1. Initialized the root in the constructor

BinarySearchTree::BinarySearchTree() {

root = nullptr;

}

1. Created a while loop to delete notes in the destructor.

BinarySearchTree::~BinarySearchTree() {

While (root != nullptr) {

Root = removeNode(root, root->bid.bidId)

}

}

1. InOrder implementation

Void BinarySearchTree::InOrder() {

inOrder(root);

}

void BinarySearchTree:: inOrder(Node\* node) {

if (node != nullptr)

inOrder(node->left);

cout << node.bid.bidId << “: “ << node->bid.title << “ | “ <<node->bid.amount << “ | “ << node->bid.fund << endl;

inOrder(node->right);

}

}

1. The posttOrder.

Void BinarySearchTree::PostOrder() {

postOrder(root);

}

Void BinarySearchTree::postOrder(Node\* node) {

If (node != nullptr) {

postOrder(node->left);

postOrder(node->right);

cout << node->bid.bidId << “ : “ << node->bid.title << “ node->bid.amount << “ | “ << node->bid.fund << endl;

}

}

1. Preorder implementation

Void BinarySearchTree::preOrder() {

preOrder(root);

}

Void BinarySearchTree::preOrder(Node\* node) {

If (node != nullptr) {

cout << node->bid.bidId << “ : “ << node->bid.title << “ node->bid.amount << “ | “ << node->bid.fund << endl;

preOrder(node->left);

preOrder(node->right);

}

}

1. Insert bid implementation.

Void BinarySearchTree::Insert(Bid bid) {

If (root == nullptr) {

Root = new Node(bid);

}

Else {

addNode(root, bid);

}

}

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

If (bid.bidId < node->bid.bidId) {

If (node->left == nullptr) {

Node->left = new Node(bid);

}

Else {

addNode(node->left, bid):

}

}

Else {

If (node->right == nullptr) {

Node->right = new Node(bid);

}

Else {

addNode(node->right, bid);

}

}

}

1. Remove bid implementation.

void BinarySearchTree::Remove(string bidId) {

root = removeNode(root, bidId);

}

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

if (node == nullptr) {

return node;

}

if (bidId < node->bid.bidId) {

node->left = removeNode(node->left, bidId);

}

else if (bidId > node->bid.bidId) {

node->right = removeNode(node->right, bidId);

}

else {

if (node->left == nullptr) {

Node\* temp = node->right;

delete node;

return temp;

}

else if (node->right == nullptr) {

Node\* temp = node->left;

delete node;

return temp;

}

Node\* temp = node->right;

while (temp && temp->left != nullptr) {

temp = temp->left;

}

node->bid = temp->bid;

node->right = removeNode(node->right, temp->bid.bidId);

}

return node;

}

1. Search bid implementation.

Bid BinarySearchTree::Search(string bidId) {

Node\* current = root;

while (current != nullptr) {

if (current->bid.bidId == bidId) {

return current->bid;

}

if (bidId < current->bid.bidId) {

current = current->left;

}

else {

current = current->right;

}

}

Bid bid;

return bid;

}