

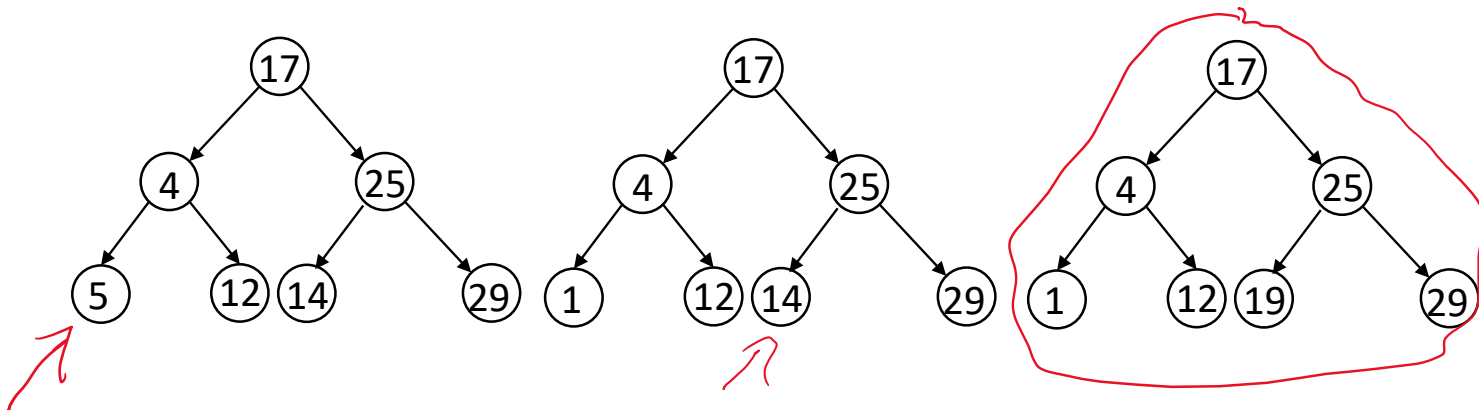
## 1. Définition

- Un arbre binaire de recherche (ABR) est une structure permettant de ranger des informations ordonnées.

Les nombres, les caractères et les chaînes de caractères, les opérateurs sont ordonnés.

- C'est un **arbre binaire** où pour tout nœud n de l'arbre, **les nœuds du fils gauche de n sont plus petits que n** et **les nœuds du fils droit sont plus grands que n**.
- Les procédures d'insertion et de suppression doivent faire respecter cette règle.

Ex 1 Entourer les arbres binaires suivants qui sont des arbres binaires de recherche



Ex 2 Insérer dans l'arbre

Insérer 6	Insérer 4	Insérer 9

### 2. Implémentation de la classe Arbre Binaire de Recherche

Un ABR est aussi un Arbre

```
class ABR(Graph) :  
  
    def __init__(self, info = None , fg = None , fd = None) :  
        self.info = str(info) # str est utile pour le tracé de l'arbre  
        self.fg = fg  
        self.fd = fd
```

En revanche la méthode pour le remplir doit satisfaire les exigences de la définition

```
def insererElement(self, e):  
    if not self.info:  
        # Le sous arbre est vide  
        self.info = e  
    else:  
        if e < int(self.info): #int permet la comparaison entre entiers  
            if self.fg : #le sous arbre existe déjà, on continue  
                self.fg.insererElement(e)  
            else: # Le sous arbre n'existe pas, on le créer  
                self.fg = ABR(e)  
        if e > int(self.info): #int permet la comparaison entre entiers  
            if self.fd :  
                self.fd.insererElement(e)  
            else:  
                self.fd = ABR(e)  
        # rien à faire si égal, déjà présent, arrêt de la récursion
```

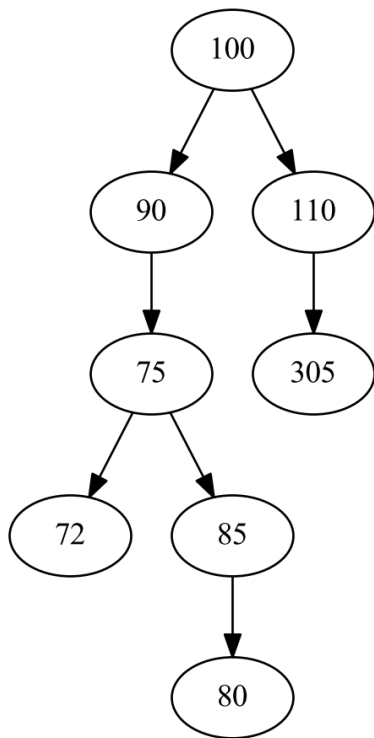
La recherche dans un ABR consiste à s'orienter vers le fils gauche si la valeur recherchée est inférieure à celle du nœud vers la droite dans le cas contraire et vérifier le cas particulier où la valeur du nœud est celle recherchée.

```
def rechercherElement(self, e):  
#Les int permettent la comparaison entre entiers  
    if not int(self.info): return (False, None)  
    if e == int(self.info): return (True, self)  
    if e < int(self.info):  
        if self.fg:  
            return self.fg.rechercherElement(e)  
        else:  
            return (False, None)  
    if e > int(self.info):  
        if self.fd:  
            return self.fd.rechercherElement(e)  
        else:  
            return (False, None)
```

Le return est important la fonction étant récursive elle a besoin de travailler sur la partie droite et gauche si besoin pour la réinjecter au tour suivant

## T NSI Les Arbres Binaires de Recherche : ABR

Ex Implémenter le code pour obtenir



On utilise aussi le mécanisme de la récursivité pour vérifier qu'un arbre est un ABR

```
def estABR(self):  
    '''  
    Vérifie l'ABR et fixe le minimum et le maximum  
    Les int() permettent ici aussi de comparer des entiers  
    '''  
    self.min = self.max = int(self.info)  
    if self.fg:  
        if not self.fg.estABR():  
            return False  
        if self.fg.max >= int(self.info):  
            return False  
        self.min = self.fg.min  
    if self.fd:  
        if not self.fd.estABR():  
            return False  
        if self.fd.min < int(self.info):  
            return False  
        self.max = self.fd.max  
    return True
```

Tester la méthode.