Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Science informatique : TD 2

Année 2022-23

Correction, terminaison, tri

Exercice 1 On considère différents algorithmes pour le calcul de x^n pour x un nombre réel et n est un entier naturel. Pour chacun des algorithmes ci-dessous, indiquez :

- si l'algorithme termine (si ce n'est pas le cas, expliquez brièvement pourquoi) ;
- si l'algorithme est correct (si ce n'est pas le cas, donnez un exemple d'entrée pour laquelle la sortie de l'algorithme est incorrecte) ;
- l'ordre de grandeur de la complexité algorithmique dans le pire des cas de l'algorithme.

Algorithme 1 :

```
def puissance(x, n):
  p = x
  for i in range(n-1):
    p = x * p
  return p
```

Algorithme 2 :

```
def puissance(x, n):
  if n == 0:
    return 1
  return x * puissance(x, n-1)
```

Algorithme 3 :

```
def puissance(x, n):
  if n == 0:
    p = 1
  else:
    if n%2 == 1:
      p = x*puissance(x*x, n//2)
    else:
      p = puissance(x*x, n//2)
```

Algorithme 4 :

```
def puissance(x, n):
  p = 1
  while n > 0:
    p = x * p
    n = n-1
  return p
```

Algorithme 5 :

```
def puissance(x, n):
  return x * puissance(x, n-1)
```

Algorithme 6 :

```
def puissance(x, n):
  if n == 0:
    return 1
  elif n == 1:
    return x
  else:
    return puissance(x*x, n//2)
```

Algorithme 7 :

```
def puissance(x, n):
  p = 1
  while n > 0:
    p = x * p
  n = n - 1
  return p
```

Algorithme 8 :

```
def puissance(x, n):
  p = 1
  for i in range(n):
    p = x * p
    return p
```

Algorithme 9 :

```
def puissance(x, n):
  p = 1
  for i in range(n):
    p = x * p
```

Exercice 2 Considérons l'algorithme d'exponentiation rapide pour calculer x^n :

```python
def puissance_rapide(x, n):
    a = 1
    b = x
    m = n
    while m > 0:
        if m % 2 == 1:
            a = a * b
        m = m // 2
        b = b * b
    return a
```

1. Montrer que l'algorithme termine.
2. Montrer que l'algorithme est correct en identifiant un invariant de boucle.
3. Estimer sa complexité dans le pire des cas.

Exercice 3 Un algorithme de tri classique est le tri par insertion, dont voici un code Python possible :

```python
def trier_par_insertion(tableau) :
    n = len(tableau)
    for i in range(1, n):
        x = tableau[i]
        # insérer x parmi les i premiers éléments
        j = i
        while (j > 0) and (x < tableau[j-1]):
            # décaler d'un élément
            tableau[j] = tableau[j-1]
            j = j-1
        # ici, x >= tableau[j-1] ou bien j==0
        tableau[j] = x
```

1. Rappelez la raison pour laquelle ce code ne retourne rien.
2. Que se passe-t-il si on remplace la troisième ligne de l'algorithme par :

```python
    for i in range(n):
```

3. Justifiez que cet algorithme termine.
4. On a vu en cours que, dans le pire des cas, la complexité du tri par insertion est en $\mathcal{O}(n^2)$. On peut raisonnablement se poser la question de savoir si on a surestimé le nombre d'opérations élémentaires. En fait, il n'en est rien. Trouver donc une suite de tableaux $(t_n)_{n\in\mathbf{N}}$ avec t_n un tableau de longueur n telle que le nombre C_n d'opérations élémentaires effectuées lors du tri du tableau t_n est de la forme $an^2 + bn + c$ avec a, b, c des constantes et $a > 0$.

Exercice 4 Étant donné un tableau de longueur $n \geq 2$ contenant des nombres entiers, on s'intéresse au problème de trouver un *pic* dans ce tableau, c'est-à-dire un indice i tel que la valeur dans le tableau à l'indice i soit supérieure ou égale aux valeurs aux indices $i-1$ et $i+1$, si ceux-ci font partie du tableau. Dans le cas où i vaut 0 ou $n-1$, seule la condition sur l'unique case voisine doit être vérifiée.

On propose l'algorithme suivant pour résoudre le problème de la recherche de pic :

```
def rechercher_pic(t):
    n = len(t)
    début = 0
    fin = n - 1
    while (fin > début):
        i = (début + fin) // 2
        if i > 0 and t[i] < t[i-1]:
            fin = i - 1
        elif i < n-1 and t[i] < t[i+1]:
            début = i + 1
        else:
            return i
    return début
```

1. Exécuter l'algorithme sur le tableau $[2, 3, 1, 0, 1, 2, 4, 1, 3, 5]$, en précisant l'évolution des variables.
2. Trouver un exemple de tableau de longueur 6 où l'algorithme renvoie 5, en précisant comment s'exécute l'algorithme sur l'exemple choisi.
3. Justifier que l'algorithme termine.
4. Justifier que l'algorithme est correct.
5. Donner la complexité de l'algorithme, en justifiant votre réponse.

Exercice 5 On considère différents algorithmes pour le calcul des moyennes d'une classe pour différentes matières. On suppose données les notes de chaque étudiant sous la forme d'une matrice dont les lignes contiennent les notes de chaque étudiant et les colonnes regroupent les notes d'une même matière. Pour chacun des algorithmes ci-dessous, indiquez :

— si l'algorithme est correct (si ce n'est pas le cas, donnez un exemple d'entrée pour laquelle la sortie de l'algorithme est incorrecte) ;

— l'ordre de grandeur de la complexité algorithmique dans le pire des cas de l'algorithme (en fonction du nombre m de matières et du nombre n d'étudiants).

Algorithme 1 :

```
def moyennes_classe(notes):
    nb_matières = len(notes)
    nb_élèves = len(notes[0])
    moyennes = [0]*nb_matières
    for m in range(nb_matières):
        for e in range(nb_élèves):
            moyennes[m] += notes[e][m] / nb_élèves
    return moyennes
```

Algorithme 2 :

```
def moyennes_classe(notes):
    nb_élèves = len(notes)
    nb_matières = len(notes[0])
    sommes = [0]*nb_matières
    for élève in notes:
        i = 0
        for note in élève:
            sommes[i] += note
            i += 1
    moyennes = []
    for note in sommes:
        moyennes.append(note/nb_élèves)
    return moyennes
```

Algorithme 3 :

```
def moyennes_classe(notes):
    nb_élèves = len(notes)
    nb_matières = len(notes[0])
    moyennes = [0]*nb_matières
    for m in range(nb_matières):
        for e in range(nb_élèves):
            moyennes[m] += notes[e][m]
            moyennes[m] /= nb_élèves
    return moyennes
```

Algorithme 4 :

```
def moyennes_classe(notes):
    nb_élèves = len(notes)
    nb_matières = len(notes[0])
    moyennes = [0]*nb_matières
    for élève in notes:
        i = 0
        while i < nb_matières:
            for j in range(nb_matières):
                if i == j:
                    moyennes[i] += élève[i] / nb_élèves
            i += 1
    return moyennes
```