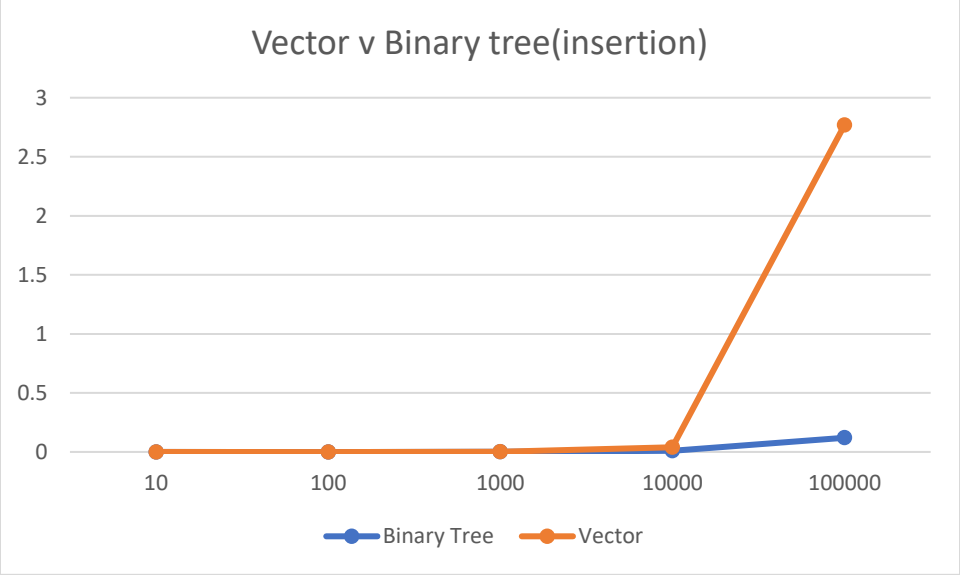
Hypothesis: Insertion into a vector with binary search will take log(n) time.

Methods: I used the Netbeans software version 8.2, I do not know the flags they use at compile time, I only cleaned, build, and ran the project. I took the iterative binary search algorithm from geeksforgeeks.com/binary-search. I created a base vector with n random numbers in it. I then iterated through that vector and inserted every number into a multiset and a vector that used binary search to find the correct index. I had to put the insertion loop into a for loop that iterated a set amount of times in order to get a non-zero value for n=10, 100, 1000, and 10000. For n=10, I looped 100000 times. For n=100, I looped 10000 times. For n=1000, I looped 1000 times. For n=10000, I looped 10 times. For each n, I compiled once and ran 10 times and averaged out the results.

Results:

n	B-Tree	Vector
10	0.000004864	0.000007674
100	0.00006971	0.00009776
1000	0.0008294	0.0012249
10000	0.01017	0.03067
100000	0.12	2.6494



Discussion: Once again, the graph is pretty useless since the data grows at such a fast rate for the selected values of n. The table is where we get the most information. Looking at the table, from n=10 to n=10000, it seems like vector insertion, using binary search and c++ vector insert, is logarithmic, then it gets to n=100000, and it turns into something completely different. I tried testing for n=1000000, but did not have the patience to see the result as I ran it for 7 minutes (480 seconds) before I killed the process. This could be because while the processors can move whole blocks of memory at once, once n gets to a certain size, it has to shift multiple blocks.

Conclusion: Under the conditions tested, insertions into a vector using binary search to find the insertion spot is bigO(n).

```cpp
#include <time.h>
#include <ctime>
#include <cstdlib>
#include<iostream>
#include<string>
#include<vector>
#include<set>
#include<unordered_set>

using namespace std;

/*
 *
 */

//Taken from geeksforgeeks.org/binary-search
int binarySearch(vector<int> arr, int l, int r, int x)
{
    while (l <= r) {

        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return l;
}

int main(int argc, char** argv) {

    srand(time(NULL));


    //multiset<int> b_tree;
```

```cpp
int main(int argc, char** argv) {

    srand(time(NULL));


    //multiset<int> b_tree;
    vector<int> random_numbers;

    constexpr int SAMPLE_SIZE = 10000;
    constexpr int NUM_LOOPS = 10;
    constexpr int RANGE = SAMPLE_SIZE*10;

    //Random numbers to insert into vector and multiset
    for (int i=0;i<SAMPLE_SIZE;i++){
        int random_number = rand() % RANGE;
        random_numbers.push_back(random_number);
    }

    //Binary tree insert
    clock_t s_time = clock();
    for(int i = 0; i < NUM_LOOPS; i++){
        multiset<int> b_tree;
        for(int j = 0; j < random_numbers.size(); j++){
            b_tree.insert(random_numbers[j]);
        }
    }
    clock_t f_time = clock() - s_time;
    cout << "Multiset Insert Time: "
         << ((double) f_time) / (double) CLOCKS_PER_SEC
         << " seconds" << endl;

    //Vector insertion
    int index;
    s_time = clock();
    for(int i = 0; i < NUM_LOOPS; i++){
        vector<int> vec;
        vec.push_back(random_numbers[0]);
        for(int j = 1; j < random_numbers.size(); j++){
            index = binarySearch(vec,0, vec.size()-1, random_numbers[j]);
            vec.insert(vec.begin()+index, random_numbers[j]);
        }
    }
    f_time = clock() - s_time;
    cout << "Vector Insert Time: "
         << ((double) f_time) / (double) CLOCKS_PER_SEC
         << " seconds" << endl;
```