Université USTHB – Bab-Ezzouar
Faculté de l'Informatique
1ére Année Master Informatique Visuelle (M1/IV)
Module : Complexité des Algorithmes
Chargé de Cours : Mr. AMANI F.

Bab-Ezzouar, 10 Mars 2021
Année universitaire 2020/2021
Semestre 1

Interrogation

Note : Le calcul d'une complexité doit être démonstratif. On n'accepte pas les réponses directes (et évasives) du type : "la complexité de l'algorithme est en $O(n)$". On doit montrer d'où provient cet $O(n)$.

Exercice 1

1- Soit une suite $u_n, n \geq 0$ définie comme suit :

$$\begin{cases} u_{n+1} = 2u_n - 3 & \forall n \geq 1 \\ u_0 = 2 \end{cases}$$

1.a- Calculer les expressions suivantes :

$1)\ u_1 - u_0;\ 2)\ u_2 - u_1;\ 3)\ u_3 - u_2;\ 4)\ u_4 - u_3;\ 5)\ u_5 - u_4.$

1.b- Conjecturer (çàd, proposer sans le démontrer) une expression algébrique pour le terme général u_n.

1.c- Démontrer par récurrence sur l'entier naturel n cette conjecture.

2- On rappelle la définition de la notation asymptotique O (grand O) :

Définition : Soient 2 fonctions numériques $f(n)\ et\ g(n)$ définies dans l'ensemble des nombres naturels N et à valeurs positives dans l'ensemble des nombres réels R :

$$f, g : N \rightarrow R^+$$

On dit que la fonction $g(n)$ est une borne supérieure asymptotique pour la fonction $f(n)$, et on note :

$$f(n) = O\big(g(n)\big)$$

si et seulement si :

$$\exists\ c \in R^{+*}, \exists\ n_0 \in N\ tels\ que : f(n) \leq cg(n) \quad \forall n \geq n_0$$

Question : Démontrer la propriété suivante sur la notation asymptotique O :

$$O\big(f(n)\big) + O\big(g(n)\big) = O(\max(f(n), g(n)))$$

Exercice 2

1- On considère le code suivant :

```
//code_x1
i = 1 ;
tant que (i <= n)
début
    j = 1 ;
    tant que (j <= n)
    début
        écrire("Salam de la section
                Master 1/IV promotion
                2020/2021") ;
        j = j + 1 ;
    fin tant que ;
    i = i * 2 ;
fin tant que ;
```

Questions :

1.a- Calculer la complexité temporelle, notée $ct(n)$, dans le cas où :

$n = 2^k, k \geq 0.$

Préciser le meilleur cas et le pire cas.

1.b- Combien de fois le message ("**Salam de la section Master 1/IV promotion 2020/2021**") est affiché.

1.c- Refaire les 2 questions précédentes dans le cas général où n est quelconque (n n'est pas une puissance de 2).

2.a- Développer un algorithme itératif qui permet de déterminer les chiffres d'un nombre composant un nombre entier naturel n. Ces chiffres doivent être enregistrés dans un tableau d'entiers noté tab.

Exemple :

Pour $n = 2021$, on a : tab =

2	0	2	1

2.b- Calculer le nombre d'instructions, noté nb, exécutées par cet algorithme.

2.c- Calculer la complexité temporelle, notée $ct(n)$, en notations exacte et asymptotique en O (Grand O), de cet algorithme.

Ind : On peut utiliser les opérations de division entière, notée **div**, et du reste de la division entière, notée **mod**.

Exercice 3

1- On rappelle le théorème 1 sur les équations de récurrence linéaires homogènes à coefficients constants :

Théorème 1 : Etant donné une équation de récurrence linéaire homogène à coefficients constants d'ordre k définie comme suit :

$$\begin{cases} c_k t(n+k) + c_{k-1} t(n+k-1) + \ldots + c_0 t(n) = 0 \\ t(n_0) = d_0 \\ t(n_0+1) = d_1 \\ \quad \ldots \\ \quad \ldots \\ t(n_0+k-1) = d_{k-1} \end{cases} \quad (1)$$

où,

- les paramètres d_i $(i = 0, \ldots, k-1)$ sont des constantes réelles définissant les conditions initiales de la récursion à partir de l'indice n_0 ;
- $t(n+k)$ désigne le terme qui détermine **l'ordre *k*** de l'équation (1).

Et, étant donné l'équation caractéristique de la relation (1) définie comme suit :

$$c_k r^k + c_{k-1} r^{k-1} + \cdots + c_1 r + c_0 = 0 \quad (2)$$

où,

- le paramètre r désigne la variable inconnue de l'équation.

Alors, la solution générale de l'équation est de la forme suivante :

$$t(n) = \sum_{i=1}^{nb} \left[r_i^n . \left(\sum_{j=0}^{m_i - 1} a_{ij}\, n^j \right) \right] \qquad (3)$$

où,

- le paramètre $nb \leq k$ désigne le nombre de racines distinctes de l'équation caractéristique (3) ;
- le paramètre r_i désigne une racine de l'équation caractéristique (2) ;
- le paramètre m_i désigne la multiplicité de la racine r_i ;
- les coefficients a_{ij} sont des constantes qui sont déterminées à partir des conditions initiales.

Question : Résoudre l'équation de récurrence linéaire suivante :

$$\begin{cases} t(n+2) - 2t(n+1) + t(n) = 0 \quad \forall n \geq 2 \\ t(0) = 2 \\ t(1) = 7 \end{cases}$$

2- On rappelle le théorème 2 sur les équations de récurrence non-linéaires à coefficients constants de type **"diviser et régner"** :

Théorème 2 (Master Théorème) : Soit l'équation de récurrence non linéaire à coefficients constants de type **"diviser et régner"** définie comme suit :

$$\boldsymbol{t(n) = at\left(\frac{n}{b}\right) + f(n)}$$

où :

- $f(n)$ est une fonction numérique définie comme suit :
  $f : N \rightarrow R^+$ telle que : $f(n) = O(n^d)$;
- d est une constante réelle positive, $d \in R,\ d \geq 0$;
- a est une constante naturelle, $a \in N,\ a \geq 1$;
- b est une constante naturelle, $b \in N,\ b > 1$;

Alors, la solution de cette équation en notation asymptotique de Landau est comme suit :

$$t(n) = \begin{cases} O\left(n^{\log_b(a)}\right) & si \quad a > b^d \\ O\left(n^d \log_e(n)\right) & si \quad a = b^d \\ O(n^d) & si \quad a < b^d \end{cases} \qquad (4)$$

Question : Résoudre l'équation de récurrence non-linéaire suivante :

$$t(n) = t\left(\frac{n}{3}\right) + n \quad \forall n \geq 1$$

Bon courage
et bonne continuation dans vos études.

Corrigé de l'interrogation

Note : Le calcul d'une complexité doit être démonstratif. On n'accepte pas les réponses directes (et évasives) du type : "la complexité de l'algorithme est en $O(n)$". On doit montrer d'où provient cet $O(n)$.

Exercice 1 (4 points/Q1=2(=0,5+0,5+1), Q2=2)

1- Soit une suite $u_n, n \geq 0$ définie comme suit :

$$\begin{cases} u_{n+1} = 2u_n - 3 & \forall n \geq 1 \\ u_0 = 2 \end{cases}$$

1.a- Calculer les expressions suivantes :

1) $u_1 - u_0$; 2) $u_2 - u_1$; 3) $u_3 - u_2$; 4) $u_4 - u_3$; 5) $u_5 - u_4$.

1.b- Conjecturer (çàd, proposer sans le démontrer) une expression algébrique pour le terme général u_n.

1.c- Démontrer par récurrence sur l'entier naturel n cette conjecture.

Solution :

Rem : Cet exercice est traité dans la série TD n° 1 / Exercice 3

1.a- Calculer les expressions suivantes :

1) $u_1 - u_0$; 2) $u_2 - u_1$; 3) $u_3 - u_2$; 4) $u_4 - u_3$; 5) $u_5 - u_4$.

Solution :

On a :

1) $u_1 = 2u_0 - 3 = 2.(2) - 3 = 4 - 3 = 1$

$\Rightarrow$ $u_1 - u_0 = 1 - 2 = -1$

2) $u_2 = 2u_1 - 3 = 2.(1) - 3 = 2 - 3 = -1$

$\Rightarrow$ $u_2 - u_1 = (-1) - (1) = -2$

3) $u_3 = 2u_2 - 3 = 2.(-1) - 3 = -2 - 3 = -5$

$\Rightarrow$ $u_3 - u_2 = (-5) - (-1) = -5 + 1 = -4$

4) $u_4 = 2u_3 - 3 = 2.(-5) - 3 = -10 - 3 = -13$

$\Rightarrow$ $u_4 - u_3 = (-13) - (-5) = -13 + 5 = -8$

5) $u_5 = 2u_4 - 3 = 2.(-13) - 3 = -26 - 3 = -29$

$\Rightarrow$ $u_5 - u_4 = (-29) - (-13) = -29 + 13 = -16$

On peut remarquer que les différences :

$u_1 - u_0 = -1$;

$u_2 - u_1 = -2$;

$u_3 - u_2 = -4$;
$u_4 - u_3 = -8$;
$u_5 - u_4 = -16$;

forment une suite géométrique de raison : $r = 2$.

1.b- Conjecturer (càd, proposer sans le démontrer) une expression algébrique pour le terme général u_n.

Solution :

En se servant de la relation trouvée en question 1, on a :

1) $u_1 - u_0 = -1 \Rightarrow u_1 = -1 + u_0 = -1 + 2 = -2^0 + 2$

2) $u_2 - u_1 = -2 \Rightarrow u_2 = -2 + u_1 = -2^1 - 2^0 + 2$

3) $u_3 - u_2 = -4 \Rightarrow u_3 = -4 + u_2 = -2^2 - 2^1 - 2^0 + 2$

4) $u_4 - u_3 = -8 \Rightarrow u_4 = -8 + u_3 = -2^3 - 2^2 - 2^1 - 2^0 + 2$

5) $u_5 - u_4 = -16 \Rightarrow u_5 = -16 + u_4 = -2^4 - 2^3 - 2^2 - 2^1 - 2^0 + 2$

On conjecture l'expression suivante pour le terme général :

$$u_n = -2^{n-1} - 2^{n-2} - \cdots - 2^1 - 2^0 + 2 \quad (1)$$

On multiplie (1) par 2, on obtient :

$$2u_n = -2^n - 2^{n-1} - \cdots - 2^2 - 2^1 + 4 \quad (2)$$

En soustrayant (1) de (2), on a :

$$u_n = -2^n + 4 - (-2^0) - 2 = -2^n + 3$$

Il convient de montrer cette conjecture pour, le cas échéant, la rendre vraie.

1.c- Démontrer cette conjecture[1].

[1] En mathématiques, une **conjecture** est une assertion pour laquelle on ne connaît pas encore de démonstration, mais que l'on croit fortement être vraie, en l'absence de contre-exemple. Quand une conjecture est démontrée, elle devient un théorème. Formulée vraisemblablement en 1637, publiée en 1670, la plus célèbre de toutes les conjectures était celle dénommée le « dernier théorème de Fermat ». Ce n'est qu'après sa démonstration par le mathématicien Andrew Wiles en 1994 que cette conjecture devint théorème. L'hypothèse de Riemann est une conjecture non démontrée ; donc, elle n'est pas un théorème [source, wikipédia, 2020].

Solution :

Soit $C(n)$ le nom de la conjecture ci-dessus :

$$u_n = -2^n + 3 \quad \forall n \geq 0 \qquad (C(n))$$

La démonstration par récurrence s'effectue en 2 étapes comme suit :

1- Etape 1 : Vérifier le cas de base : $C(0)\ est\ vraie.$

//çad, montrer que : $C(0)\ est\ vraie\ pour\ n = 0.$

On a :

$$\begin{cases} -2^0 + 3 = -1 + 3 = 2 \\ u_0 = 2 \end{cases} \Rightarrow u_0 = -2^0 + 3 \Rightarrow C(0)\ est\ vraie \qquad (1)$$

2- Etape 2 : Montrer l'hérédité :

$$C(n)\ est\ vraie \Rightarrow C(n+1)\ est\ vraie \quad \forall n \geq 0.$$

On a :

$$u_{n+1} = 2u_n - 3$$

$\Rightarrow u_{n+1} = 2(-2^n + 3) - 3$ //par hypothèse, $C(n)\ est\ vraie$

$\Rightarrow u_{n+1} = -2^{n+1} + 6 - 3$

$\Rightarrow u_{n+1} = -2^{n+1} + 3 \Rightarrow C(n+1)\ est\ vraie \qquad (2)$

//$C(n+1)\ s'écrit\ de\ la\ même\ manière\ que\ C(n).$

En conséquence,

$$(1)\ et\ (2) \Rightarrow C(n)\ est\ vraie \quad \forall n \geq 0.$$

2- On rappelle la définition de la notation asymptotique O (grand O) :

Définition : Soient 2 fonctions numériques $f(n)\ et\ g(n)$ définies dans l'ensemble des nombres naturels N et à valeurs positives dans l'ensemble des nombres réels R :

$$f, g : N \rightarrow R^+$$

On dit que la fonction $g(n)$ est une borne supérieure asymptotique pour la fonction $f(n)$, et on note :

$$f(n) = O(g(n))$$

si et seulement si :

$$\exists\, c \in R^{+*}, \exists\, n_0 \in N\ tels\ que : f(n) \leq cg(n) \quad \forall n \geq n_0$$

Question : Démontrer la propriété suivante sur la notation asymptotique O :

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

Solution :

Rem : Cet exercice est traité dans la série TD n° 1 / Exercice 7 / Question 2

On pose : $T(n) = O(f(n))$ et $H(n) = O(g(n))$

$T(n) = O\big(f(n)\big)$
$\Leftrightarrow \exists\, c_1 \in R, c_1 > 0, \exists\, n_1 \in N, n_1 \geq 0\ tels\ que :\ \forall n \geq n_1, T(n) \leq c_1 f(n)$ (1)

$H(n) = O\big(g(n)\big)$
$\Leftrightarrow \exists\, c_2 \in R, c_2 > 0, \exists\, n_2 \in N, n_2 \geq 0\ tels\ que :\ \forall n \geq n_2, H(n) \leq c_2 g(n)$ (2)

On pose : $n_0 = \max(n_1, n_2)$

En sommant (1) et (2) avec la nouvelles valeurs n_0, on a :

$\exists\, c_1 \in R, c_1 > 0, \exists\, c_2 \in R, c_2 > 0, \exists\, n_0 \in N, n_0 \geq 0\ tels\ que :$
$\forall n \geq n_0, T(n) + H(n) \leq c_1 f(n) + c_2 g(n))$

$\Leftrightarrow \exists\, c_1 \in R, c_1 > 0, \exists\, c_2 \in R, c_2 > 0, \exists\, n_0 \in N, n_0 \geq 0\ tels\ que :$
$\forall n \geq n_0, T(n) + H(n) \leq c_1 \max\big(f(n), g(n)\big) + c_2 \max\big(f(n), g(n)\big)$

$\Leftrightarrow \exists\, c_1 \in R, c_1 > 0, \exists\, c_2 \in R, c_2 > 0, \exists\, n_0 \in N, n_0 \geq 0\ tels\ que :$
$\forall n \geq n_0, T(n) + H(n) \leq (c_1 + c_2) \max\big(f(n), g(n)\big)$

En posant : $c_0 = c_1 + c_2$, on a :

$\exists\, c_0 \in R, c_0 > 0, \exists\, n_0 \in N, n_0 \geq 0\ tels\ que:$
$\forall n \geq n_0, O\big(f(n)\big) + O\big(g(n)\big) \leq c_0 max(f(n), g(n))$

$\Leftrightarrow O\big(f(n)\big) + O\big(g(n)\big) = O(max(f(n), g(n)))$

Exercice 2 (12 points/Q1=6(=2+1+3), Q2=6(=2+2+2))

1- On considère le code suivant :

```
//code_x1
i = 1 ;
tant que (i <= n)
début
    j = 1 ;
    tant que (j <= n)
    début
        écrire("Salam de la section
                Master 1/IV promotion
                2020/2021") ;
        j = j + 1 ;
    fin tant que ;
    i = i * 2 ;
fin tant que ;
```

Questions :

1.a- Calculer la complexité temporelle, notée $ct(n)$, dans le cas où :
$n = 2^k, k \geq 0$.
Préciser le meilleur cas et le pire cas.

1.b- Combien de fois le message ("**Salam de la section Master 1/IV promotion 2020/2021**") est affiché.

1.c- Refaire les 2 questions précédentes dans le cas général où n est quelconque (n n'est pas une puissance de 2).

1.a- Calculer la complexité temporelle, notée $ct(n)$, dans le cas où : $n = 2^k, k \geq 0$. Préciser le meilleur cas et le pire cas.

Solution :

Rem : Les questions 1.a et 1.b sont similaires à celles traitées dans la série TD n° 2 / Exercices 1 à 6.

Le tableau des fréquences d'exécution des instructions de l'algorithme **code x1** est comme suit (voir le commentaire explicatif ci-dessous) :

Algorithme code x1;	**Fréquence d'exécution f_i i = 1, …, L=7**
Début	
1 i = 1 ;	→ $f_1 = 1$
2 Tant que (i<=n)	→ $f_2 = log_2(n) + 2$
début	
3 j = 1 ;	→ $f_3 = log_2(n) + 1$
4 Tant que (j<=n)	→ $f_4 = nlog_2(n) + n + log_2(n) + 1$
Début	
5 écrire("Salam …") ;	→ $f_5 = nlog_2(n) + n$
6 j = j + 1 ;	→ $f_6 = 2(nlog_2(n) + n)$
fin tant que ;	
7 i = i * 2 ;	→ $f_7 = 2(log_2(n) + 1)$
fin tant que ;	
Fin. //fin de code x1	

Commentaire explicatif :

On a :

- $log_e(x)$: désigne la fonction logarithme népérien de x ;
- $log_b(x)$: désigne la fonction logarithme en base b de x et est définie comme suit : $log_b(x) = \frac{log_e(x)}{log_e(b)}$;

  en conséquence, $log_2(2) = \frac{log_e(2)}{log_e(2)} = 1.$
- $n = 2^k,\ k \geq 0$

$\Rightarrow\ log_2(n) = log_2(2^k) = klog_2(2) = k.1 = k,\ \ k \geq 0\ \ et\ \ n \geq 1$

$\Rightarrow\ log_2(n) = klog_2(2),\ \ k \geq 0\ \ et\ \ n \geq 1$

$\Rightarrow\ log_2(n) = k.1 = k,\ \ k \geq 0\ \ et\ \ n \geq 1$

$\Rightarrow\ log_2(n) = k,\ \ k \geq 0\ \ et\ \ n \geq 1$

1- Instruction de répétition n°2 : tant que (i≤n)

Il faut analyser les variations de la variable i qui joue ici le rôle de contrôleur (et non pas de compteur) de l'instruction de répétition n°2 : **tant que (i≤n)** (en fond de couleur jaune dans le tableau des fréquences d'exécution), comme suit :

Les valeurs successives de i sont :

$$\boldsymbol{i = 1 = 2^0 \rightarrow 2^1 \rightarrow 2^2\ \ldots\ \rightarrow 2^{k-1} \rightarrow n = 2^k \rightarrow 2^{k+1}}$$

//la dernière valeur $i = 2^{k+1}$ permet l'arrêt de l'instruction de répétition n°2

On remarque que l'indice en puissance varie de 0 à $k + 1$;

Il en résulte :

i prend $(k + 2)$ valeurs = $(log_2(n) + 2)$ valeurs

$\Rightarrow f_2(n) = log_2(n) + 2$

2- Instruction de répétition n°4 : tant que (j≤n)

Il faut analyser les variations des 2 variable i et j qui jouent ici le rôle de contrôleur (et non pas de compteur) des instructions de répétition n°2 et n°4 : **tant que (i≤n)** et **tant que (j≤n)** (en fond de couleur jaune dans le tableau des fréquences d'exécution), comme suit :

Les valeurs successives de j dépendent de celles de i et sont comme suit :

$$\boldsymbol{i = 1 = 2^0 \Rightarrow\ j = 1 \rightarrow 2 \rightarrow \dots \rightarrow n \rightarrow (n+1) \Rightarrow (n+1)\ fois}$$
$$\boldsymbol{i = 2 = 2^1 \Rightarrow\ j = 1 \rightarrow 2 \rightarrow \dots \rightarrow n \rightarrow (n+1) \Rightarrow (n+1)\ fois}$$
$$\boldsymbol{\dots}$$
$$\boldsymbol{\dots}$$
$$\boldsymbol{\dots}$$
$$\boldsymbol{i = 2^k = n \Rightarrow\ j = 1 \rightarrow 2 \rightarrow \dots \rightarrow n \rightarrow (n+1) \Rightarrow (n+1)\ fois}$$

//la dernière valeur $j = (n+1)$ permet à chaque fois l'arrêt de l'instruction //de répétition n°4

$\boldsymbol{i = 2^{k+1} = 2n > n} \Rightarrow$ Il y a arrêt de l'instruction de répétition n°2 ;

et, donc, l'instruction de répétition n°4 n'est pas exécutée.

Il en résulte :

$$f_4(n) = (k+1)(n+1)$$

$$\Rightarrow\ f_4(n) = (log_2(n)+1)(n+1)$$

$$\Rightarrow\ f_4(n) = nlog_2(n) + n + log_2(n) + 1$$

On calcule la somme des fréquences d'exécution des $L = 7$ instructions, notée $nbx(n)$:

$$nbx(n) = \sum_{i=1}^{L=7} f_i\,(n)$$

$$\Rightarrow nbx(n) = f_1(n) + \dots + f_7(n)$$

$$\Rightarrow nbx(n) = \dots = 4nlog_2(n) + 4n + 5log_2(n) + 7$$

En conséquence,

a - En notation exacte :

On a :

$$ct(n) = nbx(n)$$

$$\Rightarrow\ ct(n) = 4nlog_2(n) + 4n + 5log_2(n) + 7$$

b- En notation asymptotique O :

On a :

$4nlog_2(n) + 4n + 5log_2(n) + 7 = 4nlog_2(n) + 4n + 5log_2(n) + 7 \quad \forall n \geq 1$

$\Rightarrow 4nlog_2(n) + 4n + 5log_2(n) + 7 \leq 4nlog_2(n) + 4nlog_2(n) + 5nlog_2(n) + 7nlog_2(n) \quad \forall n \geq 2$

$\Rightarrow 4nlog_2(n) + 4n + 5log_2(n) + 7 \leq 20nlog_2(n) \ \forall n \geq 2$

Il en résulte que :

$\exists\, c = 20 > 0, \exists\, n_0 = 2 \geq 0 \ tels\ que:$

$\forall \geq 2, 4nlog_2(n) + 4n + 5log_2(n) + 7 \leq 20nlog_2(n)$

//ici, $f(n) = 4nlog_2(n) + 4n + 5log_2(n) + 7$ *et* $g(n) = nlog_2(n)$

$\Leftrightarrow 4nlog_2(n) + 4n + 5log_2(n) + 7 = O(log_2(n))$

$\Leftrightarrow ct(n) = O(nlog_2(n)) \ \forall n \geq 2$

En conséquence, la complexité temporelle de l'algorithme code x1 est :

$$ct(n) = \begin{cases} 4nlog_2(n) + 4n + 5log_2(n) + 7 & en\ notation\ exacte \\ O(nlog_2(n))) & en\ notation\ asymptotique \end{cases}$$

Rappel : On rappelle (voir chapitre 2/section 2.3.3) :

a- Le meilleur cas

Le meilleur cas d'exécution d'une instruction dans un algorithme correspond au nombre minimal de son exécution dans cet algorithme.

b- Le pire cas

Le pire cas d'exécution d'une instruction dans un algorithme correspond au nombre maximal de son exécution dans cet algorithme.

En conséquence, il suffit de déterminer le minimum et le maximum de la fonction $ct(n)$.

On a :

$\min(ct(n)) = min(4nlog_2(n) + 4n + 5log_2(n) + 7), n \geq 1, n \in N$

$\Rightarrow \min(ct(n)) = 11$ //car $ct(n) \geq 0\ \forall n \geq 1$

$\max(ct(n)) = max(4nlog_2(n) + 4n + 5log_2(n) + 7), n \geq 1, n \in N$

//on prend la valeur maximum de la variable n

En conséquence,

$$ct(n) = \begin{cases} 11 & au\ meilleur\ cas \\ \max(4nlog_2(n) + 4n + 5log_2(n) + 7), \quad \forall n \geq 1, n \in N & au\ pira\ cas \end{cases}$$

1.b- Combien de fois le message ("**Salam de la section Master 1/IV promotion 2020/2021**") est affiché.

Solution :

Il est clair que ce message est affiché : $\boldsymbol{f_5 = nlog_2(n) + n, \quad \forall n \geq 1, n \in N}$ **fois.**

1.c- Refaire les 2 questions précédentes dans le cas général où n est quelconque (n n'est pas une puissance de 2).

Solution :

Rappel : On rappelle le résultat suivant sur l'encadrement d'un nombre entier naturel n par des puissances successives d'un nombre entier naturel b (ici, $b = 2$).

On a :

$$\forall n \in N, \exists k \in N\ tel\ que : 2^k < n \leq 2^{k+1}$$

$$\Rightarrow \forall n \in N, \exists k \in N\ tel\ que : log_2(2^k) < log_2(n) \leq log_2(2^{k+1})$$

$$\Rightarrow \forall n \in N, \exists k \in N\ tel\ que : k.log_2(2) < log_2(n) \leq (k+1).log_2(2))$$

$$\Rightarrow \forall n \in N, \exists k \in N\ tel\ que : k < log_2(n) \leq (k+1)$$

$$//log_2(2) = 1$$

$$\Rightarrow \forall n \in N, \exists k \in N\ tel\ que : k = \lfloor log_2(n) \rfloor \ \ et\ \ 2^k < n \leq 2^{k+1}$$

//$\lfloor x \rfloor$ désigne la partie entière inférieure du nombre réel x

Exemple : $n = 17$ n'est pas une puissance de 2.
Par application de ce résultat, on a la valeur de k :

$$k = \lfloor log_2(17) \rfloor = \lfloor 4.0874 \rfloor = 4 \ \ et\ \ 2^4 = 16 < n \leq 2^5 = 32$$

Dans le cas général où n est quelconque (n n'est pas une puissance de 2), la fréquence d'exécution des instructions qui peut être modifiée en premier est : $f_2(n)$.

Les modifications éventuelles des autres fréquences dépendent de celle de $f_2(n)$.

Pour le calcul de la nouvelle fréquence de l'instruction n°2, notée $f_{22}(n)$, on reprend l'analyse (déjà présentée ci-dessus) de cette instruction, mais dans le cas général où n est quelconque (n n'est pas une puissance de 2) :

1- Instruction de répétition n°2 : tant que (i≤n)

Il faut analyser les variations de la variable i qui joue ici le rôle de contrôleur (et non pas de compteur) de l'instruction de répétition n°2 : **tant que (i≤n)** (en fond de couleur jaune dans le tableau des fréquences d'exécution), comme suit :

Les valeurs successives de i sont :

$$\boldsymbol{i = 1 = 2^0 \rightarrow 2^1 \rightarrow 2^2 \ \ldots \ \rightarrow 2^{k-1} \rightarrow 2^k \rightarrow n \rightarrow 2^{k+1}}$$

//la dernière valeur $i = 2^{k+1}$ permet l'arrêt de l'instruction de répétition n°2
//ici, n est compris entre 2^k et 2^{k+1}

On remarque que l'indice en puissance varie de 0 à $k+1$;

Il en résulte :

i prend $(k+2)$ valeurs = $(log_2(n)+2)$ valeurs

$\Rightarrow f_{22}(n) = log_2(n) + 2$

$\Rightarrow f_{22}(n) = f_2(n)$

⇒ les autres fréquences ne sont pas modifiées.

En conséquence, la complexité temporelle est la même que celle où n est une puissance de 2.

2.a- Développer un algorithme itératif qui permet de déterminer les chiffres d'un nombre composant un nombre entier naturel n. Ces chiffres doivent être enregistrés dans un tableau d'entiers noté tab.

Exemple :

Pour $n = 2021$, on a : tab =

2	0	2	1

Ind : On peut utiliser les opérations de division entière, notée **div**, et du reste de la division entière, notée **mod**.

Solution :

L'algorithme itératif, noté **"Déchiffrer_nombre"**, permettant de déterminer les chiffres composant un nombre entier naturel n est basé sur le calcul des restes successifs de la division de n par 10. On utilise les 2 opérations de division entière, notée **div**, et l'opération du reste de la division entière notée **mod**. Il est présenté sur la figure 1 ci-dessous.

Exemple : soit le nombre $n = 2021$.

On a :

$2021\ mod\ 10 = 1$ et $2021\ div\ 10 = 202$

On refait ces 2 opérations avec le nouveau quotient 202 :

$202\ mod\ 10 = 2$ et $202\ div\ 10 = 20$

On refait ces 2 opérations avec le nouveau quotient 20 :

$20\ mod\ 10 = 0$ et $20\ div\ 10 = 2$

On refait ces 2 opérations avec le nouveau quotient 2 :

$2\ mod\ 10 = 2$ et $2\ div\ 10 = 0$

On arrête les opérations lorsqu'on obtient un quotient égal à 0 (ici, on a : $2\ div\ 10 = 0$.

On remarque sur ce exemple que les restes 1, 2, 0 et 2 (en fond de couleur jaune) constituent les chiffres du nombre $n = 2021$ et sont obtenus avec l'opération du reste de la division entière notée **mod**.

Comme les chiffres 1, 2, 0 et 2 du nombre $n = 2021$ sont obtenus dans le sens inverse de l'écriture normalisée des nombres, il convient d'ajouter dans l'algorithme une étape d'inversion de ce sens.

```
Algorithme Déchiffrer_nombre ;          //Algorithme itératif de déchiffrement
                                        //d'un nombre entier naturel
Var     n, quot, reste, i, j, k, ch : entier ;
        tab : tableau[100] entier ;

 Début
   //Partie 1: Lecture des données
1  ecrire ("Donner un nombre n = ");
2  lire (n);

   //Partie 2: Traitement des données
   //Etape 1 : Déchiffrement du nombre n
3  m = n ;                  //m permet de sauvegarder n
4  i = 1 ;
5  Tant que (m > 0) faire
   Début
6       reste = m mod 10 ;
7       tab[i] = reste ;
8       i = i +1 ;
9       m = m div 10 ;
   Fin tant que ;

   //Etape 2 : Inversion du tableau tab
10  ch = i – 1 ;          //ch = nombre de chiffres du nombre n
11  j = ch ;
12  k = 1 ;
13 Tant que (j > k) faire
    Début
14      z = tab[j] ;
15      tab[j] = tab[k] ;
16      tab[k] = z ;
17      j = j - 1 ;
18      k = k + 1 ;
    Fin tant que ;

   //Partie 3: Sortie des résultats
19  k = 1 ;
20 Tant que (k <= ch) faire
    Début
21      écrire(tab[k]) ;
22      k = k + 1 ;
    Fin tant que ;
Fin ; //fin de l'algorithme Déchiffrer_nombre
```

Figure 2. Déchiffrer_nombre : Algorithme de déchiffrement d'un nombre entier naturel.

Rem 1 : On n'est pas obligé de connaître à l'avance le nombre de chiffres du nombre n. Cependant, on peut connaître ce nombre de chiffres en utilisant le résultat mathématique élémentaire et fondamental (voir question n°2 ci-dessous)).

Rem 2 : Le nombre $n = 0$ n'est pas traité dans cet algorithme. On peut ajouter aisément une étape pour prendre en compte ce cas.

2.b- Calculer le nombre d'instructions, noté nbx, exécutées par cet algorithme.

Solution :

Soit L le nombre d'instructions de l'algorithme **Déchiffrer_nombre** (ci-dessus) et soit nbx la somme des fréquences d'exécutions de toutes les instructions de cet algorithme.

On a : $L = 22$ instructions.

Pour calculer nbx, on utilise les fréquences d'exécutions f_i de chacune des $L = 22$ instructions de l'algorithme **Déchiffrer_nombre** comme il est montré sur le tableau suivant (figure 3) :

N°	Instruction I_i		Fréquence d'exécution f_i
1	ecrire('Donner un nombre n : ') ;	→	$f_1 = 1$
2	lire (n);	→	$f_2 = 1$
	//Partie 2: Traitement des données		
	//Etape 1 : Déchiffrement du nombre n		
3	m = n ;	→	$f_3 = 1$
4	**i** = 1 ;	→	$f_4 = 1$
5	Tant que (m > 0) faire	→	$f_5 = \lfloor log_{10}(n) \rfloor + 2$
	Début		
6	reste = m mod 10 ;	→	$f_6 = 2(f_5 - 1)$
7	tab[i] = reste ;	→	$f_7 = f_5 - 1$
8	i = i +1 ;	→	$f_8 = 2(f_5 - 1)$
9	m = m div 10 ;	→	$f_9 = 2(f_5 - 1)$
	Fin tant que ;		
	//Etape 2 : Inversion du tableau tab		
10	ch = i – 1 ;	→	$f_{10} = 2$
11	j = ch ;	→	$f_{11} = 1$
12	k = 1 ;	→	$f_{12} = 1$
13	Tant que (j > k) faire	→	$f_{13} = \lfloor (f_5 - 1)/2 \rfloor + 1$
	Début		
14	z = tab[j] ;	→	$f_{14} = f_{13} - 1 = \lfloor (f_5 - 1)/2 \rfloor$
15	tab[j] = tab[k] ;	→	$f_{15} = f_{13} - 1 = \lfloor (f_5 - 1)/2 \rfloor$
16	tab[k] = z ;	→	$f_{16} = f_{13} - 1 = \lfloor (f_5 - 1)/2 \rfloor$
17	j = j - 1 ;	→	$f_{17} = 2(f_{13} - 1) = 2\lfloor (f_5 - 1)/2 \rfloor$
18	k = k + 1 ;	→	$f_{18} = 2(f_{13} - 1) = 2\lfloor (f_5 - 1)/2 \rfloor$
	Fin tant que ;		
	//Partie 3: Sertie des résultats		
19	k = 1 ;	→	$f_{19} = 1$
20	Tant que (k <= ch) faire	→	$f_{20} = f_5$
	Début		
21	écrire(tab[k]) ;	→	$f_{21} = f_{20} - 1 = f_5 - 1$
22	k = k + 1 ;	→	$f_{22} = 2(f_{20} - 1) = 2(f_5 - 1)$
	Fin tant que ;		
Fin ; //fin de l'algorithme Déchiffrer_nombre			

Figure 3. Tableau des fréquences d'exécutions des instructions de l'algorithme Déchiffrer_nombre.

Commentaire explicatif :

Rappel : Le nombre de chiffres p d'un nombre $n \geq 1$ écrit en base 10 vérifie la relation suivante :

$$\boldsymbol{p = \lfloor log_{10}(n) \rfloor + 1}$$

//$\lfloor x \rfloor$ désigne la partie entière inférieure du nombre réel x

1- L'instruction de répétition n°5 : tant que (m>0)

Il faut analyser les variations de la variable m qui joue ici le rôle de contrôleur (et pas seulement de compteur) de l'instruction de répétition n°5 : **tant que (m>0)** (en fond de couleur jaune dans le tableau des fréquences d'exécution), comme suit :

Les valeurs successives de m ($m = n$) sont :

$$\boldsymbol{m\ div\ 10^0 = m \rightarrow m\ div\ 10^1 \rightarrow m\ div\ 10^2\ \ldots\ \rightarrow m\ div\ 10^{ch} = 0}$$

//la dernière valeur 0 permet l'arrêt de l'instruction de répétition n°5

où,

$ch = \lfloor log_{10}(m) \rfloor + 1 = \lfloor log_{10}(n) \rfloor + 1$ = nombre de chiffres du nombre n

Il en résulte :

m prend $(ch - 0 + 1) = \left(\left(\lfloor log_{10}(m) \rfloor + 1 \right) - 0 \right) + 1$

$= \lfloor log_{10}(m) \rfloor + 2 = \lfloor log_{10}(n) \rfloor + 2$ valeurs

$\Rightarrow \boldsymbol{f_5(n) = \lfloor log_{10}(n) \rfloor + 2}$

Exemple : On reprend l'exemple précédent $n = 2021$.

On a :

$$\boldsymbol{2021\ div\ 10^0 = 2021 \rightarrow 2021\ div\ 10^1 = 202 \rightarrow 2021\ div\ 10^2 = 20}$$

$$\boldsymbol{\rightarrow 2021\ div\ 10^3 = 2 \rightarrow 2021\ div\ 10^4 = 0}$$

où,

$\boldsymbol{4 = \lfloor log_{10}(2021) \rfloor + 1 = \lfloor log_{10}(2021) \rfloor + 1 = \lfloor 3.3056 \rfloor + 1 = 3 + 1}$

//4 est bien le nombre de chiffres du nombre $n = 2021$

On a aussi :

$f_5(2021) = \boldsymbol{\lfloor log_{10}(2021) \rfloor + 2 = 3 + 2 = 5}$

2- L'instruction de répétition n°13 : tant que (j>k)

On a (d'après l'algorithme, instruction n°11 : j = ch) :

//on rappelle que ch = nombre de chiffres du nombre n

$\boldsymbol{j = ch = \lfloor log_{10}(n) \rfloor + 1}$

Comme k est initialisé à 1, on déduit aisément :

$$f_{13}(n) = \left\lfloor \frac{ch}{2} \right\rfloor + 1 = \left\lfloor \frac{\lfloor log_{10}(n) \rfloor + 1}{2} \right\rfloor + \mathbf{1} = \left\lfloor \frac{(\lfloor log_{10}(n) \rfloor + 2) - 1}{2} \right\rfloor + \mathbf{1}$$

$$\Rightarrow f_{13}(n) = \left\lfloor \frac{f_5(n) - 1}{2} \right\rfloor + \mathbf{1}$$

3- L'instruction de répétition n°20 : tant que (k<=j)

Comme k est initialisé à 1, on déduit aisément que :

$$f_{20}(n) = ch + 1 = (\lfloor \boldsymbol{log_{10}(n)} \rfloor + \mathbf{1}) + \mathbf{1} = (\lfloor \boldsymbol{log_{10}(n)} \rfloor + \mathbf{2})$$

$$\Rightarrow f_{20}(n) = f_5(n)$$

Le nombre d'instructions exécutées nbx par l'algorithme est :

$$nbx = \sum_{i=1}^{L=20} (f_i)$$

$$\Rightarrow nbx(n) = f_1(n) + \cdots + f_{22}(n)$$

$$\Rightarrow \boldsymbol{nbx(n) = \cdots = 12 f_5(n) + 8 \left\lfloor \frac{f_5(n) - 1}{2} \right\rfloor}$$

$$\Rightarrow \boldsymbol{nbx(n) = \cdots = 12 \lfloor log_{10}(n) \rfloor + 8 \left\lfloor \frac{\lfloor log_{10}(n) \rfloor + 1}{2} \right\rfloor + 24 \quad \forall n \geq 1}$$

Rem : En appliquant quelques valeurs n on a :

- $nbx(2) = 24$; //cas d'un nombre n à 1 chiffre
- $nbx(20) = 44$; //cas d'un nombre n à 2 chiffre
- $nbx(201) = 56$; //cas d'un nombre n à 3 chiffre
- $nbx(2021) = 76$. //cas d'un nombre n à 4 chiffre

On peut vérifier que les valeurs de cette fonction $nbx(n)$ sont celles que l'on retrouve en déroulant à la main l'algorithme.

2.c- Calculer la complexité temporelle, notée $ct(n)$, en notations exacte et asymptotique en O (Grand O), de cet algorithme.

Solution :

On a :

a- En notation exacte :

On a :

$$ct(n) = nbx(n)$$

$$\Rightarrow \boldsymbol{ct(n) = 12\lfloor log_{10}(n) \rfloor + 8\left\lfloor \frac{\lfloor log_{10}(n) \rfloor + 1}{2} \right\rfloor + 24 \quad \forall\, n \geq 1}$$

b- En notation asymptotique O :

On montre aisément :

$$\boldsymbol{ct(n) = O(log_{10}(n))}$$

En conséquence, la complexité temporelle de l'algorithme **Déchiffrer_nombre** est :

$$\boldsymbol{ct(n) = \begin{cases} 12\lfloor log_{10}(n) \rfloor + 8\left\lfloor \frac{\lfloor log_{10}(n) \rfloor + 1}{2} \right\rfloor + 24 \quad \forall\, n \geq 1 & \textit{en notation exacte} \\ O(log_{10}(n)) & \textit{en notation asymptotique} \end{cases}}$$

Exercice 3 (4 points/Q1=2, Q2=2)

1- On rappelle le théorème 1 sur les équations de récurrence linéaires homogènes à coefficients constants :

Théorème 1 : Etant donné une équation de récurrence linéaire homogène à coefficients constants d'ordre k définie comme suit :

$$\begin{cases} c_k t(n+k) + c_{k-1} t(n+k-1) + \ldots + c_0 t(n) = 0 \\ t(n_0) = d_0 \\ t(n_0+1) = d_1 \\ \quad \ldots \\ \quad \ldots \\ t(n_0+k-1) = d_{k-1} \end{cases} \qquad (1)$$

où,

- les paramètres d_i $(i = 0, \ldots, k-1)$ sont des constantes réelles définissant les conditions initiales de la récursion à partir de l'indice n_0 ;
- $t(n+k)$ désigne le terme qui détermine **l'ordre *k*** de l'équation (1).

Et, étant donné l'équation caractéristique de la relation (1) définie comme suit :

$$c_k r^k + c_{k-1} r^{k-1} + \cdots + c_1 r + c_0 = 0 \qquad (2)$$

où,

- le paramètre r désigne la variable inconnue de l'équation.

Alors, la solution générale de l'équation est de la forme suivante :

$$t(n) = \sum_{i=1}^{nb} \left[r_i^n . \left(\sum_{j=0}^{m_i - 1} a_{ij}\, n^j \right) \right] \qquad (3)$$

où,

- le paramètre $nb \leq k$ désigne le nombre de racines distinctes de l'équation caractéristique (3) ;
- le paramètre r_i désigne une racine de l'équation caractéristique (2) ;
- le paramètre m_i désigne la multiplicité de la racine r_i ;
- les coefficients a_{ij} sont des constantes qui sont déterminées à partir des conditions initiales.

Question : Résoudre l'équation de récurrence linéaire suivante :

$$\begin{cases} t(n+2) - 2t(n+1) + t(n) = 0 \quad \forall n \geq 2 \\ t(0) = 2 \\ t(1) = 7 \end{cases}$$

Solution :

Rem : Cet exercice est traité dans la série TD n° 3 / Exercice 2

L'équation caractéristique de cette équation est comme suit :

$$r^2 - 2r + 1 = 0$$

On résout l'équation caractéristique :

$$r^2 - 2r + 1 = 0$$

$\Rightarrow \ (r-1)^2 = 0$

$\Rightarrow$ Il existe $nb = 1$ racine double $r_1 = 1$ de multiplicité $m_1 = 2$.

En conséquence, la solution générale de l'équation de récurrence, par application du théorème 1 (théorème des équations de récurrence linéaires homogènes et à coefficients constants), est comme suit :

$$t(n) = \sum_{i=1}^{nb} \left[r_i^n . \left(\sum_{j=0}^{m_i - 1} a_{ij}\ n^j \right) \right]$$

$$\Rightarrow t(n) = \sum_{i=1}^{1} \left[r_i^n . \left(\sum_{j=0}^{m_i - 1} a_{ij}\ n^j \right) \right]$$

$$\Rightarrow t(n) = [r_1^n . \left(\sum_{j=0}^{m_1 - 1} a_{1j}\ n^j \right)]$$

$$\Rightarrow t(n) = \left[r_1^n . \left(\sum_{j=0}^{2-1} a_{1j}\ n^j \right) \right] \qquad //m_1 = 2$$

$$\Rightarrow t(n) = \left[r_1^n . \left(\sum_{j=0}^{1} a_{1j}\ n^j \right) \right]$$

$$\Rightarrow t(n) = [\ r_1^n (a_{10} . n^0 + a_{11} . n^1)$$

$$\Rightarrow t(n) = [\ r_1^n (a_{10} + a_{11} . n)\ ]$$

$$\Rightarrow t(n) = 1^n\ (a_{10} + a_{11} . n) \qquad //r_1 = 1$$

$$\Rightarrow t(n) = a_{10} + n . a_{11} \qquad (3.1)$$

Les constantes a_{10} et a_{11} sont déduites à partir des conditions initiales de l'équation (2) comme suit :

$$\begin{cases} t(0) = a_{10} + 0 . a_{11} = 2 \\ t(1) = a_{10} + 1 . a_{11} = 7 \end{cases}$$

$$\Rightarrow \begin{cases} a_{10} = 2 \\ a_{10} + a_{11} = 7 \end{cases}$$

$$\Rightarrow \begin{cases} a_{10} = 2 \\ 2 + a_{11} = 7 \end{cases}$$

$$\Rightarrow \begin{cases} a_{10} = 2 \\ a_{11} = 7 - 2 = 5 \end{cases}$$

$$\Rightarrow \begin{cases} a_{10} = 2 \\ a_{11} = 5 \end{cases}$$

En remplaçant ces 2 constantes dans (3.1), on a la solution finale de l'équation de récurrence :

$$t(n) = 2 + n.5 \quad \forall\, n \geq 0$$

$$\Rightarrow t(n) = 5n + 2 \quad \forall\, n \geq 0 \qquad (3.2)$$

2- On rappelle le théorème 2 sur les équations de récurrence non-linéaires à coefficients constants de type **"diviser et régner"** :

Théorème 2 (Master Théorème) : Soit l'équation de récurrence non linéaire à coefficients constants de type **"diviser et régner"** définie comme suit :

$$\boldsymbol{t(n) = at\left(\frac{n}{b}\right) + f(n)}$$

où :

- $f(n)$ est une fonction numérique définie comme suit :
  $f : N \rightarrow R^+$ telle que : $f(n) = O(n^d)$;
- d est une constante réelle positive, $d \in R,\ d \geq 0$;
- a est une constante naturelle, $a \in N,\ a \geq 1$;
- b est une constante naturelle, $b \in N,\ b > 1$;

La solution de cette équation en notation asymptotique de Landau est comme suit :

$$t(n) = \begin{cases} O\left(n^{log_b(a)}\right) & si \quad a > b^d \\ O\left(n^d log_e(n)\right) & si \quad a = b^d \\ O(n^d) & si \quad a < b^d \end{cases} \qquad (4)$$

Question : Résoudre l'équation de récurrence non-linéaire suivante :

$$t(n) = t\left(\frac{n}{3}\right) + n \quad \forall n \geq 1$$

Solution :

Rem : Cet exercice est similaire à celui traité dans la série TD n° 3 / Exercice 10

On a :

$a = 1,\ b = 3,\ f(n) = n$ et $d = 1$

$\Rightarrow \begin{cases} a = 1 \\ b^d = 3^1 = 3 \end{cases} \Rightarrow a < b^d$ //Donc, on est dans le cas 3 : $a < b^d$

En conséquence, la solution générale de l'équation de récurrence, par application du théorème 2, est comme suit :

$t(n) = O(n^d)$

$\Rightarrow t(n) = O(n^1) = O(n)$