TD : Étude d'algorithmes

pfm, ENSIBS Cyber-[Log & Data], S5

2023-2024

Exercice No. 1

Somme d'éléments

Prouver en utilisant un invariant de boucle que la fonction suivante effectue correctement la somme des éléments du tableau A passé en argument.

```
def sumArray(float []A, int n):
# pre-condition : A[0:n-1] is a float array of size n >= 1
# post-condition : return a float value equal to the sum of the elements of A
  int i <- 1
  float sum <- A[0]
  while i < n:
    sum <- sum + A[i]
    i <- i + 1
  return sum
```

Préconditions:

- A est un tableau de nombres réels (indexés à partir de 0),
- n est la taille du tableau A

Post-condition: $sum = \sum_{k=0}^{n-1} A[k]$

Exercice No. 2

Puissance

On considère le pseudo code suivant:

```
float Puissance(float a, int n):
  float x <- 1.0
  for i <- 1 to n
    x <- x * a
  return x
```

Pré-conditions:

- a est un nombre réel,
- n est un entier non négatif ($n \geq 0$)

2.1 utilité

À quoi sert cet algorithme (préciser la postcondition).

2.2 Terminaison

Prouver la terminaison de cet algorithme.

2.3 Adéquation

Montrer l'adéquation de cet algorithme.

2.4 Complexité

Quelle est la complexité de cet algorithme (le démontrer) ?

2.5 Évaluation empirique

Programmer cet algorithme en Python et illustrer sa complexité empirique en traçant une courbe (temps consommé v.s. n).

Exercice No. 3

MultiplicationEntière

On rappelle ici le pseudo-code de l'algorithme *FancyMultiplication* :

```
FancyMultiply(int x, int y):
// Precondition  : x>=0, y>=0 deux entiers naturels non nuls
int prod <- 0
while x > 0
  if x is odd
    prod <- prod + y
  x <- x/2 // Division euclidienne
  y <- y + y
return prod
```

3.1 Utilité

À quoi sert cet algorithme (précisez la post-condition) ?

3.2 Terminaison

Démonter la terminaison de cet algorithme.

3.3 Adéquation

Montrer par récurrence que cet algorithme fournit en sortie le produit $x \cdot y$, pour les valeurs de x et y données en entrée.

3.4 Complexité

Quelle est la complexité de cet algorithme. Fournir une preuve de ce que vous avancez.

3.5 Évaluation empirique

Programmer cet algorithme en Python et illustrer sa complexité empirique en traçant une courbe.

Exercice No. 4

Puissance, à nouveau

On revisite ici l'exercice 2

4.1 Récursivité

Donner une version récursive à cet algorithme.

4.2 Complexité

Quelle est la complexité de cet algorithme ?

4.3 Amélioration

Existe-t-il une version plus efficace de cet algorithme ? Si oui, prouvez le.

4.4 Évaluation empirique

Programmer cet algorithme en Python et illustrer sa complexité empirique en traçant une courbe (temps consommé v.s. n) et en la comparant à celles des versions itérative et récursive initiale.

Exercice No. 5

Fusion de listes triées

On considère deux tableaux d'entier triés u et v selon l'ordre croissant, contenant respectivement n_u et n_v éléments.

5.1 Proposer un algorithme de fusion de listes

Donner le pseudo code d'un algorithme permettant de fusionner les tableaux u et v pour fournir en sortie un tableau w triéeen ordre croissant contenant tous les éléments des tableaux u et v.

5.2 Validité

Prouver la terminaison et la validité de votre algorithme.

5.3 Complexité

Donner la complexité de votre algorithme.

Exercice No. 6

Tri fusion

On considère l'algorithme du tri fusion :

```
entrée : un tableau T
sortie : une permutation triée de T
fonction triFusion(T[1, ..., n])
  si n <= 1
    renvoyer T
  sinon
    renvoyer fusion(triFusion(T[1, ..., n/2]), triFusion(T[n/2 + 1, ..., n]))
```

6.1 Complexité

Quelle est la complexité de cet algorithme (donner une preuve par récurrence).

Exercice No. 7

Tours de Hanoï

7.1 Rappeler l'algorithme

Donner le pseudo-code de l'algorithme permettant de résoudre le problème des tours de Hanoï.

7.2 Complexité

Quelle est la complexité de l'algorithme (Fournir la démonstration) ?

7.3 Évaluation empirique

Programmer cet algorithme en Python et illustrer sa complexité empirique en traçant une courbe (temps consommé v.s. n).