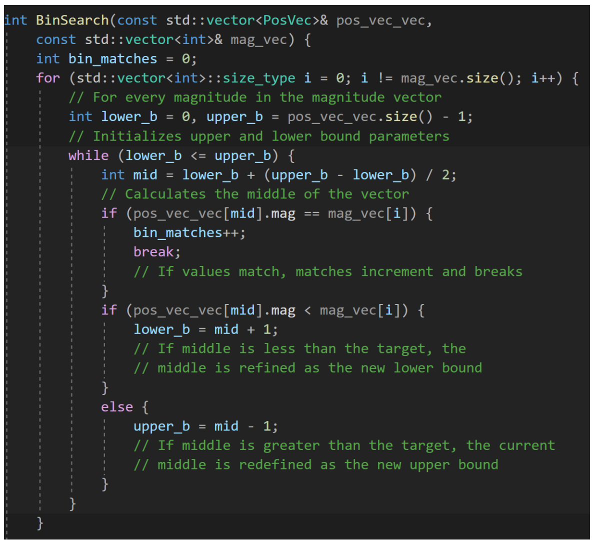
Bradley Manzo
April 8th, 2022
Prof. Porquet
ECS 36C A03 SQ

ECS 36C Programming Assignment 1 Report

Objective and Introduction:

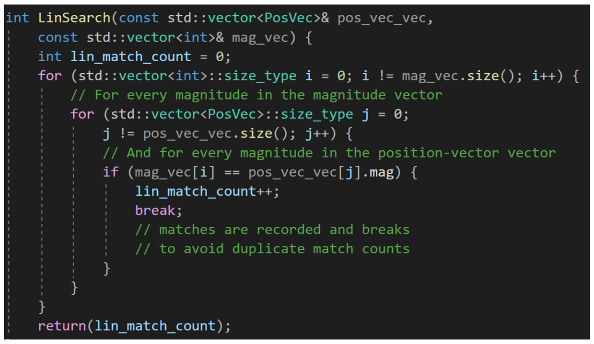
Tasked by the team of UC Davis biologists, I have written a program to search through their data set of positions and magnitudes in order to better understand the impact of climate change and its effect on wildlife during forest fires. The program allows the user to input a file of magnitudes, and a file of positions, and offers the user the choice of algorithm to calculate the number of matches in the set, printing the time to the console, and the number of matches to an output file. The two algorithms in question are referred to as Linear Search and Binary Search. By generating multiple sample sizes and recording the processing time for each algorithm, I intend on providing the team of biologists with a crash course in computational complexity as well as analyzing the pros and cons of each algorithm in practice

Procedure:



```cpp
int BinSearch(const std::vector<PosVec>& pos_vec_vec,
    const std::vector<int>& mag_vec) {
    int bin_matches = 0;
    for (std::vector<int>::size_type i = 0; i != mag_vec.size(); i++) {
        // For every magnitude in the magnitude vector
        int lower_b = 0, upper_b = pos_vec_vec.size() - 1;
        // Initializes upper and lower bound parameters
        while (lower_b <= upper_b) {
            int mid = lower_b + (upper_b - lower_b) / 2;
            // Calculates the middle of the vector
            if (pos_vec_vec[mid].mag == mag_vec[i]) {
                bin_matches++;
                break;
                // If values match, matches increment and breaks
            }
            if (pos_vec_vec[mid].mag < mag_vec[i]) {
                lower_b = mid + 1;
                // If middle is less than the target, the
                // middle is refined as the new lower bound
            }
            else {
                upper_b = mid - 1;
                // If middle is greater than the target, the current
                // middle is redefined as the new upper bound
            }
        }
    }
```

Figure 1: Binary Search



```cpp
int LinSearch(const std::vector<PosVec>& pos_vec_vec,
    const std::vector<int>& mag_vec) {
    int lin_match_count = 0;
    for (std::vector<int>::size_type i = 0; i != mag_vec.size(); i++) {
        // For every magnitude in the magnitude vector
        for (std::vector<PosVec>::size_type j = 0;
            j != pos_vec_vec.size(); j++) {
            // And for every magnitude in the position-vector vector
            if (mag_vec[i] == pos_vec_vec[j].mag) {
                lin_match_count++;
                break;
                // matches are recorded and breaks
                // to avoid duplicate match counts
            }
        }
    }
    return(lin_match_count);
```

Figure 2: Linear Search

When analyzing algorithms, it is useful to measure complexity using Big O notation: a mathematical concept which characterizes the behavior of a function in relation to its input size. Furthermore Big O can help us classify different algorithms in terms of run-time and memory complexity which is crucial to our Biologists. Linear search has a linear growth rate of O(N), meaning it has to perform an operation on every input, while binary search has a logarithmic growth rate of O(log(N)), meaning that with every successive operation, the number of inputs is halved. Using my program I will obtain an average run time from three trials for each data size of equal amounts of magnitudes and position vectors. I will then plug these into both algorithms and I will plot the magnitude of growth on a graph.

Experiment:

	10 elements	100 elements	1000 elements	10000 elements
Binary Search	3.603 us	40.981 us	556.568	1,313.16 us
Linear Search	3.179 us	214.08 us	8,117.06 us	362,340.67 us

Table 1: Simulation runtime measurements

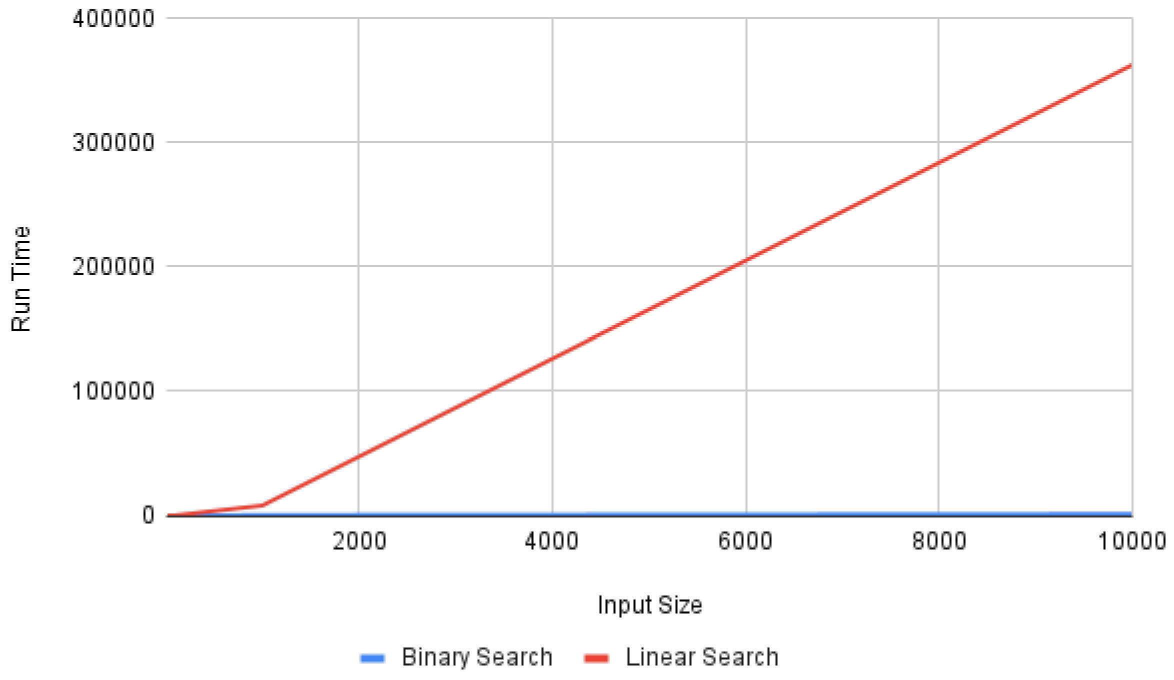


Figure 3: Graph of Run Time over Input Size

Results and Conclusion:

After simulating the runtimes in *Table 1* over 4 magnitudes, the differences in the performance of the algorithms are glaring. While it is true that the linear search's best case proves faster than binary search for small sample sizes, binary search has a much faster processing time for larger magnitudes of data. However my results also prove that my algorithms are not purely linear or logarithmic. A true linear relationship would have a constant growth rate, and a constant ratio between output and runtime, however we can see that its growth rate is increasing exponentially. This can be attributed to the nature of the assignment; since we look for a matching position magnitude input for every magnitude input, the complexity actually becomes $O(N) \cdot O(N) = O(N^2)$. Likewise, as we perform a binary search for the position magnitude for every magnitude, the computational complexity becomes $O(N) \cdot O(log(N)) = O(Nlog(N))$. This change is less noticeable however since log(N) overpowers the influence of N at large scales. While my algorithms are not entirely pure linear or logarithmic, they still demonstrate the massive difference in runtime for identical sample sizes, proving the importance of optimizing algorithms for their intended functionality.