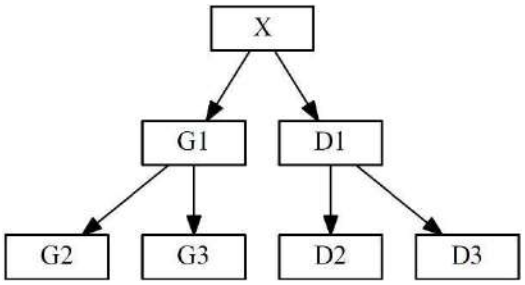
Exercice 1

Notion abordée : les arbres binaires de recherche.

Un arbre binaire est soit vide, soit un nœud qui a une valeur et au plus deux fils (le sous-arbre gauche et le sous-arbre droit).



X est un nœud, sa valeur est X.valeur

G1 est le fils gauche de X, noté X.fils_gauche

D1 est le fils droit de X, noté X.fils_droit

Un arbre binaire de recherche est ordonné de la manière suivante :

Pour **chaque** nœud X,

- les valeurs de tous les nœuds du sous-arbre gauche sont **strictement inférieures** à la valeur du nœud X
- les valeurs de tous les nœuds du sous-arbre droit sont **supérieures ou égales** à la valeur du nœud X

Ainsi, par exemple, toutes les valeurs des nœuds G1, G2 et G3 sont strictement inférieures à la valeur du nœud X et toutes les valeurs des nœuds D1, D2 et D3 sont supérieures ou égales à la valeur du nœud X.

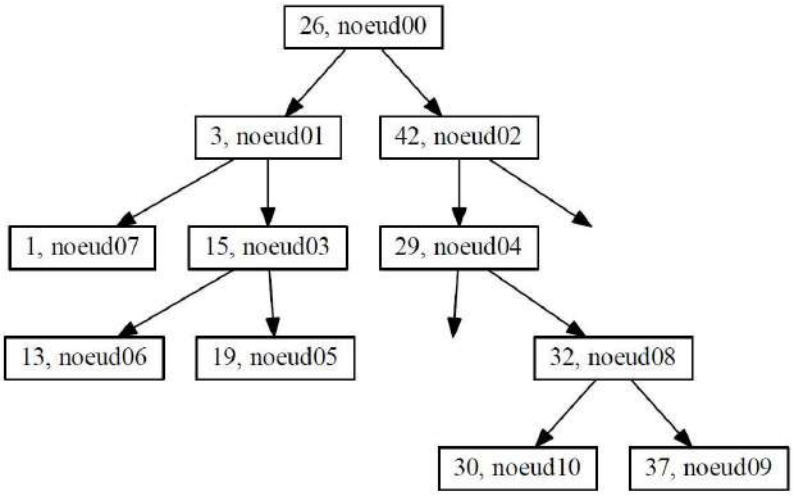
Voici un exemple d'arbre binaire de recherche dans lequel on a stocké dans cet ordre les valeurs :

[26, 3, 42, 15, 29, 19, 13, 1, 32, 37, 30]

L'étiquette d'un nœud indique la valeur du nœud suivie du nom du nœud.

Les nœuds ont été nommés dans l'ordre de leur insertion dans l'arbre ci-dessous.

`'29, noeud04'` signifie que le nœud nommé `noeud04` possède la valeur `29`.



1. On insère la valeur 25 dans l'arbre, dans un nouveau nœud nommé nœud11.
   Recopier l'arbre binaire de recherche étudié et placer la valeur 25 sur cet arbre en coloriant en rouge le chemin parcouru.
   Préciser sous quel nœud la valeur 25 sera insérée et si elle est insérée en fils gauche ou en fils droit, et expliquer toutes les étapes de la décision.

2. **Préciser** toutes les valeurs entières que l'on peut stocker dans le nœud fils gauche du nœud04 (vide pour l'instant), en respectant les règles sur les arbres binaires de recherche ?

3. Voici un algorithme récursif permettant de parcourir et d'afficher les valeurs de l'arbre :

```
Parcours(A)        # A est un arbre binaire de recherche
  Afficher(A.valeur)
  Parcours(A.fils_gauche)
  Parcours(A.fils_droit)
```

   3.a. **Écrire** la liste de toutes les valeurs dans l'ordre où elles seront affichées.
   3.b. **Choisir** le type de parcours d'arbres binaires de recherche réalisé parmi les propositions suivantes : Préfixe, Suffixe ou Infixe

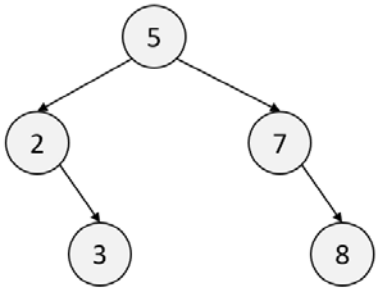
4. En vous inspirant de l'algorithme précédent, écrire un algorithme `Parcours2` permettant de parcourir et d'afficher les valeurs de l'arbre `A` dans l'ordre croissant.

EXERCICE 2 : Arbre binaire de recherche (4 points)

Cet exercice traite principalement du thème « algorithmique, langages et programmation » et en particulier les arbres binaires de recherche. La première partie aborde les arbres en mode débranché via l'application d'un algorithme sur un exemple. La suivante porte sur la programmation orientée objet. La dernière partie fait le lien avec les algorithmes de tri.

Partie A : Étude d'un exemple

Considérons l'arbre binaire de recherche ci-dessous.



1. Indiquer quelle valeur a le nœud racine et quels sont les fils de ce nœud.
2. Indiquer quels sont les nœuds de la branche qui se termine par la feuille qui a pour valeur 3.
3. Dessiner l'arbre obtenu après l'ajout de la valeur 6.

Partie B : Implémentation en Python

Voici un extrait d'une implémentation en Python d'une classe modélisant un arbre binaire de recherche.

```
class ABR:
    """Implémentation d'un arbre binaire de recherche (ABR)"""
    def __init__(self, valeur=None):
        self.valeur = valeur
        self.fg = None
        self.fd = None

    def estVide(self):
        return self.valeur == None

    def insererElement(self, e):
        if self.estVide():
            self.valeur = e
        else:
            if e < self.valeur:
                if self.fg:
```

```
                    self.fg.insererElement(e)
                else:
                    self.fg = ABR(e)
            if e > self.valeur:
                if self.fd:
                    self.fd.insererElement(e)
                else:
                    self.fd = ABR(e)
```

1. Expliquer le rôle de la fonction `__init__`.

2. Dans cette implémentation, expliquer ce qui se passe si on ajoute un élément déjà présent dans l'arbre.

3. Recopier et compléter les lignes de code surlignées ci-dessous permettant de créer l'arbre de la partie A.

```
arbre = ABR(.....)
arbre.insererElement(2)
arbre.insererElement(.....)
arbre.insererElement(7)
arbre.insererElement(.....)
```

Partie C : Tri par arbre binaire de recherche

On souhaite trier un ensemble de valeurs entières distinctes grâce à un arbre binaire de recherche. Pour cela, on ajoute un à un les éléments de l'ensemble dans un arbre initialement vide. Il ne reste plus qu'à parcourir l'arbre afin de lire et de stocker dans un tableau résultat les valeurs dans l'ordre croissant.

1. Donner le nom du parcours qui permet de visiter les valeurs d'un arbre binaire de recherche dans l'ordre croissant.

2. Comparer la complexité de cette méthode de tri avec celle du tri par insertion ou du tri par sélection.