Lycée Loges

Chapitre 6

Informatique théorique Exercices

Simon Dauguet
simon.dauguet@gmail.com

9 janvier 2025

Exercice 1 :
Étudier la terminaison, arrêt et complexité des algorithmes suivants. De plus, expliciter ce que font les deux algorithmes mystères.

```python
def moyenne(L:list) -> float :
    n = len(L)
    S = 0
    for k in range(n) :
        S = S+L[k]
    return(S/n)
```

```python
def mystere(L:list) -> list :
    tab = []
    for k in range(len(L)) :
        S = 0
        n = 0
        while n<len(L[k]) :
            S = S+L[k][n]
            n += 1
        tab = tab + [S]
    return(tab)
```

```python
def seuil(a:float, eps:float) -> int :
    u = 1
    n = 0
    while abs(u**2-a)>eps :
        u = 1/2*(u+a/u)
        n += 1
    return(n)
```

```python
def mystere2(n:int) -> bool :
    p = 2
    while p**2<n :
        if n%p==0 :
            return(False)
        else :
            p = p+1
    return(True)
```

Exercice 2 :
On considère l'algorithme de recherche d'un élément dans un tableau non trié suivant :

```python
def recherche(x:"object", L:list) -> int :
    i = 0
    while i != len(L) :
        if L[i] != x :
            i += 1
        else :
            return(i)
    return(-1)
```

Étudier la complexité de cet algorithme, sa terminaison et sa correction.

Exercice 3 :
On va étudier la somme des éléments d'une liste.

1. Écrire une fonction `sommeListe(L:list) -> float` qui calcule la somme des éléments d'une liste.
2. Expliciter un variant de boucle et montrer la terminaison de cet algorithme.
3. Expliciter un invariant de boucle et montrer que cet algorithme est correct.
4. Déterminer la complexité de cette fonction.

Exercice 4 (Extrait sujet 0 CCINP MP) :

1. Écrire une fonction `factorielle(n:int) -> int` qui prend en argument un entier naturel et qui renvoie $n!$. On ne fera pas d'algorithme récursif.
2. Écrire une fonction `seuil_facto(M:int) -> int` qui prend en argument un entier M et renvoie le plus entier n tel que $n! > M$.
3. Écrire une fonction `divisible(n:int) -> bool` qui prend en argument un entier n et qui renvoie `True` si $n+1$ divise $n!$, et `False` dans le cas contraire.
4. On considère la fonction suivante :

```
def code_mystere(n:int) -> int :
    s=0
    for k in range(1,n+1) :
        s=s+factorielle(k)
    return(s)
```

   (a) Sans tester la fonction, prévoir ce que renvoie, ce que fait cette fonction. Quelle valeur devra renvoyer `code_mystere(4)` ? Vérifier en tester ce code.
   (b) Déterminer la complexité de cette fonction. On se contentera d'une complexité naïve (donc fausse).
   (c) Proposer une amélioration du code de la fonction `code_mystere` pour que sa complexité soit linéaire.

Exercice 5 (Crible d'Ératosthène) :
Le crible d'Ératosthène est un procédé permettant de trouver la liste des nombres premiers inférieurs ou égaux à une donnée N. Le code fonctionne de la manière suivante :

- On écrit la liste L de tous les entiers entre 2 et N (on rappelle que 1 n'est jamais un nombre premier).
- Pour k compris entre 2 et $\left\lfloor \sqrt{N} \right\rfloor$, on fait :
  - Pour p entre 2 et $\lfloor N/k \rfloor$, on fait :
    - Remplacer dans la liste les multiples de k par "*" (c'est-à-dire remplacer kp par une étoile).
- Renvoyer la liste L (éventuellement après épuration des "*").

1. À l'aide de la description ci-dessus, coder la fonction `crible(n:int) -> list` reprenant l'algorithme d'Ératosthène et qui renvoie la liste de tous les nombres premiers inférieurs ou égaux à n.
2. Faire une nouvelle fonction `cribleNb(n:int) -> int` basé sur le précédent qui doit compter et afficher le nombre d'itérations dans les boucles `for`.
3. Estimer la complexité de l'algorithme d'Ératosthène (fonction `crible`).
4. Ce code n'est pas très efficace (trop naïf). En effet, une fois les multiples de 2 enlevés, il va par la suite essayer d'enlever les multiple de 4, 6 etc. Quelle instructions conditionnelles pourrait-on rajouter pour optimiser un peu cet algorithme ? Vous incorporerez un compteur qui comptera le nouveau nombre d'itérations des boucles `for`.

Exercice 6 :
On considère une liste `L` de n éléments triée dans l'ordre croissant. On cherche à construire un algorithme permettant de savoir à quel endroit de la liste se trouve une valeur donnée `v`.

1. Écrire une fonction `recherche(L:list, v:float) -> int` qui réponde au problème.
2. Étudier la complexité de cet algorithme.
3. On considère le code suivant :

```
def rechercheDicho(L:list, v:float) -> int :
    f = len(L)-1
    d = 0
    trouve = False
    while not trouve :
        i = int((d+f)/2)
        if L[i] == v :
            trouve = True
        else :
            if L[i] < v :
                d = i+1
            else :
                f = i-1
            print(i,d,f,trouve)
    return(i)
```

   (a) Faire un jeu de tests de ce programme pour $v = 19$ et `L=[1,2,3,5,7,11,13,17,19,23]`.
   (b) Étudier la terminaison et la correction de cette fonction.
   (c) Calculer sa complexité.

Exercice 7 (Extrait sujet 0 banque PT) :

1. Écrire une fonction `chiffre(n:int) -> list` qui prend en argument un entier n et qui renvoie la liste des chiffres qui le compose. Donc `chiffre(1234)` devra renvoyer `[1,2,3,4]` (ou `[4,3,2,1]`).
2. Écrire une fonction `somcube(n:int) -> int` qui renvoie la somme des cubes des chiffres qui compose n.
3. Évaluer la complexité de la fonction `somcube`.
4. Trouver tous les entiers inférieurs ou égaux à 1000 qui sont égaux à la somme des cubes de leurs chiffres.
5. Proposer un autre algorithme renvoyant la liste des chiffres qui compose un entier n (donc avec une autre méthode).

Exercice 8 (Médiane) :
Le but de cet exercice est d'étudier un algorithme donnant la médiane d'un tableau de nombres.

1. Dans un tableau, si on supprime le minimum et le maximum de ce tableau successivement jusqu'à obtenir un tableau de longueur ≤ 2, on peut en déduire facilement la médiane.
   Écrire la fonction `mediane(tab:list) -> float` qui corresponde à cette description.
2. Étudier la terminaison et la correction de cet algorithme.
3. Étudier sa complexité.

Exercice 9 (Zéros de fonction) :
On reprend le principe de la recherche de zéro d'une fonction par dichotomie.

1. On considère une fonction $f : [a, b] \to \mathbb{R}$ continue. On suppose que cette fonction ne s'annule qu'une seule fois sur $[a, b]$. Écrire une fonction `zeros(f:"function", a:float, b:float, epsilon:float) -> float` qui permet de chercher une valeur approchée d'un zéros de f sur $[a, b]$ à ϵ près par la méthode dichotomique.
2. Faire un jeu de test pour $f : x \mapsto x^3 - 2$ sur l'intervalle $[0, 2]$ avec $\epsilon = 1/10$.
3. Prouver la terminaison et la correction de cet algorithme.
4. Étudier la complexité de cet algorithme.