4_iteration_vs_recursion.cpp

```cpp
/*
    Program for Question 4: Iteration vs Recursion
    We are given an array a of length n. Our task is to implement a binary search algorithm
    using a recursive strategy, and an iterative strategy

    The algorithm for binary search is O(nlogn)
*/

#include <iostream>
#include <vector>
#include <assert.h>

int recursiveBinarySearch(std::vector<int> &vec, int low, int high, int target) {
    if (low > high || vec.size() == 0) {
        return -1;
    }

    int mid = low + (high - low) / 2;

    if (vec[mid] == target) {
        return mid;
    } else if (vec[mid] > target) {
        return recursiveBinarySearch(vec, low, mid - 1, target);
    } else {
        return recursiveBinarySearch(vec, mid + 1, high, target);
    }
}

int iterativeBinarySearch(std::vector<int> &vec, int low, int high, int target) {
    int mid;
    while (low <= high) {
        mid = low + (high - low) / 2;
        if (vec[mid] == target) {
            return mid;
        } else if (vec[mid] > target) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return -1;
}

// Test function
void testBinarySearch() {
    std::vector<std::vector<int>> testCases = {
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},        // Basic Test Case
        {2, 4, 6, 8, 10, 12, 14, 16, 18, 20},   // Element at the Beginning
        {1, 3, 5, 7, 9, 11, 13, 15, 17, 19},    // Element at the End
        {1, 3, 5, 7, 9, 11, 13, 15, 17, 19},    // Element Not Present
        {},                                     // Empty Array
        {7},                                    // Single Element Array (Found)
```

```cpp
        {3},                                // Single Element Array (Not Found)
        {1, 2, 2, 3, 4, 4, 4, 5, 6, 7},         // Array with Duplicates
    };

    std::cout << iterativeBinarySearch(testCases[0], 0, testCases[0].size(), 4) << std::endl;   // 3
    std::cout << iterativeBinarySearch(testCases[1], 0, testCases[1].size(), 2) << std::endl;   // 0
    std::cout << iterativeBinarySearch(testCases[2], 0, testCases[2].size(), 19) << std::endl;  // 9
    std::cout << iterativeBinarySearch(testCases[3], 0, testCases[3].size(), 20) << std::endl;  //-1
    std::cout << iterativeBinarySearch(testCases[4], 0, testCases[4].size(), 4) << std::endl;   //-1
    std::cout << iterativeBinarySearch(testCases[5], 0, testCases[5].size(), 7) << std::endl;   // 0
    std::cout << iterativeBinarySearch(testCases[6], 0, testCases[6].size(), 4) << std::endl;   //-1
    std::cout << iterativeBinarySearch(testCases[7], 0, testCases[7].size(), 2) << std::endl;   // 2

    std::cout << recursiveBinarySearch(testCases[0], 0, testCases[0].size(), 4) << std::endl;   // 3
    std::cout << recursiveBinarySearch(testCases[1], 0, testCases[1].size(), 2) << std::endl;   // 0
    std::cout << recursiveBinarySearch(testCases[2], 0, testCases[2].size(), 19) << std::endl;  // 9
    std::cout << recursiveBinarySearch(testCases[3], 0, testCases[3].size(), 20) << std::endl;  //-1
    std::cout << recursiveBinarySearch(testCases[4], 0, testCases[4].size(), 4) << std::endl;   //-1
    std::cout << recursiveBinarySearch(testCases[5], 0, testCases[5].size(), 7) << std::endl;   // 0
    std::cout << recursiveBinarySearch(testCases[6], 0, testCases[6].size(), 4) << std::endl;   //-1
    std::cout << recursiveBinarySearch(testCases[7], 0, testCases[7].size(), 2) << std::endl;   // 2

}

double sec() {
    return double(clock())/double(CLOCKS_PER_SEC);
}

int main() {
    // testBinarySearch();

    std::vector<int> a = { 0,1,2,3,4,5,6,7,8,9,10 };
    int n = a.size();
    int K = 20000000;

    double T1 = sec();
    for (int j = 0; j < K; j++)
        for (int i = 0; i < n; i++)
                if (iterativeBinarySearch(a, 0, n, i) != i)
                    std::cout << "\nERROR";
    double T2 = sec();

    double T3 = sec();
    for (int j = 0; j < K; j++)
        for (int i = 0; i < n; i++)
            if (recursiveBinarySearch(a, 0, n, i) != i)
                std::cout << "\nERROR";
    double T4 = sec();

    std::cout << "Run time of iterative binary search ran " << K << " times: " << T2 - T1 << "s" << std::endl;
    std::cout << "Run time of recursive binary search ran " << K << " times: " << T4 - T3 << "s" << std::endl;

    return 0;
```

```
}

/*
    CONCLUSION:

    After timing both the iterative and recursive versions of binary search, I found out that
    recursive search is slower than iterative search by a factor of 1.5x.
    Recursion is slower because it has to call a function (itself) when it traverses the tree. This adds overhead to
    the calculation. Unlike the recursive approach, the iterative approach just keeps track of a couple of variables and keeps
    searching without the need to call itself.
*/
```