2I003 TD 4 Complexité d'un algorithme 4

Exercice 1 – Ordre de grandeur en O

Question 1

Démontrer que

(i) $n^2 \in O(10^{-5}n^3)$

(ii) $25n^4 - 19n^3 + 13n^2 \in O(n^4)$

(iii) $2^{n+100} \in O(2^n)$

Question 2

Donner les relations d'inclusion entre les ensembles suivants : $O(n \log n)$, $O(2^n)$, $O(\log n)$, $O(1)$, $O(n^2)$, $O(n^3)$ et $O(n)$.

Question 3

Soit un ordinateur pour lequel toute instruction possède une durée de 10^{-6} secondes. On exécute un algorithme qui utilise, pour une donnée de taille n, $f(n)$ instructions, $f(n)$ étant l'une des fonctions précédentes ($\log n$, $n \log n$, n, n^2, n^3 et 2^n). Remplir un tableau qui donne, en fonction de la taille $n = 10$, 20, 30 et 60, et de la fonction $f(n)$, la durée d'exécution de l'algorithme.

Exercice 2 – Ordre de grandeur en O, Ω et Θ

Question 1

Démontrer que

(i) $3n^4 - 5n^3 \in \Omega(n^4)$

(ii) $2^n \in \Omega(n^2)$

Question 2

Donner sans démontration les relations d'inclusion entre les ensembles suivants : $\Omega(n \log n)$, $\Omega(2^n)$, $\Omega(\log n)$, $\Omega(1)$, $\Omega(n^2)$, $\Omega(n^3)$ et $\Omega(n)$.

Question 3

Quelles sont les relations éventuelles d'inclusion entre les ensembles suivants ? Justifiez votre réponse sans démonstration.

(i) $\Theta(n^3)$ et $\Omega(n^2)$

(ii) $O(n^2)$ et $\Omega(n^3)$

(iii) $O(n^2)$ et $\Omega(n^2)$

Exercice 3 – Ordre de grandeur

Soient f, g, S et $T : \mathbb{N} \to \mathbb{N}$.

Question 1

1. Montrer que si $f(n) \in O(g(n))$, alors $g(n) \in \Omega(f(n))$.
2. Montrer que si $f(n) \in O(g(n))$, alors $f(n) + g(n) \in O(g(n))$.
3. Montrer que $f(n) + g(n) \in \Theta(\max(f(n), g(n)))$.
4. Montrer que $O(f(n) + g(n)) = O(\max(f(n), g(n)))$.

Question 2

On suppose que $S(n) \in O(f(n))$ et $T(n) \in O(g(n))$.

1. Montrer que si $f(n) \in O(g(n))$, alors $S(n) + T(n) \in O(g(n))$.
2. Montrer que $S(n)T(n) \in O(f(n)g(n))$.

Exercice 4 – Calculs de complexité simples

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 – 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 3 – Recherche itérative du minimum dans un tableau

La fonction `RechercheMinIte` retourne la première position du minimum d'un tableau T de nombres :

```
def RechercheMinIte(T):
    n = len(T) ; imin = 0 ;  i = 1
    while i < n:
        if T[i] < T[imin] :
            imin = i
        i = i + 1
    return imin
```

Exprimer sa complexité en nombre de comparaisons ($<$) entre éléments du tableau en fonction de la taille n de T.

Question 4 – Recherche récursive du minimum dans un tableau

La fonction `RechercheMinRec` retourne la première position du plus petit des n premiers éléments d'un tableau T de nombres :

```
def RechercheMinRec(T,n):
    if (n == 1):
        return 0
    else :
        imin = RechercheMinRec(T, n - 1)
        if (T[imin] > T[n - 1]):
            return n - 1
        return imin
```

Exprimer sa complexité en nombre de comparaisons ($<$) entre éléments du tableau en fonction de n.

Exercice 5 – Tri par sélection

Le principe du tri par sélection est le suivant : étant donné un tableau de n nombres, on cherche le plus petit élément du tableau, on le place en dernière position, puis on recommence avec les $n-1$ premiers éléments du nouveau tableau, et ainsi de suite. Le tableau est ainsi trié en ordre décroissant.
On étudie maintenant la complexité de cet algorithme, dans sa version récursive et dans sa version itérative, en nombre de comparaisons ($<$) entre éléments du tableau en fonction de la taille n de du tableau.

Algorithme récursif de tri par sélection

Voici un algorithme récursif qui trie en ordre décroissant les n premiers éléments d'un tableau T (occupant donc les positions $0, \ldots, n-1$) :

```
def triSelectionRec(T, n):
    if n > 1:
        imin = RechercheMinRec(T, n)
        tmp = T[n - 1] ; T[n - 1] = T[imin] ;  T[imin] = tmp
        triSelectionRec(T, n -1)
```

La fonction `RechercheMinRec` a été étudiée dans l'exercice précédent.
Notons u_n le nombre de comparaisons entre éléments du tableau effectuées par l'appel `triSelectionRec(T,n)`.

Question 1

Donner une relation de récurrence vérifiée par la suite u_n.

Question 2

En déduire la complexité de `triSelectionRec(T,n)`.

Algorithme itératif de tri par sélection

Voici un algorithme itératif qui trie en ordre décroissant les éléments d'un tableau T :

```
def triSelectionIte(T):
    n = len(T) ; i = n
    while i > 1:
        imin = RechercheMinIte(T[0 : i])
        tmp = T[i - 1] ; T[i - 1] = T[imin] ;  T[imin] = tmp
        i = i - 1
```

La fonction `RechercheMinIte` a été étudiée dans l'exercice précédent.

Question 3
Calculer la complexité de `triSelectionIte`.

Exercice 6 – Tri à bulles

On rappelle l'algorithme de tri à bulles :

```
def Push(T, k):
    j = 1
    while j < k:
        if T[j] < T[j - 1]:
            tmp = T[j - 1]
            T[j - 1] = T[j]
            T[j] = tmp
        j = j + 1

def BubbleSort(T):
    i = 0
    n = len(T)
    while i < n:
        Push(T, n - i)
        i = i + 1
```

On étudie maintenant la complexité de l'algorithme `BubbleSort` en nombre de comparaisons ($<$) entre éléments du tableau en fonction de la taille n de T.

Question 1
Pour $1 \leq k < n$, combien `Push(T,k)` effectue-t-il de comparaisons entre éléments du tableau ?

Question 2
Calculer la complexité de `BubbleSort`.

Exercice 7 – Recherche 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 dans le pire cas 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 $T[d \dots f]$, sous l'hypothèse que $0 \leq d \leq f \leq n-1$ et que $elem \in T[d \dots f]$:

```
def RechercheDicho (elem, T, d, f):
    i = (d + f) // 2
    if (T[i] == elem):
        return i
    if (T[i] > elem):
```

```
        return RechercheDicho (elem, T, d, i - 1)
    return RechercheDicho (elem, T, i + 1, f)
```

Question 2
Exécuter l'appel `RechercheDicho (99, T, 0, 10)` pour le tableau d'entiers triés

`T=[2, 5, 9, 16, 45, 66, 78, 89, 90, 96, 99, 100].`

Représenter l'arbre des appels associé, ainsi que l'état de la pile pour un chemin de la racine à une feuille.

Question 3
Prouver la validité de la fonction si $0 \leq d \leq f \leq n-1$ et si $elem \in T[d \dots f]$.

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

Exercice 8 – 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 $\mathcal{O}(\log(\max(x, y)))$.