1. Create a design, **before** you start coding, that shows how your binary tree functions and what attributes it keeps track of to function (yes, you can add to this design once you start coding, but please get some design down to start with and make note of when you add new design features based on your implementation work🙂)

Design Overview:

General Plan:

The Binary Search Tree (BST) will be designed to store books, with each node containing book information and pointers to its left and right child nodes. The BST will support insertion, deletion, and search operations, along with various traversal methods. The program will be broken down into 5 sections, main.cpp, book.cpp, book.h, test.cpp, and test.h.

main.cpp:

int main(){

BookList* root = nullptr;

call tests

book.h:

Book: Represents a book with attributes title and id.

BookList: Node structure containing a Book and pointers to its left and right child nodes.

Headers for all the functions used in book.cpp

book.cpp:

BookList* create_node(Book book)

BookList* add_book(BookList* root, Book book)

BookList* delete_book(BookList* root, const std::string& title)

Book* search_book(BookList* root, const std::string& title)

Tree traversal functions (start with in-order)

test.h

Headers for all the tests

test.cpp

test_add

test_delete

test_search

test_traversals

2. Create some tests (at least one per function), **before** you start coding, that you want your Binary Search Tree (BST) to pass as evidence that it would be working correctly if it passed the tests.

3. Implement a binary search tree that includes:
    1. Nodes to store values,
    2. an add function that adds a new value in the appropriate location based on our ordering rules, (I likely used less than or equal to going to the left and greater than values going to the right)
    3. a remove function that finds and removes a value and then picks an appropriate replacement node, (successor is a term often used for this)
    4. we have at least one tree traversal function (I recommend starting with an in-order traversal!) **Bonus** if you implement the three common traversals (pre-order, post-order, in-order) **More Bonus** if you also include a breadth-first traversal (sometimes called a level-order search)
4. Analyze and compare the complexity of insert and search as compared to a binary tree without any order in its nodes (what is the run-time of an unordered tree...?).
5. Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).