Correction TD informatique du chapitre 10 : Arithmétique

Exercice 1 : Crible d'Erathostène

1. (a)

```
import math

def premier(n) :
    j=2
    while j<=math.sqrt(n) : #on teste les diviseurs potentiels inférieurs à la racine de n
        if n%j==0 :
            return False #si on trouve un diviseur le nombre n'est pas premier
        j+=1
    return True #on ne renvoie True qu'à la fin de la boucle, lorsque tous les diviseurs
         potentiels ont été testés.
```

(b)

```
def nbprem(n) :
    l=[] #création d'une liste vide
    for k in range(2,n+1) :
        if premier(k)==True :
            l.append(k) #ajout des nombres premiers dans la liste
    return l
```

(c)
- Pour premier(n), on effectue :
  - 1 affectation,
  - une boucle while de longueur au plus $\sqrt{n}$ avec à chaque fois au plus, une comparaison et une addition,
  - 1 renvoi

  soit au total, au plus, $2+2\sqrt{n}$ opérations.
  La complexité est donc : $O(\sqrt{n})$.
- Pour nbprem(n), on effectue :
  - 1 affectation,
  - une boucle for de longueur $n-1$ avec à chaque fois au plus, un appel à premier(n) qui est en $O(\sqrt{n})$, une comparaison, un ajout,
  - 1 renvoi

  soit au total, au plus, $2+(n-1).(\sqrt{n}+2)$ opérations.
  La complexité est donc : $O(n\sqrt{n})$, c'est-à-dire $O(n^{3/2})$.

2.

```
def erathostene(n) :
    l=[True for k in range(n+1)] #aucun nombre n'est rayé
    l[0]=False #on raye 0
    l[1]=False #on raye 1
    k=2 #on considère le nombre 2
    while k<=math.sqrt(n) :
        if l[k]==True : #on ne considère que les nombres non rayés
            for j in range(2*k,n+1,k) : #pas de k
                l[j]=False #on raye les multiples k
        k+=1
    L=[]
    for k in range(n+1) :
        if l[k]==True :
            L.append(k) #on transforme la liste de booléens en une liste de nombres
    return L
```

3. • On a vu que la complexité de `nbprem(n)` est $O(n^{3/2})$.
   • Pour `erathostene(n)`, on effectue :
     – 1 création de liste de n termes,
     – 2 affectations
     – une boucle while de longueur au plus $\sqrt{n}$ avec, au rang k, au plus une comparaison, une boucle for de longueur au plus $\frac{n}{k}$ constituée d'une affectation, puis une addition,
     – 1 affectation,
     – une boucle for de longueur $n+1$ constituée d'une comparaison et au plus d'un ajout,
     – 1 renvoi,

     soit au total, au plus, $n+2+\sum_{k=2}^{\sqrt{n}}\left(2+\frac{n}{k}\right)+1+2(n+1)+1$ opérations.

     Or : $n+2+\sum_{k=2}^{\sqrt{n}}\left(2+\frac{n}{k}\right)+1+2(n+1)+1 \leq 3n+6+2\sqrt{n}+n\sum_{k=2}^{\sqrt{n}}\frac{1}{k} \leq 3n+6+2\sqrt{n}+n\ln(\sqrt{n}) \leq 3n+6+2\sqrt{n}+\frac{n}{2}\ln n$.

     La complexité est donc : $O(n\ln(n))$.
   • Comme : $n\ln(n) = O(n^{3/2})$, la complexité la meilleure est celle obtenue en utilisant le crible d'Erathostène.

Exercice 2 : E3A - PSI - 2016

1. P0(5)=True
   P1(5)=True
   P0(9)=True
   P1(9)=False

   P0 détermine si un nombre est pair (False sauf pour 2) ou impair (True sauf pour 1).

   P1 détermine si un nombre est premier (True) ou non (False).

2. P2(N) renvoie la liste des nombres premiers inférieurs ou égaux à N de la forme k^2+1.

   P2(127)=[2, 5, 17, 37, 101]

3.

```
def nextPrime(N) :
    k=N+1
    while P1(k)==False :
        k+=1
    return k
```

4. (a)

```
def jumeau(N) :
    p=nextPrime(N)
    while P1(p+2)==False :
        p=nextPrime(p)
    return [p,p+2]
```

(b)

```
def lesJumeaux(N) :
    l=[]
    p=2
    while p+2<=N :
        if P1(p+2)==True :
            l.append([p,p+2])
        p=nextPrime(p)
    return l
```

Exercice 3 : E3A - PC - 2017

1.

```
def divise(p,q) :
    if q%p==0 :
        return True
    else :
        return False
```

2.

```
def estpremier(p) :
    for q in range(2,p) :
        if divise(q,p) :
            return False
    return True
```

3.

```
def phi(p) :
    x=0
    for q in range(2,p+1) :
        if estpremier(q) :
            x+=1
    return x
```

4. (a) On a : $\lim\limits_{n\to+\infty} \dfrac{n}{\ln(n)} = +\infty$ donc : $\lim\limits_{n\to+\infty} \dfrac{n}{\ln(n)}.\dfrac{\ln(n)\varphi(n)}{n} = +\infty$, d'où $\lim\limits_{n\to+\infty} \varphi(n) = +\infty$.
Ce qui prouve qu'il existe une infinité de nombres premiers.

(b)

```
def test(epsilon) :
    n=50
    while abs((phi(n)*log(n)/n)-1)>epsilon :
        n+=1
    return n
```

(c)

```
x=[k for k in range(50,501)]
y=[phi(k) for k in range(50,501)]
plt.plot(x,y)
plt.show()
```