```
#include "AVLtree.hpp"
#include<iostream>
#include <vector>
#include <string>

AVLTree<int>* TestInsert(){
        AVLTree<int>* t = new AVLTree<int>();
        std::vector<int> values = {50,49,48,47,46,45,44,43,42,41};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
                //t->print();
        }
        //std::cout<<t->Head->Right->Left->Right->Right->Value;
//      t->print();

        /*
        if (t->Head->Value != 50){
                std::cout << "insert failed";
        }

        if (t->Head->Left->Value != 25){
                std::cout << "insert failed on 25";
        }

        if (t->Head->Left->Left->Value != 10){
                std::cout << "insert failed on 10";
        }*/
        //std::cout<< "test";
        return t;
}

AVLTree<int>* TestLeftRotate(){
        AVLTree<int>* t = new AVLTree<int>();
        /*std::vector<int> values = {50, 51, 52,9,8,7,6,5,4,3};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }*/

        t->insert(50);
        t->insert(51);
        t->insert(48);
        t->insert(47);

        //t->print();

        t->rotateLeft(t->Head, t->Head->Right);


        return t;

}


AVLTree<int>* TestRightRotate(){
        AVLTree<int>* t = new AVLTree<int>();
        /*std::vector<int> values = {50, 51, 52,9,8,7,6,5,4,3};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }*/

        t->insert(50);
        t->insert(51);
        t->insert(48);
        t->insert(47);

        //t->print();

        t->rotateRight(t->Head, t->Head->Left);
```

```

        return t;

}


AVLTree<int>* TestDelete() {
        AVLTree<int>* t = new AVLTree<int>();
        std::vector<int> values = {50,49,48,47,46,45,44,43,42,41};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
                //t->print();
        }


        AVLNode<int>* q = t->find(44);
        t->print();
        t->erase(q);
        //t->print();
        return t;
}

AVLTree<int>* TestFind(){
        AVLTree<int>* t = new AVLTree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }
        AVLNode<int>* p = t->Head->Left->Left;
        AVLNode<int>* output = t->find(10);
        if (p != output){
                return t;
        }
        AVLTree<int>* q = new AVLTree<int>(); //it worked!
        AVLNode<int>* goodHead = new AVLNode<int>(666, nullptr, nullptr, nullptr
);
        q->Head = goodHead;
        return q;
}


AVLTree<int>* TestCopConstruct(){
        AVLTree<int>* t = new AVLTree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }

        AVLTree<int>* alsot = new AVLTree<int>(*t);
        //t->print();
        //alsot->print();

        AVLTree<int>* q = new AVLTree<int>(); //it worked!
        AVLNode<int>* goodHead = new AVLNode<int>(666, nullptr, nullptr, nullptr
);
        q->Head = goodHead;
        return alsot;
}

AVLTree<int>* TestCopAssign(){
        AVLTree<int>* t = new AVLTree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }

        AVLTree<int>* alsot = new AVLTree<int>();
        std::vector<int> values2 = {1,2,3,4,5,6,99,11,525,1245};
        for (int i = 0; i < values2.size(); i++){
                alsot->insert(values2[i]);
        }


        alsot = new AVLTree<int>(*t);
        //t->print();
        //alsot->print();

        AVLTree<int>* q = new AVLTree<int>(); //it worked!
        AVLNode<int>* goodHead = new AVLNode<int>(666, nullptr, nullptr, nullptr
);
        q->Head = goodHead;
        return alsot;
}

/*
AVLTree<int>* TestHeight(){
        AVLTree<int>* t = new AVLTree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }
        int output = t->Head->SubTreeHeight();
        if (output == 4){
                AVLTree<int>* q = new AVLTree<int>(); //it worked!
                AVLNode<int>* goodHead = new AVLNode<int>(666, nullptr, nullptr,
 nullptr);
                q->Head = goodHead;
                return q;
        }
        return t;
}
*/

int main(){
        AVLTree<int>* t;
        t = TestInsert();
        t = TestLeftRotate();
        t = TestRightRotate();
        t = TestDelete();

        //t = TestFind();
        /*t = TestCopConstruct();
        t = TestCopAssign();
        //t = TestHeight();*/
        t->print();
        return 0;
}
```

```
#include "tree.hpp"
//#include "benchmark.hpp"
#include<iostream>
#include <vector>
#include <string>

Tree<int>* TestInsert(){
        Tree<int>* t = new Tree<int>();
        t->insert(50);
        t->insert(51);
        t->insert(52);
        t->insert(53);

//      t->print();

        /*
        if (t->Head->Value != 50){
                std::cout << "insert failed";
        }

        if (t->Head->Left->Value != 25){
                std::cout << "insert failed on 25";
        }

        if (t->Head->Left->Left->Value != 10){
                std::cout << "insert failed on 10";
        }*/
        return t;
}

Tree<int>* TestDelete() {
        Tree<int>* t = new Tree<int>();
        t->insert(50);
        t->insert(25);
        t->insert(100);
        t->insert(10);
        t->insert(75);
        t->insert(76);
        t->insert(74);
        //t->print();

        t->erase(t->Head->Right->Left->Right);
        return t;
}

Tree<int>* TestFind(){
        Tree<int>* t = new Tree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }
        Node<int>* p = t->Head->Left->Left;
        Node<int>* output = t->find(10);
        if (p != output){
                return t;
        }
        Tree<int>* q = new Tree<int>(); //it worked!
        Node<int>* goodHead = new Node<int>(666, nullptr, nullptr, nullptr);
        q->Head = goodHead;
        return t;
}


Tree<int>* TestCopConstruct(){
        Tree<int>* t = new Tree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }
        Tree<int>* alsot = new Tree<int>(*t);
        //t->print();
        //alsot->print();

        Tree<int>* q = new Tree<int>(); //it worked!
        Node<int>* goodHead = new Node<int>(666, nullptr, nullptr, nullptr);
        q->Head = goodHead;
        return alsot;
}

Tree<int>* TestCopAssign(){
        Tree<int>* t = new Tree<int>();
        std::vector<int> values = {50, 25, 100, 10, 75, 76, 74};
        for (int i = 0; i < values.size(); i++){
                t->insert(values[i]);
        }

        Tree<int>* alsot = new Tree<int>();
        std::vector<int> values2 = {1,2,3,4,5,6,99,11,525,1245};
        for (int i = 0; i < values2.size(); i++){
                alsot->insert(values2[i]);
        }


        alsot = new Tree<int>(*t);
        //t->print();
        //alsot->print();

        Tree<int>* q = new Tree<int>(); //it worked!
        Node<int>* goodHead = new Node<int>(666, nullptr, nullptr, nullptr);
        q->Head = goodHead;
        return alsot;
}



int main(){
        Tree<int>* t;
        t = TestInsert();
        t = TestDelete();
        t = TestFind();
        t = TestCopConstruct();
        t = TestCopAssign();
        //t->print();
        return 0;
}
```

```
#include <chrono>
#include <iostream>
#include <random>
#include <vector>
#include "tree.hpp"


void TestInsert(){
 std::mt19937 prbg;

  for (int n = 1000; n <= 500000; n += 10000) {

    // Get the starting time point. The type is deduced because it’s hard
    // to spell (it is std::chrono::system_clock::time_point).
    auto start = std::chrono::system_clock::now();

    // The actual test.
    Tree<int>* tree = new Tree<int>();
    //std::vector<int> seq;
    for (int i = 0; i < n; ++i) {
      std::uniform_int_distribution<int> rand(0, i);

      //int num = rand(prbg);                // Generate a random number
      tree->insert(i);
      //auto iter = linear_search(seq, num); // Find the insertion point
      //seq.insert(iter, num);
    }

    // Get the current system time in nanoseconds.
    auto stop = std::chrono::system_clock::now();

    // Print the number of nanoseconds each test takes.
    std::cout << n << "," << (stop - start).count() << std::endl;
  }

}

void TestFind(){
 std::mt19937 prbg;

  for (int n = 1000; n <= 50000; n += 1000) {

    Tree<int>* tree = new Tree<int>();
    std::vector<int> seq;
    for (int i = 0; i < n; ++i) {
      std::uniform_int_distribution<int> rand(0, i);

      //int num = rand(prbg);                // Generate a random number
      tree->insert(i);
      //auto iter = linear_search(seq, num); // Find the insertion point
      seq.push_back(i);
    }
    auto start = std::chrono::system_clock::now();
    for (int f = 0; f < seq.size(); f++){
        Node<int>* test = tree->find(f);
    }
    // Get the current system time in nanoseconds.
    auto stop = std::chrono::system_clock::now();

    // Print the number of nanoseconds each test takes.
    std::cout << n << "," << (stop - start).count() << std::endl;
  }

}


int main()
{
 // TestInsert();
```

```
  TestFind();
}
```

```
#include <chrono>
#include <iostream>
#include <random>
#include <vector>
#include "AVLtree.hpp"


void TestInsert(){
 std::mt19937 prbg;

  for (int n = 1000; n <= 50000; n += 1000) {

    // Get the starting time point. The type is deduced because it’s hard
    // to spell (it is std::chrono::system_clock::time_point).
    auto start = std::chrono::system_clock::now();

    // The actual test.
    AVLTree<int>* tree = new AVLTree<int>();
    //std::vector<int> seq;
    for (int i = 0; i < n; ++i) {
      std::uniform_int_distribution<int> rand(0, i);

      int num = rand(prbg);                // Generate a random number
      tree−>insert(num);
      //auto iter = linear_search(seq, num); // Find the insertion point
      //seq.insert(iter, num);
    }

    // Get the current system time in nanoseconds.
    auto stop = std::chrono::system_clock::now();

    // Print the number of nanoseconds each test takes.
    std::cout << n << "," << (stop − start).count() << std::endl;
  }

}

void TestFind(){
 std::mt19937 prbg;

  for (int n = 1000; n <= 50000; n += 1000) {

    AVLTree<int>* tree = new AVLTree<int>();
    std::vector<int> seq;
    for (int i = 0; i < n; ++i) {
      std::uniform_int_distribution<int> rand(0, i);

      int num = rand(prbg);                // Generate a random number
      tree−>insert(n);
      //auto iter = linear_search(seq, num); // Find the insertion point
      seq.push_back(n);
    }
    auto start = std::chrono::system_clock::now();
    for (int f = 0; f < seq.size(); f++){
        AVLNode<int>* test = tree−>find(f);
    }
    // Get the current system time in nanoseconds.
    auto stop = std::chrono::system_clock::now();

    // Print the number of nanoseconds each test takes.
    std::cout << n << "," << (stop − start).count() << std::endl;
  }

}

int main()
{
  //TestInsert();
  TestFind();
}
```