TP 6
Algorithmes Dichotomiques – Sujet

Recherche dichotomique dans une liste triée

Question 1 Illustrer la méthode avec les deux exemples suivants :

- `x0 = 5` et `L` =

−3	5	7	10	11	14	17	21	30

- `x0 = 11` et `L` =

−2	1	2	7	8	10	13	16	17

.

Correction

```
# g=0, d=8
# m=4, L[m]=11 et 5 < 11, on pose g=0, d=3
# m=2, L[m]=5. On a trouvé x0

# g=0, d=8
# m=4, L[m]=8 et 8<11, on pose g=5, d=8
# m=6, L[m]=13 et 11<13, on pose g=5, d=5
# m=5, L[m]=10 et 10<11, on pose g=6, d=5. On s'arrête.
```

Question 2 Si `x0` n'est pas dans la liste `L`, donner un test d'arrêt du processus de dichotomie portant sur `g` et `d`.

Correction

```
g>d
```

Question 3 Écrire une fonction `dichotomie(x0,L)` qui renvoie `True` ou `False` selon que `x0` figure ou non dans `L` par cette méthode. On utilisera une boucle `while` que l'on interrompra soit lorsque l'on a trouvé `x0`, soit lorsque l'on a fini de parcourir la liste.

Correction

```python
def dichotomie(x0,L):
    Test=False
    n=len(L)
    g,d=0,n-1
    while g<=d and not Test:
        m=(g+d)//2
        if L[m]==x0:
            Test=True
        elif L[m]>x0:
            d=m-1
        else:
            g=m+1
        print(g,d)
    return(Test)
```

Question 4 Écrire une fonction `dichotomie_indice(x0,L)` qui renvoie l'indice de `x0` s'il existe, -1sinon.

Correction

Question 5 Tester votre fonction sur les exemples de listes ci-dessus.

Correction

Question 6 (Facultatif) Combien vaut `g-d` au i^e tour de boucle ? Si `x0` ne figure pas dans `L`, montrer que le nombre de tours de boucles nécessaires pour sortir de la fonction est de l'ordre de $\ln n$ où `n=len(L)` (cela rend la fonction beaucoup plus efficace qu'une simple recherche séquentielle pour laquelle le nombre de comparaisons pour sortir de la boucle serait de l'ordre de n).

Correction

```
# Si x0 n_est pas présent, on exécute la boucle tant que g<=d. On sort avec
     g=d+1.
# A l_entrée du 1er tout de boucle, on a d-g+1=n. A chaque tour, la valeur
    d-g+1 diminue environ de moitié. Donc après k tours de boucles, la
    longueur de l_intervalle est de l_ordre de n/2**k.
# De plus, à chaque tour de boucle, il y a 2 comparaisons.
# Au dernier tour numéro k, on a g=d soit lorsque n/2**k = 1 d_ou k=log_2(n
    ).
# On obtient donc un nombre de comparaisons équivalent à 2*ln(n)/ln(2):
    complexité logarithmique.
# Dans le cas séquentiel, on obtient une complexité linéaire, donc beaucoup
     moins intéressant.
```

Recherche d'un zéro d'une fonction

Question 7 Si l'on souhaite que g_n et d_n soient des solutions approchées de ℓ à une précision ε, quelle est la condition d'arrêt de l'algorithme ? Préciser alors la valeur approchée de ℓ qui sera renvoyée par la fonction.

Correction

```
# Pour une valeur à epsilon près, on s_arrete lorsque 0<d-g<2*epsilon et on
     renvoie (g+d)/2
```

Question 8 Écrire une fonction `recherche_zero(f,a,b,epsilon)` qui renvoie une valeur approchée du zéro de `f` sur `[a,b]` a epsilon près.

Correction

```python
def recherche_zero(f,a,b,epsilon):
    g,d=a,b
    while d-g>2*epsilon:
        m=(g+d)/2
        if f(m)*f(g)<=0:
            d=m
        else:
            g=m
```

```
    return((g+d)/2)
```

Question 9 Tester la fonction avec $f : x \mapsto x^2 - 2$ sur $[0, 2]$ et $\varepsilon = 0,001$.

Correction

```
def f(x):
    return(x**2-2)
```

```
# Avec epsilon = 1/2**p, il faut compter combien il y a de tours de boucles
    . En sortie du kieme tour de boucle, d-g vaut (b-a)/2**k. Il y a donc k
     tours de boucles avec (b-a)/2**k<=1/2**(p-1) soit k>=p-1+log_2(b-a)
    soit une complexité logarithmique encore.
```

Valeur d'un polynôme par plusieurs méthodes

Question 10 Écrire une fonction `exponaif(x,n)` d'arguments un réel x et un entier naturel n, qui renvoie la valeur de x^n par la méthode naïve $x^n = x \times x \times ... \times x$ (n termes). Compter le nombre d'opérations dans exponaïf.

Correction

```
def exponaif(x,n):
    p=1
    for i in range(n):
        p=p*x
    return(p)
```

Question 11 Quel est le nom de la variable locale dont le contenu est retourné par la fonction ?

Correction

La variable locale est `res`.

Question 12 Faire tourner « à la main » la fonction pour $x = 2$ et $n = 10$ en complétant le tableau suivant puis encadrer le nombre d'opérations dans exporapide en fonction de $\ln(n)/\ln(2)$.

Correction

	p	res	y
sortie du 1er tour de boucle	5	1	4
sortie du 2er tour de boucle	2	4	16
sortie du 3er tour de boucle	1	4	256
sortie du 4er tour de boucle	0	1024	65536

```
# Le nombre d'opérations effectuées est exactement n (1 produit à chaque
    tour)
```

On considère un polynôme $P(x) = \sum_{k=0}^{n} a_k.x^k$ que l'on modélisera en Python par la liste $P = [a_0, a_1, ..., a_n]$. Dans la suite, on prendra pour tout $k \in \mathbb{N}$, $a_k = k$.

Question 13 Ecrire une fonction `Pnaif(x,P)` d'arguments un réel `x` et `P` la liste des coefficients du polynôme, qui renvoie $P(x)$ à l'aide de la fonction `exponaif`. Compter le nombre d'opérations.

Correction

```python

def Pnaif(x,n):
    S=0
    for i in range(n):
        S=S+i*exponaif(x,i)
    return(S)

# n+n(n+1)/2 ~ n**2/2 opérations. Quadratique
```

Question 14 Faire de même pour une fonction `Prapide(x,n)` qui renvoie $P(x)$ à l'aide de la fonction `exporapide`. On admettra que la complexité est en O(n ln(n)).

Correction

```python
def Prapide(x,n):
    S=0
    for i in range(n):
        S=S+i*exporapide(x,i)
    return(S)

# O(log(i)) pour chaque i*x**i. Il reste la somme des n termes.
# D'où n+somme des log(i)=O(n.ln(n)).
```

Question 15 Écrire une fonction `horner(x,L)` de paramètres un réel x et une liste L représentant un polynôme P, renvoie la valeur de $P(x)$ par la méthode de Hörner. Compter le nombre d'opérations.

Correction

```python
def horner(x,L):
    """L est la liste des coefficients"""
    n=len(L)-1
    S=0
    for i in range(n+1):
        S=S*x+L[n-i]
    return(S)
# 2n opérations. Linéaire mais plus intéressante que ci-dessus.
```

Question 16 Définir la liste N des entiers naturels compris entre 0 et 100.

Correction

```python
N=[i for i in range(101)]
```

Question 17 Grâce à la fonction `perf_counter` de la bibliothèque `time`, écrire une fonction `Temps_calcul(x)` qui :

- définit 3 listes `Tn`, `Tr` et `Th` contenant les temps de calcul de $P(x)$ pour $P =$

$\sum_{k=0}^{n} k.x^k$ lorsque n décrit N avec respectivement la méthode naïve, la méthode rapide puis la méthode de Hörner.

- trace les trois courbes `Tn`, `Tr` et `Th` en fonction de N (on prendra $x = 2$). Interpréter le résultat (on pourrait démontrer que les temps d'exécution des trois programmes sont de l'ordre de $n**2$ pour la méthode naïve (on parle de complexité quadratique), de l'ordre de $n\ln(n)$ pour l'exporapide, et de l'ordre de n pour la méthode de Hörner (complexité linéaire)).

Correction

```
import time as t

def Temps_calcul_P(x):
    # Le polynôme est donné par une liste des coefficients.
    Tn,Tr,Th=[],[],[]
    for n in N:
        L=[k for k in range(n+1)]
        tps=t.perf_counter()
        Pnaif(x,n)
        Tn.append(t.perf_counter()-tps)
        tps=t.perf_counter()
        Sr=0
        Prapide(x,n)
        Tr.append(t.perf_counter()-tps)
        tps=t.perf_counter()
        Phorner(x,L)
        Th.append(t.perf_counter()-tps)
    plt.plot(N,Th,label='méthode horner')
    plt.plot(N,Tr,label='méthode rapide')
    plt.plot(N,Tn,label='méthode naÃ¯ve')
    plt.legend()
    plt.show()
```