Armstrong Atlantic State University
Engineering Studies
MATLAB Marina – Searching Primer

Prerequisites
The Searching Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, logic expressions, conditional structures, iteration, functions, debugging, characters and strings, cell arrays, structures, and file input and output. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, MATLAB Marina Arrays module, MATLAB Marina Logic Expressions module, MATLAB Marina Conditional Structures module, MATLAB Marina Iteration module, MATLAB Marina Functions module, MATLAB Marina debugging module, MATLAB Marina Character and Strings module, MATLAB Marina Cell Arrays module, MATLAB Marina Structures module, and MATLAB Marina File Input and Output module.

Learning Objectives
1. Be able to describe the operation of linear and binary searches.
2. Be able to use MATLAB's find function to search collections of primitive data types.
3. Be able to write search functions for collections of data of non-primitive data types such as cell arrays and structure arrays.

Terms
search, collection, key, linear search, binary search, sorted collection

MATLAB Functions, Keywords, and Operators
find

Searching
Searching involves going through a collection of data to find a match to a search criteria. Common searches are to find elements equal to, less than, or greater than a key (search value). The result of the search is an indication that a match with the key was found and may include the location in the collection where the match was found.

A linear search involves beginning with the first element of the collection and searching until a match is found or the end of the collection of data is reached. For example, the `findEqual` function shown in Figure 1 is a linear search. It searches a vector of numbers for all elements that equal a number. This could be modified to search for only the first match or to find the number of matches rather than the places.

MATLAB's `find` function is a search function that returns the indices corresponding to the non-zero entries of the array. The array passed to the `find` function is typically an array of Booleans (0s and 1s) that resulted from a logic expression.

```
function result = findEqual(data, key)
% -----------------------------------------------------
% findEqual.m
% -----------------------------------------------------
% findEqual determines the location (indices) of all
% instances of the key in a 1D array
% -----------------------------------------------------
% Syntax: result = findEqual(data, key)
%  data is the 1D array of numbers
%  key is the number to search for
%  result is an array of indices where matches were found
% -----------------------------------------------------
% Notes: an empty array is returned if no matches
% -----------------------------------------------------
n = 0;
result = [];
% go through the array and save indices matches are found at
for k = 1:1:length(data)
    if (data(k) == key)
        n = n + 1;
        result(n) = k;
    end
end
end
```

Figure 1, `findEqual` function (Linear Search)

If the collection of data is arranged in increasing or decreasing order (sorted collection), then more efficient searches can be performed. Think of how one would find a person's name in a phone book, does one start with the As if the name begins with an M?

To search a sorted collection of data:

- Start in the middle of the collection.
- If the element in the middle of the collection matches the key then the search is done.
- Otherwise determine which side of collection the key could be on and search that sub collection.
- Continue searching the sub collections until a match to the key is found or the size of sub collection is zero.

Figure 2 shows an example of a binary search function.

```
function result = findEqualBinary(data, key)
% -----------------------------------------------------
% findEqualBinary.m
% -----------------------------------------------------
% findEqualBinary determines the location (indices) of all
% instances of the key in a 1D array
% -----------------------------------------------------
% Syntax: result = findEqualBinary(data, key)
%  data is the 1D array of numbers
%  key is the number to search for
%  result is the index where match was found
% -----------------------------------------------------
% Notes: data is assumed to be in ascending order
%        only searches for first match
%        an empty array is returned if no match
% -----------------------------------------------------
low = 1;
high = length(data);
result = [];
found = 0;

% search data for key
while ((low <= high) && ~found)
    mid = floor((low + high)/2);
    if (data(mid) == key)
        % found the key
        result = mid;
        found = 1;
    elseif (key < data(mid))
        % search the left side
        high = mid - 1;
    else
        % search the right side
        low = mid + 1;
    end
end
end
```

Figure 2, `findEqualBinary` function (Binary Search)

Searching Example

Consider searching the list of integers [17, 7, 3, 14, 9, 2, 21, 8].

With a linear search:

- The search is started at the first element (index 1).
- The key is compared to the element (the 17).

- If the key matches the element, the search is complete otherwise go to the next element (index 2).
- The comparison and determination of done or move to next element is repeated until either an element matches the key or the end of the list is reached.

A search of the list [17, 7, 3, 14, 9, 2, 21, 8] for the key 14 would require four comparisons and index 4, the location of the 14, would be returned. A search of the list for the key 13 would require eight comparisons and no match would be found. The worst case for a list of size N requires N comparisons (either match is found at last index or a match is not found).

With a binary search an ordered (sorted) list is needed:

- The list sorted in ascending order is [2, 3, 7, 8, 9, 14, 17, 21]
- The search is started at the middle element (index 4.5, can use index 4 or 5 since index must be an integer). The middle element of a sorted list of numbers is the median value.
- The key is compared to the element (the 8).
- If the key matches the element, the search is complete. Otherwise search either the upper or lower half of the list. Search the upper sub list if the desired element is greater than middle element or search lower sub list if the desired element is less than middle element.
- The comparison and determination if done or search upper or lower portion of sub list is repeated until either an element matches the key or the sub list to search is empty.

A binary search of the list [2, 3, 7, 8, 9, 14, 17, 21] for the key 14 results in the following steps:

- Length of data is 8
- First sub list (the entire list), lower limit = 1, upper limit = 8, mid = 4.5 (4), compare key of 14 with element 4 (8), key is greater than element so search upper sub list
- Second sub list (the upper half), lower limit = 5, upper limit = 8, mid = 6.5 (6), compare key of 14 with element 6 (14), key matches the element so done and return index 6.

A binary search of the list [2, 3, 7, 8, 9, 14, 17, 21] for the key 13 results in the following steps:

- Length of data is 8
- First sub list (the entire list), lower limit = 1, upper limit = 8, mid = 4.5 (4), compare key of 13 with element 4 (8), key is greater than element so search upper sub list
- Second sub list (the upper half), lower limit = 5, upper limit = 8, mid = 6.5 (6), compare key of 13 with element 6 (14), key is less than element so search lower sub list.
- Third sub list (the lower half of the upper half), lower limit = 5, upper limit = 5, mid = 5 , compare key of 13 with element 5 (9), key is greater than element so search upper sub list.
- Upper sub list is empty so done and no match found, return an empty array.

The worst case of a list of size N requires N/2 comparisons, however a binary search can only be done on a sorted list.

Last modified Wednesday, November 13, 2013