Arbres Binaires de Recherche

Plan

- Structure de dictionnaire
- Arbres binaires
- Arbres binaires de recherche

Structure de dictionnaire

Un dictionnaire est un ensemble d'**entrées** constituées de **clés** associées à des données.
Toutes les clés présentes dans un dictionnaire sont **distinctes**.

Opérations que l'on veut réaliser efficacement :

- Tester la présence d'une clé
- Ajouter une entrée au dictionnaire
- Supprimer une entrée du dictionnaire

Implémentation à l'aide de tableaux

On suppose que le tableau comporte n éléments.

	Tableau non trié	Tableau trié
Recherche	$O(n)$	

Implémentation à l'aide de tableaux

On suppose que le tableau comporte n éléments.

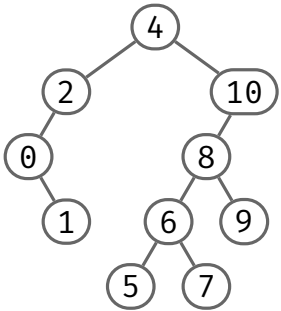
	Tableau non trié	Tableau trié
Recherche	$O(n)$	$O(\log_2 n)$

Implémentation à l'aide de tableaux

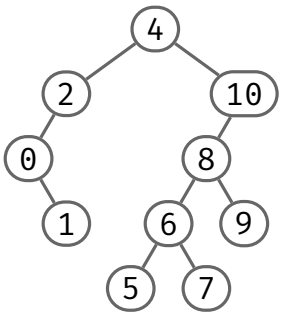
On suppose que le tableau comporte n éléments.

	Tableau non trié	Tableau trié
Recherche	$O(n)$	$O(\log_2 n)$
Ajout	$O(n)$	$O(n)$
Suppression	$O(n)$	$O(n)$

Arbres binaires



Arbres binaires



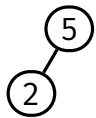

```
class ArbreBinaire:
    def __init__(self, valeur):
        self.clé    = valeur
        self.gauche = None
        self.droit  = None
```

Arbres binaires

```
a = ArbreBinaire(5)
```

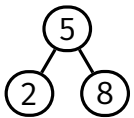
Arbres binaires

```
a = ArbreBinaire(5)
a.gauche = ArbreBinaire(2)
```



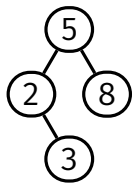
Arbres binaires

```
a = ArbreBinaire(5)
a.gauche = ArbreBinaire(2)
a.droite = ArbreBinaire(8)
```

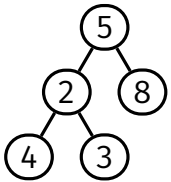


Arbres binaires

```
a = ArbreBinaire(5)
a.gauche = ArbreBinaire(2)
a.droite = ArbreBinaire(8)
a.gauche.droite = ArbreBinaire(3)
```



Parcours infixe d’un arbre binaire



Parcours infixe d'un arbre binaire

```
def parcours_infixe(arbre):
    if arbre != None:
```

Parcours infixe d'un arbre binaire

```
def parcours_infixe(arbre):
    if arbre != None:
        parcours_infixe(arbre.gauche)
```

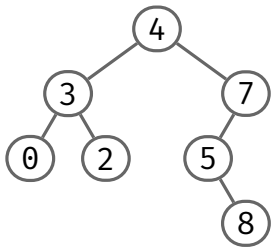
Parcours infixe d'un arbre binaire

```
def parcours_infixe(arbre):
    if arbre != None:
        parcours_infixe(arbre.gauche)
        print(arbre.clé)
```

Parcours infixe d'un arbre binaire

```
def parcours_infixe(arbre):
    if arbre != None:
        parcours_infixe(arbre.gauche)
        print(arbre.clé)
        parcours_infixe(arbre.droit)
```

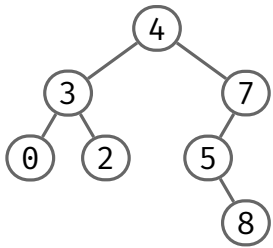
Parcours infixe d'un arbre binaire

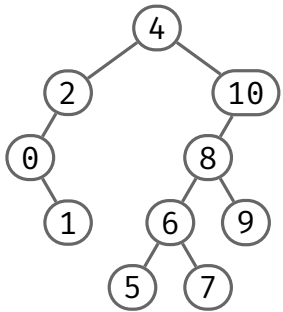


Dans quel ordre vont être traîtées les clés lors d'un parcours infixe ?

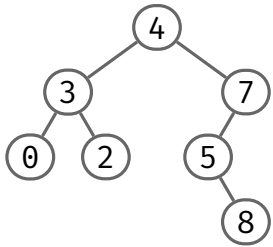
a) 4, 3, 7, 0, 2, 5, 8
b) 4, 3, 0, 2, 7, 5, 8
c) 3, 0, 2, 4, 7, 5, 8
d) 0, 3, 2, 4, 5, 8, 7

Parcours infixe d'un arbre binaire

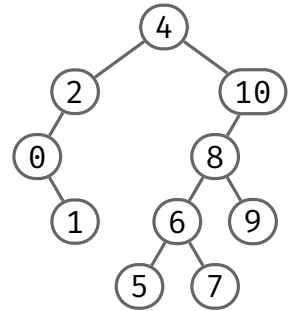




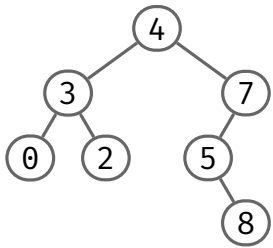
Parcours infixe d'un arbre binaire



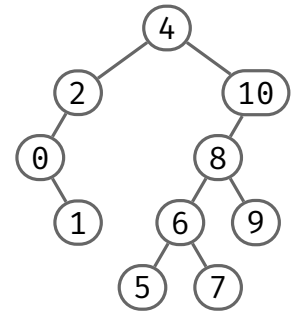
0 3 2 4 5 8 7



Parcours infixe d'un arbre binaire



0 3 2 4 5 8 7



0 1 2 4 5 6 7 8 9 10

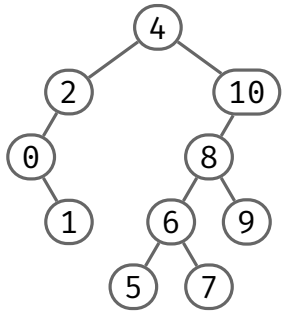
Arbre binaire de recherche

Un **arbre binaire de recherche** est un arbre binaire dont le **parcours infixe** fournit des valeurs dans un **ordre strictement croissant**.

Arbre binaire de recherche

Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

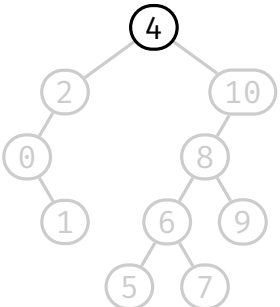
- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



Arbre binaire de recherche

Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

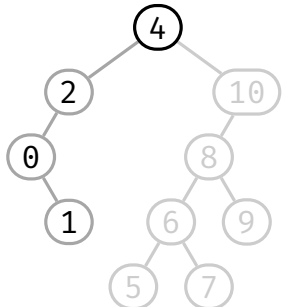
- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



Arbre binaire de recherche

Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

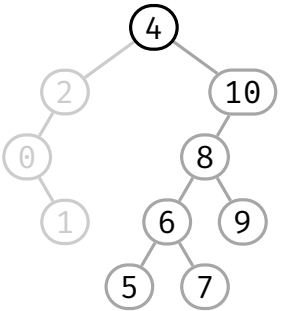
- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



Arbre binaire de recherche

Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

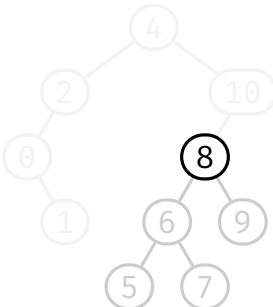
- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



Arbre binaire de recherche

Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

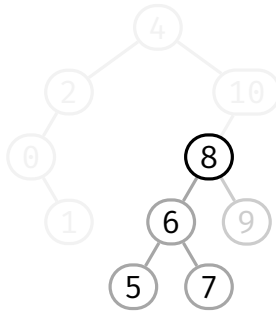
- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



Arbre binaire de recherche

Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

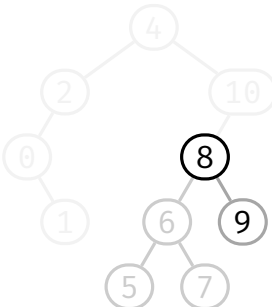
- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



Arbre binaire de recherche

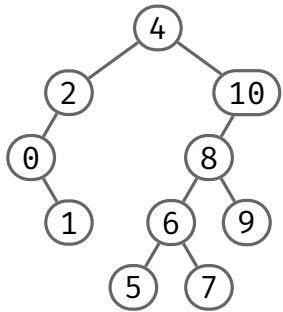
Définition La clé à la racine d'un arbre binaire de recherche (ou d'un sous-arbre) est toujours

- **plus grande** que les clés présentes dans son **fils gauche** ;
- **plus petite** que les clés présentes dans son **fils droit**.



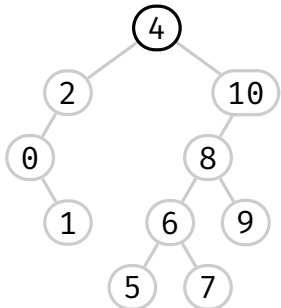
Recherche dans un a.b.r.

On recherche la clé 9.



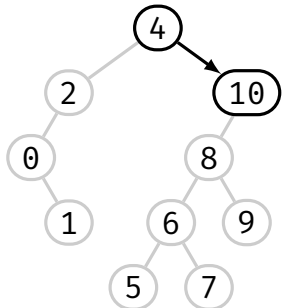
Recherche dans un a.b.r.

On recherche la clé 9.



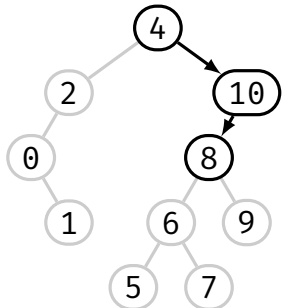
Recherche dans un a.b.r.

On recherche la clé 9.



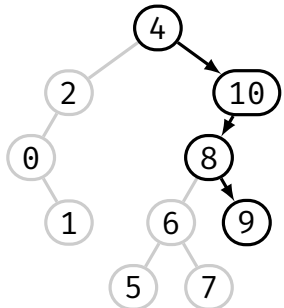
Recherche dans un a.b.r.

On recherche la clé 9.



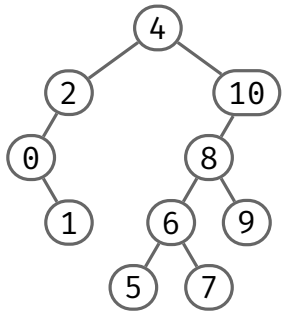
Recherche dans un a.b.r.

On recherche la clé 9.



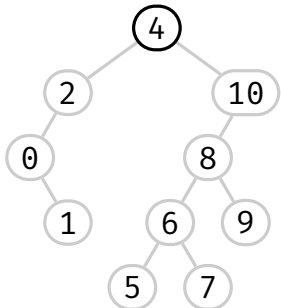
Recherche dans un a.b.r.

On recherche la clé 3.



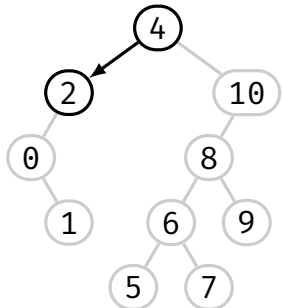
Recherche dans un a.b.r.

On recherche la clé 3.



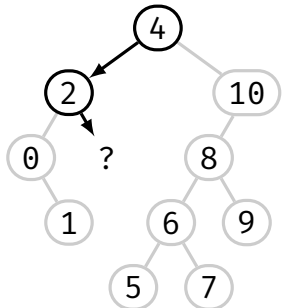
Recherche dans un a.b.r.

On recherche la clé 3.



Recherche dans un a.b.r.

On recherche la clé 3.



Recherche dans un a.b.r.

```
def est_présent(arbre, valeur):
    if arbre == None:
        return False
```

Recherche dans un a.b.r.

```
def est_présent(arbre, valeur):
    if arbre == None:
        return False
    elif arbre.clé == valeur:
        return True
```

Recherche dans un a.b.r.

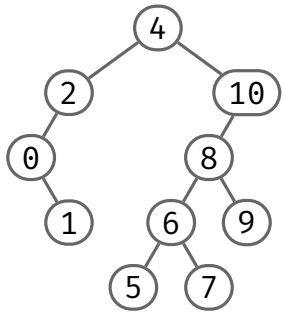
```
def est_présent(arbre, valeur):
    if arbre == None:
        return False
    elif arbre.clé == valeur:
        return True
    elif arbre.clé > valeur:
        return est_présent(arbre.gauche, valeur)
```

Recherche dans un a.b.r.

```
def est_présent(arbre, valeur):
    if arbre == None:
        return False
    elif arbre.clé == valeur:
        return True
    elif arbre.clé > valeur:
        return est_présent(arbre.gauche, valeur)
    else:
        return est_présent(arbre.droit, valeur)
```

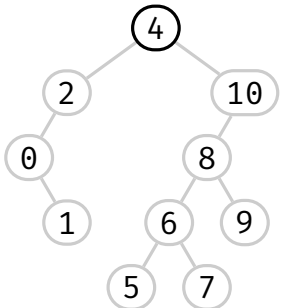
Insertion dans un a.b.r.

On insère la clé 3.



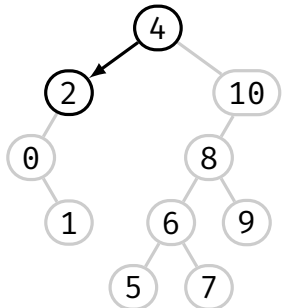
Insertion dans un a.b.r.

On insère la clé 3.



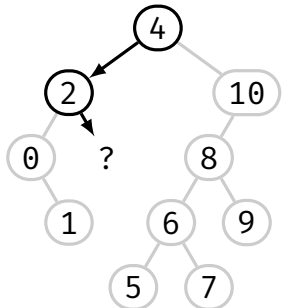
Insertion dans un a.b.r.

On insère la clé 3.



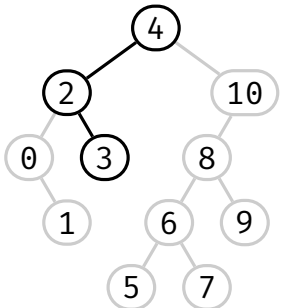
Insertion dans un a.b.r.

On insère la clé 3.



Insertion dans un a.b.r.

On insère la clé 3.



Insertion dans un a.b.r.

```
def insérer(arbre, valeur):
    # On suppose que arbre != None
    if arbre.clé < valeur:
        # On regarde dans le sous-arbre droit
```

Insertion dans un a.b.r.

```
def insérer(arbre, valeur):
    # On suppose que arbre != None
    if arbre.clé < valeur:
        # On regarde dans le sous-arbre droit
        if arbre.droit == None:
            arbre.droit = ArbreBinaire(valeur)
```

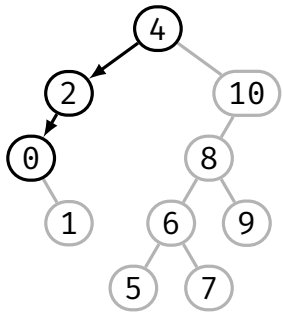
Insertion dans un a.b.r.

```
def insérer(arbre, valeur):
    # On suppose que arbre != None
    if arbre.clé < valeur:
        # On regarde dans le sous-arbre droit
        if arbre.droit == None:
            arbre.droit = ArbreBinaire(valeur)
        else:
            insérer(arbre.droit, valeur)
```

Insertion dans un a.b.r.

```
def insérer(arbre, valeur):
    # On suppose que arbre != None
    if arbre.clé < valeur:
        # On regarde dans le sous-arbre droit
        if arbre.droit == None:
            arbre.droit = ArbreBinaire(valeur)
        else:
            insérer(arbre.droit, valeur)
    elif arbre.clé > valeur:
        # On regarde dans le sous-arbre gauche
        if arbre.gauche == None:
            arbre.gauche = ArbreBinaire(valeur)
        else:
            insérer(arbre.gauche, valeur)
```

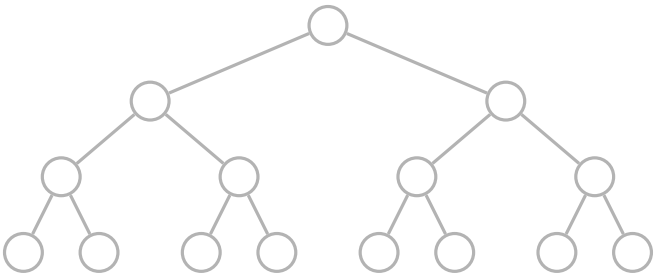
Complexité



Complexité temporelle

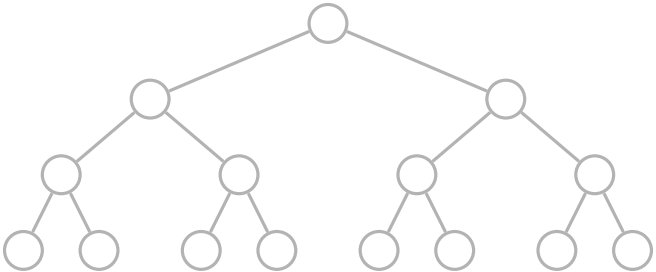
Les différentes opérations ont un **coût proportionnel à la hauteur** de l'arbre.

Formes d’arbres

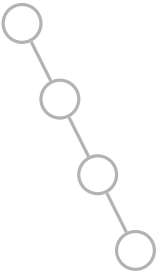


Arbre complet

Formes d’arbres



Arbre complet



Peigne