USTHB - FEI - Département D'Informatique
Master SSI **Bab-Ezzouar 04 Janvier 2017**

Epreuve finale : Algorithmique avancée et Complexité

Exercice 1 : (08 pts)

Considérons la structure d'un arbre binaire de Recherche équilibré. Il s'agit tout d'abord d'un arbre binaire ordonné tel que pour tout nœud x : Toutes les valeurs des nœuds y du sous arbre gauche de x doivent être inférieure à la valeur du nœud x, et toutes les valeurs des nœuds y du sous arbre droit de x doivent être supérieure à la valeur du nœud x. L'arbre est en plus dit équilibré si toute les feuilles de l'arbre se trouvent à la même profondeur ou ont au maximum un écart d'un seul niveau uniquement. L'objectif est d'implémenter cette structure sous forme d'un tableau en s'inspirant du principe de construction d'un Tas.

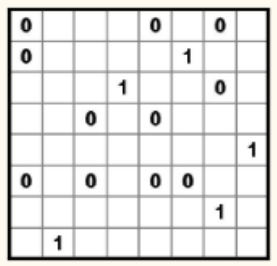
1. Illustrer les étapes de construction de l'arbre binaire de recherche équilibré sur l'exemple suivant **(4 pts)**

16	14	10	8	7	9	3	2	4	1

2. Proposer un algorithme qui implémente cette structure et calculer la complexité de l'algorithme proposé. **(4 pts)**

Exercice 2 : (12 pts)

Considérons le jeu dit Binairo (plus connu actuellement sous le nom « *Takuzu* »). C'est un jeu de réflexion basé sur la logique du Sudoku. Il s'agit d'une grille allant de 6x6 à 14x14 cases en général, et pouvant très bien avoir un nombre de lignes et de colonnes différent ou impair. Les cases de la grille ne supportent que des valeurs binaires. La grille comporte au tout début certaine valeur par défaut comme le montre l'exemple, et le but du jeu et de compléter la grille en respectant les trois règles suivantes :



0	0	1	1	0	1	0	1
0	1	1	0	0	1	1	0
1	0	0	1	1	0	0	1
1	1	0	1	0	0	1	0
0	0	1	0	1	1	0	1
0	1	0	1	0	0	1	1
1	0	1	0	1	0	1	0
1	1	0	0	1	1	0	0

1) Le nombre de 1 (Resp. de 0) est le même sur chaque ligne et sur chaque colonne
2) La grille ne doit pas comporter plus de deux chiffres identiques cote à cote.
3) Deux lignes ou deux colonnes ne peuvent être identiques.

Formellement, le problème peut être défini comme suit :

- Soit en entrée la grille Binairo de dimension (NxM) initialisée avec des valeurs par défaut à certaines positions.
- Existe-t-il un remplissage de la matrice en respectant les règles du jeu ?

Questions :

1. Décrire les étapes de construction d'une solution en spécifiant la modélisation la plus adéquate **(5 pts)**
2. Estimer approximativement la taille de l'arbre de résolution et en déduire l'ordre de complexité de l'algorithme **(2 pts)**
3. Quels sont les critères que doit satisfaire une solution donnée S' pour être valide **(1 pt)**
4. Proposer un algorithme de validation et calculer sa complexité. **(4 pts).**

Bon Courage !

Solution :

Exercice 1 :

1. Illustration :

Plusieurs manières de procéder (par construction des minimums ou maximums) ou pas validation de la relation l'ordre progressivement comme pour le tas.

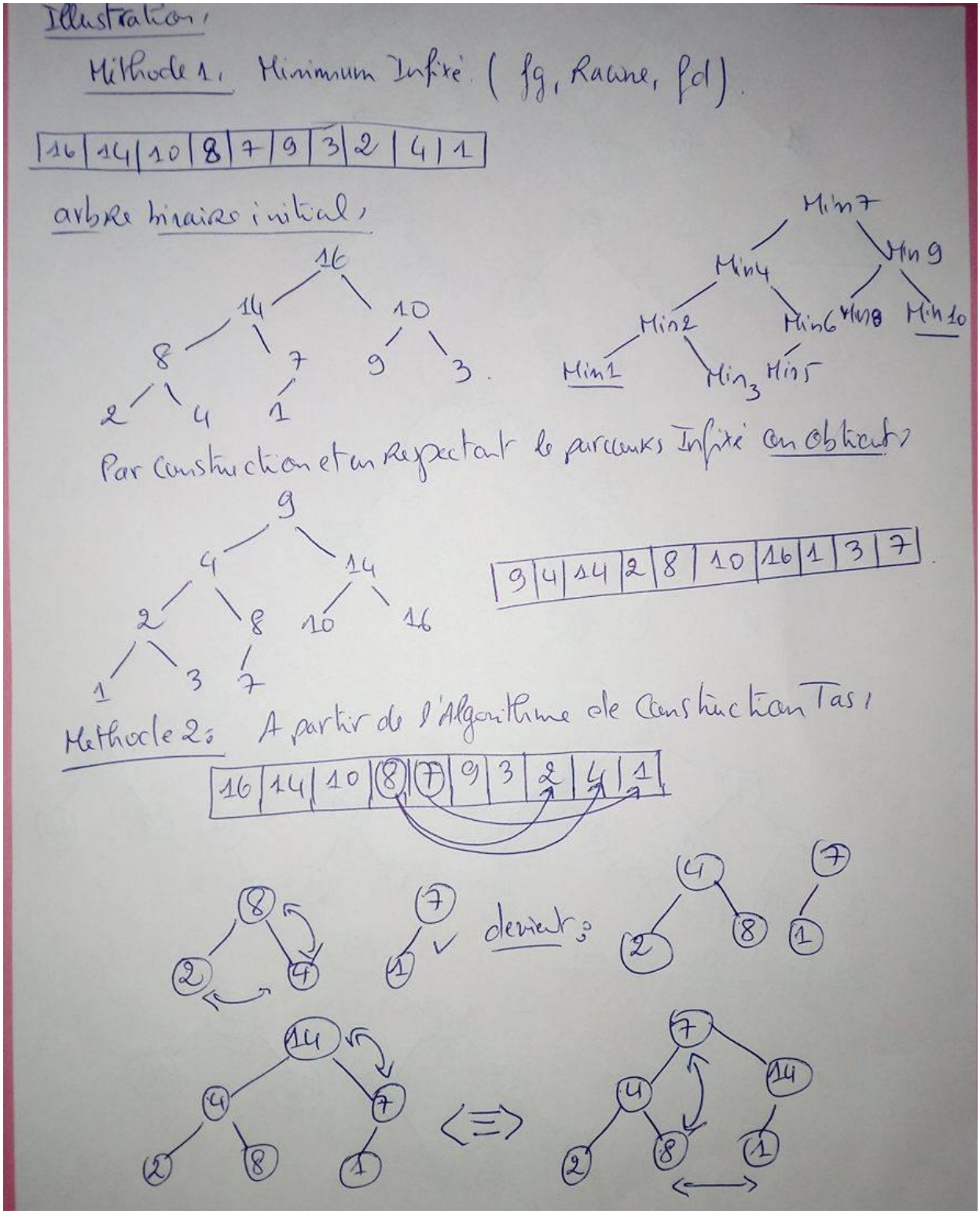
Par construction des minimums (Resp. maximum) consiste à construire l'arbre de manière infixé gauche (Resp. droit) comme le montre le déroulement qui suit.

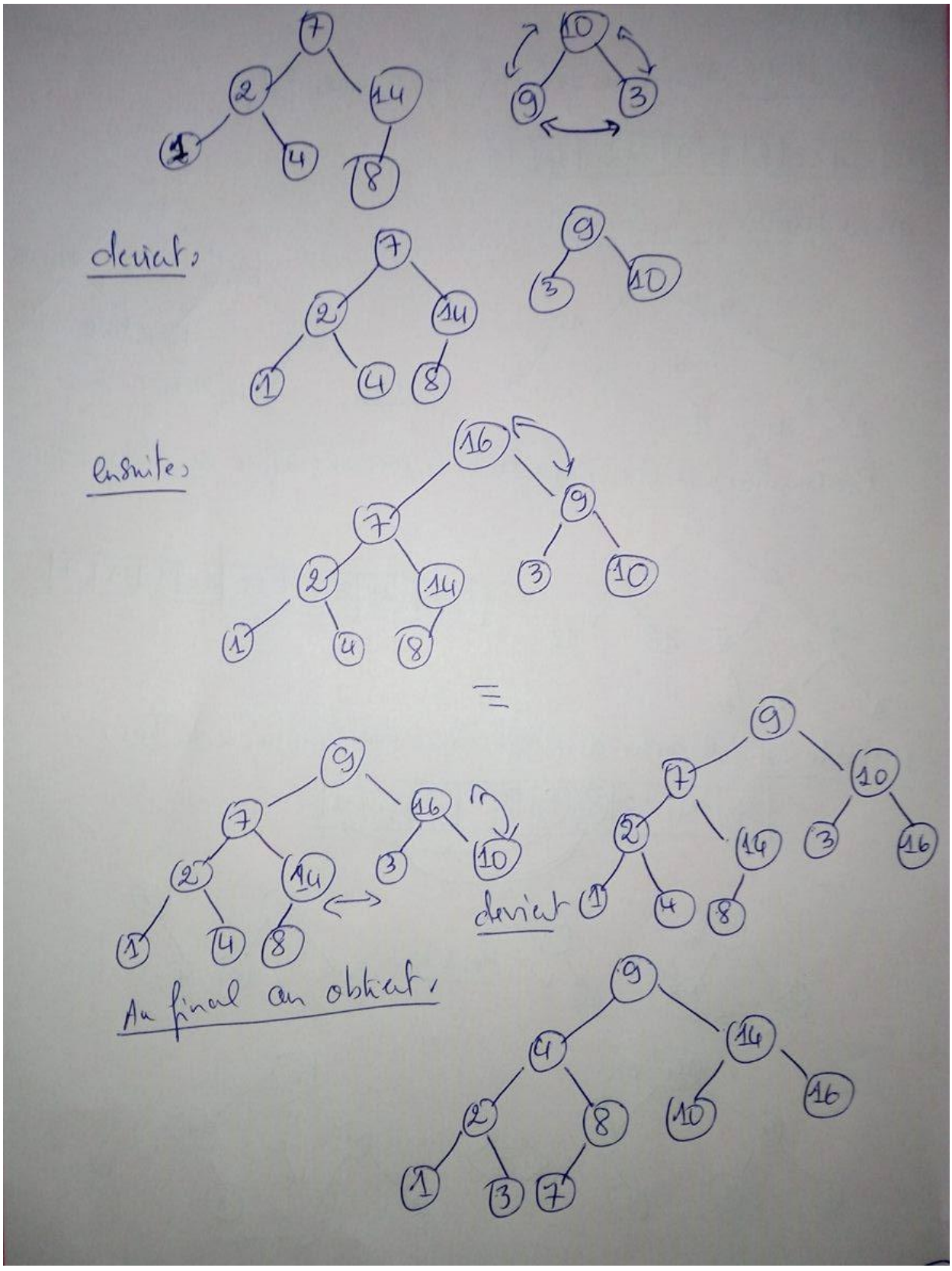
Une construction basée sur le tas consiste à vérifier progressivement la relation d'ordre totale entre tous les nœuds de l'arbre à chaque étape. En plus de la vérification locale (nœud courant, fg et fd), une vérification supplémentaire entre les descendants immédiat entre deux nœuds frères, autrement dit un fils gauche e1 devra être comparé avec le fils gauche de son frère e2 puisque e1 < (e2->fg) < e2 ; et un fils droit (noté ici e2) devra être comparé avec le fils droit de son frère gauche, puisque e2 < (e1->fd) < e1.

2. Algorithme

```
Procédure construction(n) ;
début
        pour i := n/2 à 1 par pas=-1 faire (* prendre la partie entière de n/2 *)
                Tamiser (i, n) ;
        fait;
Fin.
procédure Tamiser (i, j) ;
début
imax = 0 ;
 si (2*i) <= j (* A[i] n'est pas une feuille *)
        alors si (2*i+1) <= j alors
        si (A[2*i] > A[i] ou A[2*i+1] < A[i]) alors
        si (A[2*i] > A[i] ) alors
                                si (A[2*i] < A[2*i+1]) ou (A[2*i+1] > A[i]) alors imax = 2*i ;
                                sinon temp = A[2*i] ; A[2*i] = A[2*i+1] ; A[2*i+1] = temp ;
                                fsi ;
        sinon si (A[2*i+1] > A[2*i]) alors imax = 2*i+1 ;
                sinon temp = A[2*i] ; A[2*i] = A[2*i+1] ; A[2*i+1] = temp ; imax = 2*i+1 ;
                fsi ;
        fsi ;
        si imax <> 0 alors  temp = A[i];  A[i] = A[imax];  A[imax] = temp;
                                Tamiser (imax, j);  fsi;
fsi;
fin ;
```

Complexité O(N*log N), N étant la dimension du table puisque au pire cas, les N éléments du tableau nécessite un tamisage à chaque étape, et le processus de tamisage est logarithmique puisqu'une seule branche de l'arbre sera mise à jour à la fois.





Exercice 2 :

1. Modélisation et étapes de construction :

En entrée : Une grille binaire contenant certaines valeurs par défaut

Implémentation : Une matrice binaire de dimension NxM, initialement vide sauf pour les quelques cases contenant des valeurs par défaut.

En sortie : Une grille remplis en respectant les contraintes du problème.

Implémentation : La matrice binaire de dimension NxM entièrement remplis en maintenant les valeurs par défaut et en respectant les contraintes de remplissage

Traitement : Essayez toute les combinaisons possibles en procédant case par case, de manière verticale, de manière horizontale, ou quelconque. A chaque étape, une seule case sera remplis à la fois avec une des deux combinaisons possible (0 ou 1). Dès qu'une des conditions de remplissage ne peut être satisfaite un retour arrière est appliqué pour considérer une autre combinaison possible.

Les valeurs par défaut seront maintenu par construction, c-à-d, qu'une case contenant une valeur par défaut ne devra jamais être altérer par un retour arrière. Pour ce faire, il suffit d'avoir une liste pour enregistrer les positions (i,j) des cases en question pour que leur valeurs ne puissent jamais changer.

2. Si on note par K le nombre de valeurs par défaut initialement présentent dans la grille, le nombre de cases restantes est estimé à (N*M) – K ; Puisque la matrice est binaire, le nombre de possibilité dans chaque case est égal à 2. L'arbre de résolution est donc binaire et la taille de l'arbre est bornée par $2^{(N*M)-K}$.

Ainsi, l'ordre de complexité de l'algorithme de résolution est O ($2^{(N*M)-K}$).

3. Les critères que doit satisfaire une solution S' :
    a. Matrice binaire de dimension NxM
    b. Valeurs initiales maintenues
    c. Nombre de 1 (Resp. de 0) est le même sur toute les lignes
    d. Nombre de 1 (Resp. de 0) est le même sur toute les colonnes
    e. La grille de doit pas comporter sur de deux cases identiques cote à cote
    f. Toutes les lignes et toutes les colonnes de la grille doivent être distinctes.
4. Algorithme de validation :

```
Validation_Binairo (Solution S') : Bouléen ;
Début
i, j, k, seq, nb1_ligne, nb1_col : entier ;
seq =0 ; nb1_ligne = 0 ; nb1_col = 0 ;
i=1 ; j=1 ;
bouléen valide = vrai ;
bouléen egal = vrai ;
//toutes les lignes et les colonnes sont distinctes
Tantque (i <= N et valide = vrai)
    Faire k = i+1 ;
        tant que (K<=N et valide = vrai)
```

```
                Faire j = 1 ;
                Tant que (j <=M et egal =vrai)
                Faire si S'[i, j] = S'[k, j] alors j= j+1 ;
                        Sinon egal = faux ;     Fsi ;
                Fait ;
                Si egal = vrai alors valide = faux ;
                                Sinon k = k+1 ;
                Fsi ;
        I = i +1 ; egal = vrai ;
Fait ;
j = 1 ;
    Tantque (j <= M et valide = vrai)
        Faire k = j+1 ;
            tant que (k<=M et valide = vrai)
                Faire i = 1 ;
                Tant que (i <=N et egal =vrai)
                Faire si S'[i, j] = S'[i, k] alors i= i+1 ;
                        Sinon egal = faux ;
                        Fsi ;
                Fait ;
                Si egal = vrai alors valide = faux ;
                                Sinon k = k+1 ;
                Fsi ;
        j = j +1 ; egal = vrai ;
Fait ;
// Max(O(N*M²), O(M*N²))

//pas plus de deux nombres identiques cote à cote
i=1 ;
Tant que (i<=N-2)
Faire j =1 ;
        Tant que (j<= M-2)
        Faire si S'[i, j] = S'[i, j+1] et S'[i, j+1]= S'[i, j+2] alors valide = faux
                Sinon j = j+1 ; fsi ;
        Fait ;
i= i+1 ;
fait ;
j=1 ;
Tant que (j<=M-2)
Faire i = 1 ;
        Tant que (i<= N-2)
        Faire si S'[i, j] = S'[i+1, j] et S'[i+1, j]= S'[i+2, j] alors valide = faux
                Sinon j = j+1 ; fsi ;
        Fait ;
```

```
i= i+1 ;
fait ;
//O(N*M)

//Nombre de 1 est identique sur chaque ligne et sur chaque colonne
Pour i = 1 à N faire
        Si S'[i, 1] = 1 alors nb1_col = nb1_col +1 ; fsi ;
        Fait ;
Pour j = 1 à M faire Si S'[1, j] = 1 alors nb1_ligne = nb1_ligne +1 ; fsi ;
i=2 ;
Tant que (i<=N) et (valide = vrai)
Faire Pour j = 1 à M
                faire
                Si S'[i, j] = 1 alors nb1 = nb1 +1 ; fsi ;
                Fait
        Si nb1 <> nb1_col alors valide = faux ;
                Sinon i= i+1 ; nb1=0 ;
        Fsi ;
Fait ;
j=2 ;
Tant que (j<=N) et (valide = vrai)
Faire Pour i = 1 à M
                faire
                Si S'[i, j] = 1 alors nb1 = nb1 +1 ; fsi ;
                Fait
        Si nb1 <> nb1_ligne alors valide = faux ;
                Sinon i= i+1 ; nb1=0 ;
        Fsi ;
Fait ;
//O(N*M)

Retourner (valide) ;
Fin.
```

Au pire cas la solution est valide, et l'ordre de complexité dans le pire cas est Max($O(N*M^2)$, $O(M*N^2)$).