

Algorithmique

Correction Partiel n° 1 (P1)

INFO-SUP S1 – EPITA

8 Jan. 2019 - 10 : 00

Solution 1 (Dichotomie : "chemin" de recherche – 3 points)

- | | | |
|---|-----------------------------|-------|
| ① | 50 - 15 - 48 - 22 - 46 - 42 | OUI – |
| ② | 48 - 15 - 45 - 22 - 47 - 42 | – NON |
| ③ | 15 - 22 - 45 - 43 - 35 - 42 | OUI – |
| ④ | 22 - 45 - 43 - 15 - 35 - 42 | – NON |
-

Solution 2 (Algorithmes de recherche – 2 points)

1. Recherche séquentielle sans tenir compte de l'ordre : 13
 2. Recherche séquentielle en tenant compte de l'ordre : 9
 3. Recherche dichotomique : $8 = 2 \times 4$
-

Solution 3 (Voir Syracuse – 3 points)

Spécifications :

La fonction `Syracuse(n)` retourne la liste des valeurs de la suite de Syracuse à partir de n si $n \geq 1$. Dans le cas contraire elle retourne une liste vide.

```
1  def Syracuse(n):
2      if n <= 0:
3          return []
4      else:
5          L = [n]
6          while n != 1:
7              if n % 2 == 0:
8                  n = n // 2
9              else:
10                 n = 3 * n + 1
11                 L.append(n)
12                 return L
```

Solution 4 (Progression arithmétique – 4 points)

Spécifications :

La fonction `arithmetic(L)` vérifie si la liste L a au moins 2 éléments et suit une progression arithmétique. Elle retourne la valeur de la raison si c'est le cas, la valeur 0 sinon.

Principe

Si la liste a au moins deux éléments, la raison potentielle est donnée par la différence entre les deux premiers éléments. On parcourt la liste tant que l'élément suivant est égal à l'élément courant ajouté à la raison. Si le parcours atteint le dernier élément de la liste alors la propriété est vraie, fausse sinon. A noter que si les deux premiers éléments sont égaux, le parcours n'est pas effectué.

```
1 def arithmetic(L):
2     n = len(L)
3     if n < 2 or L[1] - L[0] == 0:
4         return 0
5     else:
6         diff = L[1] - L[0]
7         i = 1
8         while i < n and L[i-1] + diff == L[i]:
9             i += 1
10        if i == n:
11            return diff
12        else:
13            return 0
```

Solution 5 (Suppression dans liste triée – 5 points)

Spécifications :

La fonction `delete(L, x)` supprime la valeur x , si elle existe, dans la liste L strictement croissante et retourne un booléen indiquant si la suppression a pu être effectuée.

Principe :

- Recherche :
On se place sur le premier élément de la liste.
Tant qu'il reste des éléments, et que l'élément courant n'a pas dépassé x , on avance à l'élément suivant.
- Suppression :
Si on a trouvé x (si on n'a pas parcouru toute la liste et si l'élément sur lequel on s'est arrêté est x) :
 - On décale toutes les valeurs qui suivent x d'une place à gauche.
 - On "supprime" la dernière case.

```
1 def delete(L, x):
2     n = len(L)
3     i = 0
4     while i < n and x > L[i]:
5         i += 1
6     if i == n or L[i] != x:
7         return False
8     else:
9         for j in range(i, n-1):
10            L[j] = L[j+1]
11        L.pop()
12        return True
```

Solution 6 (What is it? – 3 points)

1. Résultat de l'application suivante de *what* :

```
1 >>> what ([1,3,2,8,7,2,5,4,0,6,2,15])  
2 [0, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 15]
```

2. On appelle *what*(L) avec L une liste d'entiers naturels.

- (a) À la fin de la première boucle *me* représente la valeur maximum de la liste (si L est vide, la valeur 0...)
- (b) À la fin de la troisième boucle *X* est un histogramme des valeurs de L .
- (c) La fonction retourne une copie de L triée en ordre croissant.

3. **Bonus** : Complexité : $3n + me$ avec n la longueur de L ($O(\max(n, me))$).