Exercice 1 :

1/
2^h

2/
Si n=0 l'arbre a une feuille
Tous les arbres a n – 1 nœuds internes ont n feuilles
Soit un arbre A avec nœuds
Soit 2 feuilles de profondeur max
Si on les supprime on a -1 feuille et -1 nœud
Donc A a n+1 feuilles.

Un arbre k-aire a (k-1)n+1 feuilles

3/
4/
5/
6/
Un seul

Exercice 2 :

1/
Préfixe : 1 2 4 5 8 9 3 6 7 10
Infixe : 4 2 8 5 9 1 6 3 10 7
Suffixe : 4 8 9 5 2 6 10 7 3 1

2/
Largeur : 1 2 3 4 5 6 7 8 9 10

Exercice 3 :

Exercice 4 :

1/
Chaque sommet a une arête venant de son père sauf la racine donc n-1 arêtes

2/
k-1

3/
$\log(n+1) - 1 \leq h \leq n-1$

Exercice 5 :

1/
```
def hauteur(L) :
    if L == []:return 0
    h = 0
    for i in range(len(L)) :
        tmp = hauteur(L[i])
```

```
            if h < tmp : h = tmp
        return h+1
```
La complexité est la somme des complexités es sous-arbres Θ(n)

2/
```
def hauteur(pere) :
        h = [0] * len(pere)
        for  in range(pere) :
            if pere[i] != i :
                h[i] = h[pere[i]]+1
        hMax = 0
        for i in range(h) :
            if h[i] > hMax :
                hMax = h[i]
        return hMax
```

3/
```
def hauteur(pere) :
        h = 0
        profondeur = [-1] * len(pere)
        for i i range(len(pere)) :
            remplir(i, profondeur, pere)
            h = max(h, profondeur[i])
        return h

def remplir(i, profondeur pere) :
        if profondeur[i] == -1 :
            if pere[i] == i :
                profondeur[i] = 0
            else :
                remplir(pere[i], profondeur, pere)
                profondeur[i] = 1 + profondeur(pere[i])
```

Complexité : linéaire

<u>Exercice 6 :</u>

```
def bfs(A) : #breadth first search
        l1 = [A]
        l2 = []
        while l1!= [] :
            for i in l1 :
                print i[0]
                l2 += i[1]
            l1 = l2
            l2 = []
```

<u>Exercice 7 :</u>

```
def estABR(A) :
        Ecrire l'arbre sous parcours infixe et vérifier si la liste est triée.
```