## 4\_iteration\_vs\_recursion.cpp

```
1 /*
 2
       Program for Question 4: Iteration vs Recursion
 3
       We are given an array a of length n. Our task is to implement a binary search algorithm
 4
       using a recursive strategy, and an iterative strategy
 5
 6
       The algorithm for binary search is O(nlogn)
 7
    */
 8
 9
    #include <iostream>
    #include <vector>
10
11
    #include <assert.h>
12
13
    int recursiveBinarySearch(std::vector<int> &vec, int low, int high, int target) {
14
       if (low > high || vec.size() == 0) {
15
         return -1;
       }
16
17
18
       int mid = low + (high - low) / 2;
19
20
       if (vec[mid] == target) {
21
         return mid;
22
       } else if (vec[mid] > target) {
23
         return recursiveBinarySearch(vec, low, mid - 1, target);
24
       } else {
25
         return recursiveBinarySearch(vec, mid + 1, high, target);
26
       }
27
    }
28
29
    int iterativeBinarySearch(std::vector<int> &vec, int low, int high, int target) {
30
       int mid;
31
       while (low <= high) {
         mid = low + (high - low) / 2;
32
33
         if (vec[mid] == target) {
34
            return mid;
35
         } else if (vec[mid] > target) {
            high = mid - 1;
36
37
         } else {
            low = mid + 1;
38
39
         }
40
       }
41
       return -1;
42
    }
43
44
    // Test function
45
    void testBinarySearch() {
       std::vector<std::vector<int>> testCases = {
46
                                          // Basic Test Case
47
         \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\},\
         {2, 4, 6, 8, 10, 12, 14, 16, 18, 20}, // Element at the Beginning
48
         {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}, // Element at the End
49
         {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}, // Element Not Present
50
                                   // Empty Array
51
         {},
         {7},
                                    // Single Element Array (Found)
52
```

1 of 3 3/6/24, 23:31

```
53
                                     // Single Element Array (Not Found)
 54
          \{1, 2, 2, 3, 4, 4, 4, 5, 6, 7\},\
                                           // Array with Duplicates
 55
        };
 56
 57
        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
 58
 59
        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
 60
 61
        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
 62
        std::cout << iterativeBinarySearch(testCases[6], 0, testCases[6].size(), 4) << std::endl; //-1
 63
        std::cout << iterativeBinarySearch(testCases[7], 0, testCases[7].size(), 2) << std::endl; // 2
 64
 65
        std::cout << recursiveBinarySearch(testCases[0], 0, testCases[0].size(), 4) << std::endl; // 3
 66
        std::cout << recursiveBinarySearch(testCases[1], 0, testCases[1].size(), 2) << std::endl; // 0
 67
        std::cout << recursiveBinarySearch(testCases[2], 0, testCases[2].size(), 19) << std::endl; // 9
 68
        std::cout << recursiveBinarySearch(testCases[3], 0, testCases[3].size(), 20) << std::endl; //-1
 69
 70
        std::cout << recursiveBinarySearch(testCases[4], 0, testCases[4].size(), 4) << std::endl; //-1
 71
        std::cout << recursiveBinarySearch(testCases[5], 0, testCases[5].size(), 7) << std::endl; // 0
 72
        std::cout << recursiveBinarySearch(testCases[6], 0, testCases[6].size(), 4) << std::endl; //-1
 73
        std::cout << recursiveBinarySearch(testCases[7], 0, testCases[7].size(), 2) << std::endl; // 2
 74
 75
     }
 76
 77
     double sec() {
 78
        return double(clock())/double(CLOCKS PER SEC);
 79
     }
 80
 81
     int main() {
        // testBinarySearch();
 82
 83
 84
        std::vector<int> a = { 0,1,2,3,4,5,6,7,8,9,10 };
 85
        int n = a.size();
        int K = 200000000;
 86
 87
 88
        double T1 = sec();
 89
        for (int j = 0; j < K; j++)
 90
          for (int i = 0; i < n; i++)
 91
               if (iterativeBinarySearch(a, 0, n, i) != i)
 92
                  std::cout << "\nERROR";
 93
        double T2 = sec();
 94
 95
        double T3 = sec();
 96
        for (int j = 0; j < K; j++)
 97
          for (int i = 0; i < n; i++)
 98
             if (recursiveBinarySearch(a, 0, n, i) != i)
 99
               std::cout << "\nERROR";
100
        double T4 = sec();
101
102
        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"
103
     << std::endl;
104
105
        return 0;
```

2 of 3 3/6/24, 23:31

106	}
107	
108	<i>/</i> *
109	CONCLUSION:
110	
111	After timing both the iterative and recursive versions of binary search, I found out that
112	recursive search is slower than iterative search by a factor of 1.5x.
113	Recursion is slower because it has to call a function (itself) when it traverses the tree. This adds overhead to
114	the calculation. Unlike the recursive approach, the iterative approach just keeps track of a couple of variables and keeps
115	searching without the need to call itself.
116	*/

3 of 3 3/6/24, 23:31