First I had to see if an array of size 1 million, 10 million, and 100 million was possible. Making arrays that big was possible but I had to make them as dynamic arrays, storing them on the heap to prevent stack overflow. I then filled the arrays using a for loop. Each time the i is incremented, I set n equal to n + rand() % 9 + 0, which would randomly increment the value of n by any numbers between 0 and 9. This filled the arrays with random numbers going from least to greatest.

For all the binary search functions, they take in an array, an int “left” which is the first position, an int “right” which is the last position, and an int “x” which is the number we are looking for in the array. For the iterative search, the while condition is while right does not equal to left or in other words, while the first position does not equal to the last position, check if the middle element is the number we are looking for, else bound the array. For the recursive search it’s the same logic just applied recursively.

When I thought of a linked-type binary search, I thought about linked lists, which are nodes pointing to another node. I also know that for arrays, all the elements are next to each other in memory. So for the linked-type binary search I made the function similar to the iterative function but changed the while condition to have it so that the loop checks if the first element is right next to the last element in memory. So the addresses of the two elements are being compared instead of two int values. This also meant I had to change the if condition to also check if the first or last element is the number we are looking for .

By testing the speed of all three functions, I found that the iterative binary search is the fastest, followed by the linked-type binary search and the iterative binary search which was the slowest of the three.