CSc 10200
Introduction to Computing

Lecture 15
Edgardo Molina
Fall 2013 – City College of New York
A Closer Look: Searching & Sorting

- Sorting: Arranging data in ascending or descending order for some purpose
- Searching: Scanning through a list of data to find a particular item
Search Algorithms

- Searches can be faster if the data is in sorted order
- Two common methods for searching:
  - Linear search
  - Binary search
- Linear search is a sequential search
  - Each item is examined in the order it occurs in the list
- Average number of comparisons required to find the desired item is $n/2$ for a list of $n$ items
Linear Search

- Each item in the list is examined in the order in which it occurs
- Not a very efficient method for searching
- Advantage is that the list does not have to be in sorted order
- On average, the number of required comparisons is \( n/2 \), where \( n \) is the number of elements in the list
Linear Search (continued)

- Pseudocode for a linear search

```
For all items in the list
    Compare the item with the desired item
    If the item is found
        Return the index value of the current item
    Endif
EndFor
Return -1 if the item is not found
```
Binary Search

- Binary search requires that the list is stored in sorted order
- Desired item is compared to the middle element, with three possible outcomes:
  - Desired element was found: finished
  - Desired element is greater than the middle element, so discard all elements below
  - Desired element is less than the middle element, so discard all elements above
Binary Search (continued)

- Pseudocode for a binary search

```
Set the lower index to 0
Set the upper index to one less than the size of the list
Begin with the first item in the list
While the lower index is less than or equal to the upper index
    Set the midpoint index to the integer average of the lower and upper index values
    Compare the desired item to the midpoint element
    If the desired item equals the midpoint element
        Return the index value of the current item
    Else If the desired item is greater than the midpoint element
        Set the lower index value to the midpoint value plus 1
    Else If the desired item is less than the midpoint element
        Set the upper index value to the midpoint value less 1
    Endif
EndWhile
Return -1 if the item is not found
```
Binary Search (continued)

• Binary search algorithm in C++

    // this function returns the location of key in the list
    // a -1 is returned if the value is not found
    int binarySearch(int list[], int size, int key)
    {
        int left, right, midpt;
        left = 0;
        right = size - 1;

        while (left <= right)
        {
            midpt = (int) ((left + right) / 2);
            if (key == list[midpt])
            {
                return midpt;
            }
            else if (key > list[midpt])
                left = midpt + 1;
            else
                right = midpt - 1;
        }

        return -1;
    }
Binary Search (continued)

• On each pass of binary search, the number of items to be searched is cut in half
• After $p$ passes through the loop, there are $n/(2^p)$ elements left to search
Linear and Binary Search

<table>
<thead>
<tr>
<th>Array size</th>
<th>10</th>
<th>50</th>
<th>500</th>
<th>5000</th>
<th>50,000</th>
<th>500,000</th>
<th>5,000,000</th>
<th>50,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average linear search passes</td>
<td>5</td>
<td>25</td>
<td>250</td>
<td>2500</td>
<td>25,000</td>
<td>250,000</td>
<td>2,500,000</td>
<td>25,000,000</td>
</tr>
<tr>
<td>Maximum linear search passes</td>
<td>10</td>
<td>50</td>
<td>500</td>
<td>5000</td>
<td>50,000</td>
<td>500,000</td>
<td>5,000,000</td>
<td>50,000,000</td>
</tr>
<tr>
<td>Maximum binary search passes</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td>23</td>
<td>26</td>
</tr>
</tbody>
</table>

**Table 7.3** A Comparison of `while` Loop Passes for Linear and Binary Searches
Big O Notation

• Big O Notation
  – represents “the order of magnitude of”
• Sort algorithms come in two major categories:
  – Internal sort: entire list can be resident in memory at one time
  – External sort: for very large lists that cannot be totally in memory at one time
Sort Algorithms

- Two major categories of sorting techniques exist
  - **Internal sort**: Use when data list is small enough to be stored in the computer’s memory
  - **External sort**: Use for larger data sets stored on external disk
- Internal sort algorithms
  - Selection sort
  - Exchange sort
Selection Sort

• Smallest element is found and exchanged with the first element
• Next smallest element is found and exchanged with the second element
• Process continues $n-1$ times, with each pass requiring one less comparison
Selection Sort (continued)

• Pseudocode for a selection sort

Set exchange count to zero (not required, but done to keep track of the exchanges)
For each element in the list, from the first to the next to last
  Find the smallest element from the current element being referenced to the last element by:
    Setting the minimum value equal to the current element
    Saving (storing) the index of the current element
  For each element in the list, from the current element + 1 to the last element in the list
    If element[inner loop index] < minimum value
      Set the minimum value = element[inner loop index]
      Save the index value corresponding to the newfound minimum value
    Endif
  EndFor
  Swap the current value with the new minimum value
  Increment the exchange count
EndFor
Return the exchange count
Selection Sort (continued)

- Selection sort advantages:
  - Maximum number of moves that must be made is $n-1$
  - Each move is a final move

- Selection sort disadvantages:
  - $n(n-1)/2$ comparisons are always required
  - Order of magnitude of selection sort: $O(n^2)$
Exchange (Bubble) Sort

- Successive values in the list are compared
- Each pair is interchanged if needed to place them in sorted order
- If sorting in ascending order, the largest value will “bubble up” to the last position in the list
- Second pass through the list stops comparing at second-to-last element
- Process continues until an entire pass through the list results in no exchanges
Exchange (Bubble) Sort (continued)

• Pseudocode for an exchange sort

```
Set exchange count to zero (not required, but done to keep track of the exchanges)
For the first element in the list to one less than the last element (i index)
    For the second element in the list to the last element (j index)
        If num[j] < num[j - 1]
            Swap num[j] with num[j - 1]
            Increment exchange count
        EndFor
    EndFor
EndFor
Return exchange count
```
Exchange (Bubble) Sort (continued)

- Number of comparisons = $O(n^2)$
- Maximum number of comparisons: $n(n-1)/2$
- Maximum number of moves: $n(n-1)/2$
- Many moves are not final moves
Common Programming Errors

• Failing to declare the array
• Using a subscript that references a non-existent array element (out of bounds)
• Failing to use a counter value in a loop that is large enough to cycle through all array elements
• Failing to initialize the array
Summary

• Single dimension array is a data structure that stores a list of values having the same data type.
• Array elements are stored in contiguous memory locations, and referenced by array name and index position.
• Two-dimensional array has rows and columns.
• Arrays may be initialized when they are declared.
Summary (continued)

- Arrays may be passed to a function by passing the name of the array as the argument
- Individual array elements as arguments are passed by value
- Arrays passed as arguments are passed by reference, not by value