ALGORITHMES CLASSIQUES (corrigé)

1) a)
```
def binaire(n,p) :
    L = [] ; q = n
    for i in range(p) :  L.append(q%2) ; q = q//2
    return L
```

b)
```
def binaire2(n) :
    L = [] ; q = n
    while q>0 :  L.append(q%2) ; q = q//2
    return L
```

2) a)
```
def partie(n,p) :
    L = binaire(n,p) ; E = []
    for i in range(p) :
        if L[i] == 1 :  E.append(i)
    return E
```

b)
```
def enumere(p) :  return [ partie(n,p) for n in range(2**p)]
```

3) a)
```
from numpy.linalg import *

def power(A,n) :                    # fonction récursive
    if n == 0 :  return eye(len(A))   # eye(n) renvoie la matrice I_n
    r = m%2 ; m = n//2 ; B = power(A,m)
    if r == 0 :  return dot(B,B)
    else :  return dot(A,dot(B,B))
```

b)
```
def power(A,n) :                    # fonction itérative
    L = binaire(n)     ## on utilise la fonction définie au 1) a)
    B = eye(len(A))
    while L != [] :      ## la dernière itération correspond au bit des unités
        if L.pop() == 0 :  B = dot(B,B)
        else :  B = dot(A,dot(B,B))
    return B
```

4) a)
```
def ordre(X,Y) :
    p = len(X)
    for k in range(p) :
        if X[k]<Y[k] :  return True
        if X[k]>Y[k] :  return False
    return True
```

Variante sans break :

```
def ordre(X,Y) :
    p = len(X) ;  k = 0
    while k<p and X[k] == Y[k] :  k = k+1
    return (k==p or X[k]<Y[k])    ## le cas k = p correspond au cas où X = Y
```

b)
```
def ordre(X,Y) :
```

```
        p = len(X) ; q = len(Y)
        while k<min(p,q) :
            if X[k]<Y[k] :  return True    ## correspond au cas où X ≺ Y
            if X[k]>Y[k] :  return False   ## correspond au cas où X ≻ Y
        return p<=q      ## correspond au cas où X préfixe de Y
```

5) a) `##` Avec un *break* :

```
def is(x,L) :
        for i in range(len(L))
            if L[i] == x :  return i
        return n
```

`##` Sans *break* :

```
def is(x,L) :
        i = 0 ; n = len(L)
        while i<n and L[i] != x
            i = i+1
        return i
```

b)
```
def minimum(L) :
        k = 0 ; x = L[0] ; n = len(L)
        for i in range(1,n)
            if L[i] < x :  x = L[i]
        return x
```

Variante dans le cas des entiers :

```
def minimum(L) :
        x = math.inf     ##  représente +∞
        for i in range(n)
            if L[i] < x :  x = L[i]
        return x

def indiceMin(L) :
        k = 0 ; n = len(L)
        for i in range(1,n)
            if L[i] > L[k] :  k = i
        return k
```

c)
```
def premier(L) :                ## on pourrait utiliser indiceMin(L)
        k = 0
        for i in range(len(L))
            if L[i] < L[k] :  k = i
        L[0] , L[k] = L[k] , L[0]
```

6)
```
def inverse(L) :
        S = [] ; n = len(L)
        for i in range(n-1,-1,-1) :  S.append(L[i])
```

```
        return S
```

Variante :

```
def reversed(L) :
        S = []
        while L != [] :  S.append(L.pop())
        return S
```

Remarque : Ne pas confondre avec : `for x in L : S.append(x)` qui fait une copie de L ...

7) a) `##` Version récursive :

```
def aux(x,L,i,j) :   ## teste l'appartenance à L[i : j]
     if i == j :  return False
     k = (i+j)//2    ## on a toujours i ≤ k < j
     if x == L[k] :  return True
     if x < L[k] :  return aux(x,L,i,k)
     if x > L[k] :  return aux(x,L,k+1,j)
def appart(x,L ) : return aux(x,L,0,len(L))
```

`##` Version itérative :

```
def appart(x,L) :
     i = 0 ; j =len(L)
     while i<j :
         k = (i+j)//2
         if x == L[k] :  return True
         if x < L[k] :  j = k
         if x > L[k] :  i = k+1
     return False
```

Remarque : A chaque itération, la longueur $(j - i)$ du sous-tableau considéré diminue d'un facteur 2 au moins. L'algorithme aboutit donc en $O(2^p)$ opérations, avec $p \leq 1 + \lfloor \log n \rfloor$.

b)

```
def fusion(L,M) :
        n = len(L) ; m = len(M)
        S = [] ;  i = 0 ; j = 0
        while i<n and j<m :
             if L[i]<M[j] :  S.append(L[i]) ; i = i+1
             else :  S.append(L[j]) ; j = j+1
        for k in range(i,n) :  S.append(L[k])
        for k in range(j,m) :  S.append(M[k])
        return S
```

Variante avec une boucle for :

```
def fusion(L,M) :
        n = len(L) ; m = len(M)
        S = [] ;  i = 0 ; j = 0
        for _ in range(n+m) :
```

```
            if i<n :
               if (j<m and L[i]<M[j]) or (j == m) :
                   S.append(L[i]) ; i = i+1
                else :  S.append(L[j]) ; j = j+1
           else :  S.append(L[j]) ; j = j+1
        return S
```

8) a)
```
def ensemble(L) :
        M = [] ; n = len(L)
        for i in range(n) :
            x = L[i] ; flag = True    # flag indique si i est la première occurrence de x
            for j in range(i+1,n)
                 if L[j] == x :  flag = False
            if flag :  M.append(x)    # on ajoute la première occurrence de chaque élément
        return M
```

La complexité est en $O(n^2)$.

b)
```
def reduit(L) :
        n = len(L) ; S = []
        for k in range(n-1) :
             if L[k+1] > L[k] :  S.append(L[k])
        return S.append(L[n-1])
```

c) Il suffit de trier d'abord les éléments de L (selon un ordre total arbitraire) et ensuite la suppression des doublons se fait en temps linéaire, d'où une complexité en $O(n \log n) + O(n) = O(n \log n)$.

d) L'idée est d'utiliser un dictionnaire dont les clés sont les éléments de x.
On attribue une valeur arbitraire au clé, par exemple `True`.

```
def ensemble(L) :
        dico = {}
        for x in L : dico[x] = True
        return dico.keys()
```

9) a)
```
def elementMax(L) : return L[len(L)-1]
def supprime(L) : e = L.pop() ; return e[0]
def ajoute(e,L):
        (x,k) = e
        if element(x,L) == -1 :
             n = len(L) ; L.eppend(e) ; i = n
             while i > 0 and L[i-1][1] > k :
                  L[i-1],L[i] = L[i],L[i-1]
def modif(e,L) :
        (x,k) = e ; i = element(x,L)
        if i != -1 :
             h = L[i][1] ; L[i] = e ; n = len(L)  ## h ancienne priorité de x
```

```
            if h < k :
                 while i < n-1 and L[i+1][1] < k :  L[i+1],L[i] = L[i],L[i+1]
            elif h > k :
                while i > 0 and L[i-1][1] > k :  L[i-1],L[i] = L[i],L[i-1]
        return L
def creer(E) :
        L = []
        for e in E : ajoute(e,L)
        return L
```

b) On note n la longueur de L.

`elementMax(L)` et `supprime(L)` sont en $O(1)$

`element(x,L)` est en $O(\log n)$

`ajoute(e,L)` et `modif(e,L)` sont en $O(n)$

`creer(E)` est ici en $O(n^2)$ mais on obtiendrait $O(n \log n)$ si on utilisait un tri rapide (relatif à k).

10) a)
```
def select(A,p) :
        n = len(A)
        for i in range(p) :
            k = i
            for j in range(i+1,n) :
                 if A[j]<A[k] :  k = j
            A[i],A[k] = A[k],A[i]
        return A[p-1]
```

b)
```
def aux(A,p,i,j) :
        k = pivot(A,i,j)
        if p == k-i+1 :  return A[k]
        if p < k-i+1 :  return aux(A,p,i,k)
        if p > k-i+1 :  return aux(A,p-(k-i+1),k+1,j)
```

11)
```
def pivot(A) :
        n = len(A) ; B = [0]*n ; x = A[0] ; i = 1 ; j = n-1
        for k in range(1,n) :
            if A[k] <= x :  B[i] = A[k] ; i = i + 1
            if A[k] > x :  B[j] = A[k] ; j = j - 1
               #   à la sortie, i = j car j − i décroît de 1 à chaque itération.
        A[i] = x
        A = B ; return i
```

```
def pivot2(A) :
        x = A[0] ; i = 1 ; j = n-1
        while i < j  :      #    il vaut mieux faire une seule opération à la fois :
            if A[i] <= x :  i = i + 1
            elif A[j] > x :  j = j - 1
            else :  A[i],A[j] = A[j],A[i]
```

```
            #   à la sortie, i = j ; et pour k < i, on a A[k] < x ; et pour k > j, on a A[k] > x
        if A[i] < x :  A[0],A[i] = A[i],A[0]
        elif A[i] > x :  A[0],A[i-1] = A[i-1],A[0]
            # dans la précédente instruction, il se peut que i soit égal à 1
        return i

def pivot3(A) :
        n = len(A) ; i = 1 ; j = n-1 ; x = A[0]
        while i <= j :
             if A[i] > x :
                  A[i],A[j] = A[j],A[i] ; j = j-1
             else :  i = i+1
        ##  tous les termes d'indice compris strictement entre j et n sont > x (pivot)
        if A[i] <= x :
             A[0],A[i] = A[i],A[0] ; return i
        else :
             A[0],A[i-1] = A[i-1],A[0] ; return i
```