

# TP corrigé : nombres premiers

## Informatique commune

1. Écrire une fonction `diviseurs` renvoyant la liste des diviseurs positifs d'un entier  $n$ . Par exemple, `diviseurs(36)` doit renvoyer `[1, 2, 3, 4, 6, 9, 12, 18, 36]`.

```
► def diviseurs(n):  
    res = 0  
    for d in range(1, n+1):  
        if n % d == 0: # d divise n?  
            res += 1  
    return res
```

2. En déduire une fonction `premier` déterminant si un entier est premier.

► Comme `premier(n)` appelle `diviseurs(n)`, sa complexité est  $O(n)$  également.

```
def premier(n):  
    return len(diviseurs(n)) == 2
```

3. En déduire une fonction `tous_premiers` telle que `tous_premiers(n)` renvoie la liste des nombres premiers inférieurs à  $n$ .

```
► def tous_premiers(n):  
    res = []  
    for p in range(2, n + 1):  
        if premier(p):  
            res.append(p)  
    return res
```

`tous_premiers` appelle  $n - 1$  fois la fonction `premier` qui est en  $O(n)$ , d'où une complexité totale  $O(n^2)$ .

Le crible d'Ératosthène est un algorithme plus efficace pour obtenir la liste des nombres premiers inférieurs à un entier  $n$  :

On commence par créer une liste  $L$  de taille  $n + 1$  dont tous les éléments sont `True`. On modifie la valeur de  $L[0]$  et  $L[1]$  à `False`. Puis pour chaque indice  $i$  de  $L$ , si  $L[i]$  contient `True`, alors pour chaque  $k$  multiple de  $i$ , on modifie  $L[k]$  en `False`. À la fin,  $L[i]$  vaut `True` si et seulement si  $i$  est premier.

Si vous ne comprenez pas, vous pouvez chercher plus d'explications sur Google (activité de 3ème par exemple).

4. Écrire une fonction `eratosthene` telle que `eratosthene(n)` renvoie la liste  $L$  ci-dessus.

Vérifier que votre fonction marche avec l'exemple de la question précédente.

► Pour créer la liste demandée, on peut soit commencer avec une liste vide et lui rajouter des éléments avec `L.append(...)`, soit utiliser le fait que `[e]*k` construit une liste avec  $k$  fois l'élément  $e$ .

Pour énumérer les multiples de  $i$ , on peut aller de  $i$  en  $i$  dans un `for` :

```
def eratosthene(n):  
    L = []  
    for i in range(n+1):  
        L.append(True)  
    L[0] = False  
    L[1] = False  
    for i in range(2, n+1):  
        if L[i]:  
            for k in range(2*i, n+1, i):  
                L[k] = False  
    return L
```

5. On veut déterminer expérimentalement la complexité de l'algorithme d'Ératosthène. Pour cela, créer une variable pour compter le nombre de fois que `eratosthene` met une valeur de  $L$  à `False` puis l'afficher juste avant le `return`. Comparer avec la complexité de `tous_premiers`.

6. Écrire une fonction `multiplicite` telle que `multiplicite(d, n)` renvoie le plus grand entier  $k$  tel que  $d^k$  divise  $n$ .

►

```
def multiplicite(d, n):
    res = 0
    while n % (d**res) == 0:
        res += 1
    return res-1
```

7. Écrire une fonction **decomposition** ayant un argument **n** et qui renvoie la liste **L** des diviseurs premiers de **n** avec multiplicités. On pourra stocker dans chaque élément de **L** une liste composée de deux entiers naturels : le diviseur et sa multiplicité. Par exemple **decomposition(50)** devra renvoyer la liste **[[2, 1], [5, 2]]**, puisque  $50 = 2^1 \times 5^2$ .



```
def decomposition(n):
    res = []
    L = eratosthene(n)
    for k in range(len(L)):
        if L[k]:
            m = multiplicite(k, n)
            if m != 0:
                res.append([k, m])
    return res
```