CS 106
Introduction to Computer Science I

10 / 30 / 2015

Instructor: Michael Eckmann
Today’s Topics

• Comments and/or Questions?
• Sorting
  – Bubblesort implementation
• Searching
  – Linear Search
  – Binary Search
Sorting (BubbleSort)

• Here's a graphical bubble sort algorithm.

• http://math.hws.edu/TMCM/java/xSortLab/

• There are many, many more sorting algorithms that are more efficient in that they work by making fewer comparisons. But these should give you a basic idea of sorting and how it could be accomplished.
Passing arrays into methods

• An array that is passed in as a parameter to a method CAN have its elements changed and the original array that was passed in reflects those changes.

• Recall: This is different than passing in variables of primitive types. If we pass in a variable of a primitive type, its value will remain the same when the method is complete, even if its corresponding parameter inside the method changes its value.

• Think of it this way: A COPY of a variable of a primitive type, is made when passed into a method. The method works with the copy and this copy is not “passed back out”.

• The value of the memory location of an array is passed into a method call (not the values.) The method works on the actual memory contents of the array (not a copy of the array) and therefore any changes to the values of the array will remain.
Sorting (BubbleSort)

• How might we write BubbleSort in Java? Any ideas?
• We assume the list is stored in an array.
• When we said we’d swap two elements of the list, how could we do this?
• What about how to do the correct number of passes?
public static void swap(int nums[],
    int idx1, int idx2 )
{
    int hold;

    hold        = nums[ idx1 ];
    nums[ idx1 ] = nums[ idx2 ];
    nums[ idx2 ] = hold;
}

Sorting (BubbleSort)

This is code to do ONE pass:

```java
for ( int idx = 0; idx < (list.length - 1); idx++ )
{
    if ( list[ idx ] > list[ idx + 1 ] )
    {
        swap( list, idx, idx + 1 );
    } // end if
}
```
public static void bubbleSort( int list[] )
{
    for ( int pass = 1; pass < list.length; pass++ )
    {
        for (int idx = 0; idx < (list.length - 1); idx++)
        {
            if ( list[ idx ] > list[ idx + 1 ] )
            {
                swap( list, idx, idx + 1 );
            } // end if
        } // end inner for loop (idx)
    } // end outer for loop (pass)
} // end method bubbleSort
An improvement to make this sorting algorithm not check the highest positions that are already sorted during each pass, could be made.

That is, the first pass goes through the whole array but after this happens the last position of the array is guaranteed to hold the highest value, so it doesn't need to be compared on the second pass (or any future pass.)

After the second pass, the highest two positions contain the highest two values so they both don't need to be compared on the third pass and so on...

Why is this an improvement?

What code could we change to implement this improvement?
Searching arrays

• Oftentimes it is necessary to search an array to find if there is a particular value in it.

• Assume we want to search an integer array for a particular value. What would be a good way to write a method to do this?

• Would there be any parameters? If so, what would they be?
• Would we return anything? If so, what would be good to return?
• What would we return if the value was not found?
Searching arrays

• Let's try to implement this method together.
Searching arrays (Linear search)

// Search array for specified key value

public static int linearSearch( int list[], int key )
{
    // loop through array elements
    for ( int cntr = 0; cntr < list.length; cntr++ )
    {
        // if array element equals key value,
        // return location
        if ( list[ cntr ] == key )
            return cntr;
    }
    return -1;  // key not found
}
Searching arrays (Linear search)

• Let’s analyze the linear search.
• We saw that sorting page counted the number of comparisons and the number of copies (or assignments). Let's consider comparisons here.

• How long (that is, how many comparisons) does it take to find the value?
  – What’s the minimum number of comparisons it would take?
  – What’s the maximum number of comparisons it would take?
  – Any idea on the average number of comparisons?
Searching arrays

• Can we do better? (That is, can we guarantee less comparisons on average to find a particular value?)
Searching arrays

• If the array was sorted could we modify the linear search in any way to stop sooner if we don't find the key?

• How might we implement the change to the Java code implementation of linear search?
Binary Search

• Now let's think beyond linear search.

• First question: Has anyone heard of binary search?

• What if we knew the array was sorted in ascending order?

• Could we use that to our advantage to reduce the number of comparisons a search algorithm would do on average?

• Yes --- how might you look up a person in the phone book?
Binary Search

• We could compare to the middle element of the array.

• If it is equal to the middle element, we’re done.

• If it is less than the middle element, where would we now concentrate our search?

• If it is greater than the middle element, where would we now concentrate our search?
Binary Search

- Any idea how we might write code to implement this algorithm?
Binary Search

• Any idea how we might write code to implement this algorithm?

• Let's discuss some ideas before we get right to the code.
  – What parameters might our method have?
  – What element to compare to first?
    • How do we calculate that index?
  – How do we determine what part of the array to now do a search?

• Let's take a look at an implementation and do an example call.
// method to perform binary search of an array
public static int binarySearch( int array2[], int key )
{
  int low = 0;                  // low element subscript
  int high = array2.length - 1;  // high element subscript
  int middle;                   // middle element subscript

  // loop until low subscript is greater than high subscript
  while ( low <= high )
  {
    // determine middle element subscript
    middle = ( low + high ) / 2;

    // if key less than middle element, set new high element
    else if ( key < array2[ middle ] )
      high = middle - 1;

    // key greater than middle element, set new low element
    else
      low = middle + 1;

    // if key matches middle element, return middle location
    if ( key == array2[ middle ] )
      return middle;

  } // end while loop

  return -1;   // key not found

}  // end method binarySearch
Searching arrays (Binary search)

• Let’s analyze the binary search.
• To simplify the discussion, we can count the 2 comparisons in the if/else/if/else together to be 1 comparison.
• How long (that is, how many comparisons) does it take to find the value?
  – What’s the minimum number of comparisons it would take?
  – What’s the maximum number of comparisons it would take?