Implement the ADT for a *Set* using a binary search tree (BST)

- **The ADT**
  - *add(value)*
    - *Adds an item to the set with this value*
  - *contains(value)*
    - *Returns True if a value is in the set, otherwise False*
  - ***remove(value)***
    - **Removes a value from the set** (*material covered in next session*)
  - *__len__*
    - *Returns the number of items in the set*
  - *__str__*
    - *Prints the contents of the set, ordered*
    - *Try making versions that return or print the contents of the tree pre-order and post-order as well. This can help you test whether the tree is actually getting built the way you meant it to be.*
  - ***The difference between a set and a map is, in general, that in a set the key and the data are the same value.***
- *Remember to **plan well** how to use recursion in these operations.*
  - Make separate recursive functions
    - *In addition to the value parameters they can take a node as a parameter*
    - *They can return nodes, to make sure the entire chain links up correctly after the call*
    - *In some cases you may want to keep track of a reference through a parameter or a class variable, to hold onto a certain node or value, while searching for another one.*
- ***Some extra implementations:***
  - Assume that the values are all strings and implement a ***pre-fix search***
    - Allow user to type in a string
    - Return a python list with all strings from your set that begin with that string
      - Your set includes (among many others) the strings:
        grades
        gratuity
        grandmaster flash
      - User types in: **gra**
      - Program returns:
        {“grades”, ”gratuity”, “grandmaster flash”}
  - Implement an operation that prints all values in a certain range:
    - Takes a ***min*** value and a ***max*** value
    - Prints all values that are between the values (*both included? Up to you*)