

# Exercices ABR - correction

Christophe Viroulaud

Terminale - NSI

**Algo 10**

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

Exercice 1

Exercice 2

Exercice 3

Exercice 4

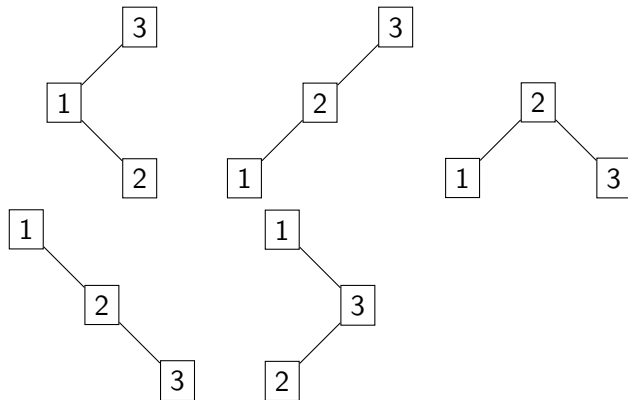
# Exercise 1

Exercice 1

Exercice 2

Exercice 3

Exercice 4



1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

Exercice 1

Exercice 2

Exercice 3

Exercice 4

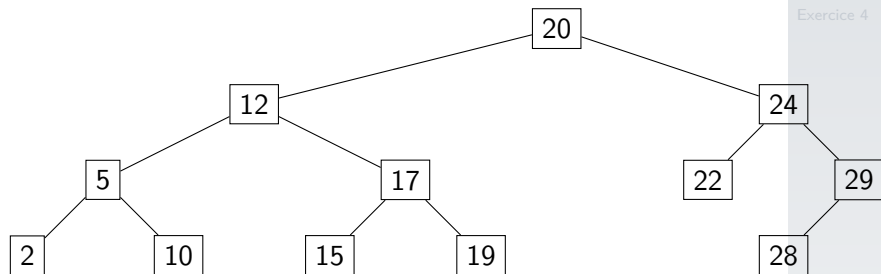
## Exercise 2

Exercice 1

Exercice 2

Exercice 3

Exercice 4



[2, 5, 10, 12, 15, 17, 19, 20, 22, 24, 28, 29]

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

Exercice 1

Exercice 2

Exercice 3

Exercice 4

## Exercice 3

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 class Noeud:
2     def __init__(self, v: int):
3         self.valeur = v
4         self.gauche = None
5         self.droit = None
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 def inserer(self, v: int) -> None:
2     if v < self.valeur: # gauche
3         if self.gauche is None: # feuille
4             self.gauche = Noeud(v)
5         else: # descente récursive
6             self.gauche.inserer(v)
7     else: # droite
8         if self.droit is None: # feuille
9             self.droit = Noeud(v)
10        else: # descente récursive
11            self.droit.inserer(v)
```



Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 arbre = Noeud(13)
2 arbre.inserer(29)
3 arbre.inserer(2)
4 arbre.inserer(49)
5 arbre.inserer(8)
6 arbre.inserer(12)
7 arbre.inserer(16)
8 arbre.inserer(30)
9 arbre.inserer(27)
10 arbre.inserer(10)
11 arbre.inserer(9)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 def rechercher(self, v: int) -> None:
2     if v == self.valeur: # trouvé
3         return True
4     elif v < self.valeur: # gauche
5         if self.gauche is None: # feuille
6             return False
7         else: # descente récursive
8             return self.gauche.rechercher(v)
9     else: # droite
10        if self.droit is None: # feuille
11            return False
12        else: # descente récursive
13            return self.droit.rechercher(v)
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 print(arbre.rechercher(16)) # True
2 print(arbre.rechercher(17)) # False
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 def minimum(self) -> int:
2     g = self.gauche
3     while g.gauche is not None:
4         g = g.gauche
5     return g.valeur
6
7
8 def minimum_rec(self) -> int:
9     if self.gauche is None:
10         return self.valeur
11     else:
12         return self.gauche.minimum_rec()
```

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 def infixe(self, noeud: object, parcours: list) -> None:
2     if n is not None:
3         self.infixe(noeud.gauche, parcours)
4         parcours.append(n.valeur)
5         self.infixe(noeud.droit, parcours)
```

1. Exercice 1

2. Exercice 2

3. Exercice 3

4. Exercice 4

Exercice 1

Exercice 2

Exercice 3

Exercice 4

# Exercise 4

Exercice 1

Exercice 2

Exercice 3

Exercice 4

```
1 def taille(a: dict, s: str) -> int:
2     if a[s][0] == '' and a[s][1] == '': # pas de fils
3         return 1
4     elif a[s][0] == '': # pas de fils gauche
5         return 1 + taille(a, a[s][1])
6     elif a[s][1] == '': # pas de fils droit
7         return 1 + taille(a, a[s][0])
8     else: # deux fils
9         return 1 + taille(a, a[s][1]) + taille(a, a[s][0])
```