Algorithmique I
Semestre 2 – LU2IN003

Leo MONBROUSSOU – Faculté des Sciences et Ingénierie
13 septembre au 14 décembre 2022

Table des matières

Thaïs MILLERET – Mono Informatique G4

TD 1 : Rappels mathématiques

Exercice 1. Logique d’automne

On considère la proposition suivante : « A l’automne, s’il a plu pendant la nuit, je vais toujours ramasser des champignons ».

Question 1

Donnez cette proposition sous la forme logique $(\boldsymbol{a} \text{ et } \boldsymbol{b}) \Rightarrow \boldsymbol{c}$.

a : à l’automne

b : il a plu pendant la nuit

c : je vais toujours ramasser des champignons

Question 2

Donnez « en français » la contraposée de cette position.

$\neg c \Rightarrow \neg(a \wedge b)$ soit $\neg c \Rightarrow (\neg a \vee \neg b)$

Je ne vais pas ramasser des champignons s’il n’est pas l’automne ou s’il n’a pas plus pendant la nuit.

Question 3

Donnez « en français » la négation de cette proposition.

$(a \wedge b) \Rightarrow c \Leftrightarrow \neg(a \wedge b) \vee c$

Négation : $(a \wedge b) \wedge \neg c$

Exercice 3. Preuve par contraposée, preuve par l’absurde, réciproque

Question 1

Soit $n \in \mathbb{N}$.

1. Démontre par contraposée que, si n^2 est pair, alors n est pair.

n^2 est pair $\Rightarrow n$ est pair

Contraposée : n est impair $\Rightarrow n^2$ est impair

Si n est impair alors $\exists k \in \mathbb{N}$ tel que $n = 2k + 1$.

$n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$

Donc $n^2 = 2k' + 1$ avec $k' = 2k^2 + 2k$. Donc n^2 est impair.

2. Quelle est la réciproque ? Est-elle vérifiée ?

Réciproque : n est pair $\Rightarrow n^2$ est pair

Si n est pair alors $\exists k \in \mathbb{N}$ tel que $n = 2k$.

$n^2 = (2k)^2 = 4k^2 = 2 \times 2k^2$

Donc $n^2 = 2k'$ avec $k' = 2k^2$. Donc n^2 est pair.

3. Quelle est la réciproque de la contraposée ? Est-elle vérifiée ?

Réciproque de la contraposée : n^2 est impair $\Rightarrow n$ est impair

Vrai

Question 2

Démontrez par l'absurde que $\sqrt{2}$ est irrationnel.

Par l'absurde,

On suppose que $\sqrt{2}$ est rationnel donc il peut s'écrire sous la forme $\frac{a}{b}$ avec $a \in \mathbb{N}$ et $b \in \mathbb{N}^*$ et a et b sont premiers entre eux.

$$\sqrt{2} = \frac{a}{b} \Rightarrow \sqrt{2}^2 = \left(\frac{a}{b}\right)^2 \Leftrightarrow 2 = \frac{a^2}{b^2} \Leftrightarrow a^2 = 2b^2$$

$b \in \mathbb{N} \Rightarrow b^2 \in \mathbb{N}$ donc a^2 est pair. Donc, d'après la question 1, a est pair ($a = 2k$).

$(2k)^2 = 2b^2 \Leftrightarrow b^2 = 2k^2$ donc b^2 est pair. Donc, d'après la question 1, b est pair ($b = 2k'$).

$$\sqrt{2} = \frac{a}{b} = \frac{2k}{2k'}$$

a et b ne sont pas premiers entre eux donc $\sqrt{2}$ n'est pas rationnel.

Question 3

Soit $n > 0$. Démontrez par l'absurde que si n est le carré d'une entier, alors $2n$ n'est pas le carré d'un entier.

$P : (\exists k \in \mathbb{N} | n = k^2) \Rightarrow (\forall \ell \in \mathbb{N}\ 2n \neq \ell^2)$

Par l'absurde,

On suppose que $\exists k \in \mathbb{N}, \exists \ell \in \mathbb{N} \mid n = k^2 \wedge 2n = \ell^2$

(Inutile que $k \in \mathbb{Z}$ et $\ell \in \mathbb{Z}$ car on utilise $k^2 = |k|^2$ et $\ell^2 = |\ell|^2$.)

$$2n = \ell^2 \Leftrightarrow 2k^2 = \ell^2 \Leftrightarrow 2 = \frac{\ell^2}{k^2} \Leftrightarrow \sqrt{2} = \sqrt{\frac{\ell^2}{k^2}} \Leftrightarrow \sqrt{2} = \frac{\ell^2}{k^2}$$

($\ell^2 \geq 0$ et $k^2 \geq 0$ donc on a le droit d'appliquer la racine carrée.)

D'après la question 2, $\sqrt{2}$ est irrationnel donc $\sqrt{2} = \frac{\ell^2}{k^2}$ est faux. Donc P est vraie.

Exercice 4. Preuve par l'absurde

On considère n ensembles $E_1, \cdots, E_n$ d'entiers tels que ces ensembles soient distincts deux à deux. Montrez la propriété suivante :

P : « Au moins l'un des ensembles $E_1, \cdots, E_n$ ne contient aucun des $n - 1$ autres ensembles ».

Rappel : démonstration par l'absurde

On suppose que l'inverse de la proposition est vraie. En utilisant uniquement des équivalences dans le raisonnement, on prouve que l'inverse de la proposition est faux donc que la proposition de départ est vraie.

$i \in [n] \Leftrightarrow i \in [\![1, n]\!] \Leftrightarrow i \in \{1,2,\cdots,n\}$

n ensembles $E_1, \cdots, E_n$ distincts deux à deux : $\forall i \in [n], \forall j \in [n] \setminus \{i\}, E_i \neq E_j$

P : $\exists i \in [n] \mid \forall j \in [n] \setminus \{i\}, \neg(E_j \subset E_i)$

Par l'absurde,

« Tous les ensembles $E_1, \cdots, E_n$ contiennent au moins un des $n-1$ autres ensembles. »

P' : $\forall i \in [n], \exists j \in [n] \setminus \{i\} \mid E_j \subseteq E_i$

Soit $i \in [n]$ fixé, $u_0 = i \to \exists j \in [n\} \setminus \{i\} \mid \underbrace{E_j}_{E_{u_1}} \in \underbrace{E_i}_{E_{u_0}}$

$$u_{n+1} \to E_{u_{n+1}} \subset E_{u_n}$$

Plus n est grand, plus E_{u_n} devient petit, jusqu'à arriver au dernier ensemble, qui lui aussi doit contenir un ensemble $E_{u_{n+1}}$, donc un des précédents ensembles, sauf que :

$E_{u_n} \subset E_{u_{n-1}} \subset \cdots \subset E_{u_0}$ (E_{u_n} est le dernier ensemble et E_{u_0} le premier), donc forcément $\exists j$ tel que ($E_{u_n} \subset E_j$ donc) $E_{u_n} = E_{u_j}$ (contradiction).

Exercice 5. Suites récurrentes homogènes

Calculer les suites récurrentes.

1. $u_n = u_{n-1} + 6u_{n-2}$ si $n \geq 2$, $u_0 = 0$, $u_1 = 1$

polynôme caractéristique : $r^2 - r - 6 = 0$ $\qquad$ $r_1 = 3$ et $r_2 = -2$

$u_n = \lambda_1 3^n + \lambda_2(-2)^n$

$$\begin{cases} \lambda_1 + \lambda_2 = 0 & (1) \\ 3\lambda_1 - 2\lambda_2 = 1 & (2) \end{cases}$$

$(1) \Leftrightarrow \lambda_2 = -\lambda_1$

$(2) \Leftrightarrow 3\lambda_1 + 2\lambda_1 = 1 \Leftrightarrow \lambda_1 = \frac{1}{5}$

$(1) \Leftrightarrow \lambda_2 = -\frac{1}{5}$

$u_n = \frac{1}{5} \times 3^n - \frac{1}{5} \times (-2)^n$

2. $u_n = 4u_{n-1} - 4u_{n-2}$ si $n \geq 2$, $u_0 = 1$, $u_1 = 4$

polynôme caractéristique : $r^2 - 4r + 4 = 0$ $\qquad$ $r = 2$

$u_n = \lambda_1 2^n + \lambda_2 n 2^n$

$$\begin{cases} \lambda_1 = 1 & (1) \\ 2\lambda_1 + 2\lambda_2 = 4 & (2) \end{cases}$$

$(2) \Leftrightarrow 2 \cdot 1 + 2\lambda_2 = 4 \Leftrightarrow \lambda_2 = 1$

$u_n = 2^n + n \cdot 2^n$

3. $F_n = F_{n-1} + F_{n-2}$ si $n \geq 2$, $F_0 = 0$, $F_1 = 1$
polynôme caractéristique : $r^2 - r - 1 = 0$

$\Delta = (-1)^2 - 4 \times 1 \times (-1) = 5 \qquad r_1 = \dfrac{1-\sqrt{5}}{2} \qquad r_2 = \dfrac{1+\sqrt{5}}{2}$

$$F_n = \lambda_1 \left(\frac{1-\sqrt{5}}{2}\right)^n + \lambda_2 \left(\frac{1+\sqrt{5}}{2}\right)^n$$

$$\begin{cases} \lambda_1 + \lambda_2 = 0 & (1) \\ \dfrac{1-\sqrt{5}}{2}\lambda_1 + \dfrac{1+\sqrt{5}}{2}\lambda_2 = 1 & (2) \end{cases}$$

$(1) \Leftrightarrow \lambda_1 = -\lambda_2$

$(2) \Leftrightarrow -\dfrac{1-\sqrt{5}}{2}\lambda_2 + \dfrac{1+\sqrt{5}}{2}\lambda_2 = 1 \Leftrightarrow \lambda_2 = \dfrac{1}{\sqrt{5}}$

$(1) \Leftrightarrow \lambda_1 = -\dfrac{1}{\sqrt{5}}$

$$F_n = -\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n + \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

4. $u_n = 5u_{n-1} - 8u_{n-2} + 4u_{n-3}$ si $n \geq 3$, $u_0 = 0$, $u_1 = 1$, $u_2 = 2$
polynôme caractéristique : $r^3 - 5r^2 + 8r - 4 = 0$
1 est solution évidente.
$(r-1)(r^2 - 4r + 4)$
2 est racine double.
$u_n = \alpha_1 \cdot 1^n + (\alpha_2 + n\alpha_3)2^n$

$$\begin{cases} \alpha_1 + \alpha_2 = 0 \\ \alpha_1 + 2\alpha_2 + 4\alpha_3 = 1 \\ \alpha_1 + 4\alpha_2 + 8\alpha_3 = 2 \end{cases} \Rightarrow \begin{cases} \alpha_1 = -2 \\ \alpha_2 = 2 \\ \alpha_3 = -\dfrac{1}{2} \end{cases}$$

Exercice 6. Suites récurrentes non homogènes

Question 1

Calculer par substitution les suites récurrentes.
1. $u_n = u_{n-1} + n$ si $n \geq 1$, $u_0 = 0$
$u_1 = u_0 + 1$
$u_2 = u_1 + 2 = u_0 + 1 + 2$
$u_3 = u_2 + 3 = u_0 + 1 + 2 + 3$
$u_4 = u_3 + 4 = u_0 + 1 + 2 + 3 + 4$

$$\forall n \in \mathbb{N}, u_n = u_0 + \sum_{i=1}^{n} \frac{n(n+1)}{2}$$

2. $u_n = 2u_{n-1} + 1$ si $n \geq 1$, $u_0 = 2$

$u_1 = 2u_0 + 1 = 2^1 u_0 + 2^0$

$u_2 = 2u_1 + 1 = 2(2u_0 + 1) + 1 = 4u_0 + 2 + 1 = 2^2 u_0 + 2 + 2^0$

$u_3 = 2u_2 + 1 = 2(2^2 u_0 + 2 + 1) + 1 = 2^3 u_0 + 2^2 + 2 + 2^0$

$$u_n = 2^n u_0 + \sum_{i=0}^{n-1} 2^i = 2^n \cdot 2 + \sum_{i=0}^{n-1} 2^i \quad \text{or} \sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1} \Rightarrow \sum_{i=0}^{n-1} 2^i = \frac{2^n - 1}{2 - 1} = 2^n - 1$$

$u_n = 2^n \cdot 2 + 2^n - 1 = 3 \cdot 2^n - 1$

3. $u_n = 2u_{n-1} + 2^n$ si $n \geq 1$, $u_0 = 3$

$u_1 = 2u_0 + 2^1$

$u_2 = 2u_1 + 2^2 = 2(2u_0 + 2^1) + 2^2 = 2^2 u_0 + 2^2 + 2^2$

$u_3 = 2u_2 + 2^3 = 2(2^2 u_0 + 2^2 + 2^2) + 2^3 = 2^3 u_0 + 2^3 + 2^3 + 2^3$

$u_n = 2^n u_0 + n2^n$

Exercice 8. Exponentielle et récurrence faible

Question 1

Démontrer par récurrence faible que pour tout $n \in \mathbb{N}^*$, $2^{n-1} \leq n! \leq n^n$.

$P(n) : \forall n \in \mathbb{N}^*, 2^{n-1} \leq n! \leq n^n$

<u>Base :</u> pour $n = 1$ $\quad 2^0 \leq 1! \leq 1^1 \Leftrightarrow 1 \leq 1 \leq 1$ vrai

Donc $P(1)$ est vraie.

<u>Induction :</u> On suppose que $P(n)$ est vraie pour tout $n \in \mathbb{N}^*$. Montrons que $P(n+1)$ est vraie.

$2^{(n+1)-1} = 2^n = 2 \cdot 2^{n-1}$ $\qquad (n+1)! = (n+1) \cdot n!$

$2^{n-1} \leq n! \Leftrightarrow 2 \cdot 2^{n-1} \leq 2 \cdot n!$ $\qquad$ or $2 \cdot n! \leq (n+1) \cdot n!$ car $n > 1$

Donc $2 \cdot 2^{n-1} \leq (n+1) \cdot n!$.

$(n+1)^{n+1} = (n+1) \cdot (n+1)^n$

$n! \leq n^n \Leftrightarrow (n+1) \cdot n! \leq (n+1) \cdot n^n$ $\qquad$ or $(n+1) \cdot n^n \leq (n+1) \cdot (n+1)^n$ car $n > 1$

Donc $(n+1) \cdot n! \leq (n+1) \cdot (n+1)^n$.

Donc $P(n+1)$ est vraie.

<u>Conclusion :</u> $P(1)$ est vraie et si $P(n)$ est vrai alors $P(n+1)$ est vraie donc pour tout $n \in \mathbb{N}^*$, $2^{n-1} \leq n! \leq n^n$.

Question 2

Soit maintenant la propriété $P(n) : 2^n > n^2$.

1. Montrez que, pour tout $n \geq 3$, $P(n) \Rightarrow P(n+1)$.

On suppose que $P(n)$ est vraie au rang $n \geq 3$. Montrons que $P(n+1)$ est vraie.

$2^{n+1} = 2 \cdot 2^n$ $\qquad 2^n > n^2 \Leftrightarrow 2 \cdot 2^n > 2n^2$

On vérifie si $2n^2 > (n+1)^2 \Leftrightarrow 2n^2 - (n+1)^2 > 0$.

$2n^2 - (n+1)^2 = 2n^2 - n^2 - 2n - 1 = n^2 - 2n - 1$ $\qquad$ On étudie le signe de $n^2 - 2n - 1$.

2. Pour quelles valeurs de $n \in \mathbb{N}$ la propriété de $P(n)$ est-elle vérifiée ?

$2n^2 - (n+1)^2 = \left((\sqrt{2}+1)n+1\right)\left((\sqrt{2}-1)n-1\right) \geq 0$ si $n \geq \dfrac{1}{\sqrt{2}-1}$

Donc $P(n)$ est vérifiée si $n \geq 3$.

Exercice 10. Importance de la base

Question 1

On considère la propriété $P(n)$: toute application f de $\{0, \cdots, n\}$ dans $\mathbb{N}$ vérifie $f(0) = 0$.
Montrer que : $\forall n \in \mathbb{N} \quad [P(n) \Rightarrow P(n+1)]$.
Peut-on en déduire que la propriété $P(n)$ est vraie pour tout $n \in \mathbb{N}$?

Supposons $P(n)$: $\forall f : \{0, \cdots, n\} \to \mathbb{N}$ on a $f(0) = 0$
Soit $f : \{0, \cdots, n+1\} \to \mathbb{N}$.
Soit g une restriction de f de $\{0, \cdots, n\} \to \mathbb{N}$ donc $g(0) = 0 \Rightarrow f(0) = 0$
Donc $P(n+1)$ est vraie.
/!\ $P(0)$ est fausse donc on ne peut pas en déduire que la propriété $P(n)$ est vraie.

Question 2

On considère la propriété $Q(n)$: toute application définie sur une ensemble à n élément est constante.

Le raisonnement suivant est-il correct ? Pourquoi ?

"$\mathcal{Q}(1)$ est vraie car toute application définie sur un singleton est constante.

Soit $n \in \mathbb{N}$, supposons que $\mathcal{Q}(n)$ soit vraie. Soit $E = \{x_0, x_1, \ldots, x_n\}$ un ensemble ayant $n+1$ éléments et f une application de E dans un ensemble F. La restriction de f à $E = \{x_0, x_1, \ldots, x_{n-1}\}$ est une application définie sur un ensemble à n éléments donc elle est constante : $\exists a \in F$ tel que $f(x_0) = f(x_1) = \ldots = f(x_{n-1}) = a$. La restriction de f à $E = \{x_1, \ldots, x_n\}$ est une application définie sur un ensemble à n éléments donc elle est constante : $\exists b \in F$ tel que $f(x_1) = \ldots = f(x_n) = b$. Comme $f(x_1) = a$ et $f(x_1) = b$, on obtient $a = b$. Par conséquent, $f(x_0) = f(x_1) = \ldots = f(x_n) = a$ et f est constante sur E.

On a donc montré que : $\forall n \geq 1 \quad [\mathcal{Q}(n) \Rightarrow \mathcal{Q}(n+1)]$.

On en conclut que $\mathcal{Q}(n)$ est vraie pour tout $n \geq 1$."

L'induction est vraie. Mais la base est mauvaise : il aurait fallu prendre deux éléments dans l'ensemble de départ.

Exercice 12. Récurrence forte

On considère la suite G_n définie par $G_n = G_{n-1} + G_{n-2} + 1$ si $n \geq 2$, $G_0 = 0$, $G_1 = 0$.

Question 1

La suite de Fibonacci F_n a été définie en cours et dans l'exercice 5. Montrer par récurrence forte la propriété $\Pi(n)$: $G_n = F_{n+1} - 1$ pour $n \geq 0$. En déduire la valeur de G_n.

Suite de Fibonacci : $F_n = F_{n-1} + F_{n-2}$ si $n \geq 2$, $F_0 = 0$, $F_1 = 1$

Démonstration par récurrence forte de $\Pi(n)$:

Base : $G_0 = 0$ $\quad F_{0+1} - 1 = F_1 - 1 = 0$ $\quad$ donc $\Pi(0)$ est vérifiée

$G_1 = 0$ $\quad F_{1+1} - 1 = F_2 - 1 = F_1 + F_0 - 1 = 0$ $\quad$ donc $\Pi(1)$ est vérifiée

Induction : Soit $n \geq 2$ tel que $\forall k \in \{0, \cdots, n-1\}$, $\Pi(k)$ est vérifiée.

$$G_n = G_{n-1} + G_{n-2} + 1 = (F_n - 1) + (F_{n-1} - 1) + 1 = \underbrace{F_n + F_{n-1}}_{F_{n+1}} - 1$$

$G_n = F_{n+1} - 1$ donc $\Pi(n)$ est vérifiée.

Valeur de G_n :

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$G_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1} - 1$$

Question 2

Montrer que tout entier n ($\forall n \geq 0$) peut s'écrire comme une somme finie de puissances de 2 toutes distinctes. $\Pi(n)$

Base : $\Pi(0)$ est vérifiée. $\quad$ (0 peut être exprimé sous la forme d'une somme vide.)

$\Pi(1) : 1 = 2^0$

Induction : Soit $n > 0$ tel que $\forall i \in \{0, \cdots, n-1\}$, $\Pi(i)$ vérifiée.

$\exists m$ tel que $2^m \leq n \leq 2^{m+1}$ $\quad$ car $u_m = 2^m$ suite strictement croissante

$2^m \leq n \leq 2^{m+1} \Leftrightarrow 0 \leq n - 2^m \leq 2^{m+1} - 2^m$

On pose $\alpha \in \{0, \cdots, n-1\}$ telle que $\Pi(\alpha)$ est vérifiée donc $\alpha = \sum_{k \in K} 2^k$.

$\alpha = n - 2^m : 0 \leq \alpha \leq 2^{m+1} - 2^m = 2^m(2-1) = 2^m$ donc $0 \leq \alpha \leq 2^m$

$$\alpha = n - 2^m \Leftrightarrow n = \alpha + 2^m = \sum_{k \in K} 2^k + 2^m$$

Que se passe-t-il si $m = k$? impossible $\quad \alpha = \sum_{k \in K} 2^k \leq 2^m$

Question 3

Montrer par récurrence forte sur n que, pour tout entier naturel $n \geq 1$, il existe deux entiers naturels p et q tels que $n = 2^p(2q+1)$. $P(n)$

Base : $1 = 2^0(2 \cdot 0 + 1)$ donc $P(1)$ est vérifiée.

Induction : Soit $n \geq 2$, supposons $P(k)$ vraie pour $1 \leq k < n$.

- Si n impair : $n = 2 \cdot \frac{n-1}{2} + 1 = 2 \cdot q + 1 = 2^p(2q+1)$ avec $p = 0$ et $q = \frac{n-1}{2}$
- Si n pair : $\frac{n}{2}$ est un entier et $P(k)$ vraie pour tout $1 \leq k < n$ donc $P\left(\frac{n}{2}\right)$ est vraie

  $$\frac{n}{2} = 2^{p_1}(2q_1 + 1)$$

  $$n = 2 \cdot \frac{n}{2} = 2 \cdot 2^{p_1}(2q_1 + 1) = 2^{p_1+1}(2_{q_1} + 1)$$

Donc $P(n)$ est vérifiée.

Question 4

Montrer l'unicité du couple $(p,q) \in \mathbb{N}^2$ tel que $n = 2^p(2q+1)$, en faisant un raisonnement par l'absurde (et non par récurrence).

Supposons $(p_1, q_1) \neq (p_2, q_2)$ tel que $n = 2^{p_1}(2q_1+1) = 2^{p_2}(2q_2+1)$.

- Si $p_1 < p_2$: $\underbrace{2q_1+1}_{\text{impair}} = \underbrace{2^{p_2-p_1}(2q_2+1)}_{\text{pair}}$ impossible

  $2q_2+1$ est impair. p_1 et p_2 sont des entiers et $p_2 - p_1 > 0$ donc $2^{p_2-p_1}$ est pair.
  pair $\times$ impair $\rightarrow$ pair
- Si $p_1 < p_2$: $\underbrace{2q_2+1}_{\text{impair}} = \underbrace{2^{p_1-p_2}(2q_1+1)}_{\text{pair}}$ impossible
- Si $p_1 = p_2$: $2q_1+1 = 2q_2+1 \Leftrightarrow q_1 = q_2$ impossible par hypothèse

On a montré l'unicité par l'absurde.

Exercice 13. Preuve par l'absurde, récurrence faible et forte

On rappelle que un nombre premier est un entier $n > 1$ qui admet exactement deux diviseurs : 1 et lui-même n.

Question 1

Démontrer par récurrence forte que tout entier supérieur ou égal à 2 admet au moins un diviseurs premier (qui peut être lui-même). $Q(n)$

Base : pour $n = 2$, 2 est premier donc $Q(2)$ est vraie.

Induction : On suppose $Q(2), \cdots, Q(n-1)$ vraies.

- Soit n est premier $\Rightarrow Q(n)$ vraie.
- Soit n n'est pas premier $\Rightarrow n = p \cdot q$ avec $2 \leq p < n$ et $2 \leq q < n$
  $Q(p)$ est vraie (d'après l'hypothèse d'induction) donc il existe p^* diviseur premier de p, donc aussi diviseur de n.

Donc $Q(n)$ est vraie.

Question 2

Démontrer qu'il existe une infinité de nombres premiers en faisant une preuve par l'absurde.

Supposons qu'il n'existe que n nombres premiers : $p_1, \cdots, p_n$.

$p = p_1 \times \cdots \times p_n + 1$ donc $(p-1)$ est divisible par $p_1, \cdots, p_n$

donc p n'est pas divisible par $p_1, \cdots, p_n$

Mais en question 1, on a montré que $\forall p \geq 2$, p admet un diviseur premier,

donc $\exists p_{n+1} \notin \{p_1, \cdots, p_n\}$ diviseur premier de p Contradiction !

Question 3

Démontrer qu'il existe une infinité de nombres premiers en faisant une preuve directe utilisant une récurrence faible.

$P(n)$: il existe au moins n nombres premiers

<u>Base :</u> $P(1)$ est vraie car 2 est un nombre premier.

<u>Induction :</u> On suppose que $P(n)$ est vraie pour $n > 1$.

Soit n nombres premiers $p_1, \cdots, p_n$.

$p = p_1 \times p_2 \times \cdots \times p_n + 1$ admet un diviseur p_{n+1} premier (d'après la question 1) donc $P(n+1)$ est vraie

Exercice 17. Ordre de grandeur en $\mathcal{O}$, $\boldsymbol{\Omega}$ et $\boldsymbol{\Theta}$

Question 1

Démontrer que

(i) $\quad n^2 \in \mathcal{O}(10^{-5}n^3)$

(ii) $\quad 25n^4 - 19n^3 + 13n^2 \in \mathcal{O}(n^4)$

(iii) $\quad 2^{n+100} \in \mathcal{O}(2^n)$

(i) $\forall n \geq 1, n^2 \leq n^3, n^2 \leq D \cdot 10^{-5} \cdot n^3$ avec $D = 10^5$

Par définition, $n^2 \in \mathcal{O}(10^{-5}n^3)$.

(ii) $\forall n \geq 1, 25n^4 - 19n^3 + 13n^2 \leq (25 - 19 + 13)n^4$ avec $D = 25 - 19 + 13$

Par définition, $25n^4 - 19n^3 + 13n^2 \in \mathcal{O}(n^4)$.

(iii) $\forall n \geq 1, 2^{n+100} = 2^n \cdot 2^{100} \leq 2^{100} \cdot 2^n$ avec $D = 2^{100}$

Par définition, $2^{n+100} \in \mathcal{O}(2^n)$.

Question 2

Donner les relations d'inclusion entre les ensembles suivants : $\mathcal{O}(n \log n)$, $\mathcal{O}(2^n)$, $\mathcal{O}(\log n)$, $\mathcal{O}(1)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$ et $\mathcal{O}(n)$.

$\forall n \geq 1, 1 \leq n \leq n^2 \leq n^3$ donc $\mathcal{O}(1) \subset \mathcal{O}(n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^3)$

$$\begin{array}{ll} \forall n \geq 3 & \left.\begin{array}{r} 1 \leq \log(n) \\ \log(n) \leq n \\ n \log n \leq n^2 \end{array}\right\} \mathcal{O}(1) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^2) \\ \forall n \geq 1 & \end{array}$$

On a $\mathcal{O}(n^3) \subset \mathcal{O}(2^n)$.

$\mathcal{O}(1) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^3) \subset \mathcal{O}(2^n)$

Question 3

Démontrer que

$(i)\quad 3n^4 - 5n^3 \in \Omega(n^2) \qquad\qquad (ii)\quad 2^n \in \Omega(n^2)$

(i) Trouvons k et n_0 tels que : $\forall n \geq n_0$, $3n^4 - 5n^3 \geq k \cdot n^2$.
$k = 1$ et $n_0 = 3$ vérifie l'inégalité. Donc $3n^4 - 5n^3 \in \Omega(n^2)$.

$(ii)\ \lim\limits_{n\to+\infty} \dfrac{2^n}{n^2} = +\infty \ \text{ donc } 2^n \in \Omega(n^2)$

Question 4

Donner sans démonstration les relations d'inclusion entre les ensembles suivants :
$\Omega(n \log n)$, $\Omega(2^n)$, $\Omega(\log n)$, $\Omega(1)$, $\Omega(n^2)$, $\Omega(n^3)$ et $\Omega(n)$.

La définition de Ω est l'opposé de la définition de $\mathcal{O}$ donc on va avoir les relations d'inclusion inverses par rapport à la question 2.

$\Omega(2^n) \subset \Omega(n^3) \subset \Omega(n^2) \subset \Omega(n \log n) \subset \Omega(n) \subset \Omega(\log n) \subset \Omega(1)$

Question 5

Quelles sont les relations éventuelles d'inclusion entre les ensembles suivants ? Justifier votre réponse sans démonstration.

$(i)\quad \Theta(n^3)$ et $\Omega(n^2)$

$\Theta(n^3) \subset \Omega(n^2)$

$(ii)\quad \mathcal{O}(n^2)$ et $\Omega(n^3)$

$\mathcal{O}(n^2) \cap \Omega(n^3) = \emptyset$

$(iii)\quad \mathcal{O}(n^2)$ et $\Omega(n^2)$

$n^2 \in \mathcal{O}(n^2)$ et $n^2 \in \Omega(n^2)$ donc $\mathcal{O}(n^2) \cap \Omega(n^2) \neq \emptyset$

mais $\begin{array}{ll} n \in \mathcal{O}(n^2) & n^3 \notin \mathcal{O}(n^2) \\ n \notin \Omega(n^2) & n^3 \in \Omega(n^2) \end{array}$

Exercice 18. Ordre de grandeur et maximum

Soient $f, g : \mathbb{N} \to \mathbb{N}$.

Question 1

Montrer que $f(n) + g(n) \in \Theta\big(\max(f(n), g(n))\big)$.

$\forall n \geq 0, \max\big(f(n), g(n)\big) \leq f(n) + g(n) \leq 2\max\big(f(n), g(n)\big)$
Donc $f(n) + g(n) \in \Theta\big(\max(f(n), g(n))\big)$.

Question 2

Montrer que $\mathcal{O}\big(f(n) + g(n)\big) = \mathcal{O}\big(\max\big(f(n), g(n)\big)\big)$.

- Montrons que $\mathcal{O}\big(f(n) + g(n)\big) \subset \mathcal{O}\big(\max\big(f(n), g(n)\big)\big)$:
  $\forall h(n) \in \mathcal{O}\big(f(n) + g(n)\big)$, $\exists k > 0, n_0 \geq 0$ tels que $\forall n \geq n_0$
  $$h(n) \leq k\big(f(n) + g(n)\big) \leq 2k \max\big(f(n), g(n)\big)$$
  (d'après la question 1)
  donc $h(n) \subset \mathcal{O}\big(\max\big(f(n), g(n)\big)\big) \Rightarrow \mathcal{O}\big(f(n) + g(n)\big) \subset \mathcal{O}\big(\max\big(f(n), g(n)\big)\big)$
- Montrons que $\mathcal{O}\big(\max\big(f(n), g(n)\big)\big) \subset \mathcal{O}\big(f(n) + g(n)\big)$:
  $\forall h(n) \in \mathcal{O}\big(\max\big(f(n), g(n)\big)\big)$, $\exists k > 0, n_0 \geq 0$ tels que $\forall n \geq n_0$
  $$h(n) \leq k \cdot \max\big(f(n), g(n)\big) \leq k \cdot \big(f(n) + g(n)\big)$$
  donc $\mathcal{O}\big(\max\big(f(n), g(n)\big)\big) \subset \mathcal{O}\big(f(n) + g(n)\big)$

Par double inclusion : $\mathcal{O}\big(f(n) + g(n)\big) = \mathcal{O}\big(\max\big(f(n), g(n)\big)\big)$

TD 2 : Terminaison et validité d'un algorithme itératif

Exercice 1

On considère la fonction FactorielleIt définie de la manière suivante pour calculer la factorielle pour $n \in \mathbb{N}$:

```
def FactorielleIt(n):
    i=1; tmp=1
    while i<=n:
        tmp=tmp*i
        i=i+1
    return tmp
```

Question 1

Montrer la terminaison de l'algorithme.

Un variant de boucle est une variable qui diminue à chaque tour de boucle jusqu'à arriver à 0 (terminaison de la boucle). Il permet de montrer la terminaison de l'algorithme.

variant de boucle : $n - 1$

La boucle est effectuée exactement n fois. Il y a dans la boucle deux instructions.

On en déduit la terminaison de l'algorithme.

Question 2

Soit tmp_i pour $i \in \{1, \cdots, n\}$ la valeur de la variable tmp aux instants suivants :

- $tmp_1 = 1$ est la valeur de tmp juste avant d'entrer dans la boucle ;
- pour $i \in \{2, \cdots, n+1\}$, tmp_i est la valeur de tmp à la fin du corps de boucle (juste après l'incrémentation de i).

Que vaut tmp_i en fonction de i ? En déduire un invariant de boucle et démontrer-le.

On considère tmp_i à la fin du corps de boucle, après l'incrémentation de i. Donc on a $tmp_i = (i-1)!$ qui est l'**invariant de boucle**. $\pi(i)$

Base : $tmp_1 = 1 = 0!$ $\quad\quad tmp_2 = 1 = 1!$

Induction : Soit $i \in [\![1, n]\!]$, supposons $\pi(i)$ vérifiée.

$tmp_{i+1} = tmp_i \times i = (i-1)! \cdot i = i!$ $\quad\quad$ donc $\pi(i+1)$ est vérifiée

Question 3

En déduire la validité de la fonction FactorielleIt.

On a démontré l'invariant de boucle. On renvoie $tmp_{n+1} = n!$ donc FactorielleIt est bien valide.

Exercice 3. Recherche séquentielle d'un élément dans un tableau non trié

Soit tab un tableau de n entiers et $elem$ un entier. Le but est d'étudier deux fonctions qui retournent l'indice i minimum tel que $tab[i] = elem$.

On suppose tout d'abord que tab n'est pas triée et que $elem$ est dans tab. On considère alors la fonction itérative suivante :

```
def Recherche(elem, tab):
    i = 0
    while elem!=tab[i] :
        i = i + 1
    return i
```

Question 1

Montrer la terminaison de la fonction Recherche.

précondition nécessaire $(*)$: $\exists i \in \{1, \cdots, n\}$ tel que $elem = tab[i]$

On ne considère que des instructions élémentaires.

$(*)$ Algorithme alternatif qui permet d'éviter la précondition :

```
def Recherche(elem, tab):
    for i in range(len(tab)):
        if (tab[i] == elem):
            return i
    return None
```

Question 2

A la fin du corps de boucle (juste après l'incrémentation de i), que peut-on dire de $elem$ et des éléments de $tab[0 \cdots i-1]$. En déduire un invariant de boucle et démontrer-le.

$elem \notin tab[0 \cdots i-1]$ $\qquad$ $\pi(i)$: $\forall i \in \{0, \cdots, n\}, \forall j \in \{0, \cdots, i-1\}, elem \neq tab[j]$

<u>Base :</u> $i = 0 \Rightarrow tab\ \underbrace{[0 \cdots 0-1]}_{\emptyset} = \emptyset$ donc on a bien $\pi(0)$

<u>Induction :</u> On suppose $\pi(k)$ vraie pour $k < n-1$.

$elem \notin tab[0 \cdots k]$

et pour $i = k+1$: le test $elem$!= $tab[i]$ renvoie vrai

donc $elem \neq tab[k+1]$

$elem \notin tab[0 \cdots k+1]$

Question 3

En déduire la validité de la fonction Recherche.

Validité : la sortie est l'indice minimal tel que $tab[i] = elem$.
On sait que $tab[i] = elem$ car le test de la boucle while est faux.
On sait que l'indice est minimal car $\pi(i)$ est vraie donc $elem \notin tab[0 \cdots i-1]$.
2 choses importantes à dire

Exercice 5. Pousser le plus grand élément d'un tableau

On étudie dans cet exercice la fonction itérative Push dont le code suit :

```
def Push(tab,k):
        j=1
        while(j<k):
                if tab[j]<tab[j-1]:
                        tmp=tab[j-1]; tab[j-1]=tab[j]; tab[j]=tmp
                j=j+1
                print("j=",j,"␣␣tab=",tab)
```

On suppose que si tab est un tableau de n éléments, $k \in \{1, \cdots, n\}$.

Question 1

Exécuter Push(tab,6) pour le tableau $tab[0 \cdots 7] = [8, 7, 12, 5, 15, 4, 3, 9]$. Que fait cette fonction ?

$tab = [8, 7, 12, 5, 15, 4, 3, 9]$ et $k = 6$
>>> Push(tab, k)

j	tab
$j = 2$	$tab = [7, 8, 12, 5, 15, 4, 3, 9]$
$j = 3$	$tab = [7, 8, 12, 5, 15, 4, 3, 9]$
$j = 4$	$tab = [7, 8, 5, 12, 15, 4, 3, 9]$
$j = 5$	$tab = [7, 8, 5, 12, 15, 4, 3, 9]$
$j = 6$	$tab = [7, 8, 5, 12, 4, 15, 3, 9]$

Elle remonte le plus grand élément dans le tableau $tab[0, \cdots, k-1]$ i.e. $tab[k-1]$ est le plus grand élément de $tab[0, \cdots, k-1]$ et on ne modifie $tab[k, \cdots, n-1]$.

Question 2

Montrer que l'appel Push(tab, k) se termine.

La boucle est effectuée k fois et on n'a que des instructions élémentaires.

Question 3

Soit $tab_1 = tab$ et pour $j \in \{2, \cdots, k\}$, tab_j désigne le tableau tab obtenu à la fin du corps de boucle de Push (après l'incrémentation de j).
Exprimer l'invariant de boucle en fonction de la suite de tableaux tab_i puis le démontrer.

Invariant de boucle : $\pi(j)$

- $tab_j[j, \cdots, n-1] = tab_1[j, \cdots, n-1]$
- $tab_j[0, \cdots, j-1]$ avec $tab[j-1]$ plus grand élément et $tab_j[0, \cdots, j-1]$ constitué des mêmes éléments que $tab_1[0, \cdots, j-1]$

<u>**Preuve par récurrence faible :**</u>

<u>Base :</u> $j = 1$, $tab_1 = tab$ (contient les mêmes éléments) et $tab_1[0]$ est le plus grand élément de $tab_1[0]$, donc $\pi(0)$ est vérifiée.

<u>Induction :</u> Supposons $j \in \{1, \cdots, k-1\}$ tel que $\pi(j)$ est vérifiée.

(1) $tab_{j+1}[j+1, \cdots, n-1] = tab_j[j+1, \cdots, n-1] = tab_1[j+1, \cdots, n-1]$

(Tel que la fonction a été écrite, on a l'égalité entre tab_{j+1} et le rang précédent tab_j. Si on a l'égalité entre tab_{j+1} et le rang précédent, alors on a l'égalité entre tab_{j+1} et le premier tableau tab_1 par hypothèse d'induction sur $\pi(j)$: $tab_j[j, \cdots, n-1] = tab_1[j, \cdots, n-1]$.)

(2) **Si $\boldsymbol{tab_j[j] < tab_j[j-1]}$:** alors par hypothèse $tab_j[j-1]$ est le plus grand élément de $tab_j[0, \cdots, j]$
  → après l'échange : $tab_{j+1}[j]$ est le plus grand élément de $tab_{j+1}[0, \cdots, j]$ et on a préservé les éléments du tableau (vrai par hypothèse)
  Si $\boldsymbol{tab_j[j] \geq tab_j[j-1]}$: $tab_{j+1}[j]$ est le plus grand élément de $tab_{j+1}[0, \cdots, j]$ (vrai par hypothèse sur $\pi(j)$) et on n'a rien fait au rang $j+1$ (pas d'échange) donc on a préservé les éléments du tableau (vrai par hypothèse).

Question 4

Démontrez que, la fonction Push(tab, k) pour $k \in \{1, \cdots, n\}$ a réorganisé les éléments du tableau tab de sorte que :

1. $tab[k-1]$ contient le plus grand élément de $tab[0 \cdots k-1]$;
2. les éléments de $tab[k \cdots n-1]$ n'ont pas été modifiés par Push(tab, k)

On a montré $\pi(j)$ avec j l'indice de j en fin de corps de boucle.
On veut montrer la même chose à l'indice k en fin de fonction i.e. en sortie de boucle.

<u>Réponse :</u>

En fin de fonction, i.e. en sortie de boucle, $j = k$ donc (d'après la réponse de la question 3) on a $\pi(k)$ vérifiée et donc 1. et 2. vérifiées.

Exercice 6. Tri à bulles

On considère l'algorithme de tri à bulles suivant :

```
def BubleSort(tab):
      i=0
      n=len(tab)
      while i<n :
            Push(tab, n-i)
            i=i+1
            print("i=", i, "␣␣tab=", tab)
```

Question 1

Appliquer cette fonction de tri au tableau de $n = 6$ éléments donné par $t[0 \cdots 5] = [18, 17, 4, 12, 1, 2]$. On ne considère que l'affichage de la fonction BubleSort (on oublie l'affichage de la fonction Push définie dans l'exercice précédent).

$tab = [18, 17, 4, 12, 1, 2] \rightarrow n = 6$
>>> BubleSort(tab)
$i = 1 \;\; tab = [17, 4, 12, 1, 2, 18]$
$i = 2 \;\; tab = [4, 12, 1, 2, 17, 18]$
$i = 3 \;\; tab = [4, 1, 2, 12, 17, 18]$
$i = 4 \;\; tab = [1, 2, 4, 12, 17, 18]$
$i = 5 \;\; tab = [1, 2, 4, 12, 17, 18]$
$i = 6 \;\; tab = [1, 2, 4, 12, 17, 18]$

Question 2

Montrer que BubleSort(tab) se termine.

La boucle s'effectue n fois, elle est constituée d'instructions élémentaires et de la fonction Push qui se termine (cf. Exercice 5).

Question 3

Soit la suite $tab_0^* = tab$ et pour tout $i \in \{1, \cdots, n\}$, tab_i^* est le tableau tab à la fin du corps de boucle de la fonction. Démontrer la propriété $\pi^*(i)$, $i \in \{1, \cdots, n\}$:

1. Le sous-tableau $tab_i^*[n - i \cdots n - 1]$ contient les i plus grands éléments de tab rangés en ordre croissant. (i)
2. Le sous-tableau $tab_i^*[0 \cdots n - i - 1]$ contient les éléments de $tab[0 \cdots n - 1]$ qui ne sont pas dans $tab_i^*[n - i \cdots n - 1]$. (ii)

Par récurrence faible

Base : $tab_1^* =$ Push(tab, n)

D'après l'exercice 5, $tab_1^*[n - 1]$ est bien le plus grand élément de tab et on a conservé les éléments de tab.

Induction : $i \in \{1, \cdots, n\}$ vérifie $\pi^*(i)$

→ On utilise le résultat de l'exercice 5.

Question 4

En déduire que BubleSort(tab) trie les n éléments de tab en ordre croissant.

Exactement comme la dernière question de l'exercice 5

→ $i = n$ en sortie de boucle donc $\pi^*(n)$ est vérifiée (cf. question précédente)

Exercice 7. Miroir d'un tableau

Le but de cet exercice est d'étudier un algorithme qui inverse les éléments d'un tableau sans utiliser une structure supplémentaire. La fonction swapp permet d'inverser les éléments $t[i]$ et $t[j]$. La fonction miroir inverse les éléments d'un tableau. Le code de ces fonctions suit. L'appel len(tab) renvoie le nombre d'éléments du tableau tab. L'instruction n%2 renvoie la valeur de n modulo 2.

```
def swapp (tab, i, j):
    aux = tab[i]; tab[i]=tab[j]; tab[j]=aux

def miroir (tab):
    n = len(tab)
    j = n/2
    if (n%2 == 0):    # Si n est pair
        i = n/2 - 1
    else:
        i = n/2
    while (j<n):
        swapp(tab, i, j)
        i = i -1
        j = j +1
        print tab
```

On considère que n / 2 est la division entière de n par 2.

Question 1

Exécuter la fonction miroir pour les tableaux $tab = [2, 7, 9, 3, 1]$ et $tab = [7, 9, 2, 4, 3, 10]$.

$tab = [2, 7, 9, 3, 1]$	$tab = [7, 9, 2, 4, 3, 10]$
>>> miroir(tab)	>>> miroir(tab)
$[2, 7, 9, 3, 1]$	$[7, 9, 4, 2, 3, 10]$
$[2, 3, 9, 7, 1]$	$[7, 3, 4, 2, 9, 10]$
$[1, 3, 9, 7, 2]$	$[10, 3, 4, 2, 9, 7]$

Question 2

Calculer le nombre exact d'itérations de la boucle principale de la fonction miroir. En déduire la terminaison de la fonction miroir pour tout tableau tab.

Nombre d'itérations :

- Initialisation de j : $\left\lfloor \frac{n}{2} \right\rfloor$
- Fin de boucle : $j = n$
- Itération : j incrémenté de 1
- Nombre d'itérations : $n - \left\lfloor \frac{n}{2} \right\rfloor$

Terminaison : on a exactement $n - \left\lfloor \frac{n}{2} \right\rfloor$ itérations de boucle avec que des instructions élémentaires (on peut considérer que swapp est un composition d'instructions élémentaires).

Question 3

Par la suite, on pose $k^* = n - \left\lfloor \frac{n}{2} \right\rfloor$. Pour toute valeur $k \in \{0, \cdots, k^*\}$, on note tab_k le tableau tab à la fin de la k-ième itération (juste après avoir incrémenté j), i_k la valeur correspondante de i et j_k celle de j. Les valeurs tab_0, i_0 et j_0 correspondent aux valeurs respectivement de tab, i et j juste avant d'entrer dans la boucle.

Démontrer par récurrence sur $k \in \{0, \cdots, k^*\}$ les propriétés suivantes :

1. $i_k = i_0 - k$ et $j_k = j_0 + k$;
2. $tab_k[0 \cdots i_k] = tab_0[0 \cdots i_k]$ et $tab_k[j_k \cdots n] = tab_0[j_k \cdots n]$;
3. $tab_k[i_k + 1 \cdots j_k - 1]$ est le miroir de $tab_0[i_k + 1 \cdots j_k - 1]$.

<u>Base :</u> $k = 0$

1. $i_0 = i_0 - 0$ et $j_0 = j_0 + 0$
2. $tab_0[0, \cdots, i_0] = tab_0[0, \cdots, i_0]$ et $tab_0[j_0, \cdots, n] = tab_0[j_0, \cdots, n]$
3. $tab_0[i_0 + 1, \cdots, j_0 - 1] = \emptyset \qquad i_0 + 1 \geq \left\lfloor \frac{n}{2} \right\rfloor > j_0 - 1$

   (Une liste vide est forcément le miroir d'elle-même.)

Donc $\pi_1(0)$, $\pi_2(0)$ et $\pi_3(0)$ sont vérifiées.

Induction : On suppose $0 \leq k < k^*$ tel que $\pi_1(k)$, $\pi_2(k)$ et $\pi_3(k)$ sont vérifiées.

1. $i_k = i_0 - k$ et $i_{k+1} = i_k - 1 = i_0 - (k+1)$ de même $j_{k+1} = j_k + 1 = j_0 + (k+1)$
2. On passe de tab_k à tab_{k+1} en inversant $tab_k[i_k]$ avec $tab_k[j_k]$ donc
   $tab_{k+1}[0, \cdots, i_{k+1}] = tab_k[0, \cdots, i_{k+1}] = tab_0[0, \cdots, i_{k+1}]$
   de même de l'autre côté : $tab_{k+1}[j_{k+1}, \cdots, n] = tab_0[j_{k+1}, \cdots, n]$
3. On inverse $tab_k[i_k]$ avec $tab_k[j_k]$ et par hypothèse $tab_k[i_k + 1, \cdots, j_k - 1]$ miroir de $tab_0[i_k + 1, \cdots, j_k - 1]$
   donc $tab_{k+1}[i_{k+1} + 1, \cdots, j_{k+1} - 1]$ miroir de $tab_0[i_{k+1} + 1, \cdots, j_{k+1} - 1]$.

Question 4

Démontrer que $i_{k^*} = -1$ et $j_{k^*} = n$. En déduire la validité de la fonction miroir.

D'après la question 3, $j_{k^*} = j_0 + k^* = \left\lfloor \frac{n}{2} \right\rfloor + n - \left\lfloor \frac{n}{2} \right\rfloor = n$ et $i_{k^*} = i_0 - k^* = i_0 - n + \left\lfloor \frac{n}{2} \right\rfloor$

Deux cas :

- n pair : $i_{k^*} = \left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right) - n + \left\lfloor \frac{n}{2} \right\rfloor = -1$
- n impair : $n = 2\left\lfloor \frac{n}{2} \right\rfloor + 1$ donc $i_{k^*} = \left(\left\lfloor \frac{n}{2} \right\rfloor\right) - n + \left\lfloor \frac{n}{2} \right\rfloor = 2\left\lfloor \frac{n}{2} \right\rfloor - n = -1$

TD 3 : Terminaison et validité d'un algorithme récursif

Exercice 1. Calcul récursif du nombre d'occurrences d'un élément dans un tableau

On considère la fonction suivante :

```
def nb_occ(tab, k, x):
    print('tab=', tab[0:k], 'x=', x)
    if k == 0:
        res=0
    else :
        if tab[k - 1] == x:
            res=nb_occ(tab, k - 1, x) + 1
        else:
            res=nb_occ(tab, k - 1, x)
    print ('tab=', tab[0:k], 'x=', x, 'nboccurences=', res)
    return res
```

On rappelle que $tab[0{:}k]$ est le sous-tableau de tab composé des k premiers éléments.

Question 1

Exécutez nb_occ(tab, k, x) pour $tab = [3,6,7,6,2,6,3]$, $k = 7$ et $x = 6$. Vous ne donnerez que les affichages obtenus.

$tab = [3,6,7,6,2,6,3]$, $k = 7$, $x = 6$

>>> nb_occ(tab, k, x)

$tab = [3,6,7,6,2,6,3]$, $x = 6$

$tab = [3,6,7,6,2,6]$, $x = 6$

$tab = [3,6,7,6,2]$, $x = 6$

$tab = [3,6,7,6]$, $x = 6$

$tab = [3,6,7]$, $x = 6$

$tab = [3,6]$, $x = 6$

$tab = [3]$, $x = 6$

$tab = []$, $x = 6$

$tab = []$, $x = 6$, nboccurrences $= 0$

$tab = [3]$, $x = 6$, nboccurrences $= 0$

$tab = [3,6]$, $x = 6$, nboccurrences $= 1$

$tab = [3,6,7]$, $x = 6$, nboccurrences $= 1$

$tab = [3,6,7,6]$, $x = 6$, nboccurrences $= 2$

$tab = [3,6,7,6,2]$, $x = 6$, nboccurrences $= 2$

$tab = [3,6,7,6,2,6]$, $x = 6$, nboccurrences $= 3$

$tab = [3,6,7,6,2,6,3]$, $x = 6$, nboccurrences $= 3$

Question 2

Démontrez que la fonction nb_occ(tab, k, x) se termine et renvoie le nombre d'occurrences de x dans le tableau tab entre les positions 0 (comprise) et $k-1$ (comprise). $P(k)$

Base : $k = 0$

nb_occ(tab, k, x) se termine (pas d'appels récursifs) et renvoie nboccurrences = 0 ce qui correspond au nombre d'occurrences entre 0 et -1 (le tableau est vide donc il y a forcément 0 occurrences de x).

Induction : Soit $k \geq 1$ et $P(k-1)$ vérifiée

Deux cas :

- $tab[k-1] == x$: res = nb_occ(tab, $k-1$, x) + 1 est le bon nombre d'occurrences par hypothèse → se termine par hypothèse
- $tab[k-1]$!= x : res = nb_occ(tab, $k-1$, x) est le bon nombre d'occurrences par hypothèse → se termine par hypothèse

Par récurrence, $P(k)$ est vérifiée.

Exercice 4. Puissance de x

Soient x et n deux entiers tels que $x > 0$ et $n \geq 0$. Le but de cet exercice est d'étudier un algorithme récursif pour calculer x^n.

Question 1

Soit $u_n(x)$, $n \geq 0$ la suite définie par :

$$u_n(x) = \begin{cases} 1 & \text{si} \quad n = 0 \\ \left(u_{\frac{n}{2}}(x)\right)^2 & \text{si} \quad n \text{ est pair} \\ \left(u_{\frac{n-1}{2}}(x)\right)^2 \times x & \text{si} \quad n \text{ est impair} \end{cases}$$

1. Calculez $u_{10}(2)$.

$u_0(2) = 1$ $\quad u_1(2) = 2$ $\quad u_2(2) = 2^2 = 4$ $\quad u_3(2) = 2^3 = 8$

$u_4(2) = 2^4 = 16$ $\quad u_5(2) = 2^5 = 32$ $\quad u_6(2) = 2^6 = 64$

$u_7(2) = 2^7 = 128$ $\quad u_8(2) = 2^8 = 256$ $\quad u_9(2) = 2^9 = 512$

$u_{10}(2) = 2^{16} = 1024$

2. Démontrez par récurrence forte sur n que $u_n(x) = x^n$. $P(n)$

Base : $\forall x \in \mathbb{N}^*, u_0(x) = 1 = x^0$

Induction : On suppose un $n > 0$ tel que $\forall i \in \mathbb{N}$ tel que $i \leq n, P(i)$ est vérifiée.

$\forall x \in \mathbb{N}^*, u_{n+1}(x) =$

- $\left(u_{\frac{n}{2}}(x)\right)^2$ si $n + 1$ est pair $= \left(x^{\frac{n+1}{2}}\right)^2 = x^{n+1}$

  $\frac{n+1}{2} \in \mathbb{N}$ et $\frac{n+1}{2} \leq n$

- $\left(u_{\frac{n-1}{2}}(x)\right)^2 \times x$ si $n + 1$ est impair $= \left(x^{\frac{n}{2}}\right)^2 \cdot x = x^{n+1}$

  $\frac{n}{2} \in \mathbb{N}$ et $\frac{n}{2} \leq n$

Question 2

Soit la fonction PuissanceRecursive définie comme suit :

```
def PuissanceRecursive (x, n):
        print "n␣vaut", n
        if (n == 0) :
                res = 1
        else :
                res = PuissanceRecursive (x, n/2)
                if (n % 2 == 0) : # Si n est pair
                        res = res*res
                else :
                        res = res*res*x
        print "PuissanceRecursive␣(", x, "," , n, ")=", res
        return res
```

Exécutez PuissanceRecursive(2, 10). Vous préciserez l'arbre des appels et l'état de la pile des exécution au moment où le dernier appel est empilé.

>>> PuissanceRecursive(2, 10)

n vaut 10

n vaut 5

n vaut 2

n vaut 1

n vaut 0

PuissanceRecursive(2, 0) = 1

PuissanceRecursive(2, 1) = 2

PuissanceRecursive(2, 2) = 4

PuissanceRecursive(2, 5) = 32

PuissanceRecursive(2, 10) = 1024

Question 3

Démontez par récurrence la validité et la terminaison de la fonction PuissanceRecursive.

On remarque que « res = res $*$ res » correspond au cas pair de la suite de la question 1 $\left(u_n(x) = \left(u_{\frac{n}{2}}(x)\right)^2\right)$ et « res = res $*$ res $*$ x » correspond au cas impair de la suite $\left(u_n(x) = \left(u_{\frac{n-1}{2}}(x)\right)^2 \times x\right)$.

En utilisant la propriété démontrée précédemment (cf. question 1), on montre la validité de la fonction.

terminaison → similaire à l'exercice 1

TD 4 : Complexité d'un algorithme

Exercice 1. Quelques calculs de complexité d'algorithmes itératifs

Question 1 : Somme des éléments d'un tableau

La fonction somTab retourne la somme des éléments d'un tableau T de nombres :

```
def somTab(T):
    res = 0
    for x in T:
        res = res + x
    return res
```

Exprimer sa complexité en nombre d'additions en fonction de la taille n de T.

L'instruction « res = res + x » correspond à $\mathcal{O}(1)$ additions. Il y a n tours de boucle avec $n = \text{len}(T)$. Donc la complexité est en $\Theta(n)$.

On utilise plutôt la notation de Landau Θ car elle fournit un encadrement de la complexité réelle (c'est la notation de Landau qui donne le plus d'information sur la complexité).

Question 2 : Recherche de l'élément minimum d'un tableau non trié

Soit tab un tableau de n éléments et les entiers d et f tels que $0 \leq d \leq f \leq n - 1$. La fonction RechercheMin retourne l'indice de l'élément minimum de tab entre les indices d et f (voir cours 2).

```
def RechercheMin(tab, d, f):
    imin=d; i=d+1
    while i<=f:
        if tab[i]<tab[imin]:
            imin=i
        i=i+1
    return imin
```

Evaluez la complexité de la fonction RechercheMin en fonction de ses paramètres.

Corps de boucle :

- 1 test
- 1 affectation et 1 addition = 2 instructions

→ exécuté $(f - d)$ fois

Donc la complexité est en $\Theta(f - d)$.

Question 3 : Complexité du tri par sélection itératif

On considère maintenant le tri par sélection dont le code suit :

```
def TriParSelection(tab):
    i=0; n=len(tab)
    while (i!= n):
        k = RechercheMin(tab, i, n-1)
        if (k!=i):
            z = tab[i]
            tab[i]=tab[k]
            tab[k]=z
        i=i+1
```

Quelle est la complexité de la fonction TriParSelection ? Justifiez votre réponse.

Corps de boucle :

- utilisation de RechercheMin → $\Theta(f-d)$ avec $f=n-1$ et $i=i$
- 1 test avec trois instructions et 1 instruction = complexité en $\Theta(1)$

Complexité :

$$C = \underbrace{\sum_{i=0}^{n-1}}_{\text{boucle}} C_i = \sum_{i=0}^{n-1}(n-1-i) \underbrace{=}_{*} \sum_{j=0}^{n-1} j = \frac{n(n-1)}{2}$$

$*$ changement de variable

Donc la complexité est en $\Theta(n^2)$.

Exercice 2. Quelques calculs de complexité d'algorithmes récursifs

Question 1 : Factorielle

La fonction fact retourne la factorielle de n :

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n -1)
```

Exprimer sa complexité en nombre de multiplications en fonction de n.

On a une seule opération par appel → comptons le nombre d'appels récursifs : il y en a n donc la complexité est en $\Theta(n)$.

Question 2 : Recherche récursive du minimum dans un tableau non trié

Soit tab un tableau de n éléments et les entiers d et f tels que $0 \leq d \leq f \leq n-1$. La fonction RechercheMinRec retourne la première position du plus petit élément de T compris entre les indices d et f (inclus).

```
def RechercheMinRec(T,d,f):
    if (d==f):
        return d
    else :
        imin = RechercheMinRec(T, d, f-1)
        if (T[imin] > T[f]):
            return f
        return imin
```

Quelle est la complexité de la fonction RechercheMinRec ? Pour cela, on pourra compter le nombre de comparaisons effectuées.

On compte le nombre de comparaisons effectué.

On fait $m = f - d$ appels récursifs :

$$C = \sum_{m} C_m \qquad \text{avec } m = f - d$$

Dans un appel récursif, on fait une comparaison :

$C_m = 1 + C_{m-1}$ et $C_0 = 0$

$C = f - d$ d'où $\Theta(f - d)$

Question 3 : Recherche récursive d'un élément dans un tableau non trié

La fonction RechercheRec retourne l'indice k maximum dans $\{0, \cdots, n-1\}$ tel que $tab[k] = elem$ (voir cours 3).

```
def RechercheRec(elem, tab, n) :
    if n==0 :
        return -1
    if tab[n-1]==elem :
        return n-1
    return RechercheRec(elem, tab, n-1)
```

Calculez la complexité de RechercheRec dans le meilleur et le pire des cas.

Meilleur des cas : $\Omega(1) \rightarrow$ élément recherché est à la fin du tableau

Pire des cas : $\mathcal{O}(n) \rightarrow$ élément pas dans le tableau

On peut donner la complexité moyenne si on connaît la distribution de l'élément. On peut par exemple considérer une distribution uniforme. Ce n'est pas toujours pertinent car on ne connaît pas forcément cette distribution à l'avance. Donner la complexité meilleur et pire cas est le plus pertinent.