

## Corrigé du TP Informatique 23

### Exercice 1

On saisit :

```
def expo(x,n):
    if n==0:
        return x**0          # cas de base avec exposant nul
    else:
        r=expo(x,n//2)**2    # propagation avec le quotient de n par 2
        if n%2==0:           # distinction selon la parité
            return r
        else:
            return x*r
```

### Exercice 2

1. Soit  $n \geq 1$  et  $k \in \llbracket 1; n \rrbracket$ . On a :

$$\forall n \geq k \geq 1 \quad \binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n}{k} \times \frac{(n-1)!}{(k-1)!((n-1)-(k-1))!}$$

Ainsi

$$\boxed{\forall n \geq k \geq 1 \quad \binom{n}{k} = \binom{n-1}{k-1} \frac{n}{k}}$$

2. On saisit :

```
def binom(n,k):
    if k>n:
        return 0
    elif k==0:
        return 1
    else:
        return n*binom(n-1,k-1)//k
```

On a l'égalité  $k \binom{n}{k} = n \binom{n-1}{k-1}$  qui prouve que  $k$  divise  $n \binom{n-1}{k-1}$ . On peut donc effectuer, après la multiplication par  $n$ , le quotient par  $k$  pour avoir un résultat entier (et non flottant, avec une division simple).

3. Les récursions s'arrêtent lorsque  $k = 0$ . C'est donc vis-à-vis de la variable  $k$  que se fait l'examen des complexités. Lors de chaque récursion, on effectue l'empilement de l'appel récursif, la transmission des arguments et un nombre constant d'opérations (une multiplication, un quotient).

On a donc  $S(k) = S(k-1) + O(1)$  et  $T(k) = T(k-1) + O(1)$

d'où 
$$S(k) = S(0) + \sum_{i=1}^k [S(i) - S(i-1)] = O(k)$$

et de même pour la complexité temporelle. On conclut

La fonction `binom(n,k)` admet une complexité temporelle et spatiale en  $O(k)$ .

**Remarque :** On fait l'hypothèse simplificatrice que multiplication et quotient sont à coût constant.

### Exercice 3

1. On saisit :

```
def rech_dicho(elt,L,deb,fin):
    """rech_dicho(elt:int,L:list)->(bool,int)
    Renvoie le résultat de la recherche dichotomique
    de elt dans L[deb:fin+1] liste triée :
    * si L[k]==elt      -> (True,k)
    * si elt absent de L[deb:fin+1] -> (False,k)"""
    milieu=(fin+deb)//2
    if fin-deb<0:
        return False,deb
    else:
        if L[milieu]==elt:
            return True,milieu
        if elt<L[milieu]:
            return rech_dicho(elt,L,deb,milieu-1)
        else:
            return rech_dicho(elt,L,milieu+1,fin)
```

2. On saisit :

```
def rech(elt,L):
    """rech_dicho(elt:int,L:list)->(bool,int)
    Renvoie le résultat de la recherche dichotomique
    de elt dans L liste triée :
    * si L[k]==elt      -> (True,k)
    * si elt absent de L -> (False,k)"""
    return rech_dicho(elt,L,0,len(L)-1)
```

3. On saisit :

```
def est_triee(L):
    """est_triee(L:list)->bool
    Renvoie True si L est triée, False sinon"""
    n=len(L)
    for k in range(n-1):
        if L[k]>L[k+1]:
            return False
    return True

def rech(elt,L):
    assert est_triee(L)
    return rech_dicho(elt,L,0,len(L)-1)
```

## Exercice 4

1. On saisit :

```
def dichot(f,a,b,eps):
    c=(a+b)/2
    if b-a<eps:
        return c
    else:
        if f(a)*f(c)<=0:
            return dichot(f,a,c,eps)
        else:
            return dichot(f,c,b,eps)
```

2. On expérimente :

```
>>> dichot(lambda t:t**2-2,0,2,1e-5)
1.4142112731933594
>>> dichot(np.sin,2,4,1e-5)
3.141590118408203
```

3. On saisit :

```
def rech_dichot(f,a,b,eps):
    assert f(a)*f(b)<=0
    return dichot(f,a,b,eps)
```

On expérimente :

```
>>> rech_dichot(lambda t:t**2-2,0,2,1e-5)
1.4142112731933594
>>> rech_dichot(lambda t:t**2-2,0,1,1e-5)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    rech_dichot(lambda t:t**2-2,0,1,1e-5)
  File "D:\GIT CPGE\info_MPSI_PCSI\listes_exo\REC\EX017.py", line 29, in rech_dichot
    assert f(a)*f(b)<=0
AssertionError
```

## Exercice 5

On saisit :

```
def pgcd(a,b):
    if b==0:
        return a
    else:
        return pgcd(b,a%b)
```

## Exercice 6

On saisit :

```
def val(n,p):
    if n%p!=0:
        return 0
    else:
        return val(n//p,p)+1
```

Dans le meilleur des cas, si  $p \nmid n$ , la complexité est en  $O(1)$  puisqu'il y a un seul appel de `val`. Le pire de cas correspond à la situation où  $n$  est une puissance de  $p$ , *i.e.*  $n = p^\alpha$  avec  $\alpha$  entier non nul. La puissance décroît de 1 à chaque récursion et on a donc  $\alpha = \log_p(n)$  récursions d'où une complexité dans le pire des cas en  $O(\log n)$ . Ainsi

La fonction `val` admet une complexité en  $O(1)$  dans le meilleur des cas et en  $O(\log n)$  dans le pire des cas.

## Exercice 7

1. On saisit :

```
def phi(n):
    res=0
    for k in range(1,n+1):
        if pgcd(k,n)==1:
            res+=1
    return res
```

2. On a  $\forall n \in \mathbb{N}^* \quad \varphi(n) = n - \sum_{d|n, d < n} \varphi(d)$

On saisit :

```
def phi_rec(n):
    if n==1:
        return 1
    else:
        aux=0
        for d in range(1,n):
            if n%d==0:
                aux+=phi_rec(d)
        return n-aux
```