Evaluation de l'Algorithmique au lycée

version 0.9 février 2012
IREM Clermont Fd

Les retours des collègues confrontés à l'enseignement de l'algorithmique dans les classes de lycée, et la lecture des activités produites, semblent mettre en évidence des difficultés à évaluer cette partie dans le cadre du programme de mathématiques.
Que doit-on évaluer ? Comment l'évaluer ? Comment rendre cette évaluation constructive pour l'élève ? Comment placer cette évaluation au sein d'une évaluation de mathématiques ?...
Cette liste de questions n'est pas exhaustive et beaucoup d'interrogations de différents ordres se sont présentées aux enseignants de Mathématiques. Nous n'avons pas la prétention d'apporter une réponse tranchée à chacune de ces questions ni même d'en faire l'inventaire complet. Cependant nous espérons, tout de même, apporter quelques éléments de réflexion sur lesquels appuyer la construction de séquences pédagogiques autour de l'algorithmique.
Dans un premier temps, nous pensons important, de mettre en relief plusieurs axes à développer, axes avec des dépendances fortes.

Une évaluation de l'algorithmique au lycée :
- *Quoi ?*
- *Comment ?*
- *Objectif et place dans l'enseignement de mathématiques ?*

Nous allons essayer d'apporter quelques premiers éléments de réflexion :

Quoi ?

La réponse à cette première question va fortement influencer le contenu des autres réponses, mais il semble acquis auprès des collègues que les compétences suivantes doivent être ciblées par une évaluation :

✓ **Lire un algorithme**

Être capable de lire un algorithme est une compétence qui nécessite de comprendre les concepts entrant en jeu dans son écriture. Ces notions sont clairement signalées dans le texte du programme officiel, et entrent dans le cadre d'une progression établie sur l'année de seconde. Elles sont ensuite confortées dans le cycle terminal. Cette compétence devra donc faire l'objet d'une évaluation permanente, et non limitée au début de l'apprentissage, évaluation d'autant plus importante qu'elle va permettre de faire un choix entre différentes stratégies d'apprentissage comme nous le verrons par la suite.

✓ Exécuter un algorithme

Cette compétence vient en complément de la précédente. Elle est aussi une étape essentielle dans la phase d'appropriation d'un algorithme. Pour s'assurer de la compréhension, et donc d'une lecture juste, d'un algorithme, l'exécution « à la main » de celui-ci est un excellent indicateur.
Devant un algorithme difficile à appréhender, nous avons tous ressenti le besoin de le « faire tourner à la main » [1].
Notre expérience en tant qu'enseignant nous permet de mettre en place cette stratégie de façon naturelle. Ce point va s'imposer beaucoup moins naturellement aux élèves, il apparaît indispensable d'insister dessus, et de leur transmettre cette habitude. Le gain ne se limitera pas alors à l'algorithmique mais aura des chances de trouver des répercussions dans l'apprentissage des mathématiques en général.

✓ Comprendre un algorithme

Lire et exécuter un algorithme est possible, sans forcément en comprendre le sens et la fonction.
« Que fait cet algorithme ? », « Pourquoi avoir choisi cette structure ? », « Quel est le rôle de cette instruction ? » etc...
Nous avons là des questions qui ne relèvent plus du simple cadre de la lecture ou de l'exécution d'un algorithme mais qui en demande une

1. Le choix de cette formulation, que nous retrouvons dans les retours des collègues n'est pas anodine, et soulève bien l'importance de cette étape dans le processus d'apprentissage. On y retrouve l'idée de l'apprentissage par l'essai et aussi par l'observation rendant l'élève plus actif.

analyse plus critique.
Cette compétence va ainsi faire intervenir des capacités d'analyse portant sur :

- **la correction d'un algorithme**
  Il n'est évidemment pas raisonnable de présenter des preuves de correction d'algorithmes aux élèves. Cependant, sans utiliser ces outils, nous arrivons à relever des erreurs dans un algorithme. Pour cela, plusieurs mécanismes entrent en jeu. Notre expérience nous permet de repérer des erreurs dans les endroits critiques, ou, en tout cas, là où elles se produisent fréquemment. Nous avons aussi, par notre vécu, acquis de bonnes pratiques améliorant l'efficacité de cette démarche. À ce sujet, la question « Pourquoi cet algorithme est-il faux ? » est souvent plus riche d'enseignement que la question « Pourquoi cet algorithme est-il correct ? ».
  Mais, malgré ces précautions, il n'est pas rare, après passage sur une machine, de découvrir d'autres erreurs [2]. Ce passage dans un langage de programmation est important dans le cadre de la formation et peut déboucher sur des questions intéressantes, comme nous le verrons dans la suite.
- **la terminaison d'un algorithme**
  Ici encore, sans faire de développement de cette notion, il est intéressant que les élèves se posent ce genre de questions : « L'exécution de cet algorithme se termine-t-elle ? », « Quel argument nous permet de penser que cet algorithme va rendre un résultat ? » etc ...
  Ces questions sont d'autant plus intéressantes, que, comme nous le verrons, elles vont pouvoir servir de support à des notions mathématiques enseignées dans nos classes.
- **l'efficacité d'un algorithme**
  Nous utilisons volontairement ce terme plutôt que celui de complexité d'un algorithme. Il n'est pas question dans les classes de lycée d'entrer dans un développement de la notion de complexité d'un algorithme, mais les élèves peuvent déjà prendre conscience, que si deux algorithmes effectuent la même tâche, ils ne le font pas de la même façon. Des questions peuvent être posées sur le nombre d'instructions réalisées par un algorithme, sur la nécessité d'un test donné, etc ...
  Nous utiliserons donc le terme d'efficacité afin d'éviter des confusions éventuelles avec la notion de complexité d'un algorithme.

2. Ne perdons pas de vue que l'exécution correcte d'un algorithme sur quelques valeurs n'est pas la garantie d'un algorithme correct.

✓ **Ecrire un algorithme**

Les compétences précédentes sont un passage obligé avant de pouvoir écrire un algorithme.
En fin de terminale, nous pensons raisonnable d'être en mesure de demander à un élève l'écriture d'un algorithme simple relevant d'un problème classique.
Par exemple : « Écrire un algorithme retournant la somme des entiers de 1 à n, où n est un entier donné. », « Déterminer le plus petit entier naturel n, tel que $u_n > A$ avec A un réel donné et (u_n) une suite, tendant vers $+\infty$, donnée. » etc ...
Si ces exercices semblent accessibles aux élèves, il en est autrement de l'écriture d'un algorithme plus complexe, et une évaluation, dans ce cas, ne peut se faire que dans un cadre très particulier :
- temps suffisant pour son écriture. Ainsi, dans le cadre d'algorithmes non triviaux, cet exercice semble être exclu en temps limité ;
- maîtrise et recul suffisants par rapport aux concepts enseignés. Ce qui laisse entendre que ce type d'exercice, s'il est mis en place, ne peut l'être qu'en fin d'un cycle de formation ;

Cette forme d'exercice soulève des difficultés importantes. On peut, par contre, envisager des variantes [3] plus adaptées :
- algorithmes « à trous » à faire remplir par les élèves [4] ;
- réécriture d'un algorithme en imposant certaines conditions ;
- etc...

Nous illustrerons ces points par différents exemples dans la suite.

Comment évaluer ces compétences ?

Les compétences précédentes étant ciblées, nous pouvons présenter quelques exemples de stratégies permettant de les évaluer. Nous rappelons qu'il ne s'agit pas d'imposer une pratique, mais seulement de donner des pistes de réflexion, afin d'aider les collègues dans la construction de leur enseignement.

3. On peut d'ailleurs constater, que si les premières activités algorithmiques produites par les collègues, faisaient figurer quelques fois l'écriture complète d'un algorithme, les plus récentes, elles, ne le font plus.

4. Ici, la frontière entre comprendre un algorithme et écrire un algorithme devient difficile à percevoir.

En fonction du contenu, les stratégies à mettre en œuvre vont s'avérer différentes. Concernant la lecture d'un algorithme, de simples exercices de reconnaissance pourront tout à fait convenir. On peut, dans un algorithme donné par exemple, faire reconnaître les instructions liées à des affectations, les blocs d'instructions liés à des structures conditionnelles ou des structures itératives. Concrètement, la présentation d'un algorithme de ce genre :

```
lire n
c ← 0
pour k allant de 1 à n faire
    | si k divise n alors c ← c + 1
retourner c
```

peut donner lieu à ces questions :

- « *souligner les intructions liées à une affectation* »
- « *encadrer la (ou les) structures itératives.* »

etc...

Ces questions vont constituer une partie importante de l'évaluation en début de formation mais ne seront pas à négliger non plus en cours et en fin de formation. Si elles n'apparaissent pas toujours dans les activités proposées par les enseignants, elles font bien souvent l'objet d'un traitement à l'oral. Un passage à l'écrit, cependant, ne nous semble pas superflu. Il permet en tout cas de poser un premier diagnostique fort utile.

Le fait de savoir exécuter un algorithme est par contre une compétence davantage ciblée dans les activités. Les formes retenues, le plus souvent, consistent à demander une exécution « à la main » d'un algorithme donné.

Par exemple, toujours à partir de l'algorithme :

```
lire n
c ← 0
pour k allant de 1 à n faire
    | si k divise n alors c ← c + 1
retourner c
```

On peut considérer la question suivante :

« Quelle valeur retourne cet algorithme, s'il est exécuté pour $n = 5$? »

Un passage à la programmation dans un logiciel pourvu d'un mode « debug » est intéressant ici. Il permet de d'illustrer, ce que représente en termes de calculs, un algorithme de ce genre.
L'avantage est de permettre à l'élève de se faire une représentation plus concrète de l'exécution d'un algorithme, que l'on pourrait qualifier de basique. La représentation que s'en font les élèves est souvent très abstraite et il ne faut pas hésiter, lorsque c'est possible, à illustrer ces notions, et montrer que ces algorithmes correspondent à une réalité.
Il faut aussi garder à l'esprit qu'un élève peut être capable d'exécuter, pas à pas, un algorithme sans forcément en comprendre sa fonction. C'est pourquoi, il nous apparait nécessaire de bien distinguer cette compétence de celle relevant de la compréhension d'un algorithme.
On peut, par exemple, modifier l'exercice précédent pour aller dans ce sens de la façon suivante :

Contexte de l'activité : établir, puis démontrer, une conjecture sur la parité du nombre de diviseurs d'un entier.

On se propose de mettre en œuvre un algorithme qui, pour un entier naturel n, non nul, choisi par l'utilisateur, affiche le nombre de diviseurs de n.
On donne quatre versions :

lire n
$c \leftarrow 0$
pour k **allant de** 1 **à** n **faire**
  si k *divise* n **alors** $c \leftarrow c+1$
afficher c

Algorithme 1 :

lire n
$c \leftarrow 2$
pour k **allant de** 2 **à** $n-1$ **faire**
  si k *divise* n **alors** $c \leftarrow c+1$
afficher c

Algorithme 2 :

lire n
$c \leftarrow 1$
pour k **allant de** 2 **à** n **faire**
  si k *divise* n **alors** $c \leftarrow c+1$
afficher c

Algorithme 3 :

lire n
$c \leftarrow 1$
pour k **allant de** 1 **à** $n-1$ **faire**
  si k *divise* n **alors** $c \leftarrow c+1$
afficher c

Algorithme 4 :

Parmi ces quatre versions, une ne répond pas au problème. Laquelle et pourquoi ?

On peut ensuite compléter par une question portant sur l'efficacité de l'algorithme :

Des trois versions correctes, laquelle semble la plus judicieuse ? Justifier.

L'exercice nécessite de vraiment se plonger dans la lecture et l'exécution des algorithmes. Les élèves, pour pouvoir répondre, vont devoir non seulement exécuter ces algorithmes mais aussi en faire une analyse plus critique, analyse renforcée par la deuxième question.
D'autre part, ce genre d'exercice a l'intérêt de présenter les choses sous forme de défi. Lors de cette séquence, on a pu constater que les élèves se prenaient réellement au jeu, mettant en place des stratégies différentes pour trouver la bonne réponse.
On peut aussi, au travers d'exercices du type « algorithmes à trous », s'assurer de la compréhension de points précis.
Par exemple, dans cet exercice :

Compléter l'algorithme suivant afin qu'il retourne la somme des entiers naturels de 0 à n, pour n donné :

lire n
$s \leftarrow 0$
pour k **allant de** 0 **à** n **faire**
    $s \leftarrow$.....
retourner s

De cette façon, on met l'accent sur le rôle d'une variable d'accumulation, tout en commençant à mettre en place certains automatismes, et certains algorithmes de base que l'élève pourra, par la suite, exploiter dans des situations plus complexes.
On peut également demander aux élèves de corriger un algorithme comportant une erreur.
Ce qui donnerait dans le cas précédent :

Corriger l'algorithme suivant afin qu'il retourne la somme des entiers naturels de 0 à n, pour n donné :

lire n
$s \leftarrow 0$
pour k **allant de** 0 **à** n **faire**
    $s \leftarrow s+1$
retourner s

La difficulté supplémentaire est ici de trouver l'erreur pour pouvoir la corriger. Il est intéressant dans ce type d'exercice de chosir des erreurs souvent relevées dans les copies. Il s'agira tout de même de rester raisonnable quant à la difficulté des corrections demandées.
Les exercices précédents sont aussi une bonne préparation à l'écriture d'algorithmes. Cependant le manque de recul dans l'enseignement de l'algorithmique au lycée et la frontière, souvent mince, entre les mathématiques et l'algorithmique, dans les activités proposées, vont engendrer des difficultés supplémentaires dans l'évaluation des algotihmes écrits par les élèves. Face à des réponses érronées, il n'est pas toujours aisé de cibler le problème. L'erreur est-elle engendrée par des difficultés de compréhension en mathématiques ou [5] en algorithmique ?
Deux exemples, relevés dans des copies :

Contexte : on demande un algorithme permettant de déterminer le plus petit rang n pour lequel $u_n \leq 65$*, où* (u_n) *est la suite définie par* $u_0 = 100$ *et* $u_{n+1} = \frac{1}{2}u_n + 30$.

Variables
n est du type nombre
u est du type nombre
Début algorithme
n prend la valeur 0
u prend la valeur 100
si $u_n > 65$ **alors**
    u prend la valeur $\frac{1}{2}u + 30$
    n prend la valeur $n+1$
Afficher n

5. « ou » inclusif, bien sûr !

Variables
n est du type nombre
u_n est du type nombre
Début algorithme
n prend la valeur 0
u_n prend la valeur $60+40\times\left(\frac{1}{2}\right)^n$ [a]
Si $u_n > 65$ **alors** u_n prend la valeur $60+40\times\left(\frac{1}{2}\right)^n$
Afficher n

a. cette expression est le résultat de l'une des questions de l'exercice.

Plusieurs remarques viennent à l'esprit et nous reviendrons ultérieument sur la forme des ces algorithmes, conséquence d'habitudes prises sous le logiciel ALGOBOX.
Intéressons nous au contenu, et, en particulier, à la structure itérative qui n'est mise en place dans aucun de ces algorithmes.
Face à ces productions, la première réaction a été de penser que les élèves ne maîtrisaient pas ces structures et donc que le problème ne relevait que de l'algorithmique. Après avoir intérrogé les élèves, il s'est avéré que le problème était plutôt la conséquence d'une mauvaise compréhension des suites définies par récurrence.
La structure itérative, à mettre en place ici, est fortement liée à la définition de la suite. Ainsi, des difficultés en mathématiques vont se traduire immédiatement par de mauvaises traductions algorithmiques. Dans ce cas précis, il a été intéressant d'opérer une remédiation sur les deux plans, à savoir en algorithmique et sur les suites définies par récurrence. Le passage à un algorithme a permis de cibler une difficulté en mathématiques, et le fait de pouvoir exécuter les algorithmes, produits par les élèves, leur a permis de prendre conscience de leurs erreurs de raisonnement. À charge, ensuite, à l'enseignant d'organiser leur réflexion afin de corriger ces erreurs.
Dans les exemples précédents, les difficultés des élèves étaient visibles mais une réponse exacte n'est pas toujours la preuve d'une bonne compréhension des notions abordées par l'élève. En tout cas elle n'en est pas la garantie, et c'est pourquoi dans tous les cas de figure, l'analyse de ces réponses ne pourra être complète qu'à partir de questions judicieusement élaborées.
Sans vouloir en dresser une liste exhaustive, on peut présenter quelques stratégies souvent utilisées en situation :

Une question, dans laquelle on demande ce que produit l'algorithme si on remplace une instruction par une autre, permet à l'élève de prendre conscience du rôle et de la portée de cette même instruction.

Par exemple, considérons un algorithme de la méthode de Monté-Carlo appliquée à l'estimation du nombre de triplets (a,b,c), de l'ensemble $\mathscr{E} = \{0,1,2,\ldots,9\}$, vérifiant $a < b < c$:

$S \leftarrow 0$
pour k **allant de** 1 **à** 10^4 **faire**
  $a \leftarrow$ entier aléatoire compris entre 0 et 9
  $b \leftarrow$ entier aléatoire compris entre 0 et 9
  $c \leftarrow$ entier aléatoire compris entre 0 et 9
  si $a < b$ *et* $b < c$ **alors** $S \leftarrow S + 1$
retourner $\dfrac{S}{10^4} \times 10^3$

Les questions, que l'on peut tirer d'un tel algorithme sont très nombreuses :

- *« Que se passe-t-il si on modifie l'ordre des instructions de la façon suivante : »*

$S \leftarrow 0$
$a \leftarrow$ entier aléatoire compris entre 0 et 9
$b \leftarrow$ entier aléatoire compris entre 0 et 9
$c \leftarrow$ entier aléatoire compris entre 0 et 9
pour k **allant de** 1 **à** 10^4 **faire**
  si $a < b$ *et* $b < c$ **alors** $S \leftarrow S + 1$
retourner $\dfrac{S}{10^4} \times 10^3$

- *« Que se passe-t-il si on supprime l'instruction $S \leftarrow S + 1$? »*

D'autres questions, impliquent davantage le programme de mathématiques :
On peut, par exemple, donner cet algorithme, et demander ce qu'il va produire :

$S \leftarrow 0$
pour k **allant de** 1 **à** 10^4 **faire**
    $a \leftarrow$ entier aléatoire compris entre 0 et 9
    $b \leftarrow$ entier aléatoire compris entre 0 et 9
    $c \leftarrow$ entier aléatoire compris entre 0 et 9
    si $a < b$ *et* $b < c$ **alors** $S \leftarrow S + 1$
retourner $\dfrac{S}{10}$

☞ *La dernière instruction est donnée volontairement sous forme simplifiée.*
Ou encore, poser cette question :

« L'algorithme précédent, ne permet pas une estimation acceptable du nombre recherché, que peut-on modifier dans cet algorithme pour remédier à ce problème ? »

Ici, on a une question en rapport direct avec la méthode de Monté Carlo. Afin de placer l'élève en situation, il ne faut pas hésiter à passer à une programmation de l'algorithme. Le recours, à un ordinateur et à des essais successifs va permettre à l'élève de prendre conscience des problèmes mathématiques soulevés dans l'activité (estimation, loi des grands nombres, etc ...). À penser que ce soit encore nécessaire d'en fournir, nous avons encore un exemple de ce que peut apporter l'algorithmique à notre enseignement de mathématiques.
D'autres questions vont relever à la fois du cadre du programme de mathématiques et d'algorithmique :
Par exemple :

« Compléter l'algorithme suivant, afin qu'il retourne une estimation du nombre de triplets recherché : »

$S \leftarrow 0$
pour k **allant de** 1 **à** 10^4 **faire**
    $a \leftarrow$ entier aléatoire compris entre 0 et 9
    $b \leftarrow$ entier aléatoire compris entre 0 et 9
    $c \leftarrow$ entier aléatoire compris entre 0 et 9
    si ... **alors** $S \leftarrow S + 1$
retourner $\dfrac{S}{10^4} \times 10^3$

La réponse attendue va porter sur la clause du test qui, elle-même, ne peut être bien perçue que si la méthode de Monté Carlo est comprise de l'élève.

Si on souhaite ne faire porter la question que sur ce test, en se débarassant du problème mathématique, on peut alors demander une réécriture de l'algorithme en utilisant des structures conditionnelles imbriquées, ou inversement. L'objectif étant de passer de l'une de ces deux versions à l'autre :

$S \leftarrow 0$
pour k **allant de** 1 **à** 10^4
faire
  …
  si $a < b$ **alors**
    si $b < c$ **alors**
      $S \leftarrow S+1$
retourner $\dfrac{S}{10^4} \times 10^3$

$S \leftarrow 0$
pour k **allant de** 1 **à** 10^4
faire
  …
  si $a < b$ *et* $b < c$ **alors**
    $S \leftarrow S+1$
retourner $\dfrac{S}{10^4} \times 10^3$

Le passage d'une structure à l'autre, en imposant le type d'instruction à utiliser, oblige les élèves à vraiment analyser certaines parties de l'algorithme. Il permet d'insister sur des points en particulier. Il a aussi l'avantage que les élèves se prennent assez bien au jeu, et apprécient le côté ludique de ce genre de questions.
Toujours sur cette idée, on peut reconsidérer totalement le problème et conduire les élèves à un algorithme non plus probabiliste mais déterministe. L'utilisation conjointe de structures itératives et conditionnelles va s'avérer nécessaire pour résoudre le problème. Les élèves arrivent assez facilement à un premier algorithme de ce genre :

$n \leftarrow 0$
pour a **allant de** 0 **à** 9 **faire**
  pour b **allant de** 0 **à** 9 **faire**
    pour c **allant de** 0 **à** 9 **faire**
      si $a < b$ *et* $b < c$ **alors** $n \leftarrow n+1$
retourner n

Il s'agit de réaliser l'inventaire des triplets, vérifiant la relation donnée. L'exercice devient encore plus intéressant lorsque les élèves se rendent compte que beaucoup de passages dans les structures itératives sont inutiles, et génèrent des tests pour rien. L'efficacité, au sens où nous l'avons défini précédemment, n'est pas optimale. Rapidement, les élèves proposent différentes solutions pour arriver à celle-ci :

```
n ← 0
pour a allant de 0 à 9 faire
    pour b allant de a+1 à 9 faire
        pour c allant de b+1 à 9 faire
            si a < b et b < c alors n ← n+1
retourner n
```

Puis en dernier lieu, ils se rendent compte que la structure itérative est inutile :

```
n ← 0
pour a allant de 0 à 9 faire
    pour b allant de a+1 à 9 faire
        pour c allant de b+1 à 9 faire
            n ← n+1
retourner n
```

Enfin, à partir de ce dernier résultat, les élèves établissent une formule générale du dénombrement de ces triplets. Ici, le travail algorithmique les a finalement accompagné et leur a servi de support dans ce calcul de dénombrement.
Pour revenir à l'efficacité de notre algorithme, et sensibiliser un peu plus les élèves à cette notion, on peut aussi envisager de faire comparer les deux méthodes (probabiliste et déterministe) de notre problème. Cette comparaison est alors réalisée en fonction du nombre d'éléments de notre ensemble $\mathscr{E}$ de départ.
Sous Python, un programme comme celui-ci permet de réaliser ces comparaisons :

```
from random import randint as tirage
from time import *
n=100
print 'méthode probabiliste pour n=',n,'\n'
t0=time()
S=0   #désigne le nombre de triplets vérifiant la relation
for i in range(1000000):
    a=tirage(0,n)
    b=tirage(0,n)
    c=tirage(0,n)
    if a<b and b<c:
        S=S+1
t1=time()
print S/1000000.0*(n+1)**3,'\n'
print 'temps de calcul ',t1-t0,'\n\n'

print 'méthode déterministe pour n=',n,'\n'
t0=time()
S=0   #désigne le nombre de triplets vérifiant la relation
for a in range(0,n+1):
    for b in range(a+1,n+1):
        for c in range(b+1,n+1):
            S=S+1
t1=time()
print S,'\n'
print 'temps de calcul ',t1-t0
```

Les élèves vont constater, que pour des valeurs de n relativement faibles, la méthode déterministe, si elle donne des résultats exacts, devient très consommatrice en terme de temps, à la différence de la méthode probabiliste qui est toujours d'un temps d'exécution constant (si on ne modifie pas le nombre de tirages).
Ici encore, l'algorithmique permet d'illustrer de façon concrète, et relativement simple, des problèmes mathématiques essentiels.
Ce problème d'efficacité d'un algorithme peut aussi revenir sous d'autres formes, et nous obliger à prendre du recul face à certaines raisonnements mathématiques.
Considérons cet exercice :

On suppose que l'on dispose d'une fonction booléenne `ISPRIME` *qui teste la primalité d'un nombre entier.* $ISPRIME(n) = \begin{cases} 1 \text{ si } n \text{ est premier} \\ 0 \text{ sinon} \end{cases}$

« En utilisant la fonction `ISPRIME`*, construire une fonction, notée* nbp*, qui, pour un entier n donné, retourne le nombre d'entiers premiers compris entre* 2 *et* n*. »*
« Puis construire une fonction, notée $nbpI$*, qui, pour deux entiers* a *et* b *donnés, avec* $a \leq b$*, retourne le nombre d'entiers premiers appartenant à l'intervalle* $[a;b]$*. »*

Si les solutions mathématique et algorithmique de la première question sont de même nature :

$$nbp(n) = \sum_{k=2}^{n} ISPRIME(k), \quad \text{pour } n \geq 2.$$

```
lire n
S ← 0
pour k allant de 2 à n faire
    si ISPRIME(k) alors
        S ← S + 1
retourner S
```

La seconde question peut fournir des solutions différentes.
La solution mathématique, que l'on qualifie souvent d'élégante, consiste à utiliser la fonction précédente pour construire $nbpI$ de la façon suivante :

$$nbpI([a;b]) = nbp(b) - nbp(a-1)$$

ce qui donne, traduit sous forme algorithmique :

```
lire a
lire b
retourner nbp(b) − nbp(a − 1)
```

Une solution, très simple à écrire, mais qui dans le cas de l'algorithmique n'est pas acceptable en terme d'efficacité.
Il est intéressant d'amener les élèves à se poser des questions liées à cette efficacité :

« En comptant les appels à la fonction nbp, combien de fois la fonction ISPRIME est-elle appelée ? »
« Cette fonction est-elle exécutée plusieurs fois sur des mêmes entiers ? »

Une traduction directe d'une solution mathématique ne fournit pas toujours l'algorithme le plus adapté. L'utilisation de structures itératives ne peut se faire sans considérer ces effets de bord. Il est essentiel que les élèves prennent conscience de ces points, et qu'ils essaient de les intégrer dans leur pratique.
Ce sont des points incontournables dans le cadre de la programmation d'algorithmes. Et même si la programmation n'est pas une finalité de l'algorithmique, elle en reste une suite logique. Ces problèmes de calculs font partie de l'algorithmique, il apparait intéressant de sensibiliser les élèves à ces notions, et de leur faire comprendre que deux algorithmes ne se valent pas forcément, même s'ils répondent, tous les deux, à un problème donné.

On peut ainsi, envisager de demander la réécriture de l'algorithme dans une version plus efficace permettant de répondre à la seconde question, en minimisant le nombre d'appels à la fonction `ISPRIME` :

```
lire a
lire b
S ← 0
pour k allant de a à b faire
    si ISPRIME(k) alors
        S ← S + 1
retourner S
```

D'autres questions, relevant davantage de compétences mathématiques [6], peuvent permettre d'améliorer encore l'efficacité de cet algorithme.

Le fait de demander à un élève de corriger, ou compléter, un algorithme, dépasse en pratique le cadre de la compétence *comprendre un algorithme*. Ce type d'exercice reste l'étape préalable au passage à l'écriture d'un algorithme.
Outre les difficultés prédemment soulevées, que nous allons retrouver dans un exercice d'écriture, il est intéressant aussi d'analyser les choix faits par les élèves. Ce type d'exercice va en effet davantage laisser le champ libre aux élèves, et il est intéressant d'en tenir compte dans la nature des réponses produites, même si cela n'est pas toujours aisé.
Par exemple, dans ce contexte :

Écrire un algorithme retournant le plus petit entier n tel que $u_n > 50$ avec (u_n) la suite définie par $u_0 = 1$ et $u_{n+1} = \frac{3}{2}u_n + \frac{1}{3}$.

un élève répond :

6. Par exemple, avec des tests ne portant uniquement que sur des entiers impairs, et en traitant le cas de l'entier 2 à part.

Variables
n est du type nombre
u est du type nombre
F est du type nombre
Début algorithme
lire F `# il faut choisir F très grand pour être sûre`
n prend la valeur 0
u prend la valeur 1
pour k **allant de** 1 **à** F **faire**
    $u \leftarrow \frac{3}{2}u + \frac{1}{3}$
    si $u \leq 50$ **alors** n prend la valeur $n+1$
Afficher n

Si on passe sur l'erreur [7] du test non effectué au premier rang, on peut constater que le choix de la structure itérative la plus adaptée n'a pas été évidente pour l'élève. À la vue du commentaire, un problème a été perçu, par l'élève mais il ne voit pas comment le traiter sous forme algorithmique. On peut remarquer de plus que, dans ce cas précis, l'utilisation d'une boucle **Pour** lui a vraisemblablement beaucoup compliqué la tâche.
Ce problème n'aurait pas été mis en évidence, si une certaine liberté n'avait pas été laissée à l'élève.
Par exemple, dans ce contexte :

Écrire un algorithme de résolution d'une équation du second degré à partir de ses coeeficients donnés.

comment réagir face à cette réponse :

7. A-t-on le droit de considérer que c'est une erreur ? Ici, la définition de u_n est intrinsèque à l'algorithme, et le fait de savoir que $u_0 = 1 < 50$, nous dispense de ce test.

#a, b, c et Δ désignent des réels
Lire a, b et c
Δ reçoit $b^2 - 4ac$
si $\Delta < 0$ **alors**
    afficher « l'équation n'admet pas de solution réelle »
si $\Delta = 0$ **alors**
    x_0 reçoit $-\dfrac{b}{2a}$
    afficher « l'équation admet une solution : »
    afficher x_0
si $\Delta > 0$ **alors**
    x_1 reçoit $\dfrac{-b-\sqrt{\Delta}}{2a}$
    x_2 reçoit $\dfrac{-b+\sqrt{\Delta}}{2a}$
    afficher « l'équation admet deux solutions : »
    afficher x_1 et x_2

Dans sa réponse, l'élève a choisi de faire séparer chacun des tests sur le signe de Δ dans des structures itératives distinctes. Nous savons que si Δ est strictement négatif, ce test sera effectué inutilement deux fois mais l'algorithme reste correct.
Plusieurs interprétations peuvent être tirées de cette réponse :

- l'élève ne maitrise pas suffisament les structures conditionnelles imbriquées
- l'élève s'est contenté de cette réponse car l'algorithme répond au problème demandé

Après avoir questionné l'élève, il s'est avéré que ce choix était délibéré et motivé par le seul soucis de lisibilité de l'algorithme. Cet argument justifie tout à fait ce choix.
Nous n'avons pas encore une pratique et un recul important dans l'enseignement de l'algorithmique au lycée. C'est pourquoi il convient d'être très prudent quant aux conclusions que l'on peut tirer de la réponse écrite d'un élève. Cependant les indications que l'on peut tirer de certaines réponses sont souvent très pertinentes et peuvent cibler des difficultés bien précises. Ainsi, même si l'exercice d'écriture d'un algorithme est difficile à donner dans une épreuve en temps limité, il est important de ne pas l'occulter.

Lors de l'écriture d'un algorithme, les élèves vont être confrontés au problème de mise en forme. Les recommandations dans les programmes du lycée nous incitent à éviter tout excés de formalisme, ce qui nous apparait une bonne chose. Il est important de garder une liberté d'écriture. La mise en forme d'un algorithme ne doit pas être un frein pour l'élève à l'élaboration d'une solution d'un problème mathématique mais plutôt l'occasion de proposer une réponse dans un langage qui lui est plus naturel ou en tout cas plus accessible.
Cependant l'écriture d'un algorithme et sa traduction dans un langage de programmation vont très vite faire apparaître la nécessité d'éviter tout non dit ou tout implicite. C'est une situation à mettre en lien avec les mathématiques. Pour les élèves, le formalisme et la rigueur demandés dans les démonstrations mathématiques apparaissent souvent artificiels, ou comme un simple exercice de style. Nous avons ici une occasion supplémentaire de légitimer cette façon de faire. Il s'agit de montrer aux élèves que cette façon de procéder pour l'écriture des algorithmes, apporte une sécurité tangible dans la résolution d'un problème. La moindre erreur dans l'écriture d'un algorithme va se traduire par des effets de bords souvent désastreux, mais, ici, facilement observables par les élèves.
D'autre part, cette pratique permet aussi de fournir un travail lisible, et compréhensible par d'autres. Les élèves ne perçoivent pas toujours une production écrite comme un média de communication. Les exercices, que l'on peut mettre en place, pour souligner cet aspect sont nombreux :

- faire lire, ou programmer, l'algorithme d'un élève par un camarade ;
- refaire lire à son auteur un algorithme après avoir laissé un temps suffisament long s'écouler depuis son écriture ;
- faire deviner la fonction d'un algorithme mal ou peu commenté ;
- etc...

L'objectif, ici, est d'illustrer l'importance d'un langage avec suffisament de normes pour qu'il soit compréhensible indépendament de l'auteur, du lecteur ou de l'instant, tout en évitant les excés. À ce titre, le choix des logiciels va avoir beaucoup d'importance.
Par exemple, cet exercice tiré d'un manuel scolaire [8] met bien en relief ce dernier point :

8. Hyperbole, Nathan

Morgane a trouvé dans le répertoire « Géométrie » de son ordinateur, le programme écrit ci-dessous avec le langage ALGOBOX.

```
1   VARIABLES
2     X1 EST_DU_TYPE NOMBRE
3     Y1 EST_DU_TYPE NOMBRE
4     X2 EST_DU_TYPE NOMBRE
5     Y2 EST_DU_TYPE NOMBRE
6     X3 EST_DU_TYPE NOMBRE
7     Y3 EST_DU_TYPE NOMBRE
8     U1 EST_DU_TYPE NOMBRE
9     V1 EST_DU_TYPE NOMBRE
10    U2 EST_DU_TYPE NOMBRE
11    V2 EST_DU_TYPE NOMBRE
12    R EST_DU_TYPE NOMBRE
13  DEBUT_ALGORITHME
14    LIRE X1
15    LIRE Y1
16    LIRE X2
17    LIRE Y2
18    LIRE X3
19    LIRE Y3
20    U1 PREND_LA_VALEUR X2-X1
21    V1 PREND_LA_VALEUR Y2-Y1
22    U2 PREND_LA_VALEUR X3-X1
23    V2 PREND_LA_VALEUR Y3-Y1
24    R PREND_LA_VALEUR U1*V2-U2*V1
25    SI (R==0) ALORS
26      DEBUT_SI
27      AFFICHER "......"
28      FIN_SI
29      SINON
30        DEBUT_SINON
31        AFFICHER "......."
32        FIN_SINON
33  FIN_ALGORITHME
```

a) Expliquer à Morgane le rôle de ce programme.
b) Proposer le texte des affichages du programme.

Ce programme ALGOBOX met en évidence les problèmes de lisibilité que peut engendrer un excés de formalisme. Ce logiciel est un formidable outil d'initiation et d'introduction à l'algorithmique. Pour pouvoir remplir sa mission, il oblige une rédaction et une structure très rigides, qui, pour quelqu'un d'initié, peut devenir peu pratique voire encombrante. On ne pourrait baser son

enseignement sur son utilisation exclusive [9]. Nous pensons très important, à ce titre, d'utiliser différents logiciels. Il s'agit alors de montrer aux élèves qu'il n'existe pas un seul langage mais plusieurs, et que certains sont plus adaptés que d'autres pour certaines tâches.

Objectif et place de l'évaluation de l'algorithmique dans l'enseignement de mathématiques

Les notions de base en algorithmique sont largement explicitées dans le programme de mathématiques. Elles peuvent paraître ambitieuses pour un enseignant non expérimenté, mais cela nous semble le minimum pour pouvoir réaliser des activités intéressantes, ou, en tout cas, mettant en œuvre des notions de mathématiques enseignées au lycée.
D'un autre point de vue, cela peut paraître pauvre, pour d'autres enseignants, mais ces notions sont déjà suffisament nombreuses pour pouvoir construire des activités intéressantes et aussi préparer les élèves à de futures notions. Ne perdons pas de vue, que nous n'avons pas en notre présence que de futurs informaticiens.
Il vaut mieux s'assurer d'une parfaite compréhension de ces notions de base, les autres s'imposeront alors plus facilement aux élèves qui devraient continuer dans cette voie.
Certains points, propres à l'algorithmique, vont devoir faire aussi l'objet d'une attention toute particulière. En effet certains concepts ne semblent pas uniquement relever du cadre algorithmique, mais aussi du cadre des mathématiques ou voire relever de la programmation.
Par exemple, les affectations multiples dans un algorithme soulèvent des problèmes qui sont traités de façon transparente en mathématiques. On peut illustrer ce propos en considérant cet exercice (Antilles S 2004) :

9. Cette remarque est valable pour tout logiciel.

On définit les suites (a_n) et (b_n) par $a_0 = 1$, $b_0 = 7$ et
$$\begin{cases} a_{n+1} &= \dfrac{1}{3}(2a_n + b_n) \\ b_{n+1} &= \dfrac{1}{3}(a_n + 2b_n) \end{cases}$$
Soit D une droite munie d'un repère $\left(\text{O} ; \vec{\text{i}}\right)$.
Pour tout $n \in \mathbb{N}$, on considère les points A_n et B_n d'abscisses respectives a_n et b_n.

1. Placez les points A_0, B_0, A_1, B_1, A_2 et B_2.
2. Soit (u_n) la suite définie par $u_n = b_n - a_n$ pour tout $n \in \mathbb{N}$. Démontrez que (u_n) est une suite géométrique dont on précisera la raison et le premier terme.
   Exprimez u_n en fonction de n. ...

Son adaptation, dans le cadre de l'algorithmique peut amener les élèves, à élaborer un algorithme permettant les calcul des termes a_n et b_n pour n donné.
Face à cette production :

...
tant que $k < n$ **faire**
    $a \leftarrow \dfrac{2a+b}{3}$
    $b \leftarrow \dfrac{a+2b}{3}$
    $k \leftarrow k+1$
...

il est facile d'analyser l'erreur produite, et d'envisager des remédiations, comme par exemple l'éxécution de l'algorithme pas à pas, en relevant l'état des variables à chaque instruction.
Mais comment réagir face à un algorithme juste, comme dans ces deux versions :

...
tant que $k < n$ **faire**
    $a, b \leftarrow \dfrac{2a+b}{3}, \dfrac{a+2b}{3}$
    $k \leftarrow k+1$
...

...
tant que $k < n$ **faire**
    $temp \leftarrow a$
    $a \leftarrow \dfrac{2a+b}{3}$
    $b \leftarrow \dfrac{temp+2b}{3}$
    $k \leftarrow k+1$
...

La première version est beaucoup plus facile à lire que la seconde mais elle laisse de côté une partie du problème d'affectation multiple, ou la considère comme déja résolue.
Doit-on alors accepter les deux versions, ou seulement l'une des deux ?
Les avis semblent partagés mais nous pensons important de n'en rejeter aucune. Le premier algorithme peut tout à fait être validé, si on accepte l'idée d'affectation multiple [10] à travers le symbole « $\leftarrow$ » dans nos algorithmes. Il s'agit alors de s'assurer que le problème de l'affectation multiple dans nos deux suites a bien été perçu par les élèves.

L'implicite, dans la définition mathématique de nos deux suites, est fort et ne doit pas être laissé de côté. Il est intéressant de soulever le problème auprès des élèves, et le terrain de l'algorithmique s'y prête particulièrement bien.
Ces questions d'implicite ne sont pas toujours simples. Il est pourtant nécessaire de résoudre tous ces problèmes, si on souhaite, que les élèves puissent construire une réflexion sur des bases rigoureuses et solides. Pour les enseignants, il n'est pas toujours facile non plus de prévenir ces implicites.
Par exemple, la traduction du calcul de cette fonction f définie sur $\mathbb{R}$ ainsi :

si $x < 1$ alors $f(x) = 3x - 2$
si $x \geq 1$ et $x \leq 2$ alors $f(x) = x^2$ sinon $f(x) = 2x$

peut conduire à ce genre de réponse(observée dans des copies d'élèves) :

lire x
si $x < 1$ **alors** retourner $3x - 2$ **si** $x \geq 1$ *et*
$x \leq 2$ **alors**
    retourner x^2
sinon
    retourner $2x$

10. Certains langages, comme Python, autorise ce type d'affectation

Au travers de l'évaluation de l'algorithmique, nous avons l'occasion de présenter des points importants en mathématiques, points que l'on n'avait pas l'habitude, ou des difficultés, à illustrer.
Un autre exemple porte sur la nature même des problèmes traités par l'algorithmique. Les algorithmes ne traitent que des ensembles discrets et finis de valeurs et, dans certains cas, l'apport des mathématiques va nous permettre de dépasser ce cadre [11].
Considérons ce problème :

Résoudre dans $\mathbb{N}^3$*, l'équation* $a^2+b^2+c^2=2386$.
On ne retiendra que les triplets solutions $(a;b;c)$ *vérifiant* $a \leq b \leq c$.

On ne peut écrire un algorithme de recherche des solutoins dans $\mathbb{N}^3$ tout entier.
On peut par contre se débarasser du problème d'infini, en montrant que les solutions de l'équation appartiennent à un ensemble borné.
À partir de $a \leq b \leq c$, on montre facilement que $3a^2 \leq a^2+b^2+c^2$
Par conséquent, pour $3a^2 > 2386$, c'est-à-dire $a \geq 17$, aucun triplet $(a;b;c)$ n'est solution.
De même, $a^2+b^2+c^2 \geq c^2$ donc pour $c^2 > 2386$, c'est-à-dire $a \geq 49$, aucun triplet $(a;b;c)$ n'est solution.
On a ainsi $0 \leq a \leq 16$ et $0 \leq c \leq 48$ sachant que $a \leq b \leq c$.
Il ne reste plus qu'un nombre fini de valeurs à explorer, ce qui peut se traiter à l'aide d'un algorithme exhaustif :

```
pour a allant de 0 à 16 faire
    pour b allant de a à 48 faire
        pour c allant de b à 48 faire
            si a²+b²+c² = 1386 alors afficher (a;b;c)
```

Question subsidiaire : déterminer un meilleur encadrement de b afin de réduire le nombre d'itérations dans l'algorithme précédent.

De la même façon, ce sont des résultats mathématiques qui nous permettent de valider certains algorithmes et nos élèves peuvent déjà en prendre conscience.
Les notions, présentées au lycée, sur les suites convergentes fournissent des exemples intéressant traitant de la terminaison d'un algorithme. On peut présenter les choses ainsi :

11. Réciproquement, l'apport de l'algorithmique, et d'un traitement automatisé, a permis aux mathématiques de résoudre certains problèmes sur lesquels elles butaient.

D'après Baccalauréat S Nouvelle-Calédonie mars 2011

Soit (u_n) la suite définie par $\begin{cases} u_0 &= 1 \\ u_{n+1} &= u_n - \ln\left(u_n^2+1\right) \text{ pour } n \in \mathbb{N}. \end{cases}$

Partie A

Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = x - \ln\left(x^2+1\right)$.

1. Résoudre dans $\mathbb{R}$ l'équation $f(x) = x$.
2. Étudier le sens de variation de la fonction f sur l'intervalle $[0\,;\,1]$. En déduire que si $x \in [0\,;\,1]$ alors $f(x) \in [0\,;\,1]$.

Partie B

1. Démontrer par récurrence que, pour $n \geqslant 0$, $u_n \in [0\,;\,1]$.
2. Étudier le sens de variation de la suite (u_n).
3. Démontrer que la suite (u_n) converge vers 0.
4. On propose l'algorithme suivant :

```
u ← 1
k ← 0
tant que u > 0,05 faire
    k ← k + 1
    u ← u − ln(u² + 1)
retourner k
```

   (a) Que retourne cet algorithme à la fin de son exécution ?

   (b) Quel résultat mathématique, démontré précédemment, nous permet d'affirmer que l'exécution de cet algorithme se termine.

Dans la dernière question, nous avons une application concrète de la convergence d'une suite. Sans ce résultat, sur la suite (u_n), nous n'avons aucun moyen de nous assurer de la terminaison, et donc de la validité, de cet algorithme.
Toujours pour pousser les élèves dans cette réflexion, pourquoi ne pas leur propose de débattre [12] sur la validité de cet algorithme :

12. Il ne s'agit en aucun cas d'utiliser cet exercice dasn le cadre d'une évaluation sommative.

$a \leftarrow 0$
$k \leftarrow 0$
tant que $a \neq 3$ **faire**
    $a \leftarrow$ nombre aléatoire entre 1 et 6
    $k \leftarrow k+1$
retourner k

Pour pousser le vice, on peut faire programmer et exécuter cet algorithme [13] . L'exécution de ce programme va bien sûr toujours se terminer alors que l'algorithme pose un problème du point de vue de sa terminaison. Cet algorithme, simple à écrire, permet de soulever d'intéressantes questions auprès des élèves.

Conclusion à faire On peut constater que l'écriture correcte de l'algorithme ne peut se faire que si les notions mathématiques développées sont bien perçues. Ici, la frontière entre évaluation de l'algorithmique et évaluation en mathématiques va être difficile à distinguée, et doit-elle l'être ?
Cette évaluation, dans beaucoup de situations, ne pourra être que conjointe à celle des mathématiques, fournissant ainsi un outil pertinent d'analyse au service de notre enseignement. **Développer l'idée d'une évaluation sommative raisonnable : concept algorithme + concept mathématique =difficulté^2 pour les élèves, de plus encore trop tôt (algo en seconde première et terminale pas assez ancrée dans les pratiques) pour poser des évaluations difficiles.**
Par contre évaluation comme outil de diagnostique -> pertinent avec de réels intérêts dans l'évaluation des maths

Quelques exemples de ce que pourrait être l'évaluation au bac de l'algorithmique

Faire figurer le sujet réalisé par Malika.

autres exemples :

D'après sujet de bac TS spécialité :

1. On a réalisé les calculs suivants dans un logiciel de calcul formel :

13. Cet exemple a été présenté dans un premier temps, à des élèves, sans le calcul du nombre d'itérations. L'objectif était de privilégier le problème de terminaison mais cette situation a beaucoup perturbé les élèves : un algorithme qui ne retourne aucune valeur ! On peut ainsi constater que, par nos pratiques, nous avons malheureusement déjà réduit la représentation que les élèves peuvent se faire d'un algorithme…

```
pour k de 0 jusque 11 faire
print(simplifier(developper((1+sqrt(6))^k)));
fpour;
```

☞ *sqrt(6) désigne* $\sqrt{6}$.

On a obtenu alors l'affichage suivant :

# 1	`1`	*# 5*	`28*sqrt(6)+73`	*# 9*	`4088*sqrt(6)+10033`
# 2	`sqrt(6)+1`	*# 6*	`101*sqrt(6)+241`	*# 10*	`14121*sqrt(6)+34561`
# 3	`2*sqrt(6)+7`	*# 7*	`342*sqrt(6)+847`	*# 11*	`48682*sqrt(6)+119287`
# 4	`9*sqrt(6)+19`	*# 8*	`1189*sqrt(6)+2899`	*# 12*	`167969*sqrt(6)+411379`

(a) Quels résultats mathématiques peut-on tirer de la ligne *# 5* et de la ligne *# 7* ?

(b) Appliquer l'algorithme d'Euclide à 847 et 342. Que peut-on en déduire ?

2. Montrer par récurrence, que, pour tout entier naturel n, il existe deux entiers a_n et b_n tels que

$$\left(1+\sqrt{6}\right)^n = a_n + b_n\sqrt{6}.$$

☞ *on vérifiera, en particulier, que* $a_{n+1} = a_n + 6b_n$ *et que* $b_{n+1} = a_n + b_n$.

3. À partir des données de l'énoncé, préciser les valeurs de a_n et b_n, pour $n = 4$ et $n = 6$.

4. Démontrer que :
si 5 ne divise pas $a_n + b_n$, alors 5 ne divise pas non plus $a_{n+1} + b_{n+1}$.
En déduire que, pour tout entier naturel n, 5 ne divise pas $a_n + b_n$.

5. Démontrer que, si a_n et b_n sont premiers entre eux, alors a_{n+1} et b_{n+1} sont premiers entre eux.
En déduire que, pour tout entier n, a_n et b_n sont premiers entre eux.

6. On souhaite écrire un algorithme réalisant le calcul de a_n et b_n pour un entier n donné.
Compléter, dans ce sens, l'algorithme donné dans l'annexe.

ANNEXE

lire n
$a \leftarrow 1$
$b \leftarrow 0$
$k \leftarrow 0$
tant que ..... **faire**
    $temp \leftarrow a$
    $a \leftarrow a + 6 \times b$
    $b \leftarrow temp + b$
    $k \leftarrow k + 1$
retourner a et b

D'après sujet Baccalauréat S Métropole 16 septembre 2010 :
Soit (u_n) la suite définie par $u_0 = 5$
et pour tout nombre entier naturel n, par $u_{n+1} = \dfrac{4u_n - 1}{u_n + 2}$.
Si f est la fonction définie sur l'intervalle $]-2\,;\,+\infty[$ par $f(x) = \dfrac{4x-1}{x+2}$, alors on a, pour tout nombre entier naturel n, $u_{n+1} = f(u_n)$.
On donne en annexe 1 (à rendre avec la copie) une partie de la courbe représentative $\mathscr{C}$ de la fonction f ainsi que la droite Δ d'équation $y = x$.

1. (a) Sur l'axe des abscisses, placer u_0 puis construire u_1, u_2 et u_3 en laissant apparents les traits de construction.

(b) Quelles conjectures peut-on émettre sur le sens de variation et sur la convergence de la suite (u_n) ?

2. (a) Démontrer par récurrence que, pour tout nombre entier naturel n, on a $u_n - 1 > 0$.

(b) *Dans cette question, toute trace de recherche, même incomplète, ou d'initiative même non fructueuse, sera prise en compte dans l'évaluation.*

Valider par une démonstration les conjectures émises à la question 1. b.

3. On propose l'algorithme suivant :

$u \leftarrow 5$
$k \leftarrow 0$
tant que $u > 1,1$
faire
    $k \leftarrow k + 1$
    $u \leftarrow \dfrac{4u-1}{u+2}$
retourner k

(a) Que retourne cet algorithme à la fin de son exécution ?

(b) Quel résultat mathématique, démontré précédemment, nous permet d'affirmer que l'exécution de cet algorithme se termine.

4. Dans cette question, on se propose d'étudier la suite (u_n) par une autre méthode, en déterminant une expression de u_n en fonction de n.

Pour tout nombre entier naturel n, on pose $v_n = \dfrac{1}{u_n - 1}$.

(a) Démontrer que la suite (v_n) est une suite arithmétique de raison $\dfrac{1}{3}$.

(b) Pour tout nombre entier naturel n, exprimer v_n puis u_n en fonction de n.

(c) En déduire la limite de la suite (u_n).