TP info 10 - Corrigé

I1 - Palindrôme

Complexité : O(n)

où n est la taille de la chaine de caractères

```
def est_un_palindrome(s):
    """
    chaine: une chaine de caracteres, type: string
    renvoie True si chaine est un palindrome, False sinon
    """
    n=len(s)
    for i in range(n//2):
        if s[i]!=s[n-1-i]:
            return False
    return True#par defaut s est un palindrome
```

I2 - Evaluation d'un polynôme

```
Algorithme 1 : Evaluation naïve d'un polynôme
Entrées : liste P, réel x
Sorties : réel val
fonction Eval_pol(P,x)
    val=P[0]
    pour i de longueur(P)-1 faire
        val=val+P[i]× x**i
    val=val+P[0]
    retourner val
```

Coût (n) où n = longueur(P) = degré du polynôme

Addition, multiplication, affectation : O(1) coût unitaire.

Exponentiation : O(log(n))

Dans la boucle :

- Coût de l'itération : $1 + 1 + \log(i)$ ($= 2 + \log(i)$)
- coût total : $\sum_{i=1}^{n} (2+\log(i)) = 2n + \sum_{i=1}^{n} \log(i) = 2n + \log(n!)$

Or $2n = O(\log(n!))$

Finalement la complexité : $O(\log(n!))$
$\sim O(n \log(n))$

2.

Entrées : liste P, réel x
Sorties : réel val
fonction Horner(P,x)
    val=0
    /*On parcourt la liste à l'envers*/
    pour i de longueur(P)-1 à 1 faire
        val=(val+P[i])× x
    val=val+P[0]
    retourner val

Coût $C(n)$:

$C(n)$ ≈ coût de la boucle

$C(n) \approx 2n$

d'où la complexité : $O(n)$

3/ La complexité de l'algo de Horner est un peu meilleur que la complexité de l'algorithme naturel..

I 3- Tri par sélection

```
def rech_pos_min(l,debut,fin):
    m=l[debut]
    pos_min=debut
    i=debut
    while i<=fin:
        if l[i]<m:
            m=l[i]
            pos_min=i
        i=i+1
    return pos_min
```

1/ La fct° rech_pos_min renvoie l'indice du minimum de la portion du tableau comprise entre les indices début et fin

∀ entrée, on parcourt la portion [début, fin] du tableau. Posons k = fin - début, alors la complexité vaut : $O(k)$

2/ Appliquons tri sélection à [25, 3, 6, 92, 0, 14]

n=6

i	p	
0	4	[0, 3, 6, 92, 25, 14]
1	1	[0, 3, 6, 92, 25, 14]
2	2.	[0, 3, 6, 92, 25, 14]
3	5	[0, 3, 6, 14, 25, 92]
4	4	[0, 3, 6, 17, 25, 92]
5	5	[0, 3, 6, 17, 25, 92]

la fonction tri-sélection : trie le tableau dans l'ordre croissant.

4. Complexité

$C(n)$ = coût de la boucle.

$$= \sum_{i} c(i)$$

$c(i)$: coût de l'itération i

avec $c(i) = O(n-i)$

$$C(n) \sim \sum_{i=0}^{n-1} (n-i) = n^2 - \underbrace{\sum_{i=0}^{n-1} i}_{\frac{(n-1)n}{2}}$$

$$\sim \frac{n^2}{2} + \frac{n}{2}$$

D'où la complexité $O(n^2)$.

3/ Correction :

Invariant de boucle : $\mathcal{I}(i)$: le sous-tableau $t[0:i]$ est trié.

Initialisation :

$i = 0$, $t[0:0]$: tableau vide. Par convention il est trié

Hérédité : P_i : $t[0:i]$ est trié $\Rightarrow P_{i+1}$: $t[0:i+1]$ est trié

$$t[0:i+1] = t[0:i] + \underbrace{t[i:i+1]}_{\text{1 seul élément } t[i]}$$

Par hypothèse de récurrence $t[0:i]$ trié, il faut montrer $t[i]$ min de $t[i:n]$. C'est assuré par la correction de la fct° recherche_min

donc $(P_{i+1}) \Rightarrow (P_i)$

D'où P_i : $t[0,i]$ trié à la $i^{ème}$ itération est un invariant de boucle.

(3) A la sortie de la boucle $i = n$ donc : le tableau $t[0:n]$ est trié i.e. l'intégralité du tableau est trié.

I 4 - Etude d'une suite convergente

Convergence $\sum_{n=1}^{\infty} \frac{(-1)^n}{n}$?

1/
```
def calcul_serie(u,N):
    """
    u: fonction definissant les terme de la suite
    N: rang du dernier terme calcule
    serie : terme de la serie definie par u
    """
    serie=[]
    s=0
    for i in range(N+1):
        s=s+u(i)
        serie.append(s)
    return serie
```

i	s	serie
x	0	[ ]
0	u(0)	[u(0)]
1	u(0)+u(1)	[u(0), u(0)+u(1)]
⋮	⋮	⋮

Complexité :
- unité de mesure : opérateur arithmétique
- u(i) coût constant a priori
- addition : coût constant.

$\Rightarrow$ complexité $O(N)$ linéaire

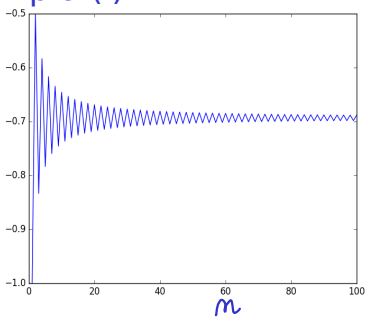
2/ On appelle la fonction calcul_serie en passant la fonction définissant la suite u ou sa valeur absolue $v = |u|$.

```
def f(n):
    return (-1)**n/n

def g(n):
    return fabs(f(n))
```

```
l=calcul_serie(f,100)
plot(l)
```

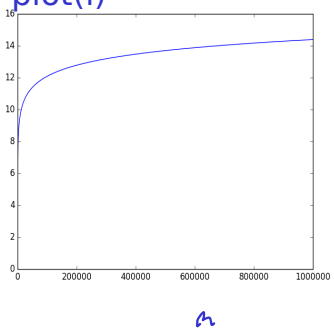
$S_n = \sum_{k=1}^{n} \frac{(-1)^k}{k}$



Conjecture : S_n est convergente

$\sum_{k=1}^{n} \frac{1}{k}$

```
l=calcul_serie(g,1000000)
plot(l)
```



Conjecture : S_n n'est pas absolument convergente.