# Laboratoire 9 – Arbre Binaire de Recherche

## Objectifs ASD

Mettre en œuvre en C++ la structure de donnée d'arbre binaire de recherche, et en particulier

- Insérer un élément
- Trouver l'élément le plus petit
- Trouver un élément par sa clé
- Supprimer l'élément le plus petit
- Supprimer un élément quelconque
- Parcourir l'arbre avec les parcours pré-ordonné, symétrique et post-ordonné
- Trouver un élément par son rang
- Retourner le rang d'un élément
- Construire et affecter par copie et par déplacement, effacer la structure
- Equilibrer l'arbre en le linéarisant puis arborisant en place.

Pour la plupart de ces opérations, une mise en œuvre récursive sera la plus simple.

Comprendre et documenter la complexité de toutes ces opérations.

## Objectifs C++

- Mettre en œuvre une classe C++ générique
- Utiliser les opérateurs `new` et `delete` classique.
- Maitriser mieux le passage de pointeur par référence.
- Ecrire les constructeur et destructeur, constructeur de copie et opérateur d'affectation, constructeur et opérateur de déplacement
- Tester l'état de la structure et les paramètres reçus et lever les exceptions appropriées quand ils ne permettent pas l'exécution d'une méthode.
- Offrir une garantie forte sur l'ensemble des méthodes

## Consignes

Ce laboratoire se déroule sur une période de 3 semaines. Des résultats intermédiaires doivent être fournis à la fin des deux premières semaines sous la forme des codecheck 1 et 2 testant votre classe C++. Le résultat final est en partie évalué via codecheck 3.

- **Codecheck 1** teste le constructeur par défaut, le destructeur, les méthodes `insert(value_type)`, `contains(value_type)`, `min()`, `deleteMin()`, et `deleteElement(value_type)`, ainsi que les trois types de parcours en profondeur.
- **Codecheck 2** effectue les mêmes tests que le CC1, mais affiche également le contenu du champ `nbElements` de chaque `Node`. Il teste également les méthodes `size()`, `nth_element(size_t)` et `rank(const_reference)`. Enfin, il teste les constructeurs et affectations par copie et par déplacement, ainsi que la méthode `swap(BinarySearchTree&)`
- **Codecheck 3** effectue les memes tests que le CC2, mais teste également les algorithmes de linéarisation et arborisation, les exceptions générées par votre code ainsi que les garanties offertes en cas d'exceptions générées lors de la copie d'éléments dans `insert`, `operator=` et dans le constructeur de copie.

L'interface publique de la classe à mettre en œuvre, la sous-classe `Node` stockant les éléments, ainsi que l'ensemble des attributs privés de la classe sont fournis et ne peuvent être modifiés. Ces attributs sont le seul pointeur `_root` pointant vers la racine de l'arbre, ou valant `nullptr` dans le cas d'un arbre vide.

Pour le dernier codecheck uniquement, toutes les méthodes, publiques ou privées, doivent être commentées dans le style doxygen, y compris un champ `@remark` documentant la complexité algorithmique de la fonction.