Devoir surveillé n°5 – NSI
Correction

Exercice 1

1. Le code est le suivant :

```
def est_maj(x,L):
    n = 0
    for e in L:
        if e == x:
            n+=1
    return (n>len(L)//2)
```

2. Le code est le suivant :

```
def maj(L):
    n = len(L)
    if n == 1:
        return L[0]
    else:
        m = n//2
        x = maj(L[:m])
        y = maj(L[m:])
        if est_maj(x,L):
            return x
        elif est_maj(y,L):
            return y
        else:
            return -1
```

Exercice 2

1. **Questions préliminaires**
   (a) À l'indice 1 du tableau on trouve 8, à l'indice 3 on trouve 7.
   Nous avons $1 < 3$ alors que $8 > 7$, nous avons donc bien une inversion.
   (b) À l'indice 2 du tableau on trouve 3, à l'indice 3 on trouve 7.
   Nous avons $2 < 3$ et $3 < 7$, nous n'avons donc pas d'inversion.
2. **Méthode itérative.**
   (a) i.
   - Cas n°1 : 0
   - Cas n°2 : 1
   - Cas n°3 : 2

   ii. Si on considère l'élément `b` situé à l'indice `i` dans le tableau `tab`. La fonction `fonction1` permet de déterminer le nombre d'éléments plus grands que `b` situés dans le tableau à un indice supérieur à `i`.
   (b) voici le code attendu :

```
def nombre_inversions(tab):
    nb_inv = 0
    n = len(tab)
    for i in range(n-1):
        nb_inv = nb_inv + fonction1(tab, i)
    return nb_inv
```

(c) L'ordre de grandeur de la complexité en temps de l'algorithme est $O(n^2)$.

3. **Méthode récursive.**

(a) Le tri fusion a une complexité en $O(n \log_2(n))$.

(b) Le code attendu peut être le suivant (valable dans beaucoup de langages) :

```
def moitie_gauche(tab):
    n = len(tab)
    nvx_tab = []
    if n==0:
        return []
    mil = n//2
    if n%2 == 0:
        lim = mil
    else :
        lim =mil+1
    for i in range(lim):
        nvx_tab.append(tab[i])
    return nvx_tab
```

Une autre possibilité, plus courte :

```
def moitie_gauche(tab):
    return [tab[i] for i in range(len(tab)//2+len(tab)%2)]
```

ou encore plus court :

```
def moitie_gauche(tab):
    return tab[:(len(tab)+1)//2]
```

(c) Voici le code attendu :

```
def nb_inversions_rec(tab):
    if len(tab) > 1:
        tab_g = moitie_gauche(tab)
        tab_d = moitie_droite(tab)
        return nb_inv_tab(tri(tab_g), tri(tab_d)) +\
               nb_inversions_rec(tab_g) +\
               nb_inversions_rec(tab_d)
    else:
        return 0
```

Voici les codes complets avec des tests pour les deux exercices :

```
## Exercice 1 (élément majoritaire)

def est_maj(x,L):
    n = 0
    for e in L:
        if e == x:
            n+=1
    return (n>len(L)//2)

def maj(L):
    n = len(L)
    if n == 1:
        return L[0]
    else:
        m = n//2
        x = maj(L[:m])
        y = maj(L[m:])
        if est_maj(x,L):
            return x
        elif est_maj(y,L):
            return y
        else:
            return -1

assert maj([1,2,1,3,1]) == 1
assert maj([1,2,1,3,4]) == -1

## Exercice 2 (inversions)

def fonction1(tab, i):
    nb_elem = len(tab)
    cpt = 0
    for j in range(i+1, nb_elem):
        if tab[j] < tab[i]:
            cpt += 1
    return cpt

def nombre_inversions(tab):
    nb_inv = 0
    n = len(tab)
    for i in range(n-1):
        nb_inv = nb_inv + fonction1(tab, i)
    return nb_inv

assert nombre_inversions([1, 5, 7]) == 0
assert nombre_inversions([1, 6, 2, 7, 3]) == 3
assert nombre_inversions([7, 6, 5, 3]) == 6

tri = sorted
```

```python
def moitie_gauche(tab):
    return tab[:(len(tab)+1)//2]

def moitie_droite(tab):
    return tab[(len(tab)+1)//2:]

def nb_inv_tab(tab1, tab2):
    n=0
    for e in tab1:
        n+=len([x for x in tab2 if x<e])
    return n

def nb_inversions_rec(tab):
    if len(tab) > 1:
        tab_g = moitie_gauche(tab)
        tab_d = moitie_droite(tab)
        return nb_inv_tab(tri(tab_g), tri(tab_d)) +\
               nb_inversions_rec(tab_g) +\
               nb_inversions_rec(tab_d)
    else:
        return 0

assert nb_inversions_rec([1, 5, 7]) == 0
assert nb_inversions_rec([1, 6, 2, 7, 3]) == 3
assert nb_inversions_rec([7, 6, 5, 3]) == 6
```