LU2IN003 Complexité d'un algorithme 4

Exercice 1 – Quelques calculs de complexité d'algorithmes itératifs

Question 1 – Somme des éléments d'un tableau

La fonction `somTab` retourne la somme des éléments d'un tableau T de nombres :

```
def somTab(T):
    res = 0
    for x in T:
        res = res + x
    return res
```

Exprimer sa complexité en nombre d'additions en fonction de la taille n de T.

Question 2 – Recherche de l'élément minimum d'un tableau non trié

Soit tab un tableau de n éléments et les entiers d et f tels que $0 \leq d \leq f \leq n-1$. La fonction `RechercheMin` retourne l'indice de l'élément minimum de tab entre les indices d et f (voir cours 2) :

```
def RechercheMin(tab, d, f):
    imin=d; i=d+1
    while i<=f:
        if tab[i]<tab[imin]:
            imin=i
        i=i+1
    return imin
```

Evaluez la complexité de la fonction `RechercheMin` en fonction de ses paramètres.

Question 3 – Complexité du tri par sélection itératif

On considère maintenant le tri par sélection dont le code suit :

```
def TriParSelection(tab):
    i=0; n=len(tab)
    while (i!= n):
        k = RechercheMin(tab, i, n-1)
        if (k!=i):
            z = tab[i]
            tab[i]=tab[k]
            tab[k]=z
        i=i+1
```

Quelle est la complexité de la fonction `TriParSelection` ? Justifiez votre réponse.

Exercice 2 – Quelques calculs de complexité d'algorithmes récursifs

Question 1 – Factorielle

La fonction `fact` retourne la factorielle de n :

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n -1)
```

Exprimer sa complexité en nombre de multiplications en fonction de n.

Question 2 – Recherche récursive du minimum dans un tableau non trié

Soit tab un tableau de n éléments et les entiers d et f tels que $0 \leq d \leq f \leq n-1$. La fonction `RechercheMinRec` retourne la première position du plus petit élément de T compris entre les indices d et f (inclus).

```
def RechercheMinRec(T,d,f):
    if (d==f):
        return d
    else :
        imin = RechercheMinRec(T, d, f-1)
        if (T[imin] > T[f]):
            return f
        return imin
```

Quelle est la complexité de la fonction `RechercheMinRec` ? Pour cela, on pourra compter le nombre de comparaisons effectuées.

Question 3 – Recherche récursive d'un élément dans un tableau non trié

La fonction `RechercheRec` retourne l'indice k maximum dans $\{0, \ldots, n-1\}$ tel que $tab[k] = elem$ (Voir cours 3).

```
def RechercheRec(elem, tab, n) :
   if n==0 :
      return -1
   if tab[n-1]==elem :
      return n-1
   return RechercheRec(elem, tab, n-1)
```

Calculez la complexité de `RechercheRec` dans le meilleur et le pire des cas.

Exercice 3 – Tri par sélection récursif

Soit tab un tableau de n éléments et les entiers d et f tels que $0 \leq d \leq f \leq n-1$. On considère l'algorithme de tri par sélection suivant :

```
def triSelectionRec(T, d,f):
    if d<f:
        print ('A␣l␣ppel␣T[', d, ',' , f, ']=', T[d:f+1])
        imin = RechercheMinRec(T, d, f)
        tmp = T[d] ; T[d] = T[imin] ;  T[imin] = tmp
        triSelectionRec(T, d+1,f)
        print ('En␣sortie␣T[', d, ',' , f, ']=', T[d:f+1])
```

On supposera que `RechercheMinRec(T, d, f)` se termine et retourne pour les entiers d et f tels que $0 \leq d \leq f \leq n-1$, un indice minimal $i \in \{d, \ldots, f\}$ tel que $T[i]$ est le plus petit élément du sous tableau de T de d à f.

Question 1

Exécutez `triSelectionRec(T, 0,6)` pour le tableau `tab=[3,1,8,5,1,4,3]`. Vous ne donnerez que les affichages ! !

Question 2
Démontrez la terminaison et la validité de la fonction `triSelectionRec`. Pour cela, on supposera que `RechercheMinRec(T, d, f)` se termine et retourne pour les entiers d et f tels que $0 \leq d \leq f \leq n-1$, un indice minimal $i \in \{d, \ldots, f\}$ tel que $T[i]$ est le plus petit élément du sous tableau de T de d à f.

Question 3
Evaluez le nombre de comparaisons effectuées par la fonction `RechercheMinRec`. En déduire la complexié de `RechercheMinRec`

Exercice 4 – Recherche d'un élément dans un tableau trié de nombres

Dans cet exercice la complexité est comptée en nombre de comparaisons ($=$, $<$, $>$) entre l'élément cherché et les éléments du tableau.

Recherche séquentielle

Question 1
Quelle est la complexité d'une recherche séquentielle dans un tableau non trié ? dans un tableau trié ?

Recherche dichotomique

On suppose maintenant que T est un tableau de n nombres trié en ordre croissant. La fonction récursive `RechercheDicho (elem,T,d,f)` retourne une position de $elem$ dans $T[d \ldots f]$ si elle existe, -1 sinon.

```
def RechercheDicho (elem, T, d, f):
    print ('A␣l␣ppel␣T[', d, ',' , f, ']=', T[d:f+1], 'et␣elem=',elem)
    if (d<=f):
        i = (d + f) // 2
        if (T[i] == elem):
            res=i
        elif (T[i] > elem):
            res= RechercheDicho (elem, T, d, i - 1)
        else:
            res=RechercheDicho (elem, T, i + 1, f)
    else:
        res=-1
    print ('En␣sortie␣res=',res)
    return res
```

Question 2
Exécuter les appels `RechercheDicho(40,T,0,6)` et `RechercheDicho(12,T,0,6)` pour le tableau d'entiers triés `T=[1,6,12,17,34,45,65]`.

Question 3
Prouver la terminaison et la validité de la fonction.

Question 4
Déterminer la complexité de la fonction `RechercheDicho` dans le pire cas.

Exercice 5 – Somme des produits

Pour $n \in \mathbb{N}$, on veut calculer la somme $SomProd(n)$ de tous les produits $i * j$ pour $1 \leq j \leq i \leq n$. Autrement dit :

$$SomProd(n) = \sum_{1 \leq j \leq i \leq n} i * j$$

Cette somme vaut 0 si $n = 0$.

Question 1 – Somme des produits, en itératif

La fonction `somProdIte(n)` retourne la somme de tous les produits $i * j$ pour $1 \leq j \leq i \leq n$:

```
def somProdIte(n):
    res = 0
    for i in range(1, n + 1):
        for j in range(1, i + 1):
                  res = res + i * j
    return res
```

Calculer sa complexité en nombre d'opérations arithmétiques ($+$, $*$).

On dispose d'une fonction `somRec` ainsi définie :

```
def somRec(n):
    if n == 0:
        return 0
    else:
        return n + somRec(n - 1)
```

Cette fonction cacule la somme des entiers de 1 à n, en effectuant n additions (*cf.* cours).

Question 2 – Somme des produits, en récursif

On définit la fonction `somProdRec` :

```
def somProdRec(n):
    if n == 0:
        return 0
    else:
        return somProdRec(n - 1) + n * somRec(n)
```

1. Prouver la terminaison et la validité de `somProdRec`.
2. Calculer sa complexité en nombre d'opérations arithmétiques ($+$, $*$).

Exercice 6 – Calcul du PGCD

La fonction `PGCD` retourne le plus grand diviseur commun des entiers x et y :

```
def PGCD(x, y):
     if y == 0 :
         res = x
     else:
         res = PGCD(y, x % y)
     return res
```

On étudie maintenant la complexité de la fonction `PGCD`, comptée en nombre de calculs de restes, c'est-à-dire en nombre d'appels à la fonction `%`.

Soit $c(x, y)$ le nombre d'appels à la fonction `%` effectués par $PGCD(x, y)$ pour x, y entiers naturels.

Dans les questions 1 à 5 on fera l'hypothèse que $x > y \geq 0$.

Question 1

Montrer que, pour tous $x > y \geq 0$:

$$c(x, y) = \begin{cases} 0 & \text{si } y = 0 \\ 1 + c(y, x \,\%\, y) & \text{sinon} \end{cases}$$

L'objet des deux questions suivantes est de montrer que, pour tous x, y entiers naturels tels que $x > y \geq 0$

$$\mathcal{P}(k) : \quad (c(x,y) = k) \Rightarrow (x \geq F_{k+2}).$$

On rappelle la définition des nombres de Fibonacci : $F_0 = 0$, $F_1 = 1$, $F_k = F_{k-1} + F_{k-2}$ si $k \geq 2$.

Question 2

Montrer que $\mathcal{P}(k)$ est vérifiée pour $k = 0$ et $k = 1$.

Question 3

Montrer, par récurrence forte sur k, que $\mathcal{P}(k)$ est vérifiée pour tout $k \geq 0$.

Question 4

Montrer que $F_{k+2} \geq \phi^k$ où ϕ est le nombre d'or ($\phi = \frac{1+\sqrt{5}}{2} \approx 1,618$).
On rappelle que $\phi^2 = \phi + 1$.

Question 5

1. En déduire que, si $x > y \geq 0$ et si $k = c(x,y)$, alors $x \geq \phi^k$.

2. On rappelle que $\phi > 1$, ce qui implique :

$$\exists \alpha > 0 \text{ tel que } \forall x > 0, \quad \log_\phi(x) = \alpha \log(x).$$

Montrer que, pour tous x, y entiers naturels tels que $x > y \geq 1$, la complexité de $PGCD(x,y)$ est en $\mathcal{O}(\log(x))$.

Question 6

Montrer que, pour tous x, y entiers naturels non nuls, la complexité de $PGCD(x,y)$ est en $\mathcal{O}(\log(\max(x,y)))$.