Lycée Faidherbe, 2019-2020

Devoirs d'informatique

MPSI

Version du 19 février 2021

Table des matières

I DS1 **I-1**
1 Questions de cours I-1
2 Problème : répartition des votes I-3
3 Solutions I-7

II DS2 **II-1**
1 Une suite II-1
2 Décomposition de Fibonacci II-2
3 Recherche II-2
4 Solutions II-3

Devoir I

DS1

Recommandations

- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- Tout fonction demandée dans l'énoncé peut être utilisée d ans les questions ultérieures même si elle n'a pas été écrite.
- Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.
- L'essentiel des points est attribué à une bonne *logique* : algorithme, stratégie de boucle, etc.
- Il y a des points si le programme tourne sans erreur, mais ce n'est pas la majorité du barème. L'idée est qu'écrire un code sur papier n'est pas confortable et que, devant un ordinateur, beaucoup d'erreurs seraient immédiates à repérer et à corriger.
- Il peut y avoir quelques points bonus pour une bonne «hygiène informatique» : ne pas faire plusieurs fois le même calcul, éviter de manipuler des valeurs numériques directement, etc

1 Questions de cours

Exercice I.1
Le code ci-dessous devrait permuter les valeurs de `x` *et* `y`*. Que faut-il changer pour qu'il fonctionne correctement ?*

```
x = 1
y = 2
y = x
temp = y
x = temp
```

Exercice I.2
On donne `x=7` *et* `y=3`*. Que valent, respectivement,* `x/y` *et* `x//y` *?*

Exercice I.3
Soit `x` *une variable qui vaut 3. Que vaut* `x>0 and x%2==0` *?*

Exercice I.4
On écrit une fonction pour tester la parité d'un entier `n`.
Qu'est-ce qui ne va pas dans le code ci-dessous ?

```
def est_pair(n):
    if n%2 == 0:
        return False
    else:
        return True
```

Proposer un code qui s'écrit en 2 lignes.

Exercice I.5
Dans le code suivant, que valent `x` *et* `y` *à la fin de l'exécution ?*

```
def f(x):
    x = 2
    x = x + 1
    return x

x = 4
y = f(x)
```

Exercice I.6
Complétez le code de la fonction `factorielle` *ci-dessous, qui doit calculer la factorielle de* `n`.
Les parties à compléter sont remplacées par `&&&&`.

```
def factorielle(n):
    facto = 1
    for i in range(&&&&):
        facto = facto * &&&&
    return facto
```

Exercice I.7
Combien de tours de boucle fait le code ci-dessous ?

```
n = 6
for i in range(n):
    for j in range(i, n):
```

Exercice I.8
Soit une liste `ls = [-8, 3, 5, 0, 3, 7]`. *Que vaut* `ls[2:4]` *?*

Exercice I.9
Écrivez une fonction `maximum` *qui reçoit une liste de nombre* `ls` *en entrée et renvoie la valeur de son élément le plus grand en sortie.*

Exercice I.10
Écrivez une fonction `nb_jours` *qui reçoit en entrée un entier représentant un mois (1 pour janvier, 2 pour février, ...) et renvoie le nombre de jours du mois en sortie.*
On supposera que l'année est normale (non bissextile).
Exemple : `mois(4)` *renvoie 30.*

Exercice I.11
Écrivez une fonction `signe` *qui reçoit un entier* `n` *en entrée, et renvoie* `1` *s'il est strictement positif,* `-1` *s'il est strictement négatif,* `0` *s'il est nul.*

Exercice I.12
Soit la suite définie par $u_{n+1} = u_n^2 + u_n + 2$ *et* $u_0 = 2$. *Écrivez une fonction* `suite_u` *qui reçoit un entier* `n` *et renvoie en sortie la valeur du terme de rang* `n` *de la suite.*

2 Problème : répartition des votes

Présentation du problème

Lors des élections d'assemblées, il existe deux manières de désigner les vainqueurs.

1. Le scrutin majoritaire découpe le corps électoral en petites entités dans chacune desquelles celui qui a le plus de voix remporte le siège. C'est le cas des élections législatives en France.
2. Le scrutin proportionnel attribue les sièges aux différentes listes selon leur nombre de voix. C'est le cas des élections européennes en France

On peut aussi combiner les deux types. C'est le cas des élections municipales en France.

Nous allons nous intéresser au partage des sièges dans le cas de proportionnelles : on doit allouer un nombre fini de sièges qu'on ne peut pas fractionner selon les résultats.
Un exemple simple est celui de l'attribution de 10 sièges à deux listes notées A, B et C : la liste A a réuni 4673 voix, la liste B 3661 voix et la liste C 1666 voix.
Doit-on élire 4 ou 5 candidats de la liste A ? 3 ou 4 de la liste B ? 1 ou 2 de la liste C ?
La réponse dépendra de la règle de partage qui est choisie (le choix, bien entendu, doit intervenir **avant** les élections).
Nous allons donner quelques règles possibles dans ce sujet et en écrire les algorithmes de calcul.

On peut noter que ce problème se retrouve dans d'autres situations quand il s'agit de distribuer des ressources discrètes :

- nombre de députés par état aux USA, en fonction de la population
- affectations de professeurs dans les différentes académies, en fonction du nombre de lycéens
- attributions de centres de calculs aux laboratoires universitaires, en fonction du nombre de thésards ...

Notations

Dans toute la suite les listes électorales sont supposées indexées par les entiers de 0 à $n-1$.
On travaillera avec 2 objets :

1. le nombre de sièges, qui sera représenté par `S`,
2. les voix obtenues par chaque liste candidate qui seront représentées par une liste (tableau) `voix`. La liste électorale d'indice i a reçu `voix[i]` suffrages.

Le nombre de listes candidates n'est connu que par la longueur de `voix`.
L'exemple ci-dessus correspond à `S0 = 10` et `voix0 = [4673, 3661, 1666]`.

2.1 Les incontestables

Si on reprend l'exemple ci-dessus, on voit qu'il y a 10 sièges pour 10000 électeurs donc 1 siège pour 1000 électeurs. La liste A qui rassemble 4673 électeurs doit donc pouvoir avoir au moins 4 sièges, la liste B au moins 3 sièges et la liste C au moins 1. Il restera alors 2 sièges à pourvoir.
On peut généraliser ce calcul.

- On calcule le nombre total de votes valides : `nE`.
- On calcule le quotient électoral : QE.
  C'est le nombre de votes divisé par le nombre de sièges.
- On crée une liste `elus` de même taille que `voix`.
- Pour chaque i, `elus[i]` prend pour valeur la partie entière du rapport entre le nombre de voix de la liste candidate i et le quotient électoral. C'est le nombre d'élus certains.

Voici le squelette de la fonction que l'on va écrire.

```
1  def base(voix, S):
2      n = len(voix)
3      nE = somme(voix)
4      QE = ***********
5      elus = [0]*n
6      for i in range(***):
7          elus[i] = ******
8      return elus
```

Les portions avec des `*` seront remplies et on va définir `somme`.

La fonction partie entière existe sous le nom de `floor` dans le module `math`.

Exercice I.13
Donner l'instruction d'importation du module `math` *qui permet d'utiliser la partie entière par* `m.floor`*.*

Exercice I.14 — Ligne 3
Écrire une fonction `somme(liste)` *qui renvoie la somme des termes de la liste passée en paramètre.*

`somme(voix0)` renvoie `10000`

Exercice I.15 — Lignes 4, 6 et 7
Compléter les ligne 4, le calcul du quotient électoral, 6 et 7, le calcul du nombre d'élus certains.

`base(voix0, S0)` renvoie `[4, 3, 1]`.

2.2 Plus fort reste

Après avoir pourvu les sièges certains, il va rester des sièges non encore attribués, sauf quand tous les résultats des listes candidates sont des multiples entiers du quotient électoral, ce qui est très peu probable. La première idée qui permet de distribuer les sièges restants de manière qui peut être équitable est le **plus fort reste**.

- On attribue les sièges certains.
- On calcule, pour chaque liste candidate, le nombre de voix restantes. On l'obtient en ôtant au nombre de voix obtenues k fois le quotient électoral où k est le nombre de sièges obtenus dans la méthode ci-dessus. Ces restes sont mémorisés dans une liste `reste`.
- On calcule le nombre de sièges à pourvoir `resteS` ; c'est le nombre total de siège moins la somme des nombres de sièges certains pour chaque liste candidate.
- On effectue alors `resteS` fois les opérations
  - on calcule l'indice du reste maximal : i
  - on ajoute un siège à la liste candidate d'indice i
  - on donne la valeur 0 à `reste[i]`.

Dans l'exemple les restes forment la liste `[673, 661, 666]` : on attribue un siège supplémentaire à chacune des listes A et C.
Les résultats sont alors `[5, 3, 2]`

Voici une proposition pour la fonction, obtenue en étendant la fonction `base`

```
1  def plusFortReste(voix, S):
2      n = len(voix)
3      nE = somme(voix)
4      QE = *****
5      elus = [0]*n
6      restes = [0]*n
7      for i in range(*****):
8          elus[i] = *****
9          restes[i] = *****
10     resteS = *****
11     for k in range(resteS):
12         i = indiceMax(restes)
13         elus[i] = *****
14         restes[i] = 0
15     return elus
```

Exercice I.16
Écrire une fonction `indiceMax(liste)` *qui renvoie le premier indice en lequel la liste atteint sa valeur maximale.*

`indiceMax([4, 3, 9, -2, 7, 11, 5, -3, 11, 8])` renvoie `5`.

Exercice I.17
Compléter le programme ci-dessus.

2.3 Paradoxe de l'Alabama

Dans les années 1880, la méthode retenue pour le calcul du nombre de membres du congrès par état était celle du plus fort reste. Après le recensement de 1880, C. W. Seaton, chef de service au bureau de recensement américain, effectua des simulations du nombre de sièges à affecter à chaque État pour des chambres dont la taille varierait entre 275 et 350 sièges. Il découvrit alors un phénomène curieux.
Nous allons illustrer ce phénomène avec un exemple simple. On se donne `voix1 = [4200, 4200, 1400]`

Exercice I.18
Calculer, à la main, `plusFortReste(voix1, 10)` *et* `plusFortReste(voix1, 11)`.
Si on augmente le nombre de sièges, la troisième liste en perd un !

2.4 Plus fort quotient, méthode de Jefferson

Pour éviter le paradoxe précédent, on peut attribuer les sièges restant à pourvoir non pas en fonction des restes mais en fonction des représentativités, on calcule à combien de voix correspondent les élus de chaque liste et on attribue les élus supplémentaires aux listex qui ont le plus grand taux.
Le principal inconvénient est que le taux n'est pas calculable si une liste n'a pas encore d'élus.

La méthode de Jefferson (ou de d'Hondt) modifie le taux calculé.

- On attribue les sièges certains.
- On calcule, pour chaque liste candidate, le quotient du nombre de voix par le nombre de sièges attribués **augmenté de 1**. Ces rapports sont mémorisés dans une liste `quotients`.
- On calcule le nombre de sièges à pourvoir `resteS` ; c'est le nombre total de siège moins la somme des nombres de sièges certains pour chaque liste candidate.
- On effectue alors `resteS` fois les opérations
  - on calcule l'indice du quotient maximal : i
  - on ajoute un siège à la liste candidate d'indice i
  - on change la valeur de `quotients[i]` en tenant compte de l'élu supplémentaire (inutile de modifier les autres quotients, ils sont inchangés).

Voici ce que donne la distribution pour l'exemple initial.

	liste A	liste B	liste C
voix	4673	3661	1666
certains	4	3	1
quotient	934.6	915.25	833.0
élus après le premier calcul	5	3	1
quotient	778.83	915.25	833.0
élus après le deuxième calcul	5	4	1

Exercice I.19
Écrire une fonction `Jefferson(voix, S)` *qui attribue les sièges selon cette méthode.*

2.5 Plus fort quotient, méthode de Huntington-Hill

Une autre méthode est utilisée aux états-unis pour déterminer le nombre d'élus au congrès pour chaque état. Elle est applicable pour des élections si le nombre de listes est faible devant le nombre de sièges. On peut obtenir ce résultat en définissant un seuil de résultat en-dessous duquel une liste n'aura aucun élu.
Elle ne part pas des élus certains.

- On attribue un siège à chaque liste : `elus = [1]*n`
- Il reste `S1 = S - n` sièges à pourvoir.
- On calcule, pour chaque liste candidate, le quotient $\frac{\texttt{voix[i]}}{\sqrt{e_i(e_i+1)}}$ où e_i est la valeur de `elus[i]`. Ces rapports sont mémorisés dans une liste `quotients`.
- On attribue les `S1` sièges un-par-un en l'attribuant à la liste qui a le plus fort quotient et en modifiant ensuite ce quotient.

Voici ce que donne la distribution pour `voix = [4540, 3560, 1900]` et `S = 10`.

	liste A	liste B	liste C
voix	4673	3661	1666
élus	1	1	1
quotient	3304.31	2588.72	1178.04
élus	2	1	1
quotient	1907.74	2588.72	1178.04
élus	2	2	1
quotient	1907.74	1494.60	1178.04
élus	3	2	1
quotient	1348.98	1494.60	1178.04
élus	3	3	1
quotient	1348.98	1056.84	1178.04
élus	4	3	1
quotient	1044.91	1056.84	1178.04
élus	4	3	2
quotient	1044.91	1056.84	680.14
élus	4	4	2

Exercice I.20
Écrire une fonction `HH(voix, S)` *qui attribue les sièges selon cette méthode.*

3 Solutions

Solution de l'exercice I.1 - À la ligne 3, la valeur de `y` est écrasée avant d'avoir été sauvegardée dans `temp`, donc à la fin `x` et `y` valent toutes les deux 1. Il faut donc permuter les lignes 3 et 4.

```
x = 1
y = 2
temp = y
y = x
x = temp
```

Solution de l'exercice I.2 - `x/y` réalise la division en flottant et renvoie donc `2.33`. `x//y` renvoie le quotient de la division euclidienne, donc `2`.

Solution de l'exercice I.3 - `x` est positive mais pas paire, donc ceci renvoie `False`.

Solution de l'exercice I.4 - Le code a inversé les réponses : il renvoie `False` quand `n` est pair. Donc il faut permuter les deux `return`.

```
def est_pair(n):
    return n%2 == 0:
```

Solution de l'exercice I.5 - La variable `x` créée ligne 6 est une variable globale. Celle créée ligne 2 dans la fonction est une variable locale. Dans une fonction, les variables locales sont prioritaires sur les variables globales, donc `x` globale est masquée par `x` locale jusqu'à la fin de la fonction. Donc la ligne 3 ajoute 1 à `x` locale, et la fonction renvoie 3. À la fin de l'exécution, `x` vaut donc toujours 4, et `y` vaut 3.

Solution de l'exercice I.6 -

```
def factorielle(n):
    facto = 1
    for i in range(i):
        facto = facto * (i+1)
    return facto
```

Une autre solution possible est

```
n = 6
for i in range(n):
    for j in range(i, n):
```

Solution de l'exercice I.7 - La boucle extérieure fait n tours, et la boucle intérieure $n-i$ tours. Donc le nombre total de tours est :

$$N = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} 1 = \sum_{i=0}^{n-1}(n-i) = n^2 - \frac{n(n-1)}{2} = \frac{n(n+1)}{2} = 21.$$

Solution de l'exercice I.8 - C'est une liste à $4-2=2$ éléments qui vaut `[5, 0]`.

Solution de l'exercice I.9 - On commence par désigner le premier élément de la liste comme «candidat maximum», puis on parcourt la liste en mettant à jour ce candidat quand on en trouve un plus grand :

```
def maximum(ls):
    max_temp = ls[0]
    for e in ls:
        if e > max_temp:
            max_temp = e
    return max_temp
```

Solution de l'exercice I.10 - Les différents cas étant exclusifs entre eux, utilisez `elif`.

```
def nb_jours(mois):
    if mois == 2:
        return 28
    elif mois in [4, 6, 9, 11]:
        return 30
    else:
        return 31
```

Une autre solution

```
def nb_jours(mois):
    jours = [31, 28, 31, 30, 31, 31, 30, 31, 30, 31, 30, 31]
    return jours[mois + 1]
```

Solution de l'exercice I.11 - Là encore, utilisez `elif`.

```
def signe(n):
    if n > 0:
        return 1
    elif n < 0:
        return -1
    else:
        return 0
```

Solution de l'exercice I.12 - Comme on ne veut que le terme de rang `n`, il n'est pas utile de stocker tous les termes précédents. On peut donc minimiser le nombre de variables utilisées en «recyclant» la même variable d'un tour de boucle à l'autre.

```
def suite_u(n):
    u = 2
    for _ in range(1, n+1):
        u = u**2+u+2
    return u
```

Solution de l'exercice I.13 -

```
import math as m
```

Solution de l'exercice I.14 -

```
def somme(liste):
    n = len(liste)
    som = 0
    for i in range(n):
        som = som + liste[i]
    return som
```

ou

```
def somme(liste):
    som = 0
    for x in liste:
        som = som + x
    return som
```

Solution de l'exercice I.15 -

```
def base(voix, S):
    n = len(voix)
    nE = somme(voix)
    QE = nE/S
    elus = [0]*n
    for i in range(n):
        elus[i] = m.floor(voix[i]/QE)
    return elus
```

Solution de l'exercice I.16 -

```
def indiceMax(liste):
    n = len(liste)
    maxi = 0
    i_max = 0
    for i in range(1, n):
        if liste[i] > maxi:
            maxi = liste[i]
            i_max = i
    return i_max
```

Solution de l'exercice I.17 -

```
def plusFortReste(voix, S):
    n = len(voix)
    nE = somme(voix)
    QE = nE/S
    elus = [0]*n
    restes = [0]*n
    for i in range(n):
        elus[i] = m.floor(voix[i]/QE)
        restes[i] = voix[i] - elus[i]*QE
    resteS = S - somme(elus)
    for k in range(resteS):
        i = indiceMax(restes)
        elus[i] = elus[i] + 1
        restes[i] = 0
    return elus
```

Solution de l'exercice I.18 -

```
plusFortReste(voix1, 10) -> [4, 4, 2]
plusFortReste(voix1, 11) -> [5, 5, 1]
```

Solution de l'exercice I.19 -

```
def Jefferson(voix, S):
    n = len(voix)
    nE = somme(voix)
    QE = nE/S
    elus = [0]*n
    quotients = [0]*n
    for i in range(n):
        elus[i] = m.floor(voix[i]/QE)
        quotients[i] = voix[i]/(elus[i] + 1)
    resteS = S - somme(elus)
    for k in range(resteS):
        i = indiceMax(quotients)
        elus[i] = elus[i] + 1
        quotients[i] = voix[i]/(elus[i] + 1)
    return elus
```

Solution de l'exercice I.20 -

```
def HH(voix, S):
    n = len(voix)
    elus = [1]*n
    quotients = [0]*n
    for i in range(n):
        quotients[i] = voix[i]/2**0.5
    S1 = S - n
    for k in range(S1):
        i = indiceMax(quotients)
        elus[i] = elus[i] + 1
        e = elus[i]
        quotients[i] = voix[i]/(e*(e+1))**0.5
    return elus
```

Devoir II

DS2

1 Une suite

On considère la suite $(S_n)_{n\in\mathbb{N}^*}$ définie par $S_n = \sum_{k=1}^{n} \frac{1}{3k+1}$.
On a $S_1 = 0,25$, $S_2 \simeq 0,39$, $S_3 \simeq 0,49$, $S_4 \simeq 0,57$, $S_5 \simeq 0,63$.

Exercice II.1
Écrire une fonction `S(n)` *qui renvoie le terme* S_n.

Exercice II.2
Combien d'opérations (additions, multiplications, divisions) sont effectuées lors du calcul de `S(n)` *?*

On prouve en mathématiques que la suite tend vers $+\infty$.
On veut trouver le premier entier n tel que $S_n \geqslant A$.
On propose la fonction suivante.

```
def seuil(A):
    n = ??
    ??? S(n) < A:
        n = ???
    return n
```

Exercice II.3
Compléter le code.

Exercice II.4
Si le résultat de `seuil(A)` *est* N*, déterminer, en fonction de* N*, le nombre d'opérations effectuées. On comptera bien sur les opérations effectuées pour tous les calculs des* $S(n)$.

Exercice II.5
Écrire la fonction `seuil(A)` *d'une autre manière (sans utiliser la fonction* `S`*) afin que le nombre d'opérations soit plus petit : il sera majoré par* $4N$ *si le résultat de* `seuil(A)` *est* N.

Dans la partie suivante on pourra utiliser le fait que le dernier terme d'une liste est déterminé par l'indice -1 (et l'avant dernier par l'indice -2). Par exemple, pour `L = [7, 4, 3, 5, 8]`, `L[-1]` renvoie `8` et `L[-1]` renvoie `5`.

2 Décomposition de Fibonacci

Rappel : la suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par

$$F_0 = 0,\ F_1 = 1 \text{ et } F_{n+2} = F_{n+1} + F_n \text{ pour } n \in \mathbb{N}$$

Exercice II.6
Écrire une fonction `fibo(n)` *qui renvoie une liste contenant les nombres de Fibonacci* F_0, F_1, …, F_n. *Cette liste sera de longueur* $n+1$.

`fibo(14)` doit renvoyer `[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]`.

Exercice II.7
Écrire une fonction `petitsFibo(n)` *qui renvoie une liste contenant les nombres de Fibonacci* F_0, F_1, …, F_p *inférieurs à* n.
On ne connaît pas la longueur de cette liste à l'avance. On pourra supposer qu'on a $n \geqslant 1$.

`petitsFibo(200)` doit renvoyer `[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]`.

On appelle $\phi(n)$ le dernier élément de la liste `petitsFibo(n)`. C'est le plus grand nombre de Fibonacci inférieur ou égal à n, c'est aussi l'unique nombre de Fibonacci F_p tel que $F_p \leqslant n < F_{p+1}$.

Exercice II.8
Prouver que, si $\phi(n) = F_p$, *alors* $n - \phi(n) < F_{p-1}$.
En déduire que, pour tout entier $n \geqslant 1$ *il existe des indices* i_1, i_2, …, i_p *tels que*
- $n = F_{i_1} + F_{i_2} + \cdots + F_{i_p}$,
- $2 \leqslant i_1 \leqslant i_2 - 2$, $i_2 \leqslant i_3 - 2$, …, $i_{p-1} \leqslant i_p - 2$.

On pourra utiliser une récurrence généralisée.
On appelle décomposition de Fibonacci de n *une suite* $(F_{i_1}, F_{i_2}, \ldots, F_{i_p})$ *vérifiant ces propriétés.*

Exercice II.9
Écrire une fonction `decomp(n)` *qu renvoie la suite des nombres de Fibonacci d'une décomposition de Fibonacci de* n. *On pourra renvoyer le résultat sous forme d'une liste décroissante.*

3 Recherche

À partir d'une liste `L` dont les éléments sont triés par ordre croissant et d'un élément `a` on veut trouver l'indice i_0 tel que `L[i] <= a < L[i+1]`.
Par exemple pour `L0 = [4, 8, 14, 14, 18, 19, 26, 41]` et `a = 20`, la réponse est $i_0 = 5$
et pour `a = 14`, la réponse est $i_0 = 3$.
Dans les questions on supposera que la liste est triée et qu'on a `L[0] <= a < L[-1]`.

Exercice II.10
Proposer une fonction `position(L, a)` *qui calcule* i_0 *en comparant* `L[i]` *à* `a` *tant qu'on a* `L[i] <= a`.

Exercice II.11
Proposer une fonction `position1(L, a)` *qui calcule* i_0 *beaucoup plus rapidement en s'inspirant de la recherche par dichotomie dans une liste triée.*
Avec l'exemple ci-dessus `position1(L0, a)` *n'effectuera que 3 comparaisons.*

Exercice II.12
Prouver que votre fonction `position1` *renvoie bien le résultat souhaité.*
Estimez le nombre de comparaisons effectuées.

4 Solutions

Solution de l'exercice II.1 - 2 points

```
def S(n):
    u = 0
    for k in range(1, n+1):
        u = u + 1/(3*k+1)
    return u
```

Solution de l'exercice II.2 - 1 point
On effectue n passages dans la boucle, chacun demande 4 opérations. Il y a donc $4n$ opérations.

Solution de l'exercice II.3 - 1 point

```
def seuil(A):
    n = 1
    while S(n) < A:
        n = n + 1
    return n
```

Solution de l'exercice II.4 - 2 points
On calcule `S(n)` pour $1 \leqslant n \leqslant N$ d'où $\sum_{n=1}^{N} 4n = 2N^2 + 2N$ opérations et on ajoute 1 à n pour $1 \leqslant n < N$. Il y a donc $2N^2 + 3N - 1$ opérations en tout.

Solution de l'exercice II.5 - 2 points

```
def seuil1(A):
    n = 1
    S = 1/4
    while S < A:
        n = n + 1
        S = S + 1/(3*n+1)
    return n
```

On effectue $4N - 3$ opérations.

Solution de l'exercice II.6 - 2 points

```
def fibo(n):
    F = [0]*(n+1)
    F[0] = 0
    F[1] = 1
    for i in range(2,n+1):
       F[i] = F[i-1] + F[i-2]
    return F
```

Solution de l'exercice II.7 - 2 points

```
def petitsFibo(n):
    F = [0, 1]
    while F[-1] + F[-2] <= n:
        F.append(F[-1] + F[-2])
        n = n + 1
    return F
```

Solution de l'exercice II.8 - 2 points
On a $\phi(n) = F_p \leqslant n < F_{p+1}$ donc $0 \leqslant n - \phi(n) < F_{p+1} - F_p = F_{p-1}$.
1 admet la décomposition $1 = F_2$.
On suppose que tout entier k vérifiant $1 \leqslant k \leqslant n-1$ pour $n \geqslant 2$ admet une décomposition de Fibonacci.
On a $n \geqslant 2$ donc $\phi(n) \geqslant 2$ et $k = n - \phi(n) \leqslant n - 2 \leqslant n - 1$;

- si $k = 0$, $n = \phi(n) = F_p$ est une décomposition de Fibonacci de n,
- si $k \geqslant 1$, il admet une décomposition de Fibonacci $k = F_{i_1} + F_{i_2} + \cdots + F_{i_q}$ avec $F_{i_q} \leqslant k < F_{p-1}$ donc $i_q \leqslant p-2$ et $n = F_{i_1} + F_{i_2} + \cdots + F_{i_q} + F_p$ est bien une décomposition de Fibonacci de n car les inégalités sont vérifiées pour les i_k et $i_q \leqslant p-2$.

Solution de l'exercice II.9 - 2 points

```
def decomp(n):
    decFibo = []
    k = n
    while k > 0:
        L = petitsFibo(k)
        phi = L[-1]
        decFibo.append(phi)
        k = k - phi
    return decFibo
```

Solution de l'exercice II.10 - 2 points

```
def position(L, a):
    i = 0
    while L[i] <= a:
        i = i + 1
    return i - 1
```

Solution de l'exercice II.11 - 2 points

```
def position1(L, a):
    av = 0
    ap = len(L) - 1
    while ap - av > 1:
        c = (av + ap)//2
        if L[c] > a:
            ap = c
        else:
            av = c
    return av
```

Solution de l'exercice II.12 - 2 points
On a toujours `L[av] <= a < L[ap]`.
Si la longueur est majorée par 2^p, on effectue au plus p comparaisons.