

**[CYBER1][2024-2025] Rattrapage**  
Algorithmique 2**NOM :** \_\_\_\_\_**PRÉNOM :** \_\_\_\_\_

Vous devez respecter les consignes suivantes, sous peine de 0 :

- |  |  |
|--|--|
| I) Lisez le sujet en entier avec attention | V) Écrivez lisiblement vos réponses (si nécessaire en majuscules)                              |
| II) Répondez sur le sujet                  |  |
| III) Ne trichez pas                        | VI) Vous devez écrire les algorithmes et structures en langage C (donc pas de Python ou autre) |
| IV) Ne détachez pas les agrafes du sujet   |  |

## 1 Arbres Binaires (8 points)

### Questions

#### 1.1 Dessinez un arbre répondant aux critères imposés (2 points)

##### 1.1.1 (0,5 point) Arbre binaire localement complet

*(hauteur minimale : 3)*

##### 1.1.2 (0,5 point) Arbre filiforme

*(hauteur minimale : 4)*

---

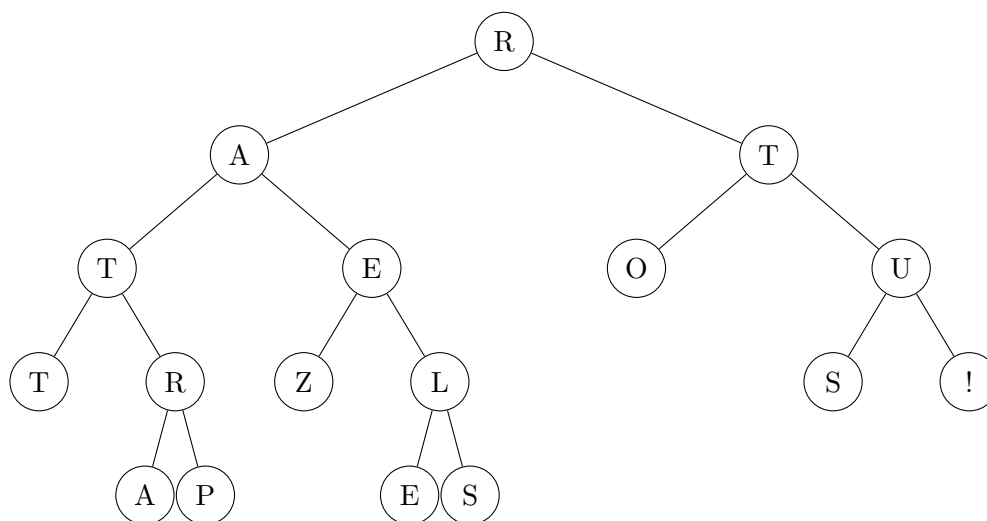
##### 1.1.3 (0,5 point) Arbre binaire parfait

*(hauteur minimale : 2)*

##### 1.1.4 (0,5 point) Arbre binaire presque complet

*(hauteur minimale : 3)*

## 1.2 Répondez aux différentes questions concernant l'arbre suivant (4 points)



### 1.2.1 (1,5 point) Indiquez toutes les propriétés que possède cet arbre :

Arité :

Taille :

Hauteur :

Nb feuilles :

☐

Arbre binaire strict / localement complet

☐

Arbre binaire (presque) complet

☐

Arbre binaire parfait

☐

Arbre filiforme

☐

Peigne gauche

☐

Peigne droit

### 1.2.2 (2 points) Écrivez les clés lors d'un parcours profondeur main gauche de l'arbre dans les 3 ordres ainsi que lors d'un parcours largeur :

Parcours profondeur :

ordre préfixe : \_ \_ \_ \_ \_

ordre infixé : \_ \_ \_ \_ \_

ordre suffixe : \_ \_ \_ \_ \_

Parcours largeur :

ordre : \_ \_ \_ \_ \_

### 1.2.3 (0,5 point) Indiquez la profondeur et le numéro hiérarchique des nœuds suivants :

	Profondeur	N° hiérarchique
L		

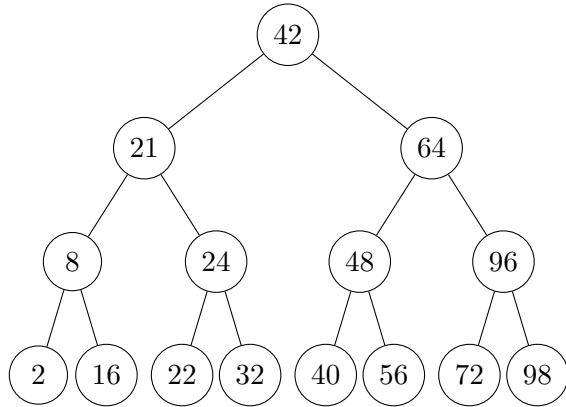
	Profondeur	N° hiérarchique
P		



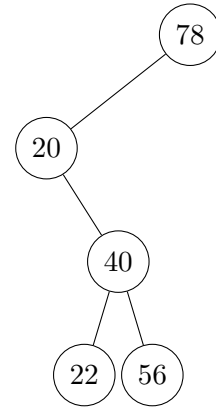
## 2 Arbres Binaires de Recherche (6 points)

### Questions

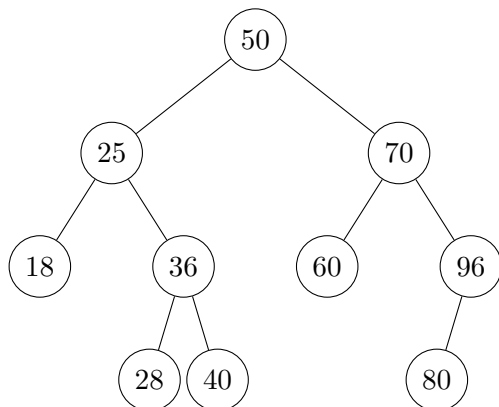
2.1 (1 point) Indiquez pour chacun des cas suivants s'il s'agit d'un arbre binaire de recherche ou non :



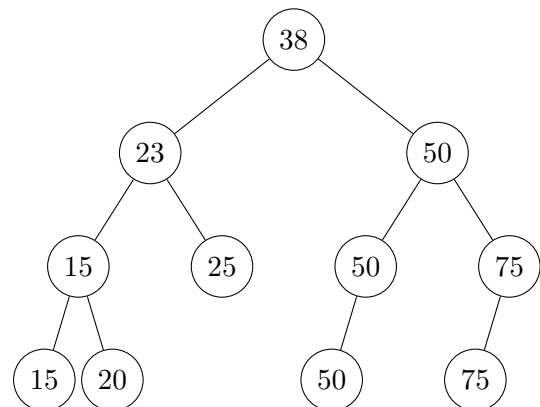
☐ Est un ABR    ☐ N'est pas un ABR



☐ Est un ABR    ☐ N'est pas un ABR



☐ Est un ABR    ☐ N'est pas un ABR



☐ Est un ABR    ☐ N'est pas un ABR

2.2 (3 points) Dessinez le résultat des opérations successives :

- *InsertLeaf*(node \*R, int val) insère l'élément « val » en feuille dans l'ABR « R »
- *InsertRoot*(node \*R, int val) insère l'élément « val » en racine dans l'ABR « R »
- *RemoveFromBST*(node \*R, int val) supprime l'élément « val » de l'ABR « R »

Étape 1  
R = InsertLeaf(NULL, 50)

Étape 2  
R = InsertLeaf(R, 30)

Étape 3  
R = InsertLeaf(R, 80)

*Étape 4*  
 $R = \text{InsertLeaf}(R, 20)$

*Étape 5*  
 $R = \text{InsertLeaf}(R, 60)$

*Étape 6*  
 $R = \text{InsertLeaf}(R, 40)$

---

*Étape 7*  
 $R = \text{InsertLeaf}(R, 25)$

*Étape 8*  
 $R = \text{InsertLeaf}(R, 70)$

---

*Étape 9*  
 $R = \text{RemoveFromBST}(R, 30)$

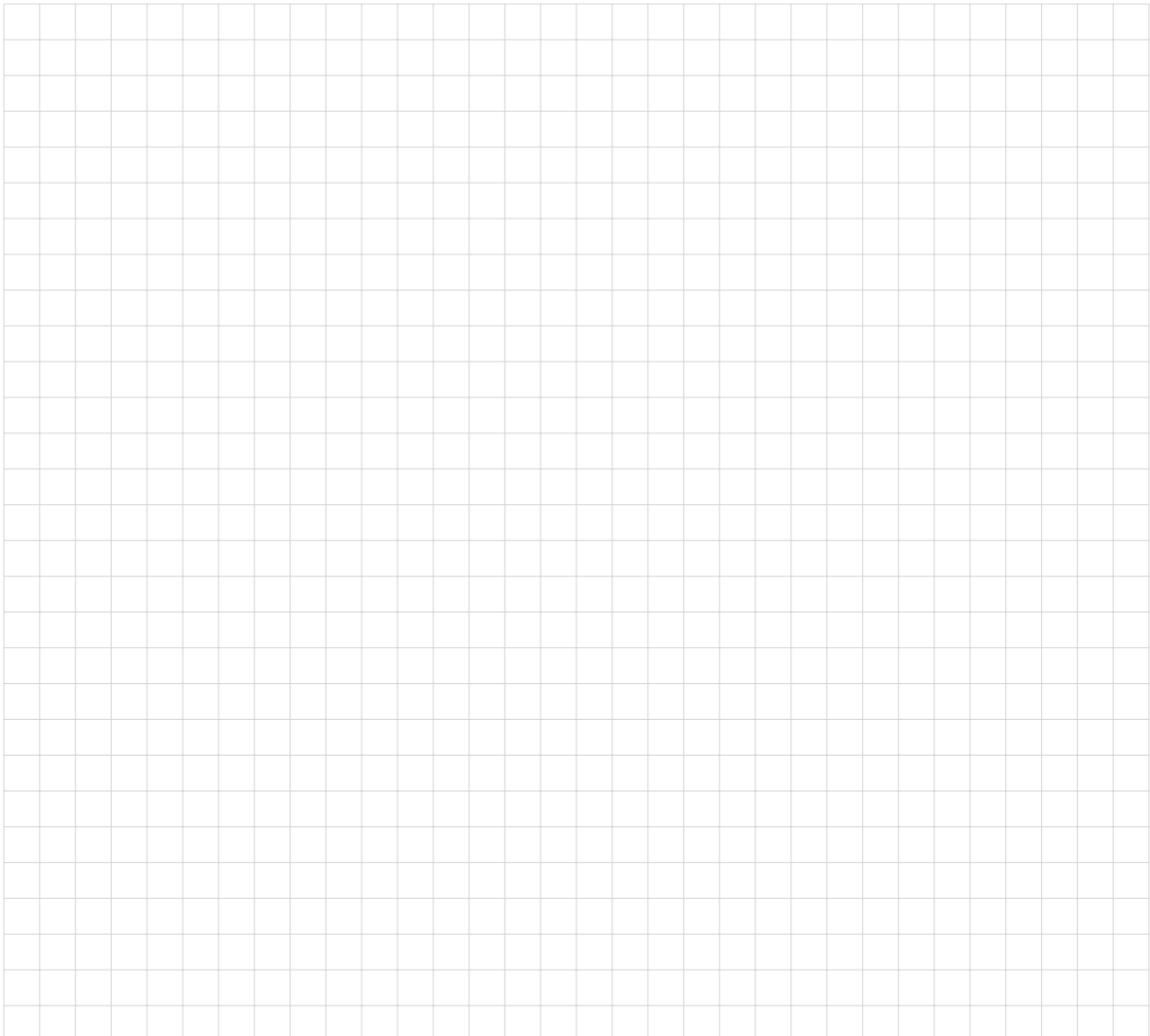
*Étape 10*  
 $R = \text{RemoveFromBST}(R, 80)$

*Étape 11* $R = \text{InsertRoot}(R, 30)$ *Étape 12* $R = \text{InsertRoot}(R, 35)$ 

---

## Algorithmes

**2.3 (2 points)** Écrivez une fonction *AjoutFeuille* ajoutant récursivement en feuille un élément dans un arbre binaire de recherche :



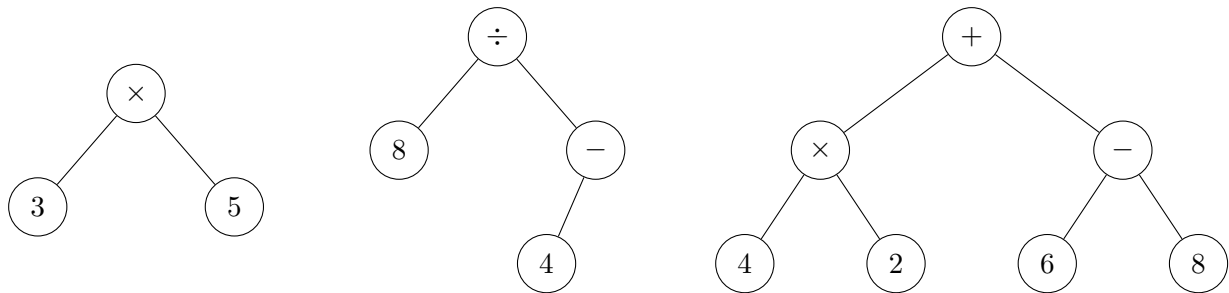
### 3 Problème (6 points)

En mathématiques, vous avez appris tout au long de vos études primaires et secondaires à utiliser plusieurs opérateurs communs : «  $+$  », «  $-$  », «  $\times$  », «  $\div$  ». Ces opérateurs sont tous des opérateurs *binaires*, c'est-à-dire qu'ils prennent deux paramètres : «  $3 + 5$  » peut également s'écrire *addition*(3, 5).

«  $+$  » et «  $-$  » sont également des opérateurs *unaires* lorsqu'ils représentent le signe de la valeur absolue qui suit : «  $-8$  » peut aussi s'écrire *negatif*(8).

Dans cet exercice, vous allez transformer des arbres en formules mathématiques, puis l'inverse, et enfin, écrire l'algorithme de parcours et d'exécution de ces opérations.

#### 3.1 (2 points) Transformez les arbres suivants en leurs formules mathématiques associées :



(0,5 pt) Formule :

(0,5 pt) Formule :

(0.5 pt) Formule :

(0.5 point) Que remarquez-vous concernant les opérateurs et les nombres par rapport à leur placement dans les arbres ?

#### 3.2 (2 points) Transformez chacune des formules suivantes en un arbre binaire :

(0,5 point)

$$3 \times (4 - 5)$$

(0,5 point)

$$(-5) \div (8 + 3)$$

(1 point)

$$(6 \times (-4)) + ((-8) \div (5 - 3))$$

### 3.3 (2 points) Écrivez une fonction « *exec\_maths\_tree* » exécutant l'expression représentée par l'arbre binaire donné en paramètre, et renvoyant le résultat :

Chaque nœud de cet arbre est une structure contenant du texte en tant que valeur, vous pouvez utiliser les fonctions suivantes pour tester ou extraire le contenu de la valeur. Les nœuds ne contiendront jamais autre chose que l'un des 4 opérateurs ou un nombre, et l'arbre ne sera jamais vide.

```
typedef struct math_node
{
    char *value;
    struct math_node lc;
    struct math_node rc;
} math_node;
```

**int UnaryOp(char \*op, int val)** : fonction exécutant l'opérateur fournit en paramètre sur la valeur.

**int BinaryOp(char \*op, int v1, int v2)** : fonction exécutant l'opérateur fournit en paramètre sur les 2 valeurs.

**int atoi(char \*text)** : fonction transformant le texte donné en paramètre en un entier.

```
int exec_maths_tree(math_node *root)
```