Binary Search Tree Pseudocode

Dakota Keyes

CS300

struct Bid {

string bidId; // unique identifier

string title;

string fund;

struct Node {

Bid bid;

Node\* left; // pointer for values < parent

Node\* right; // pointer for values < parent

class BinarySearchTree {

void addNode(Node\* node, Bid bid);

void inOrder(Node\* node);

Node\* removeNode(Node\* node, string bidId);

BinarySearchTree();

virtual ~BinarySearchTree();

void InOrder();

void Insert(Bid bid);

void Remove(string bidId);

Bid Search(string bidId);

//Traverse Tree inOrder

void BinarySearchTree::InOrder() {

this->inOrder(root);

//Inserting a bid

void BinarySearchTree::Insert(Bid bid) {

if (root == nullptr) {

root = new Node(bid);

} else {

this->addNode(root, bid);

//Removing Bid

void BinarySearchTree::Remove(string bidId) {

this->removeNode(root, bidId);

//Searching for Bid

Bid BinarySearchTree::Search(string bidId)

//Add bid to Node

Void BinarySearchTree::addNode(Node\* node, Bid bid) {

// If node > bid, add to left subtree

if (node->left == nullptr) {

node->left = new Node(bid); }

// If left node exists, keep traversing down left subtree

else { this->addNode(node->left, bid);

void BinarySearchTree::inOrder(Node\* node)

// Traverse left side first. inOrder(node->left);

// Traverse right side after each level's been checked. inOrder(node->right);

Print (BidId, BidTitle, BidAmount, BidFund)

Node\* BinarySearchTree::removeNode(Node\* node, string bidId) {

//Loop to Remove Node

If( node == nullptr)

Return;

If(node->bid.Id) < 0)

Node->left = removeNode

Else (node->bidId) > 0)

Node->right = removeNode