TP4_correction

December 6, 2017

Correction du TP 4
Activité 1 :
Q3 et Q4

```
In [1]: def somme(L,a): #evaluation de la  complexite
            S=0 #1 affectation
            N=len(L) #1 autre
            for k in range(len(L)):
                S=S+L[k]                    #1 opération + 1 affectation que l'on fai
            return S
```

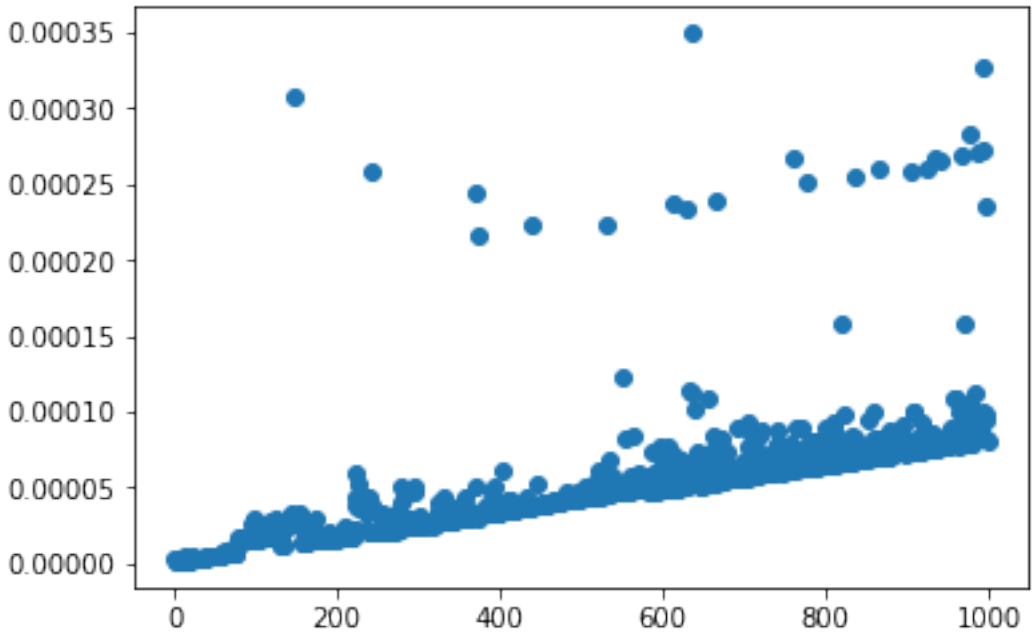
Bref si l'on doit exprimer la complexité on calcule au final l'expression :
$\sum_{k=0}^{N-1} 2 + 2 = 2N + 2 = O(N)$.
Q5

```
In [2]: from time import time
        from pylab import *
        from random import sample



        #Les graphiques


        L=range(10**(8))


        def temps(n,programme,a):
            A=sample(L,n)
            t1=time()
            programme(A,a)
            t2=time()
            return t2-t1


        def echantillon_complexite(programme,a):
            #for k in range(1,10**(4)):
```

```
    #    plot(k,temps(k),'o')
L1=range(1,10**(3))
L2=[temps(k,programme,a) for k in L1]
plot(L1,L2,'o')
show()
```

On écrit ici un script que l'on pourra appliquer dans chaque particulier. Par exemple ici, on saisit :

```
In [3]: echantillon_complexite(somme,0)
```



On voit alors une tendance linéaire (droite) malgré quelques points où le calcul a été un peu plus long.

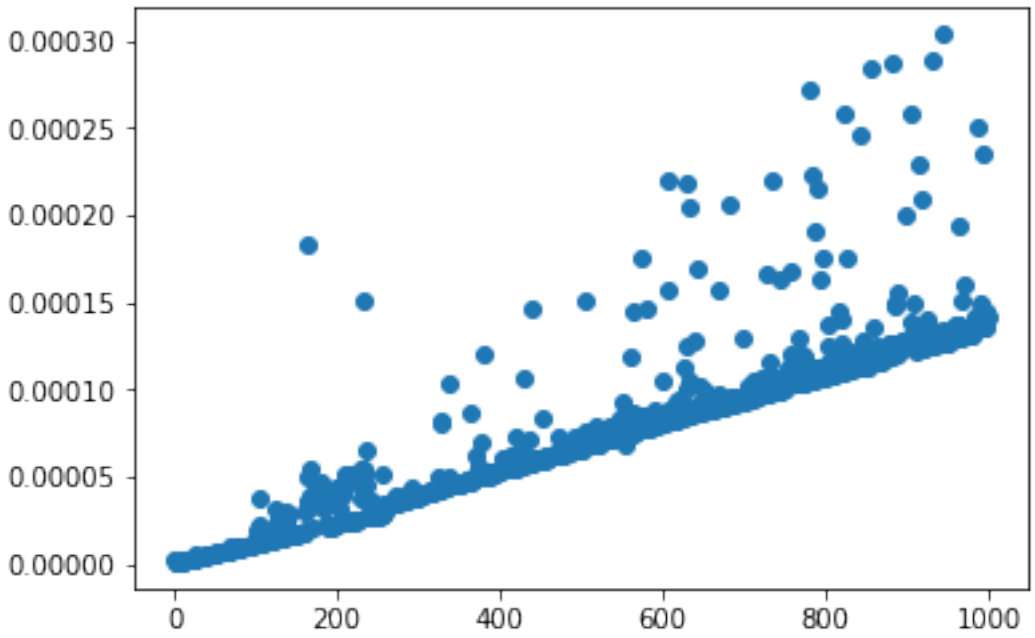
Activité 2 :

Q2 et Q3

```
In [4]: def find(L,a):#évaluation complexité (au pire et au mieux)
            k=0# 1 instruction d'affectation
            N=len(L)# 1 instruction d'affectation
            while L[k]!=a and k<=N-2:       #au mieux rien a faire (a en première po
                k=k+1#1 opération et une instruction d'afectation
            if k<=len(L)-2:
                return("trouvé en position",k)
            elif k==len(L)-1 and L[k]==a:
                return("trouvé en position",k)
            else :
                return(a,"n'est pas dans la liste")
```

La complexité au mieux est donc 2 et au pire : $\sum_{k=0}^{N-2} 2 + 2 = 2N$.

```
In [9]: echantillon_complexite(find,100)
```

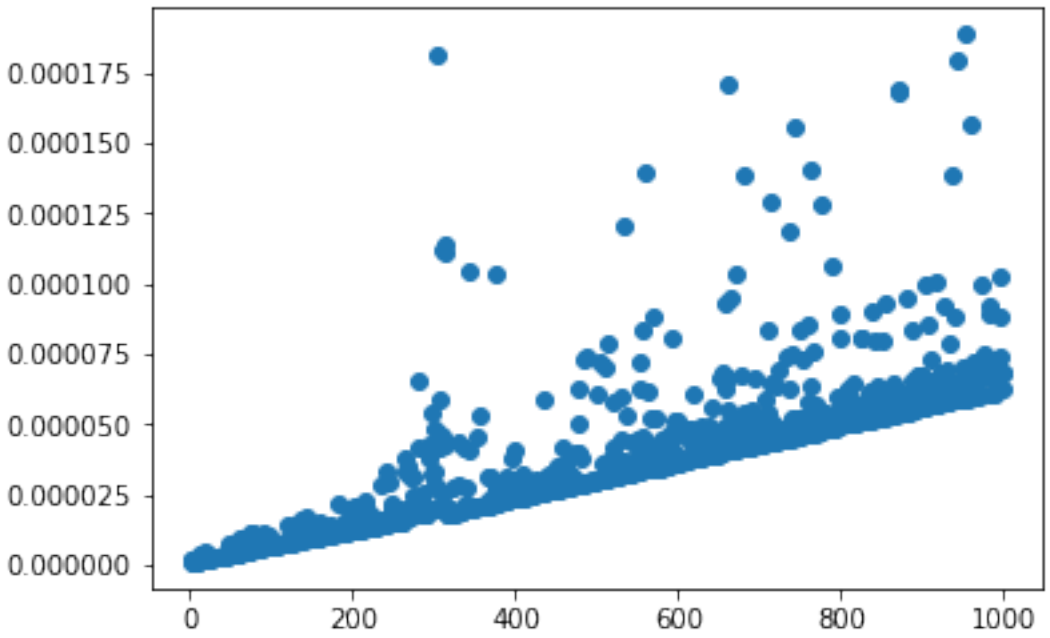


On constate donc expérimentalement toujours cette tendance linéaire.
Activité 4 :
Q2 et Q3

```
In [10]: def max(L,a): #évaluation complexite au mieux et au pire
             x=L[0]#1 instruction d'affectation
             N=len(L)#1 instruction d'affectation
             for k in range(N):#itérations de 0 à N-1
                 if L[k]>x:#1 instruction de comparaison
                     x=L[k]    #au pire une affectaion sinon rien
             return x
```

Ceci nous donne une complexité au mieux de $2 + \sum_{k=0}^{N-1} 1 = N + 2$ et au pire de $2 + \sum_{k=0}^{N-1} 2 = 2N + 2$.
Q4

```
In [11]: echantillon_complexite(max,0)
```

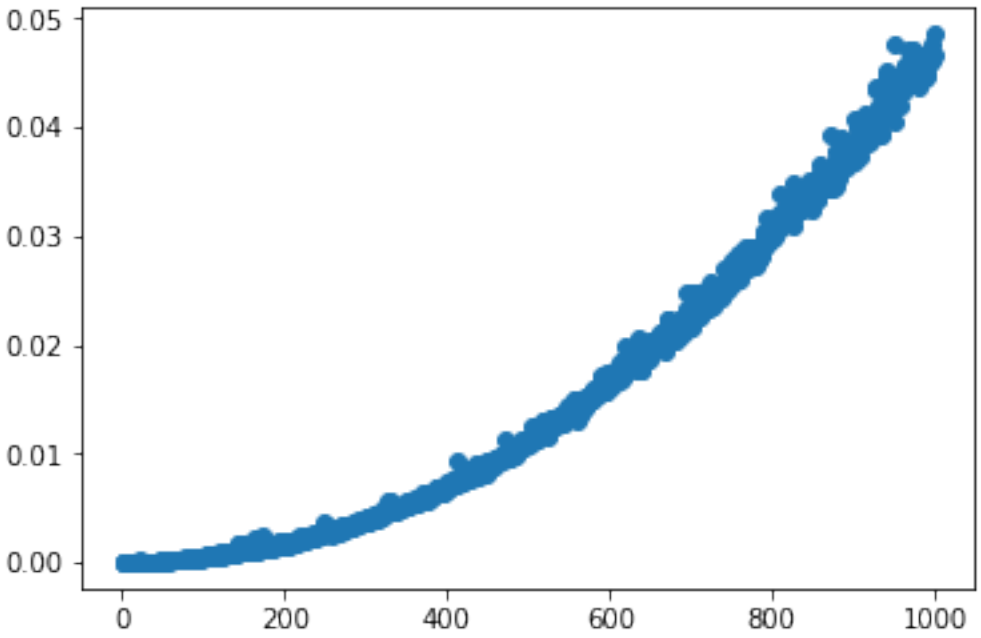


Encore et toujours cette mise en évidence d'une complexité linéaire !
Activité 5 :
Q2 et Q3

```
In [12]: def insert(T,a): #évaluation complexité
             N=len(T)     #1 affectation
             for i in range(1,N): #N-1 itérations
                    x=T[i]     #1 affectation
                    j=i-1     #1 autre
                    while (j>=0)and(T[j]>x):  #2 conditions de tests et au pire i-1
                        T[j+1]=T[j]    #1 affectation au pire ou 0 sinon
                        j=j-1       #1 affectation au pire ou 0 sinon
                    T[j+1]=x#1 affectation
             return T
```

On en déduit une complexité au pire égale à :
$\sum_{i=1}^{N-1}(2+\sum_{j=0}^{i-1}4)+1) = 2(N-1)+\sum_{i=1}^{N-1}4i+N-1 = 3N-3+2(N-1)N = 2N^2+N-3$,
et une complexité au mieux égale à :
$\sum_{i=1}^{N-1}(2+2+1) = 5(N-1)$
Q4

```
In [13]: echantillon_complexite(insert,0)
```



On vérifie donc expérimentalement une complexité quadratique car on reconnaît un arc de parabole, ce qui correspond à une expression de degré 2.

In [ ]: