Corrigé du TP n°6 - Algorithmes dichotomiques - Corrigé

1. Recherche dichotomique

Programmons l'algorithme de recherche dichotomique.
Dans la suite, E est un ensemble totalement ordonné, `L` est un tableau **trié** d'éléments de E et `x` est un élément de E.

Q1 : *(Rappel)* Écrire une fonction `recherche(L,x)` qui opère une recherche de complexité linéaire de `x` dans `L`.

Q2 : *Version récursive de la recherche dichotomique :*
En vous basant sur la description du cours, écrire une fonction `dicho_rec(L,x)` récursive de la recherche dichotomique de `x` dans `L`.

Q3 : *Estimation empirique de la complexité :*

a) Importer le module `random` et écrire une fonction `liste_triee(n)` qui renvoie une liste triée de nombres entiers tirés aléatoirement et uniformément entre 0 et 100. *En attendant l'étude des algorithme de tri, on pourra se servir de la méthode `L.sort()` pour trier la liste `L`*

b) Modifier les fonctions `recherche(L,x)` et `dicho_rec(L,x)` de telle sorte qu'elle renvoie un compteur `c` qui compte le nombre de comparaisons effectuées dans chacune des fonctions. On appellera `recherche_complexite(L,x)` et `dicho_rec_complexite(L,x)` ces nouvelles fonctions.

c) Comparer les résultats de `recherche_complexite(liste_triee(n),101)` et `dicho_rec_complexite(liste_triee(n),101)` pour `n`$= 2^k$ où $k = 5, 10, 15, 20, 100, \ldots$. Qu'en pensez-vous ?
Proposer un nom de complexité pour l'algorithme de recherche dichotomique.

Q4 : Proposer une version itérative de la recherche dichotomique.

Correction.

Q1 :

```python
def recherche(L,x):
    for e in L:
        if x == e:
            return True
    return False
```

Q2 :

```python
### 1ere version avec slicing, peu économique en mémoire
def dicho_rec(L,x):
    if len(L)==0:
        return False
    else:
        m = len(L)//2
        if x == L[m]:
            return True
        elif x < L[m]:
            return dicho_rec(L[:m],x)
        else:
            return dicho_rec(L[m+1:],x)

#2eme version, plus économe en mémoire
def dicho_rec_ind(L,x,deb,fin):
    if deb>=fin:
        return False
    else:
        m = (deb+fin)//2
        if x == L[m]:
            return True
        elif x < L[m]:
            return dicho_rec_ind(L,x,deb,m)
        else:
            return dicho_rec_ind(L,x,m+1,fin)

def dicho_rec2(L,x):
    return dicho_rec_ind(L,x,0,len(L))
```

Q3 : *Estimation empirique de la complexité :*

a)

```python
import random as rd
def listetriee(n):
    L=[rd.randint(0,100) for i in range(n)]
    L.sort()
    return L
```

b)

```python
def recherche_compteur(L,x):
    c=0
    for e in L:
        c+=1
        if x == e:
            return c
    return c

def dicho_rec_c(L,x,c):
    if len(L)==0:
        return c
    else:
        m = len(L)//2
        c+=1
        if x == L[m]:
            return c
        elif x < L[m]:
            return dicho_rec(L[:m],x,c)
        else:
            return dicho_rec(L[m+1:],x,c)

def dicho_rec_compteur(L,x,c):
    return dicho_rec_c(L,x,0)
```

Q4 :

```python
def dicho_it(L,x):
    deb = 0
    fin = len(L)
    while deb<fin:
        m = (deb+fin)//2
        if x == L[m]:
            return True
        elif x < L[m]:
            fin = m
        else:
            deb = m+1
    return False
```

2. Exponentiation rapide

Programmons l'algorithme d'exponentiation rapide.
Dans la suite, `x` est un nombre et `n` est un entier positif. On n'aura bien-sûr jamais recours à l'opérateur ** dans les questions qui suivent !

Q1 : *(Échauffement)* Écrire une fonction `puissance(x,n)` qui renvoie la valeur de `x` à la puissance `n` de complexité linéaire.

Q2 : *Version récursive de l'exponentiation rapide :*
En vous basant sur la description du cours, écrire une fonction `expo_rec(x,n)` récursive de l'exponentiation rapide.

Q3 : *Estimation empirique de la complexité :*

a) Modifier les fonctions `puissance(x,n)` et `expo_rec(x,n)` de telle sorte qu'elle renvoie un compteur `c` qui compte le nombre de multiplication de chaque fonction. On appellera `puissance_complexite(x,n)` et `expo_rec_complexite(x,n)` ces nouvelles fonctions.

b) Comparer les résultats de `puissance_complexite(2,n)` et `expo_rec_complexite(2,n)` pour `n` $= 2^k$ où $k = 5, 10, 15, 20, 100, \ldots$. Que pensez-vous de la complexité de l'exponentiation rapide ?

Q4 : Proposer une version itérative de l'exponentiation rapide.

Correction.

Q1 :

```
def puissance(x,n):
    p=1
    for i in range(1,n+1):
        p = p*x #ou p*=x
    return p
```

Q2 :

```
def expo_rec(x,n):
    if n == 0:
        return 1
    else:
        p = expo_rec(x,n//2)
        if n%2 == 0:
            return p*p
        else:
            return p*p*x
```

Q3 : *Estimation empirique de la complexité :*

a)

```python
def puissance(x,n):
    c=0
    p=1
    for i in range(1,n+1):
        c+=1
        p = p*x
    return c

def expo_rec_c(x,n):
    if n == 0:
        return 0
    else:
        p = expo_rec(x,n//2)
        if n%2 == 0:
            return p+1
        else:
            return p+1
```

Q4 :

```python
def expo_it(x,n):
    N = n
    X = x
    p=1
    while N > 0:
        if N%2 == 1:
            p = p*X
        X = X*X
        N = N//2
    return p
```