```
// BST.h

#pragma once
#include <iostream>
#include <initializer_list>
using namespace std;



class BST {

private:
    struct Node {
        int val; // The value stored in the node
        int size; // number of nodes in the subtree rooted here
        Node *left; // points to the left subtree
        Node *right; // points to the right subtree
        Node(int v, Node *l = nullptr, Node *r = nullptr) : val(v), size(1),
            left(l), right(r) {};
    };

    Node* root; //root node

public:
    /*********************************************************************
    BST()
    Parameters: none
    Complexity: O(1)

    Default constructor. Makes an object of the BST class and sets the
    root node to nullptr.
    *********************************************************************/
    BST() {
        root = nullptr;
    }

    /*********************************************************************
    BST()
    Parameters: initializer_list<int>
    Complexity: O(n)

    User defined constructor. Makes an object of the BST class and inserts
    every element in the list into the tree.
    *********************************************************************/
    BST(initializer_list<int> lst) {
        for (auto i : lst)
            insert(i);
    }

    /*********************************************************************
    ~BST()
    Parameters: none
    Complexity: O(n)

    Default destructor. Calls the clear function and passes it a pointer to
    the root node. Deletes every node in the tree and removes dangling
    pointers.
    Complexity is O(n) because it has to delete every node in the tree.
    *********************************************************************/
    ~BST() {
        clear(root);
    }

    /*********************************************************************
    insert()
    Parameters: int
    Complexity: O(n)

    Public insert method. Calls the private method and passes it a pointer to
    the root node and the integer to be inserted.
    Complexity is O(n) because the tree may not be balanced.
    *********************************************************************/
    void insert(int v) {
        insertAux(root, v);
    }

    /*********************************************************************
    remove()
    Parameters: int
    Complexity: O(n)

    Public method to delete a node from the tree using recursion
    It calls the private search function to make sure the node exists before
    trying to delete it and therefore decrement all sizes along the path.
    Then it call the private delete node method and passes it a pointer to a
    node and the integer to be removed.
    Complexity is O(n) because the tree may not be balanced.
    *********************************************************************/
    void remove(int v) {
        if (searchAux(root, v) == true)
            removeAux(root, v);
    }

    /*********************************************************************
    inOrderPrint()
    Parameters: ostream&
    Complexity: O(n)

    Public method to print the contents of a tree in order of ascending value.
    Calls the private method and passes it a pointer to the root node
    and the ostream type to use for output.
    Complexity is O(n) because it has to go to every node in the tree and
    print its value.
    *********************************************************************/
    void inOrderPrint(ostream& os)const {
        if (root == nullptr)  // Tree has no data
            os << "<empty>";
        else
            inOrderPrintAux(root, os);
        os << endl;
    }

    /***********************************************************************
    numNodes()
    Parameters: none
    Complexity: O(n)

    Public method to return an int with the total number of nodes in the tree.
    Calls the private method and passes it a pointer to the root node.
    Complexity is O(n) because the method has to go to every node
    in the tree.
    ************************************************************************/
    int numNodes() {
        return numNodesAux(root);
    }

    /***********************************************************************
    search()
    Parameters: int
    Complexity: O(n)

    Public method to search for an int in a tree.
    Returns a bool that says whether the int was found.
    Calls the private method and passes it a pointer to the root node
    and the int to search for.
    Complexity is O(n) because the tree may not be balanced.
    ************************************************************************/
    bool search(int v) {
        return searchAux(root, v);
    }

    /***********************************************************************
    rank()
    Parameters: int
    Complexity: O(n)

    Public method to return an integer's rank in the tree.
    The rank is its position in the sorted tree. For example, the smallest
    int in a tree would be rank 1. The largest would be equal to the number
    of nodes in the tree. If the integer is not found in the tree, it
    returns 0.
    Calls the private method and passes it a pointer to the root node and
    the int whose rank will be returned.
    Complexity is O(n) because the method may have to go through every node
    in the tree.
    ************************************************************************/
    int rank(int v) {
        return rankAux(root, v);
    }

    /**********************************************************************
    range()
    Parameters: 2 ints
    Complexity: O(n)

    Public method to return an int with the range between two ints in a tree.
    The range is the number of nodes in the tree with values that are
    greater than or equal to the first argument (i),
    and less than the second (j). If the arguments create an impossible range
    or the tree is empty, it returns 0.
    Calls the private method and passes it a pointer to the root node, and
    two ints whose range will be returned.
    Complexity is O(n) because the method may have to go through every node in
    the tree.
    ***********************************************************************/
    int range(int i, int j) {
        return rangeAux(root, i, j);
    }


private:
    void clear(Node*& r);
    void insertAux(Node*&, int);
    void removeAux(Node*&, int);
    void inOrderPrintAux(Node*, ostream&)const;
    int numNodesAux(Node*&) const;
    bool searchAux(Node*&, int)const;
    int rankAux(Node*&, int, int = 1);
    int rangeAux(Node*&, int, int);
};
```