DS 1 : jeudi 5 novembre

Correction

Exercice 1 *(SQL : Station de ski)*.

Correction :

1°
```sql
SELECT nom
FROM Domaines
```

2°
```sql
SELECT DISTINCT nom
FROM Bornes
```

3°
```sql
SELECT COUNT()
FROM Domaines
```

4°
```sql
SELECT id_passage
FROM Passages
WHERE id_forfait=4 AND passage=1
```

5°
```sql
SELECT nom, prenom
FROM Clients
GROUP BY id_client
HAVING COUNT()>=2
```

6°
```sql
SELECT DISTINCT nom
FROM Bornes
WHERE id_borne NOT IN (SELECT DISTINCT id_borne
                       FROM Bornes
                       WHERE id_domaine = 3)
```

7°
```sql
SELECT nom, prenom
FROM Forfaits
    JOIN Clients ON Clients.id_forfait = Forfaits.id_forfait
WHERE Forfaits.date = "2020-12-31"
```

8°
```sql
SELECT Passages.date, Passages.heure
FROM Passages
    JOIN Forfaits ON Passages.id_forfait=Forfaits.id_forfait
    JOIN Domaines ON Forfaits.id_domaine=Domaines.id_domaine
WHERE nom="Débutant"
```

9°
```sql
SELECT DISTINCT heure
FROM Passages
    JOIN Bornes ON Passages.id_borne=Bornes.id_borne
WHERE nom="TK Chavanette"
```

10°
```sql
SELECT id_passage
FROM Passages
    JOIN Forfaits ON Passages.id_forfait = Forfaits.id_forfait
    JOIN Bornes ON Passages.id_borne = Bornes.id_borne
WHERE Forfaits.id_domaine=Bornes.id_domaine
```

11°
```sql
SELECT id_passage
FROM Passages
WHERE id_passage NOT IN (SELECT id_passage
                         FROM Passages
                             JOIN Forfaits ON Passages.id_forfait = Forfaits.id_forfait
                             JOIN Bornes ON Passages.id_borne = Bornes.id_borne
                             WHERE Forfaits.id_domaine=Bornes.id_domaine)
```

Exercice 2 *(E3A PSI, 2018)*.

Correction :

1° Par exemple (avec la condition que L doit être non vide) :

```
def maxi(L):
    res = L[0]
    for k in range(1,len(L)):
        if L[k] > res:
            res = L[k]
    return res
```

2° Par exemple :

```
def ind(L):
    res = []
    for k in range(len(L)):
        if L[k] != 0:
            res.append(k)
    return res
```

3° Par exemple (avec la condition que L doit être non vide) :

```
def nb_oc(L):
    M = maxi(L)
    res = [0]*(M+1)
    for x in L: # On peut aussi parcourir les indices
        res[x] += 1
    return res
```

4° 4°.1. Lors de l'appel `nb_oc(L)` (donnée en 3°), la liste `L` est parcourue deux fois : une première pour l'appel le calcul de `M` et une seconde fois pour le remplissage de res.

4°.2. La fonction donnée en 3° satisfait donc à la condition (le nombre d'appel est constant égal à deux donc indépendant de `M`).

5° 5°.1. On trouve $L_1 = [1,4,3,3,2,2,3,1,1,0]$, $L_2 = [1,4,3,3,2,2,3,1,1,0] = L_1$, et donc par récurrence directe, $L_k = L_1$ pour tout $k \geq 1$.

5°.2. Les éléments d'indice impaire de L_1 sont les nombres qui apparaissent dans L_0 et sont donnés par ordre décroissant, or dans L_1 les éléments d'indice impaire sont 4, 1 puis 2, ils ne sont donc pas décroissant, ainsi B ne peut pas être la liste L_1 d'une suite de Robinson, il n'existe donc pas de L_0 tel que $L_1 = B$.

5°.3. Une liste L_0 qui donne une telle liste L_1 doit donc avoir deux 4 et un 0, les L_0 qui donnent un tel L_1 sont donc exactement les trois listes : $[0,4,4]$, $[4,0,4]$ et $[4,4,0]$.

5°.4. Par exemple (avec la condition que L doit être non vide) :

```
def rob(A,n):
    L = A
    for k in range(n):
        T = nb_oc(L)
        I = ind(T)
        L = []
        for l in len(I):
            i=I[len(I)-1-l] # On parcours I dans l'autre sens
            L.append(T[i])
            L.append(i)
    return L
```

On peut aussi mettre `for i in I[::-1]` pour parcourir directement `I` dans l'autre sens (et non ses indices).

Exercice 3.

Correction :

1° (a) `def somme(L,i,j):`

```
    res=0
    for k in range(i,j+1):
        res+=L[k]
    return res
```

(b)
```
def somme_min1(L):
    res=L[0]
    n=len(L)
    for i in range(n):
        for j in range(i,n):
            test=somme(L,i,j)
            if test<res:
                res=test
    return res
```

(c) L'appel effectue $j-i+1$ opérations, le nombre d'opérations est donc $\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} j-i+1 = \sum_{i=0}^{n-1}\sum_{j=1}^{n-i} j = \sum_{i=0}^{n-1} \frac{(n-i)(n-i+1)}{2} = O(n^3)$.

2° (a)
```
def cumsum(t):
    n=len(t)
    res=[t[0]]
    for k in range(1,n):
        res.append(res[k-1]+t[k])
    return res
```

(b) Cela correspond à `S[j]-S[i]+L[i]` (ou `S[j]-S[i-1]`, mais le cas $i=0$ doit se traiter à part).

(c)
```
def somme_min2(L):
    S=cumsum(L)
    n=len(L)
    res=L[0]
    for i in range(n):
        for j in range(i,n):
            test=S[j]-S[i]+L[i]
            if test<res:
                res=test
    return res
```

(d) On est en $O(n^2)$

3° (a)
```
def somme_min3(L):
    s=0
    s_cur=0
    for e in L:
        s_cur=s_cur+e
        if e<s_cur+e:
            s_cur=e
        if s_cur<s:
            s=s_cur
    return s
```

(b) Il est clairement en $O(n)$.