```python
#ex1 : File

#1
def creerFile(n):
    return [0,0,[0]*n]
#2
def estVide(F):
    return F[0]-F[1]==0

#3
def estPleine(F):
    return F[0]-F[1]==len(F[2])

#4
def enfiler(F, x):
    assert not estPleine(F), "File Pleine"
    n=len(F[2])
    queue = F[0]
    F[2][queue % n]=x
    F[0]+=1

#5
def defiler(F):
    assert not estVide(F), "File vide"
    n=len(F[2])
    tete = F[1]
    indice = tete % n
    F[1]+=1
    return F[2][indice]

#6
def sommet(F):
    assert not estVide(F), "File vide"
    n=len(F[2])
    tete = F[1]
    indice = tete % n
    return F[2][indice]

#Exercice 2: Nombres de Hamming
#1
def Nombres_Hamming(n):
    f2 = creerFile(n)
    f3 = creerFile(n)
    f5 = creerFile(n)
    enfiler(f2, 1)
    enfiler(f3, 1)
    enfiler(f5, 1)
    for i in range (n):
        #On détermine le plus petit des trois têtes de file,
        #que l'on note k et que l'on imprime à l'écran ;
        k=min(sommet(f2),sommet(f3),sommet(f5))
        print(k)
        #On retire cet élément des files où il se trouve ;
        if sommet(f2)==k:
            defiler(f2)
        if sommet(f3)==k:
            defiler(f3)
        if sommet(f5)==k:
            defiler(f5)
        #on insère en queue des files f2, f3 et f5 les entiers 2k, 3k et 5k.
        enfiler(f2,2*k)
        enfiler(f3,3*k)
        enfiler(f5,5*k)

#tester Nombres_Hamming(n)
n=18
Nombres_Hamming(n)
```

```python
#Exercice 3
#1
def init_pile(n):
    p =[0]*(n+1)
    p[0]=1
    return p
#test
p= init_pile(5)

#2
def pile_vide(p):
    return p[0]==1



#3
def pile_pleine(p):
    return p[0]==len(p)

#4
def empiler(p,x):
    assert not pile_pleine(p)
    sommet = p[0]
    p[sommet] = x
    p[0]+=1

#test
L=[2,5,4,8,9]
for i in L:
    empiler(p,i)

#5
def depiler(p):
    assert not pile_vide(p)
    sommet = p[0]-1
    p[0]-=1
    return p[sommet]

#6
def copier(p):
    p_temp = init_pile(len(p)-1)
    p_copie = init_pile(len(p)-1)

    while not pile_vide(p):
        empiler(p_temp,depiler(p))

    while not pile_vide(p_temp):
        elt = depiler(p_temp)
        empiler(p,elt)
        empiler(p_copie,elt)

    return p_copie


#7
def echanger(p):
    elt1 = depiler(p)
    elt2 = depiler(p)
    empiler(p,elt1)
    empiler(p,elt2)

#8
def enroler(p,n):
```

```
    if len(p)-1 < n :
        print(n,"dépasse la taille de la pile")
    else:
        elt = depiler(p)
        p_temp = init_pile(n-1)
        for i in range(n-1):
            empiler(p_temp,depiler(p))

        empiler(p,elt)
        for i in range(n-1):
            empiler(p,depiler(p_temp))


#9
def superposition (p1, p2):
    p_temp = init_pile(len(p1) + len(p2) - 2)
    while not pile_vide(p2):
        empiler(p_temp,depiler(p2))
    while not pile_vide(p1):
        empiler(p_temp,depiler(p1))
    return p_temp

#10
def inserer (x, p):
    #La pile p est supposée triée en ordre croissant
    #le sommet de p est le plus grand element
    assert not pile_pleine(p),"Pile peine, impossible d'insérer "+str(x)
    if pile_vide(p):
        empiler(p,x)
    elif p[p[0]-1] <= x :
        empiler(p,x)
    else:
        p_temp = init_pile(len(p)-1)
        while not pile_vide(p) and p[p[0]-1] > x:
            sommet = depiler(p)
            empiler(p_temp,sommet)
        empiler(p,x)
        while not pile_vide(p_temp):
            empiler(p,depiler(p_temp))



#11
def tri_insertion_en_place(p) :
    #copier p
    p_temp = copier(p)
    #vider p
    while not pile_vide(p):
        depiler(p)
    #trier p par insertion
    while not pile_vide(p_temp):
        sommet = depiler(p_temp)
        inserer (sommet, p)

#test
p= init_pile(5)
from random import randint
L=[randint(4,12) for i in range(5)]
for i in L:
    empiler(p,i)
print(p)
tri_insertion_en_place(p)
print(p)
```