

SEARCHING A LIST OF NUMBERS (1)

```
// Search a list stored in an array
// maximum list size
const int MAXSIZE=100;
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
. . . .
void main ()
{
    // list to search
    int list[MAXSIZE];

    // length of list
    int size;

    // number to find in list
    int goal;
```

```
// position of number in list
int pos;
// file containing list
fstream listfile;
```

- Search a list stored as a one-dimensional array.
- Maximum size for the list is defined as a constant at the beginning of the program to make it easy to change.
- List of numbers read from a file.
- Search the list for user-specified goal numbers.

Programming 1

Lecture 14 -- 1

SEARCHING A LIST OF NUMBERS (2)

```
// read list from file
listfile.open("list.txt", ios::in);
size = 0;
listfile >> list[size];
while (!listfile.eof()) {
    size = size + 1;
    listfile >> list[size];
};
// search list for goal numbers
cout << "Enter a goal number: ";
cin >> goal;
while (goal >=0) {
    pos = search(list, size, goal);
    if (pos >=0) {
        cout << goal
            << " found at position "
            << pos << endl << endl;
    }
}
```

```
else {
    cout << goal
        << " not found"
        << endl << endl;
}
cout << "Enter goal number: ";
cin >> goal;
}
```

- Call function search to search the list.
 - returns the position in the list (subscript) if found
 - returns -1 if not found
- Loop until user enters a negative number.

Programming 1

Lecture 14 -- 2

LINEAR SEARCH

```
int search(int list[ ], int len, int target)
{
    int i;        // loop variable
    // loop to search list for target
    for (i=0; i<len; i=i+1) {
        if (list[i] == target) {
            // found it
            return i;
        }
    };
    // didn't find it so return -1
    return -1;
}
```

- ❑ Scan the list from beginning to end looking for the goal number element by element.
- ❑ When it is found, return the subscript of where it was found.
- ❑ Works for a list in any order.
- ❑ How much work is done?
 - don't find → look at all numbers in the list
 - find (worst case) → look at all numbers in the list
 - find (best case) → look at one number in list
 - find (average case) → look at half the numbers in the list

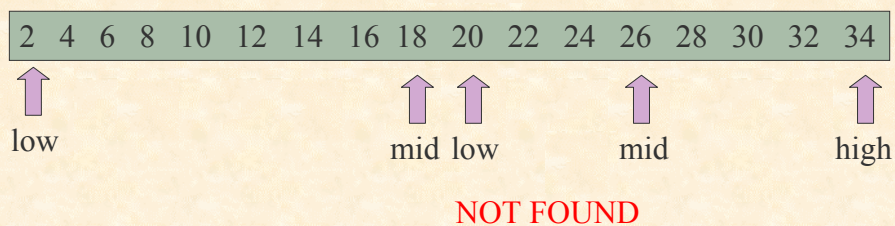
"BIG O" NOTATION

- ❑ Need a measure of the amount of work required by an algorithm relative to the size of the input.
 - allows comparison of different algorithms to solve the same problem
 - make it relative to the input size because you would expect all algorithms to do more work when given more input
- ❑ The linear search algorithm is said to be an $O(n)$ algorithm for both the average case and the worst case, where n is the length of the list to search.
 - $O(n)$ is read "order n "
 - this means that the amount of work done by the algorithm is proportional to $c*n$ where c is a constant
 - we don't necessarily know or care what a c is, but we know that it is a constant
 - c is the overhead of searching one number in the list

BINARY SEARCH - GOAL NOT IN LIST

- ❑ What happens when the goal number is not in the list.
- ❑ The high and low pointers will eventually switch so that the low pointer is to the right of the high pointer.
- ❑ This is the indicator that the goal number is not in the list.

Find 23 in the list:



Programming 1

Lecture 14 -- 7

BINARY SEARCH

```
int search(int list[ ], int len, int target)
{
    int low; // left end of search range
    int high; // right end of search range
    int mid; // midpoint of search range

    // initial search range
    low = 0;
    high = len-1;

    // loop to search list for target
    while (low <= high) {
        mid = (low+high)/2;
        if (list[mid] == target) {
            // found it
            return mid;
        }
    }
```

```
        else if (list[mid] < target) {
            // must be in upper half
            low = mid + 1;
        }
        else {
            // must be in lower half
            high = mid - 1;
        }
    };

    // didn't find it so return -1
    return -1;
}
```

- ❑ The rest of the program is the same as for linear search.

Programming 1

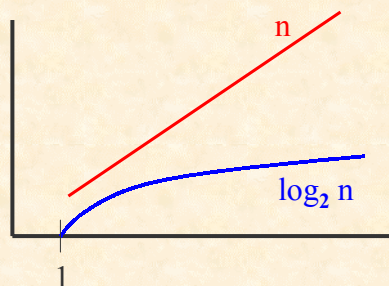
Lecture 14 -- 8

WHAT IS THE BIG O MEASURE FOR BINARY SEARCH ?

- ❑ To answer this question, we need to know how many iterations are done of the while loop.
- ❑ During each iteration of the loop, half the list is thrown away.
- ❑ In the worst case, we keep throwing away half the list until we get down to a list with one thing in it.
- ❑ How many times can you cut a list of length n in half until there is only one number left in the list ?
 - $n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow n/16 \rightarrow \dots$
- ❑ If n is a power of 2 to start (say $n = 2^k$, for some integer k), then you can divide n by 2 exactly k times before it becomes 1.
- ❑ Note that k is the logarithm of n for the base 2, i.e., $k = \log_2 n$.
- ❑ Thus, the binary search algorithm is $O(\log_2 n)$ where n is the length of the list.

COMPARISON OF LINEAR AND BINARY SEARCH

- ❑ Linear search is simple and $O(n)$, while Binary search is more complex and $O(\log_2 n)$, where n is the length of the list.



- ❑ As n gets large, the difference between n and $\log_2 n$ gets large.
 - this means that for long lists, binary search is much more efficient than linear search for searching a list
 - the extra complexity of binary search is worthwhile because of its significantly better performance