Search Algorithms

Common Problems

- There are some very common problems that we use computers to solve:
  - **Searching** through a lot of records for a specific record or set of records
  - **Sorting**, or placing records in a desired order
- At times we need to use both of these techniques as part of solving the same problem.

Common Problems

- There are numerous algorithms to perform searches and sorts.
- Over the remaining lessons in this course, we will briefly explore a few common search and sort algorithms.
  - We begin with search algorithms as applied to simple arrays.
  - Techniques can be extended to arrays of structures.

Search Algorithms

- Search: A search algorithm is a method of locating a specific item of information in a larger collection of data.
- There are two primary algorithms used for searching the contents of an array:
  - Linear or Sequential Search
  - Binary Search

Linear Search

- This is a very simple algorithm.
- It uses a loop to sequentially step through an array, starting with the first element.
- It compares each element with the value being searched for (key) and stops when that value is found or the end of the array is reached.

Searching an Array of structs

- Same process as for a simple array
- Use one of the fields to search

- Returns position/index in array as before

Linear Search

- Algorithm pseudocode:

```
set found to false; set position to –1; set index to 0
while index < number of elemts. and found is false
    if list[index] is equal to search value
        found = true
        position = index
    end if
    add 1 to index
end while
return position
```

Linear Search Function

```
int itemInArray(int arr, int item)
{
    int index = -1;

    for (int i = 0; i < arr.length; i++)
    {
       if (item == arr[i])
          { index = i;
            break;}
    }
    return index;
}
```

Linear Search Example

- Array `numlist` contains:

17	23	5	11	2	29	3

- Searching for the the value `11`, linear search examines `17, 23, 5,` and `11`
- Searching for the the value `7`, linear search examines `17, 23, 5, 11, 2, 29,` and `3`

Linear Search Tradeoffs

- Benefits:
  - Easy algorithm to understand
  - Array can be in any order
- Disadvantages:
  - Inefficient (slow): for array of N elements, examines N/2 elements on average for value in array, N elements for value not in array

Binary Search

Requires array elements to be in order

1. Divides the array into three sections:
   - middle element
   - elements on one side of the middle element
   - elements on the other side of the middle element
2. If the middle element is the correct value, done. Otherwise, go to step 1. using only the half of the array that may contain the correct value.
3. Continue steps 1. and 2. until either the value is found or there are no more elements to examine

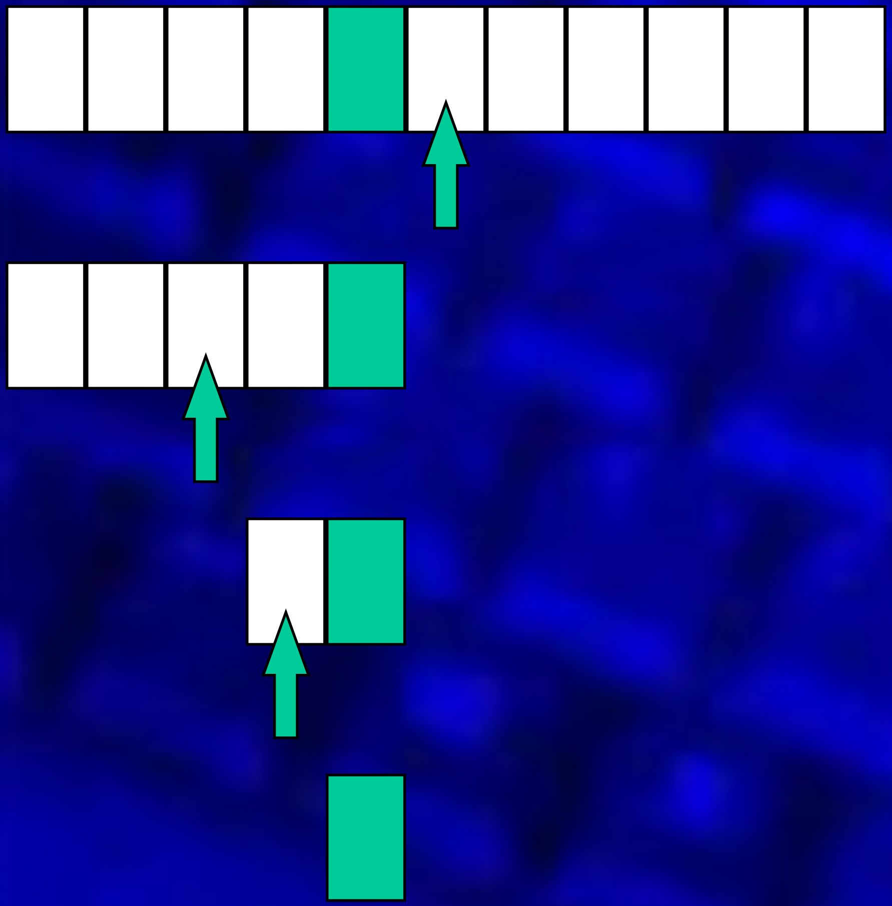
Binary Search Example

- Array `numlist2` contains:

2	3	5	11	17	23	29

- Searching for the the value `11`, binary search examines `11` and stops
- Searching for the the value `7`, linear search examines `11, 3, 5,` and stops

How a Binary Search Works



Always look at the center value. Each time you get to discard half of the remaining list.

Is this fast ?

main function

Binary Search Program

```
int main(void)
{
    int tests[] = {101, 142, 147, 189, 199,};
    int results, empID;
    cout<<"Enter the Employee ID you wish to search for: ";
    cin>>empID;
    results = binarySearch(tests,  empID);
    if (results == -1)
            cout<<"That number does not exist in the array.\n";
    else
    {
            cout<<"That ID is found at element"<< results;
            cout<<" in the array\n");
    }
    return 0;
}
```

Binary Search Program

```
int bsearch(int a int b)
{         if (a.length == 0) {
                   return -1;                }
          int low = 0;               int high = a.length-1;
          while(low <= high )
 {  int middle = (low+high) /2;
                   if (b> a[middle] )
{   low = middle +1;            }
else if (b< a[middle])
{   high = middle -1; }
else { // The element has been found
                          return middle;                 }
}         return -1;     }
```

How Fast is a Binary Search?

- Worst case: 11 items in the list took 4 tries
- How about the worst case for a list with 32 items ?
  - 1st try - list has 16 items
  - 2nd try - list has 8 items
  - 3rd try - list has 4 items
  - 4th try - list has 2 items
  - 5th try - list has 1 item

How Fast is a Binary Search?

- <u>List has 250 items</u>
- 1st try - 125 items
- 2nd try - 63 items
- 3rd try - 32 items
- 4th try - 16 items
- 5th try - 8 items
- 6th try - 4 items
- 7th try - 2 items
- 8th try - 1 item

- <u>List has 512 items</u>
- 1st try - 256 items
- 2nd try - 128 items
- 3rd try - 64 items
- 4th try - 32 items
- 5th try - 16 items
- 6th try - 8 items
- 7th try - 4 items
- 8th try - 2 items
- 9th try - 1 item

What's the Pattern?

- List of 11 took 4 tries
- List of 32 took 5 tries
- List of 250 took 8 tries
- List of 512 took 9 tries

- $32 = 2^5$ and $512 = 2^9$
- $8 < 11 < 16$ $2^3 < 11 < 2^4$
- $128 < 250 < 256$ $2^7 < 250 < 2^8$