La suite de Syracuse

1. Contexte pédagogique

1.1. Présentation de l'activité

Cette activité se positionne entre le milieu et la fin de l'année scolaire de première et se déroule sur une séance de 2h, idéalement, à effectif réduit.

Nous avons choisi comme support la suite de Syracuse qui, pour un nombre entier N > 0 est définie par récurrence, de la manière suivante :

$$u_0 = N$$

$$\forall\, n \in \mathbb{N} \qquad u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

La conjecture affirme que pour tout N, il existe un indice n tel que u_n = 1.

Exemple : Suite de Syracuse pour N = 15

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}	u_{19}	u_{20}	
15	46	23	70	35	106	53	**160**	80	40	**20**	**10**	5	16	8	4	2	**1**	4	2	**1**	...

1.2. Les pré-requis

Nous avons voulu créer une activité qui permettait de réinvestir tous ce qui avait été vu jusque là :

- Les bases de la programmation (création de listes / tableaux, boucles while et for, fonction (def), …) ;
- L'utilisation de la bibliothèque Matplotlib ;
- Les algorithmes de tri (par insertion, par sélection) et leur terminaison ;
- Opérations sur les tableaux : parcours séquentiel, lire les éléments,…

Cela permettra de faire émerger les difficultés encore présentes chez les élèves. On pourra ainsi en profiter pour réaliser une évaluation formative, avec remédiation derrière, ou une évaluation sommative en cours d'activité (programme python) ou à la fin de la séance sous forme de QCM.

1.3. Les objectifs

Les objectifs de cette activité sont les suivants :

- Étudier la terminaison d'un algorithme ;
- Ouverture sur la complexité d'un algorithme, étudié en terminale.

Références des programmes :

NSI Première

Contenus	Capacités attendues	Commentaires
Parcours séquentiel d'un tableau	**Écrire un algorithme de recherche d'un extrémum**, de calcul d'une moyenne.	

Recherche dichotomique dans un tableau trié	Montrer la **terminaison** de la recherche dichotomique à l'aide d'un variant de boucle.	

NSI Terminale

Contenus	Capacités attendues	Commentaires
Algorithmes sur les arbres binaires et sur les arbres binaires de	Calculer la taille et la hauteur d'un arbre	Une structure de données récursive adaptée est utilisée

2. Début de la séance

Le professeur vous donne un nombre et vous devez trouvez ceux qui suivent en respectant les consignes suivantes :

> Si le nombre est pair, on le divise par deux pour trouver le suivant ;
>
> Si le nombre est impair, on le multiplie par 3 et on ajoute 1 pour trouver le suivant.

À vous de jouer !!!

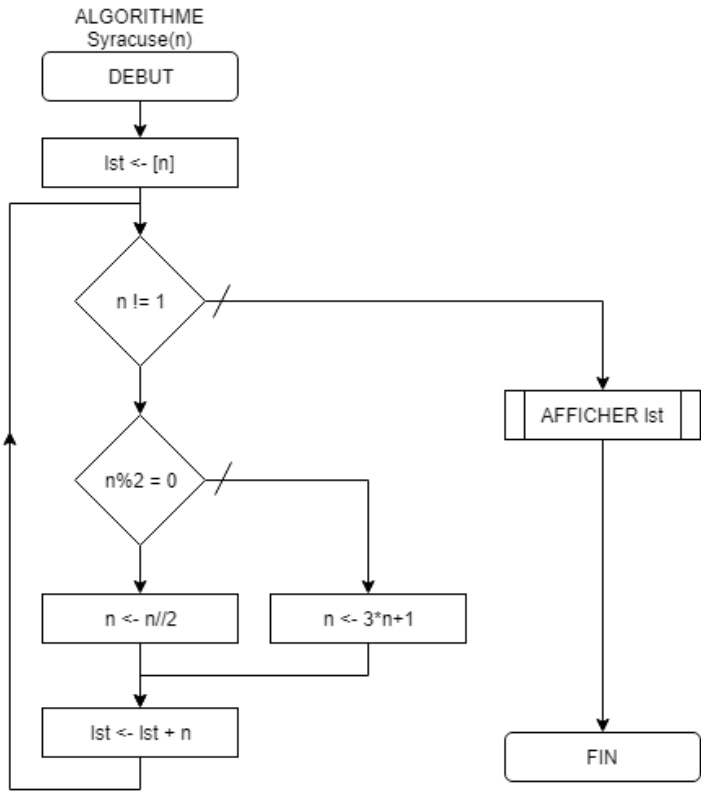
pour vérifier les résultats des élèves :

In [ ]:
```
syracuse(14)
```

3. Représentation de l'algorithme

- Représenter l'algoritme de Syracuse en pseudo-code ou sous forme d'algorigramme.
- L'algorithme doit retourner une liste avec l'ensemble des valeurs calculées par la suite de Syracuse.

In [ ]:
```
ALGORITHME Syracuse(n)
|    lst <- [n]
|    TANT QUE n != 1
|    |    SI n%2 = 0 ALORS
|    |    |    n <- n//2
|    |    SINON
|    |    |    n <- 3*n+1
|    |    FIN SI
|    |    lst <- lst + n
|    FIN TANT QUE
|    AFFICHER lst
FIN ALGORITHME
```



4. Programmation de l'algorithme en python

- Programmer en python l'algorithme de Syracuse.
- Tester les valeurs trouvées manuellement.
- Tester de plus grandes valeurs.

In [41]:
```
def syracuse(n):
    lst = [n]
    while n != 1:
        if n%2 == 0:
            n = n//2
        else :
            n = 3*n+1
        lst.append(n)
    return lst
```

In [ ]:
```
syracuse(32)
```

5. Représentation graphique de la suite de Syracuse

- Représenter graphiquement les valeurs d'une suite de syracuse pour n donné puis justifier le vocabulaire employé : temps de vol et altitude

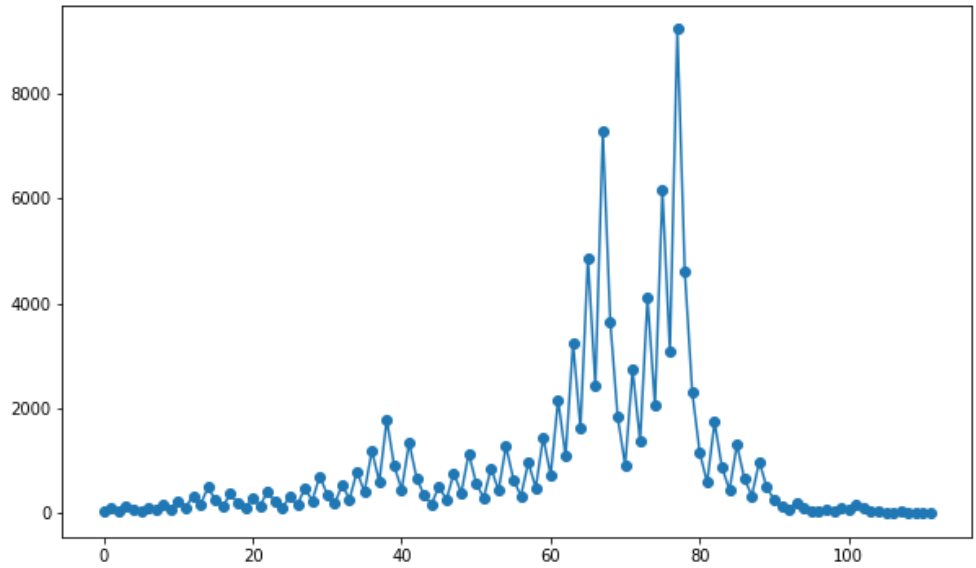
In [34]:
```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
%pylab inline
pylab.rcParams['figure.figsize'] = (10, 6)

def trace_syracuse(n):
    y = syracuse(n)
    x = range(len(y))

    plt.plot(x,y,'-',marker='o')
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [35]:
```
trace_syracuse(27)
plt.show()
```



6. Outils d'analyse

Temps de vol et altitude maximale

Le **"temps de vol"** de la suite de Syracuse du nombre n désigne le nombre de valeurs de cette suite avant de tomber sur la valeur 1 (on compte 1 et on ne compte pas n).

- Ecrire une fontion qui prend pour paramètre le nombre n et qui retourne le temps de vol.
- Testez votre fonction.

In [46]:
```
def temps_de_vol(n):
    cpt = 0
    while n != 1:
        if n%2 == 0:
            n = n//2
        else :
            n = 3*n+1
        cpt += 1
    return cpt
```

In [47]:
```
temps_de_vol(6)
```

Out[47]:
```
8
```

"L'altitude maximale" désigne la plus grande valeur atteinte.

- Ecrire une fonction qui prend pour paramètre le nombre n et qui retourne l'altitude maximale.

In [ ]:
```
def altitude_max(n):
    k =  max(syracuse(n))
    return k
```

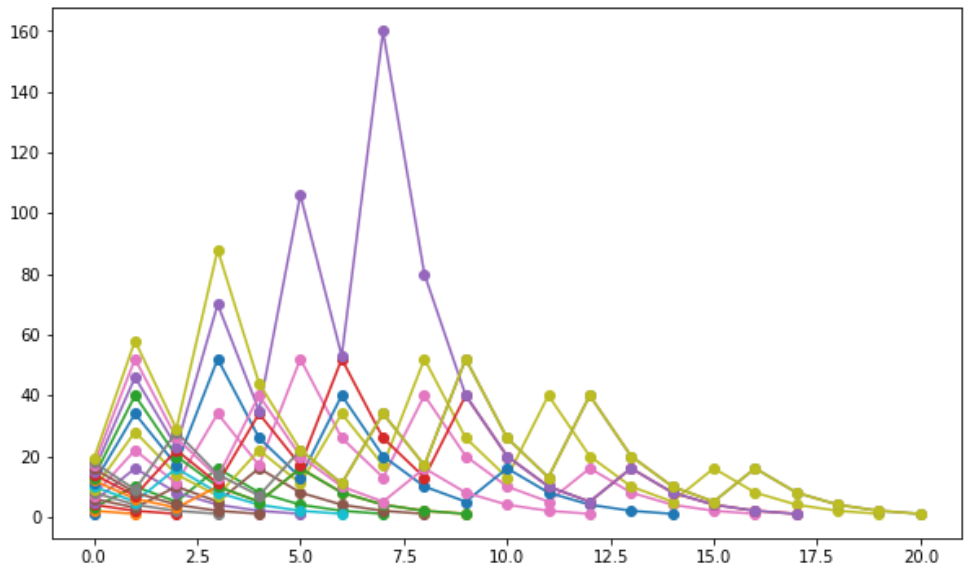
- Testez votre fonction

In [ ]:
```
altitude_max(6)
```

Représentation de n courbes.

- Représenter sur un même graphique les suites de Syracuse pour toutes les valeurs de n de 1 à 20

In [48]:
```
for i in range(1,20):
    trace_syracuse(i)
plt.show()
```



Terminaison de la suite de Syracuse

Jusqu'à présent, que pouvez-vous dire du comportement de toutes les suites de Syracuse que nous avons testées ?

Pouvons-nous généraliser cette remarque ?

Petit rappel

- Reconnaissez-vous l'algorithme ci-dessous ?
- Que peut-on dire sur sa terminaison ?

In [ ]:
```
def mystere(T):
    n = len(T)
    for i in range(1,n):
        j = i-1
        x = T[i]
        while j>=0 and x<T[j]:
            T[j+1] = T[j]
            j -= 1
        T[j+1] = x
```

lien (http://lwh.free.fr/pages/algo/tri/tri_insertion.html)

Retour à Syracuse

Êtes-vous capable de prouver sa terminaison?

In [7]:
```
def syracuse(n):
    lst = [n]
    while n != 1:
        if n%2 == 0:
            n = n//2
        else :
            n = 3*n+1
        lst.append(n)
    return lst
```

Jusqu'à présent, personne n'a réussi à démontrer que toute suite de Syracuse tombe à 1. On admet ceci, on conjecture et cette conjecture porte justement un nom: c'est la conjecture de Syracuse

In [49]:
```
from ipywidgets import*
def trace_syracuse(n):
    y = syracuse(n)
    x = range(len(y))
    plt.plot(x,y,'-',marker='o')
    print("n = ",n)
    plt.show()
interact_manual(trace_syracuse,n=(1,10**4,1))
```

Out[49]:
```
<function __main__.trace_syracuse(n)>
```

8. Ouvertures possibles

Contexte scientifique du problème

À l'heure actuelle, tous les entiers jusqu'à 82×2^{60} ont été testés.

In [10]:
```
87*2**60
```

Out[10]:
```
100304170900795686912
```

Mais il est possible qu'un contre-exemple soit caché après !

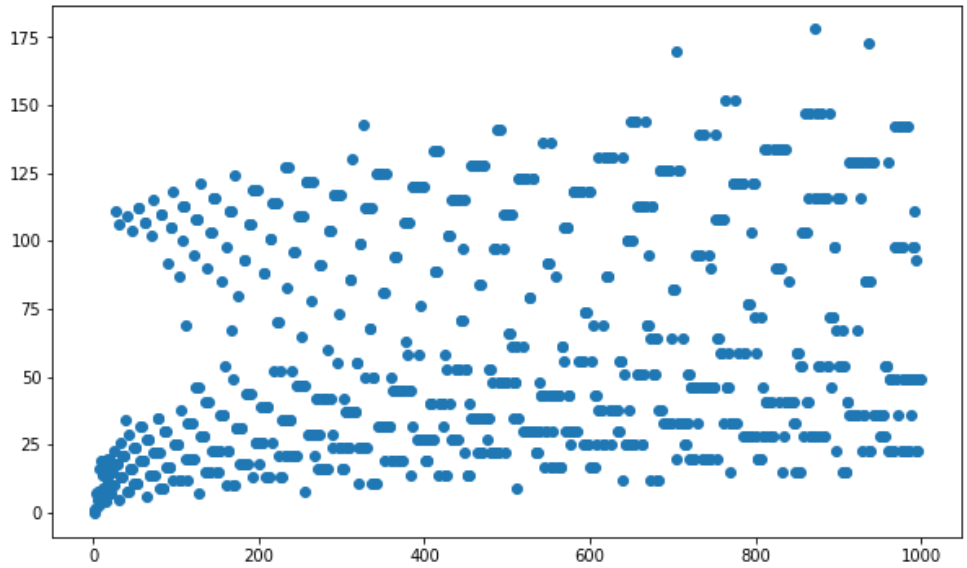
Vous pouvez participer à sa recherche : https://boinc.thesonntags.com/collatz/ (https://boinc.thesonntags.com/collatz/)

Il est aussi possible que cette propriété soit vraie mais indémontrable ...

Quelques curiosités

- Observons l'évolution du temps de vol en fonction de n.

In [13]:
```
n = 1000
x = range(1,n)
y = [temps_de_vol(k) for k in x]
plt.plot(x,y,'o')
plt.show()
```



- Dans quelle mesure, pour un nombre n donné, peut-on prévoir son temps de vol ?

Tentative de réponse par étude aléatoire sur un intervalle

principe :

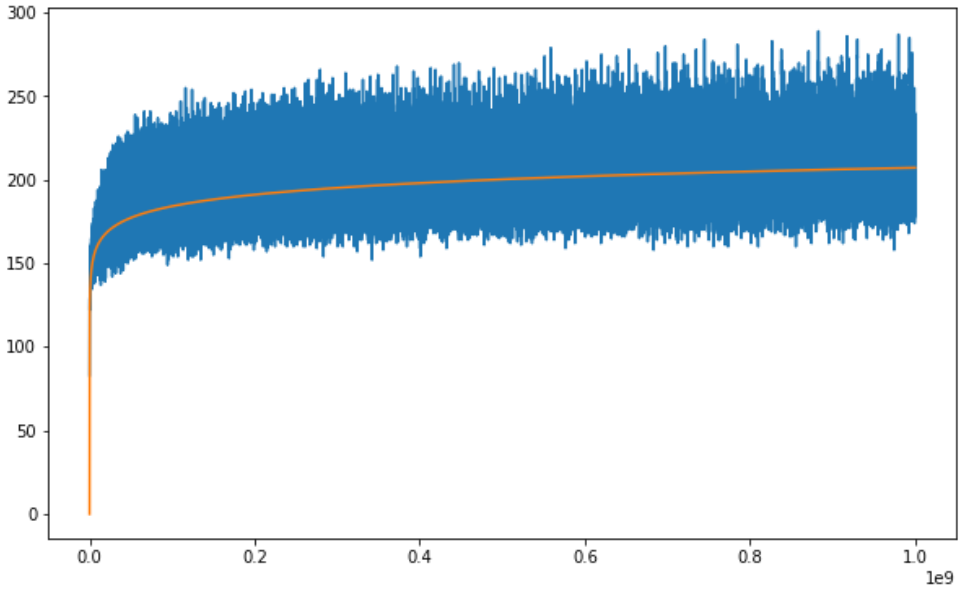
pour estimer le temps de vol d'une valeur n, on va observer comment se comportent *en moyenne* les valeurs voisines de n (choisies aléatoirement) sur un intervalle de taille fixée.

In [ ]:
```
def temps_moyen(nb_val, n_min, n_max):
    s = 0
    for k in range(nb_val):
        s += compte_syracuse(randint(n_min, n_max))
    return s / nb_val



interval = 10**5
val_max = 10**9
nb_point_interval = 200
x = range(1, val_max, interval)
y = [temps_moyen(nb_point_interval, n_min, n_min+interval) for n_min in
x]
z = [10*log(k) for k in x]

plt.plot(x,y,'o')
plt.plot(x,z)
plt.show()
```



Tentative de réponse par étude du temps moyen

Pour une valeur n donnée, on calcule la moyenne des temps de vol de toutes les valeurs inférieures ou égales à n.

In [14]:
```
def temps_de_vol_moyen_val_inferieures(n):
    s = 0
    for i in range(1,n+1):
        s += temps_de_vol(i)
    return s/n
```

In [15]:
```
temps_de_vol(17)
```

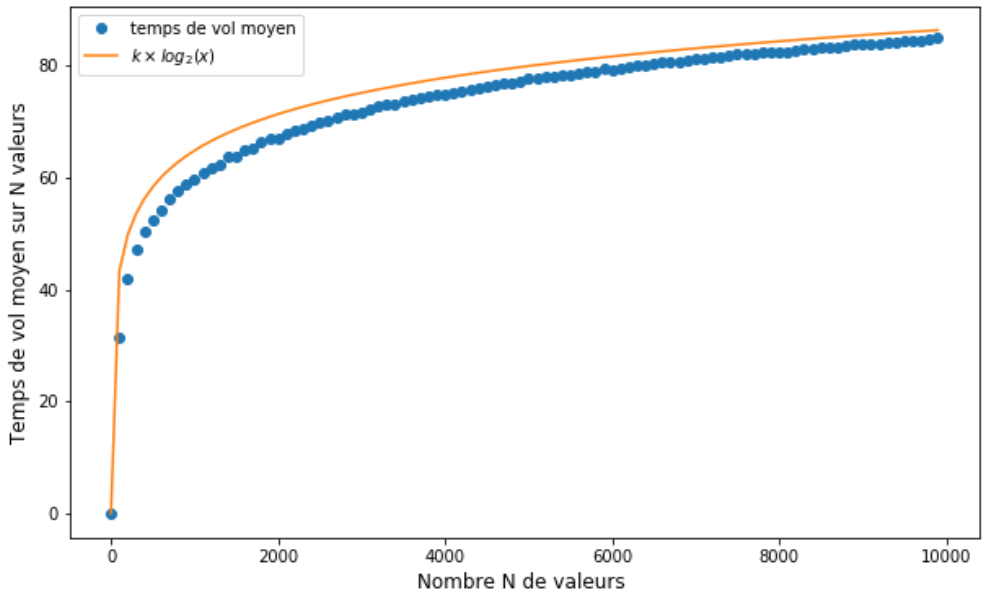
Out[15]: 12

Observation graphique

In [24]:
```
Nmax = 10**4
pas = Nmax // 100 #100 subdivisions
x = range(1,Nmax, pas)
y = [temps_de_vol_moyen_val_inferieures(i) for i in x]
plt.plot(x,y,'o',label='temps de vol moyen')

# approximation par un log
k = 6.5 #fit pour 10**7 : 6.677381507056237
z1 = [k*math.log2(i) for i in x]
plt.plot(x,z1,label=r'$k \times log_2 (x) $')

plt.xlabel('Nombre N de valeurs ', fontsize='large')
plt.ylabel('Temps de vol moyen sur N valeurs',fontsize='large')
plt.legend(loc='upper left')
plt.show()
```



In [ ]:

In [1]:

Impératif vs récursif

In [50]:
```
def syr_normal(n):
    while n != 1:
        if n % 2 == 0:
            n = n//2
        else :
            n = 3*n + 1
    return None


def syr_recursif(n):
    if n == 1 :
        return None
    else :
        if n % 2 == 0:
            syr_recursif(n//2)
        else :
            syr_recursif(3*n+1)
```

In [51]:
```
%timeit syr_normal(200)
```

```
7.71 µs ± 103 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [52]:
```
%timeit syr_recursif(200)
```

```
11.9 µs ± 54.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each
)
```

In [ ]:
```

```