Exercice 1 : Donner tous les ABR formés de trois nœuds contenant les entiers 1, 2, 3.

Exercice 2 :

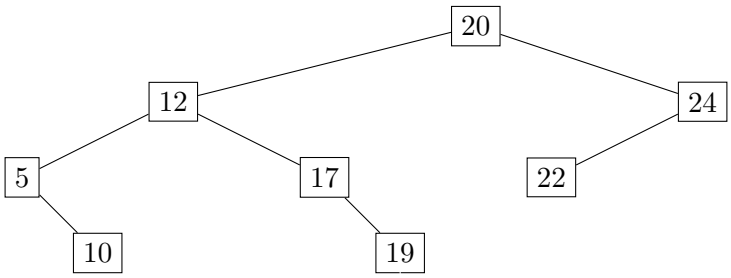

FIGURE 1 – Un Arbre Binaire de Recherche (*ABR*)

1. Compléter cet ABR en insérant dans l'ordre les valeurs 2, 15, 29, 28.
2. Donner le résultat d'un parcours infixe de cet ABR.

Exercice 3 : Ajout de méthodes

1. Dans la classe ABR construite en cours, ajouter la méthode **minimum(self) → int** qui renvoie le minimum de l'ABR.
2. Écrire une méthode *récursive* **maximum_rec(self, n : Noeud) → int** qui renvoie le maximum de l'ABR.
3. Écrire la méthode **maximum(self) → int** qui utilise la méthode précédente pour renvoyer le maximum de l'ABR.
4. Écrire la méthode **infixe_rec(self, n : Noeud, parcours : list) → list** qui renvoie *parcours*, le parcours infixe de l'arbre.
5. Écrire la méthode **infixe(self) → list** qui appelle la méthode *infixe_rec* et renvoie le parcours infixe de l'arbre.
6. **Pour les plus avancés :** Réécrire la méthode *infixe* avec une **fonction** infixe_rec interne à la méthode *infixe*.

Exercice 4 : Comparaison de tris

1. Écrire la fonction **tri_selection(tab : list) → list** qui renvoie le tableau trié.
2. Écrire la fonction **tri_rapide(tab : list) → list** qui renvoie le tableau trié.
3. Écrire la fonction **tri_ABR(tab : list) → list** qui construit l'ABR à partir du tableau puis effectue un parcours infixe et renvoie le parcours.
4. Construire par compréhension un tableau de 5000 entiers aléatoires compris entre 0 et 1000.
5. Écrire la fonction **duree_tri(fonction, tab : list) → float** qui renvoie la durée d'exécution du tri de *tab* par *fonction*.
6. Mesurer la durée d'exécution des trois tris.
7. Quelle est la complexité de la fonction *tri_ABR* si l'arbre est équilibré ?
8. Que devient cette complexité si le tableau de départ est déjà trié ?

Christophe Viroulaud