DS2 INFORMATIQUE : CORRIGÉ

1. `L=[0,1,0,1,0,1,0,1]`   (La première case a pour indice 0)

2. *Note :* certains ont cru que la liste `L` de cette question était celle de la question précédente. Si tel était le cas, la fonction `occupe` n'aurait pas besoin de dépendre de `L`, mais uniquement de `i`. Comme on demandait une fonction `occupe(L,i)`, il fallait comprendre que `L` était une liste quelconque. Cependant, le sujet était ambigu, donc des points ont été accordés s'il y a eu méprise et que la réponse était juste.

```
def occupe(L,i):
    if L[i]==1:          # Plus simple encore : return L[i]==1
        return True
    else:
        return False
```

3.

```
import random     # randint devient utilisable avec random.randint
def file_alea(n):
    L=[]
    for k in range(n):      # on n'utilie pas la variable k
        L.append( random.randint(0,1) )
    return L
```

4. Chaque case peut être ou bien occupée, ou bien inoccupée : il y a donc 2 cas possibles par case. Comme il y a n cases, on a donc $\underbrace{2\times\cdots\times 2}_{n \text{ fois}} = 2^n$ cas possibles.

5.

```
1 def egal(L1,L2):
2     if len(L1)!=len(L2):  # pour que L1=L2, il faut avoir la même taille
3         return False
4     n = len(L1)           # à ce stade, L1 et L2 ont la même taille n
5     for i in range(n):
6         if L1[i]!=L2[i]:  # si les cases i de L1 et L2 diffèrent...
7             return False
8     return True
```

6. Le pire cas arrive, par exemple, lorsque les listes `L1` et `L2` sont égales (en effet, on éxécute alors la totalité du script jusqu'à `return True`). On fait alors :
   - une opération élémentaire à la ligne 2,
   - n opérations élémentaires à la ligne 6.

   On fait ainsi $n+1$ opérations élémentaires, donc on obtient une complexité d'ordre n.

7. `avancer([0,1,1,0],False)` renvoie `[0,0,1,1]`, et donc l'instruction de la question renvoie `[1,0,0,1]`.

8.

```python
def avancer_fin(L,m):
    n=len(L)
    L1 = L[0:m]              # L1 correspond aux cases de L jusqu'à m-1
    L2 = L[m:n]              # L2 correspond aux cases de L à partir de m
    L2_av = avancer(L2,False) # à partir de la case m, les voitures bougent
    return L1+L2_av          # mais pas les voitures des cases 0 à m-1
```

9.

```python
def avancer_debut(L,b,m):
    n=len(L)
    L1 = L[0:m+1]  # la dernière case de L1 est vide (indice m)
    L2 = L[m+1:n]
    L1_av = avancer(L1,b) # les voitures des cases 0 à m-1 bougent
    return L1_av+L2       # mais pas les voitures à partir de la case m+1
```

10. On cherche la première case libre en reculant à partir de $m-1$: on notera k l'indice correspondant à cette case libre.
    - Si $k=m-1$, la case $m-1$ est libre et toute les voitures avant m peuvent avancer normalement.
    - Si $k \leq m-2$, alors il y a des voitures sur toutes les cases $k+1,\dots,m-1$: elle sont bloquées et n'avanceront pas. Par contre, les voitures des cases 0 à $k-1$ peuvent avancer normalement.

```python
def avancer_blocage(L,b,m):

    # On cherche l'indice k de la première case vide avant m
    k=m-1
    libre = False
    while libre==False and k>=0:
        if L[k]==0:
            libre=True
        else:
            k=k-1
    # k est maintenant l'indice d'une case vide (sauf si k vaut -1)

    if k==-1:           # la file est remplie : tout est bloqué
        return L
    else:
        M = avancer_debut(L,k)
        return M        # les autres voitures ne bougent pas
```

11.

```
def avancer_files(L1,b1,L2,b2):
    n = len(L1) # n est impair
    m = (n-1)/2 # indice du milieu

    # Les voitures sur la file L1 avancent toujours normalement
    # Si b1 vaut True, alors une voiture arrive en case 0 pour L1
    M1 = avancer(L1,b1,m)

    # On vérifie s'il y a une voiture en case m qui bloque la file verticale
    if M1[m]==0:                  # pas de blocage
        M2 = avancer(L2,b2,m)    # les voitures de L2 avancent normalement

    else:                         # il y a blocage
        # Les voitures de L2 à partir de la case m avancent normalement
        M2b = avancer_fin(L2,m)
        M2a = avancer_blocage(L2,b2,m)
        M2 = M2a[0:m+1] + M2b[m+1:n]

    return [M1,M2]
```

12. Cette instruction retourne

```
[ [0,0,1,0,1] , [1,1,0,1,0] ]
```

13.

$$r(M) = 2^5 + 2^2 = 32 + 4 = 36 \qquad r(N) = r(M) + 1 = 37$$

14.

```
def rep(L):
    s = 0
    for i in range(len(L)):
        s = s + L[i]*2**(n-1-i)
    return s
```

15.

$$\begin{aligned} r(L) &= \left(\sum_{i=0}^{n-2} 2^{n-1-i} L_i \right) + L_{n-1} \\ &= 2 \left(\sum_{i=0}^{n-2} 2^{n-2-i} L_i \right) + L_{n-1} \end{aligned}$$

et comme $0 \leq L_{n-1} < 2$, il s'agit bien de la division euclidienne recherchée. Ainsi, le quotient est $\sum_{i=0}^{n-2} 2^{n-2-i} L_i$ et le reste est L_{n-1}.

Une rédaction rigoureuse nécessiterait de traiter séparément le cas $n = 1$, pour lequel le quotient vaut 0, et le cas $n > 1$, pour lequel le quotient est l'expression ci-dessus, puis enfin remarquer que l'expression du cas $n > 1$ est valable pour $n = 1$ (avec la convention qu'une somme sur le vide vaut 0). Cependant, pour les épreuves d'informatique, on est moins regardant sur ce genre de « détails » mathématiques.

16.

$$L' = [0, L_0, L_1, \dots, L_{n-2}]$$

$$r(L') = 2^{n-2}L_0 + 2^{n-3}L_1 + \cdots + 2L_{n-3} + L_{n-2} = \sum_{i=0}^{n-2} 2^{n-2-i}L_i$$

On reconnait que $r(L')$ est le quotient de la division euclidienne de $r(L)$ par 2. Ainsi, on peut écrire :

```
rp = r // 2
```

17.

$$L' = [1, L_0, L_1, \dots, L_{n-2}]$$

$$r(L') = 2^{n-1} + 2^{n-2}L_0 + 2^{n-3}L_1 + \cdots + 2L_{n-3} + L_{n-2} = 2^{n-1} + \sum_{i=0}^{n-2} 2^{n-2-i}L_i$$

Ainsi, on peut écrire :

```
rp = 2**(n-1) + r // 2
```

18. Par la question 15, L_{n-1} est le reste de la division euclidienne de $r(L)$ par 2. On peut ainsi obtenir l'élément d'indice $n-1$ de L. Ensuite, on remarque que le quotient de cette division est précisément

$$r(L') = 2^{n-2}L_0 + 2^{n-3}L_1 + \cdots + 2L_{n-3} + L_{n-2}$$

et donc on peut obtenir L_{n-2} comme le reste de la division euclidienne de $r(L')$ par 2, et ainsi de suite. Le code est le suivant.

```python
def inv_rep(r,n):
    if r>=2**n:
        return None   # si len(L)=n, alors r(L) < 2**n

    L=[]
    for i in range(n-1,-1,-1):  # [n-1,n-2,...,0]
        q = r//2
        reste = r//2
        L[i] = reste
        r = q
    return L
```

19.

```python
def avance(L,b):

    r = rep(L)
    if b==False:
        rp = r//2
    else:
        rp = 2**(n-1) + r // 2

    n = len(L)
    Lp = inv_rep(rp,n)
    return Lp
```