Université Hassan II de Casablanca
École Nationale Supérieur des Arts et Métiers
Matière : Algorithme Avancée

TDP 1
Rappels

Partie 01 : TD

Exercice 1: complexité

1. Prouver par récurrence que :
   a. $1 + 2 + \ldots + n = n\,(n+1)\,/2$
   b. $1^2 + 2^2 + \ldots + n^2 = n\,(n+1)\,(2n+1)\,/\,6$

2. Donnez la complexité en O() des programmes suivants (Borne supérieure).

- ```
  Pour i allant de 1 à n faire
      Pour j allant de 1 à n faire
          x ← x+3
      FinPour
  FinPour
  ```
- ```
  Pour i allant de 1 à n faire
      Pour j allant de 1 à n faire
           Pour k allant de 1 à n faire
                x ← x+4
           FinPour
      FinPour
  FinPour
  ```
- ```
  Pour i allant de 5 à n-5 faire
      Pour j allant de i-5 à i+5 faire
          x ← x+2
      FinPour
  FinPour
  ```
- ```
  i ← n
  TantQue i > 1 faire
   x ← x+a
   i ← i/2
  FinTantQue
  ```
- ```
  Pour i allant de 1 à n faire
      Pour j allant de 1 à i faire
          x ← x+3
      FinPour
  FinPour
  ```
- ```
  Pour i allant de 1 à n faire
      Pour j allant de 1 à i faire
            Pour k allant de 1 à j faire
                 x ← x+4
            FinPour
      FinPour
  FinPour
  ```

Exercice 2: Preuve d'algorithme

On considère l'algorithme suivant :

```
Données : un entier naturel a et un entier b
Résultat : un entier p
p ← 0
m ← 0
Tant que m < a Faire
      p ← p + b
      m ← m + 1
Fin Tant que
```

1. Donner la valeur finale de p lorsque $(a\ ;\ b) = (3\ ;\ 5)$ et $(a\ ;\ b) = (4\ ;-3)$. Que fait cet algorithme ?
2. Justifier que cet algorithme se termine. Quelle est la valeur de m à la fin de la boucle ?
3. Vérifier que «$p = mb$» est un invariant de boucle.

Exercice 3 : Preuve d'algorithme [5 points] [~ 25 minutes]

On considère l'algorithme suivant :

```
Entrées : un entier naturel x et un entier naturel y
Résultat : un nombre r
Variables r,s,t,w : Entier
Début
r ← 0
s ← x
Tant que s > 0 faire
  r ← r + y
  s ← s - 1
FinTantQue
w ← x - 1
```

```
t ← r
Tant que w > 0 faire
  r ← r + t
  w ← w - 1
FinTantQue
Fin
```

1. Calculer la valeur finale de r lorsque (x ; y) = (3; 2) et (x ; y) = (4; 3).
2. Justifier que l'algorithme se termine en précisant le variant de la première et la deuxième boucle TantQue.
3. Démontrer par récurrence que :
   - "$r_i = (x - s_i)y$" est un invariant de la première boucle TantQue.
   - "$r_i = (x - w_i)xy$" est un invariant de la deuxième boucle TantQue.
4. Que fait cet algorithme (valeur de la variable r à la fin) ?

Partie 02 : TP

- **Fonctions** (solutions en langage C, puis Python)

Exercice 4 :

1. Soit T un tableau d'entier de taille N. Ecrire une fonction qui copie toutes les composantes strictement positives de T dans un tableau TPOS et toutes les valeurs strictement négatives dans un tableau TNEG.
   Indication : les tableaux TPOS et TNEG sont passés comme paramètres de la fonction.

2. Somme des chiffres
   a. Ecrire une fonction itérative **Som_Chiff** qui calcule la somme des chiffres d'un entier
      Exemple : somme chiffres 194 est 14
   b. Ecrire la foncrion **Som_Chiff_rec** la version récursive de Som_Chiff

3. Écrire une fonction **Decal_lettre_chaine** qui décale de n positions les lettres d'une chaine de caractère **ch** de longueur **L (on suppose que n < L)**. Le résultat est rangé dans une chaine CHD.
   L'entete est:
   `Fonction Decal_lettre_chaine(ch[]:caracatere,L:Entier,n:Entier,CHD[]:caractere):vide`
   Exemple: si ch = "Rattrapage" et n=2 alors CHD = "geRattrapa"

4. Écrire une fonction récursive qui calcule le nombre d'occurrence d'un élément e dans une liste L

5. Suppression d'espaces
   a. Ecrire une fonction SED qui supprime les espaces au début d'une chaine de caractères (s'ils existent)
   b. Ecrire une fonction SEDR la version récursive de SED
   c. Ecrire une fonction SEF qui supprime les espaces à la fin d'une chaine de caractères (s'ils existent)
   d. En déduire une fonction SEDF qui supprime les espaces au début et à la fin d'une chaine de caractères (s'ils existent)

6. Écrire une fonction récursive qui vérifie si une chaine de caractère est palindrome ou pas

- **Matrices** :

Exercice 5: [3 points][~ 15 minutes] (Examen 2020)
Le triangle de Pascal est une matrice triangulaire inférieure telle que :

- contient `1` sur la diagonale,
- contient `1` sur le premier élément de chaque ligne
- les autres éléments sont construits comme indiqué sur l'exemple ci-dessous :

Exemple du TRIANGLE DE PASCAL de degré 5 :

N=0	1					
N=1	1	1				
N=2	1	2	1			
N=3	1	3	3	1		
N=4	1	4	6	4	1	
N=5	1	5	10	10	5	1

Écrire une fonction **TrPascal** qui prend en paramètre un entier N et une matrice triangulaire inférieure MP de dimension N+1 et retourne 1 si MP représente un TRIANGLE DE PASCAL de degré N et 0 sinon.

Exercice 6 :

a. Une matrice **Antisymétrique** est une matrice carrée opposée à sa transposée, càd $A^\top = -A$

Exemples : $\begin{pmatrix} 0 & 4 \\ -4 & 0 \end{pmatrix}$ ou $\begin{pmatrix} 0 & 2 & -5 \\ -2 & 0 & 9 \\ 5 & -9 & 0 \end{pmatrix}$

- Écrire une fonction **ANTISYM** qui prend en paramètre une matrice carré **M** de taille **n**x**n** et retourne 1 si la matrice M est antisymétrique et 0 sinon. (Faire le parcours nécessaire élément par élément de la matrice)
- Proposer une solution plus simple en utilisant les fonctionnalités du langage Python

b. Une matrice **diagonale** est une matrice carrée dont les coefficients en dehors de la diagonale principale sont nuls.
Ecrire une fonction qui retourne si une matrice carrée M est diagonale ou pas

c. Une matrice de **Hankel** est une matrice carrée dont les valeurs le long des diagonales ascendantes de gauche à droite sont constantes.

Ex : $\begin{pmatrix} 3 & 7 & -1 & 2 & 6 \\ 7 & -1 & 2 & 6 & 18 \\ -1 & 2 & 6 & 18 & 4 \\ 2 & 6 & 18 & 4 & -5 \\ 6 & 18 & 4 & -5 & 125 \end{pmatrix}$

Ecrire une fonction **HANKEL**() qui retourne 1 si une matrice carrée M de taille nxn est de Hankel, sinon retourne 0.

<u>Exercice 7 :</u>

Voici un algorithme qui réalise la multiplication de 2 nombres entiers naturels N et M en n'utilisant que des multiplications/divisions par 2 selon la conception suivante :

À chaque étape, N est divisé par 2 (division entière) et M est multiplié par 2. Si N est impair, la valeur de M est ajoutée au futur résultat. Si N est strictement positif, on s'arrête à N = 1.

<u>Exemple:</u> 321 * 457

```
N          M
321       457      N est impair donc futur résultat=457
160       914      N est pair donc on n'ajoute pas 914
 80      1828      N est pair donc on n'ajoute pas 1828
 40      3656      N est pair donc on n'ajoute pas 3656
 20      7312      N est pair donc on n'ajoute pas 7312
 10     14624      N est pair donc on n'ajoute pas 14624
  5     29248      N est impair donc futur résultat = 457 + 29248 = 29705
  2     58496      N est pair donc on n'ajoute pas 58496
  1    116992      N est impair donc résultat = 29705 + 116992 = 146697
```

1. **<u>En justifiant votre réponse</u>**, calculer le produit N*M en utilisant l'algorithme de multiplication par shifting ci-dessus pour les cas suivants : (une réponse non justifiée est fausse)
    - N = 21 , M = 5
    - N = 107, M = 3
    - N = 5 , M = 21
2. Ecrire une fonction itérative **prod_shift_iter** qui prend en paramètres deux entiers, respectivement, N et M et retourne leur produit. La fonction doit prendre en considération qu'un des deux paramètres peut être nul.
   N.B. Attention aux initialisations dans la fonction.
3. Quelle est la complexité de cette solution (nombre de divisions) ? Justifier votre réponse.
4. Ecrire une fonction **prod_shift_rec** la version récursive de la fonction **prod_shift_iter.**
5. Ecrire un programme qui lit deux entiers positifs N et M puis, pour calculer leur produit, fait appel à la fonction itérative **prod_shift_iter** de sorte à ce que le nombre d'itérations soit **<u>minimal</u>**.

<u>Exercice 8:</u>

On considère qu'un texte est une chaine de caractère suffisament longue. Soient les suppositions suivantes:

- Un texte peut être vide.
- Si un texte n'est pas vide, il est composé d'une ou plusieurs phrases.
- Une phrase commence par une lettre ou un chiffre et se termine par un point. Il n'y a pas d'espace avant le point.
- Les phrases d'un texte sont séparées par un espace.
- Les mots d'une phrase sont séparés par un espace.

a. Écrire une fonction **NPT** qui retourne le nombre de phrases d'un texte.

b. Écrire une fonction **NMT** qui retourne le nombre de mots d'un texte.

c. Écrire la fonction **Txt_Tab** qui met les phrases du texte **TX** dans **TCH[]**, un tableau de chaines de caractères**.** Les chaines de caractères, éléments du tableau TCH[], **<u>n'ont pas nécessairement la même longueur</u>.** La fonction Txt_Tab doit avoir l'entête suivant:

```
Fonction Txt_Tab(TX : chaine de caractères, TCH[] : chaines de caractères) : vide
```

<u>Exemple 3:</u> si TX = "Aujourd'hui c'est l'examen d'Algorithmique. Bonne chance. Chacun pour soi et Dieu pour tous." alors il faut que le tableau TCH soit tel que :
TCH[0] = "Aujourd'hui c'est l'examen d'Algorithmique."
TCH[1] = "Bonne chance."

TCH[2] = “Chacun pour soi et Dieu pour tous.”

Indication: On accepte l’affectation d’une chaine de caractèrs à une autre, i.e ch1 ← ch2

d. En déduire une fonction **Inv_Txt** qui inverse l’ordre des phrases d’un texte.
Exemple 4: le text TX dans l’exemple 3 deviendra TX = “Chacun pour soi et Dieu pour tous. Bonne chance. Aujourd’hui c’est l’examen d’Algorithmique.”