

Chapter 32 – Binary Search

1. Binary Search of an Ordered List

Binary search of an ordered (sorted) list is much more efficient than any traversal search. Using binary search, one search of a sorted list can be performed in $\log_2 n$ time for the worst case. This is a significant improvement over the worst case n time of a single search of an ordered list using the traversal search.

The idea behind the binary search is based on the game of "guess a number". In the game, one player selects an integer between two limits and the other attempts to guess the number. For each guess, the first player must tell the second player if the guess is high or low. As anyone who has played this game knows, the most productive strategy is to start by selecting the number in the middle between the two limits. If the guess is low, the guess becomes the new high limit. If the guess is high, the guess becomes the new low limit. In this manner, the second player eliminates half of the remaining possibilities with each guess.

Binary search of an ordered list stored in an array works the same way. It "guesses" the index of the element sought by choosing the index in the middle of the limits, then checks the value stored at that index to see if the item sought is at that location or if the value at that location is higher or lower than the value being sought. If the list is n long, the algorithm starts by selecting the index $\frac{1}{2}n$ and comparing the key value stored there against the item being sought. If the value stored at the index is higher than the value being searched for, index-1 becomes the new upper limit in which to search. (In effect, it becomes the new n .) If the value stored at the index is lower than the value being searched for, index+1 becomes the new minimum limit. Next, the index between the new upper and lower index limit pair is selected and the process is repeated. This goes on until a matching value is found or the upper and lower limits cross, indicating that there is no match.

The following function is an example of the binary search algorithm. It will return the index of the element containing the item sought or, if the item is not found in an element of the array, a value of -1.

```

long bsearch(elementtype array[ ], int listlen, elementtype item) {
    long first = 0;
    long last = listlen-1;
    long mid;
    while (first <= last) {
        mid = (first + last) / 2;
        if (array[mid] < item) first = mid + 1;
        else if (array[mid] > item) last = mid - 1;
        else return mid;
    }
    return -1;
}

```

Analysis on Time: Using binary search, one search of a sorted list can be done in $\log_2 n$ time in the worst case. If the search is repeatedly carried out, this means that a binary search algorithm grows in time at $O(n \log_2 n)$. How was this arrived at? The process is simple. The power of two that produces the number that is equal to or closest to but larger than n is the maximum number of iterations that will be involved in a single binary search. For example, 2^{10} is 1,024. A list of length above 2^9 (512) and equal to or less than 1,024 elements can be searched in at most 10 iterations using the binary search. $\log_2 1024$ reverses this and conveniently gives 10 as its answer. (If your calculator does not have a \log_2 function, you can calculate $\log_2 n$ by dividing $\log_{10} n$ by $\log_{10} 2$.)

Programming Assignment 32.1

Alter the search routines of Programming Assignment 31.1 so that binary search is used. However, do not alter the insertion sort!