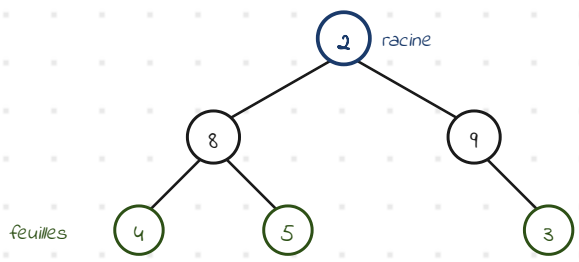
Algorithmes sur les arbres binaires & ABR

NSI · Terminale · Bac

Vocabulaire · arbre binaire (AB)

Chaque noeud a **au plus 2 fils** · racine, fils gauche (SAG), fils droit (SAD). Définition **récursive**.



Mesure	Définition
Taille N	nombre de noeuds
Hauteur H	$\lfloor \log_2 N \rfloor \leq H \leq N-1$

Parcours en profondeur (DFS) · récursifs

Préfixe
R · G · D
2-8-4-5-9-3

Infixe
G · R · D
4-8-5-2-9-3

Suffixe
G · D · R
4-5-8-3-9-2

```
def parcours_prefixe(A):
    if A is not None:
        print(A.etiquette)    # avant
        parcours_prefixe(A.gauche)
        parcours_prefixe(A.droit)

def parcours_infixe(A):
    if A is not None:
        parcours_infixe(A.gauche)
        print(A.etiquette)    # entre
        parcours_infixe(A.droit)
```

Complexité des 3 parcours : $O(N)$ · chaque noeud visité 1 fois.

Parcours en largeur (BFS) · file FIFO

Visite **niveau par niveau**, gauche → droite. Itératif avec une **file**.

ordre : 2 - 8 - 9 - 4 - 5 - 3

```
def parcours_largeur(A):
    F = File()
    F.enfiler(A)
    while F.taille() != 0:
        a = F.defiler()
        if a is not None:
            print(a.etiquette)
            F.enfiler(a.gauche)
            F.enfiler(a.droit)
```

Complexité : $O(N)$ · utile pour **plus court chemin** (graphes non pondérés) et affichage niveau par niveau.

Classe Noeud · Python

```
class Noeud:
    def __init__(self, e, g=None, d=None):
        self.etiquette = e
        self.gauche = g
        self.droit = d

    def est_feuille(self):
        return not self.gauche and not self.droit
```

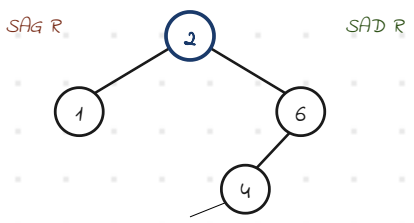
→ Arbre vide = **None**

→ Chaque **Noeud** est à la fois un noeud **et** le sous-arbre dont il est la racine.

Hauteur récursive : `1 + max(h(g), h(d))` · arbre vide → 0.

Arbre Binaire de Recherche (ABR)

Pour tout noeud de clé k : **SAG** $\leq k$ et **SAD** $> k$ · propriété valable **en tout noeud**.



Parcours infixe d'un ABR → liste les clés dans **l'ordre croissant**.

Recherche · insertion en $O(H)$

```
def etq_presente(A, e):
    if A is None: return False
    if e == A.etiquette: return True
    if e < A.etiquette:
        return etq_presente(A.gauche, e)
    return etq_presente(A.droit, e)

def ajouter(A, e):
    if A is None: return Noeud(e)
    if e <= A.etiquette:
        return Noeud(A.etiquette,
                     ajouter(A.gauche, e), A.droit)
    return Noeud(A.etiquette,
                 A.gauche, ajouter(A.droit, e))
```

→ Insertion crée toujours une **feuille**, jamais un noeud interne.

Équilibre · meilleur vs pire cas

Cas	H	Recherche
Équilibré	$\log_2 N$	$O(\log_2 N)$
Parfait	$N = 2^{H+1}-1$	$O(\log_2 N)$
Filiforme	$N-1$	$O(N)$

Insérer dans **l'ordre trié** → ABR filiforme : aussi lent qu'une liste chaînée · AVL et rouge-noir s'auto-équilibrent.

Synthèse · pièges fréquents

1. Confondre **hauteur** et **taille** · H = profondeur max des feuilles.
2. Oublier que l'efficacité d'un ABR dépend de son **équilibre**.
3. Bien retenir l'ordre **R/G/D** qui change entre préfixe, infixe, suffixe.
4. Vérifier la propriété ABR sur **tous les noeuds**, pas seulement la racine.

Algo	Structure	Complexité
Parcours	AB quelconque	$O(N)$
Recherche	ABR équilibré	$O(\log_2 N)$
Insertion	ABR filiforme	$O(N)$