale au maximum de ℓ=`[L[0],...,L[k]]`,
ii. la valeur de **`indice`** est égale à l'indice de la première apparition du maximum dans ℓ.

Montrons que la propriété $\mathcal{P}(k)$ est un invariant de boucle

Démonstration :

- Avant la boucle **`for`**, `L[0]` est bien le maximum de la liste `[L[0]]` et 0 est bien la première apparition de `L[0]`. $\mathcal{P}(0)$ est donc vraie avant la boucle.

- Soit $k \in [\![0, n-1]\!]$, supposons $\mathcal{P}(k)$ vraie.
  Donc **`maximum`** est égale au maximum de la liste `[L[0],...,L[k]]` et que **`indice`** est égale à l'indice de la première apparition du maximum.

  Si `L[k+1]>maximum` alors `L[k+1]` est supérieur (strictement) à `L[0],...,L[k]`.
  Après la k+1-ième itération, **`maximum`** sera égale à `L[k+1]` et sera donc bien égale au maximum de la liste `[L[0],...,L[k+1]]`,
  `indice` sera égale à `k+1` et sera égale à l'indice de la première apparition du maximum.

  Si `L[k+1]` $\leq$ `maximum` alors il ne se passer rien. Après la k+1-ième itération, **`maximum`** reste bien égale au maximum de la liste `[L[0],...,L[k+1]]`,
  `indice` reste bien égale à l'indice de la première apparition du maximum.
  Donc $\mathcal{P}(k+1)$ vraie.

(e) Correction

Après la boucle, $\mathcal{P}(n)$ est donc vraie. Ainsi, La valeur de **`maximum`** est égale à au maximum de la liste `L` et la valeur de **`indice`** est égale à l'indice de la première apparition de ce maximum.

3. Factorielle

(a) Algorithme

```
Entrée : n  #un entier naturel
prod ⟵ 1
POUR i allant de 1 à n FAIRE
   prod ⟵ prod × i
Sortie : prod
```

(b) Terminaison

La boucle est une boucle **`for`**, il y a exactement n itérations. L'algorithme se termine bien.

(c) Complexité

```
Entrée : n  #un entier naturel
prod ⟵ 1      [1 opération]
POUR i allant de 1 à n FAIRE   [n itérations]
   prod ⟵ prod × i      [2 opérations]
Sortie : prod
```

1 opération est effectuée avant la boucle **`for`** (1 affectation).

La boucle effectue n itérations. Pour chacune d'elle, 2 opérations sont effectuées : 1 produit + 1 affectation.

Donc

$$c(n) = 1 + 2 \times n = 2n + 1$$

Finalement,

$$\boxed{c(n) = O(n)}$$

(d) Invariant de boucle

Notons $\mathcal{P}(k)$ la propriété suivante :
Après la k-ième itération, la valeur de **`prod`** est égale à $k!$

Montrons que la propriété $\mathcal{P}(k)$ est un invariant de boucle

Démonstration :

- Avant la boucle **`for`**, la valeur de **`prod`** est égale à 1 donc à 0 !
  donc $\mathcal{P}(0)$ est vraie avant la boucle.

- Soit $k \in [\![0, n-1]\!]$, supposons $\mathcal{P}(k)$ vraie. Et donc **`prod`** est égale à k !

  Après la k+1-ième itération, **`prod`** est égale à $k! \times (k+1) = (k+1)!$
  Donc $\mathcal{P}(k+1)$ vraie.

(e) Correction

Après la boucle, $\mathcal{P}(n)$ est donc vraie. Ainsi, La valeur de **`prod`** est égale à $n!$

4. Fibonacci

(F_n) est la suite définie par $F_0 = 0$, $F_1 = 1$ et pour tout $n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.

(a) Algorithme

```
Entrée : n  #un entier naturel non nul
A ⟵ 0
B ⟵ 1
POUR i allant de 1 à n-1 FAIRE
   A , B ⟵ B , A+B
Sortie : B
```

(b) Terminaison

La boucle est une boucle **`for`**, il y a exactement $n - 1$ itérations. L'algorithme se termine bien.

(c) Complexité

```
Entrée : n  #un entier naturel non nul
A ⟵ 0      [2 opérations]
B ⟵ 1
POUR i allant de 2 à n FAIRE [n-1 itérations]
   A , B ⟵ B , A+B      [3 opérations]
Sortie : B
```

2 opérations sont effectuée avant la boucle **`for`** (2 affectations).

La boucle effectue $n - 1$ itérations. Pour chacune d'elle, 3 opérations sont effectuées : 1 somme + 2 affectations.

Donc

$$c(n) = 2 + 3 \times (n - 1) = 3n - 1$$

Finalement,

$$\boxed{c(n) = O(n)}$$

(d) Invariant de boucle

Notons $\mathcal{P}(k)$ la propriété suivante :
Après l'itération $n^o\, k$,
— la valeur de **A** est égale à F_{k-1},
— la valeur de **B** est égale à F_k

Montrons que la propriété $\mathcal{P}(k)$ est un invariant de boucle

Démonstration :

- Avant la boucle **for**, la valeur de **A** est égale à 0 donc à F_0 et la valeur de **B** est égale à 1 donc à F_1.
  donc $\mathcal{P}(1)$ est vraie avant la boucle.

- Soit $k \in [\![1, n]\!]$, supposons $\mathcal{P}(k)$ vraie.
  Et donc **A** est égale à F_{k-1} et **B** est égale à F_k

  Après l'itération $n^o\, k+1$,
  — **A** est égale à $B = F_k$,
  — **B** est égale à $A + B = F_{k-1} + F_k = F_{k+1}$
  Donc $\mathcal{P}(k+1)$ vraie.

(e) Correction

Après la boucle, $\mathcal{P}(n)$ est donc vraie. Ainsi, La valeur de **B** est égale à F_n

5. Recherche dichotomique dans une liste triée

(a) Algorithme

```
Entrées : L et x  #une liste triée et un réel

N ⟵ longueur de L
iMaj ⟵ N-1     # indice majorant
iMin ⟵ 0     # indice minorant
iMilieu ⟵ (iMaj-iMin)//2     # indice milieu

TANT QUE L[iMilieu] != x ET iMaj-iMin>0 FAIRE
   SI L[iMilieu] > x ALORS
      iMaj ⟵ iMilieu-1
   SINON
      iMin ⟵ iMilieu+1
   iMilieu ⟵ (iMaj-iMin)//2

SI iMaj-iMin>0 ALORS
   AFFICHER "x appartient à la liste"
   RETOURNER iMilieu
SINON
   AFFICHER "x n'appartient pas à la liste"
```

(b) Terminaison

Les valeurs successives de la variable `D=iMaj-iMin` sont positives et strictement décroissante via la commande `iMaj ⟵ iMilieu-1` ou `iMin ⟵ iMilieu+1`

Il y a donc un nombre fini d'itérations. L'algorithme se termine bien.

(c) Complexité

6 opérations sont effectuées avant la boucle **`while`** (4 affectations, 1 différence et 1 division).

Pour chacune des itérations, 4 opérations sont effectuées : 1 test, 1 somme (ou 1 différence) + 2 affectations.

Après la boucle, il y a 2 ou 3 opérations.

Combien y a t-il d'itérations dans la boucle **`while`** ?

D'une itération à l'autre, `iMaj-iMin < iMaj-iMin /2`

Dans le pire des cas, (lorsque la boucle **`while`** s'arrête lorsque `iMaj-iMin`≤ 0) il y a k itérations où k est le plus petit entier vérifiant

$$\frac{N}{2^k} < 1$$

or

$$\frac{N}{2^k} < 1 \iff k > \frac{\ln(N)}{\ln(2)} \iff k \geq \left\lfloor \frac{\ln(N)}{\ln(2)} \right\rfloor + 1$$

donc

$$c(n) \leq 6 + 4 \times \left(\left\lfloor \frac{\ln(N)}{\ln(2)} \right\rfloor + 1 \right) + 2$$

Finalement,

$$\boxed{c(n) = O\big(\ \ln(N)\ \big)}$$

(d) Invariant de boucle

La propriété suivante (notée $\mathcal{P}(k)$) est un invariant de boucle
Après l'itération $n^o\, k$,
— `iMaj-iMin` est strictement inférieur à $\frac{N}{2^k}$
— `L[iMin]`$\leq$`x`$\leq$`L[iMaj]` si x appartient à la liste.

6. Ecriture en base deux

(a) Algorithme

```
Entrée : n  #un entier naturel non nul
D ⟵ n    # Dividende
P ⟵ 1    # puissance de 10 successives
E ⟵ 0    # Ecriture en base deux
TANT QUE D > 0 FAIRE
   E ⟵ E + (D % 2) * P
   D ⟵ (D // 2)
   P ⟵ P * 10
Sortie : E
```

(b) Terminaison

Les valeurs successives de la variable D sont positives et strictement décroissante via la commande `D ⟵ (D // 2)` . Il y a donc un nombre fini d'itérations. L'algorithme se termine bien.

(c) Complexité

3 opérations sont effectuées avant la boucle **`while`** (3 affectations).

Pour chacune des itérations, 8 opérations sont effectuées : 1 somme + 2 produits + 2 division euclidienne + 3 affectations.

Combien y a t-il d'itérations dans la boucle **`while`** ?

Il y en a autant qu'il y a de chiffres dans l'écriture en base deux de n. Il y a p chiffres dans cette écriture où p est le plus petit entier vérifiant $2^p > n$

Donc p vérifie

$$2^p > n \geq 2^{p-1}$$

or

$$2^p > n \geq 2^{p-1} \iff 1 + \frac{\ln(n)}{\ln(2)} \geq p > \frac{\ln(n)}{\ln(2)}$$

donc

$$3 + 8 \times \frac{\ln(n)}{\ln(2)} < c(n) \leq 3 + 8 \times \left(1 + \frac{\ln(n)}{\ln(2)}\right)$$

Finalement,

$$\boxed{c(n) = O\big(\ln(n)\big)}$$

(d) Invariant de boucle

La propriété suivante (notée $\mathcal{P}(k)$) est un invariant de boucle
Après l'itération $n^o\, k$,
— `D` est égale au quotient de la division euclidienne de n par 2^k
— `P` est égale à 10^k
— `E` est égale à $\sum_{i=0}^{k-1} a_i\, 2^i$ où $a_p a_{p-1} ... a_k ... a_0$ est l'écriture binaire de n

(e) Correction

Après la boucle, $\mathcal{P}(p)$ est donc vraie. Ainsi, La valeur de `E` est égale à $a_p a_{p-1} ... a_0$

7. Exponentiation rapide

(a) Algorithme

```
Entrée : n et a  # n ∈ ℕ* et a ∈ ℝ
D ⟵ n    # Dividende
C ⟵ a    # carrés successifs de a
prod ⟵ 1    # produit
TANT QUE D > 0 FAIRE
   SI D % 2 = 1 ALORS
      prod ⟵ prod * C
   D ⟵ (D // 2)
   C ⟵ C**2
Sortie : E
```

(b) Terminaison

Les valeurs successives de la variable D sont positives et strictement décroissante via la commande `D ⟵ (D // 2)` . Il y a donc un nombre fini d'itérations. L'algorithme se termine bien.

(c) Complexité

3 opérations sont effectuées avant la boucle **`while`** (3 affectations).

Pour chacune des itérations, 1 test et éventuellement 4 opérations sont effectuées : 1 produit + 1 division euclidienne + 2 affectations.

Combien y a t-il d'itérations dans la boucle **`while`** ?

Il y en a autant qu'il y a de chiffres dans l'écriture en base deux de n. Il y a donc p opérations où p vérifie

$$1 + \frac{\ln(n)}{\ln(2)} \geq p > \frac{\ln(n)}{\ln(2)}$$

donc

$$3 + 4 \times \frac{\ln(n)}{\ln(2)} < c(n) \leq 3 + 4 \times \left(1 + \frac{\ln(n)}{\ln(2)}\right)$$

Finalement,

$$\boxed{c(n) = O\big(\ln(n)\big)}$$

(d) Invariant de boucle

La propriété suivante (notée $\mathcal{P}(k)$) est un invariant de boucle
Après l'itération $n^o\, k$,
— `D` est égale au quotient de la division euclidienne de n par 2^k
— `C` est égale à a^k
— `prod` est égale à $a^{\sum_{i=0}^{k-1} a_i\, 2^i}$ où $a_p a_{p-1} ... a_0$ est l'écriture binaire de n

(e) Correction

Après la boucle, $\mathcal{P}(p)$ est donc vraie. Ainsi, La valeur de `prod` est égale à $a^{\sum_{i=0}^{p-1} a_i\, 2^i} = a^n$