Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 3
Code UE : SPO1U57

Année 2023–2024

Algorithmes – correction et complexité

Exercice 1 (Correction d'un algorithme)

1. Chacun des algorithmes suivants admet une erreur. Déterminez-la. Le paramètre d'entrée `n` est un entier positif.

```
def algo1(n):
    b = 1
    while n != 0:
        b = 2 * b
        n = n - 2
    return b
```

```
def algo2(n):
    b = 1
    a = 1
    while a < n:
        b = 2 + a
    return b
```

2. Écrivez une fonction qui prend en paramètres deux entiers `x` et `y` et qui retourne le maximum de `x` et `y`. Notons que l'entier m est le maximum de x et y s'il s'agit de l'unique entier tel que $m \geq x$ et $m \geq y$ et ($m = x$ ou $m = y$).

   Montrez que votre algorithme termine et qu'il est correct.

3. On veut écrire une fonction qui calcule la factorielle d'un entier positif `n` passé en entrée : on rappelle que la factorielle de l'entier $n > 0$ est l'entier $n!$ défini par $1 \times 2 \times 3 \times \cdots \times n$. L'algorithme suivant admet une erreur.

```
def factorielle(n):
    fact = 1
    i = 0
    while i < n:
        fact = fact * i
        i = i + 1
    return fact
```

   (a) Déterminez l'erreur et corrigez-la.

   (b) Déroulez l'exécution de `factorielle(4)` en la présentant sous la forme d'un tableau permettant de suivre l'évolution des variables instruction par instruction. En particulier, vous indiquerez les valeurs des variables `i`, et `fact` à la fin de chaque itération de la boucle. L'itération 0 indiquera les valeurs de ces variables avant la première itération.

   Quelle est la valeur de `fact` à la fin de l'exécution ?

   (c) (*À faire à la maison*) Vous devriez vous apercevoir qu'à la fin de la i-ième itération de la boucle, la variable `fact` contient $i!$. En déduire une preuve par récurrence que votre algorithme est correct.

Exercice 2 (Complexité d'un algorithme)

1. Déterminez le nombre d'affectations effectuées par chacun des algorithmes suivants pour $i = 3$. Généralisez ce résultat pour tout $i \in \mathbb{N}$.

```
def algo1(i):
    n = 2**i   # calcule 2 puissance i
    a = 1
    b = 1
    while a < n:
        b = 2 * b
        a = a + 1
    return b
```

```
def algo2(i):
    n = 2**i
    a = 1
    b = 1
    while a < n:
        b = 2 * b
        a = a * 2
    return b
```

```
def algo3(i):
    n = 2**i
    a = 1
    b = 1
    while n != 1:
        b = n * b
        n = n // 2
    return b
```

```
def algo4(i):
    a = 1
    b = 1
    for j in range(i):
        for k in range(i):
            b = 2 * b
    return b
```

2. Quel algorithme est le plus lent ? Quel le plus rapide ?

3. On considère maintenant l'algorithme suivant, qui prend en entrée un entier n strictement positif :

```
def mystère(n):
    i = 1
    a = 1
    while i < n:
       a = a + 1
       i = 2 * i
    if a % 2 == 0:
       for j in range(n):
          a = a + 5
    return a
```

   (a) Donnez deux valeurs de n telles que la condition `a % 2 == 0` est évaluée à `True` et à `False` respectivement.

   (b) Expliquez pourquoi cet algorithme termine sur toute entrée.

   (c) Évaluez le nombre d'affectations de l'algorithme en fonction de n *dans le pire des cas*, puis simplifiez l'expression avec la notation « grand O ».

   (d) Évaluez le nombre d'affectations de l'algorithme en fonction de n *dans le meilleur des cas*, puis simplifiez l'expression avec la notation « grand O ».