Today’s Topics

• Comments and/or Questions?
• Sorting
  – Bubblesort implementation
Sorting

- Sorting is an important topic in computer science.
- We'll discuss what sorting is and several algorithms for doing it.
- There are a myriad of ways to sort.
- Arrays of data lend themselves easily to being sorted.
- Sorting can be done in ascending or descending order.
Sorting

• Original unsorted: 38, 41, 22, 12, 67
• We want the list to be: 12, 22, 38, 41, 67
• To accomplish this, one way would be to:
  – Go through the whole list once and find the lowest value.
  – Take it out and put it in a new list.
  – Go through the remaining list of numbers and find the lowest
  – Take that one out and put it at the end of the new list.
  – And so on ... until there's nothing left in the list.
• Does everyone agree that this will achieve a sorted list.
• We didn't say how exactly we'd find the lowest value each time ---
  the next slide describes a different way to sort and is more detailed.
Sorting

• Another way would be to:
  – compare the item in the first position to all the other items in the list and swap them in place if the item in the first position is greater than the item in the other position.
  – the second pass does the same thing but with the item in the second position.
  – third pass does the same but with the 3rd item.
  – and so on, until the last position is reached.

• Why would this work or not work?
Sorting

• This algorithm doesn’t require a new list like the first one mentioned. All the sorting is done within the one list.

• After the first pass, what is guaranteed about the list?

• After the second pass, what is guaranteed about the list?

• Etc.
Sorting

• Original unsorted: 38, 41, 22, 12, 67

• compare 38 to 41. 38 is not > 41, so leave them in place.
• compare 38 to 22. Swap them. So, now list is 22, 41, 38, 12, 67
• compare 22 to 12. Swap them. So, now list is 12, 41, 38, 22, 67
• compare 12 to 67. 12 is not > 67, so leave them in place.

• After first pass: 12, 41, 38, 22, 67
Sorting

• After first pass: 12, 41, 38, 22, 67
• Now start at second position.

• compare 41 to 38. Swap them. So, now list is 12, 38, 41, 22, 67
• compare 38 to 22. Swap them. So, now list is 12, 22, 41, 38, 67
• compare 22 to 67. Leave them.

• After second pass: 12, 22, 41, 38, 67
Sorting

• After second pass: 12, 22, 41, 38, 67

• Now start at third position.

• compare 41 to 38. Swap them. So, now list is 12, 22, 38, 41, 67

• compare 38 to 67. Leave them.

• After third pass: 12, 22, 38, 41, 67
• After third pass: 12, 22, 38, 41, 67

• Now start at fourth position.

• compare 41 to 67. Leave them.

• After fourth pass: 12, 22, 38, 41, 67

• Done now.
Another algorithm for sorting is the BubbleSort:
- Compares consecutive numbers in the list
- Swaps them if the two numbers are not in ascending order
- Compare each consecutive pair of numbers in the first pass.
- Next pass, start at first again but go only up until the next to last element, and so on…
- Do $n - 1$ passes, where $n$ is length of the list
Sorting (BubbleSort)

- Original unsorted: 38, 41, 22, 12, 67

- compare 38 to 41. 38 is not > 41, so leave them in place.
- compare 41 to 22. Swap them. So, now list is 38, 22, 41, 12, 67
- compare 41 to 12. Swap them. So, now list is 38, 22, 12, 41, 67
- compare 41 to 67. Leave them.

- After first pass: 38, 22, 12, 41, 67
Sorting (BubbleSort)

• After first pass: 38, 22, 12, 41, 67

• Now start at first position again.
• compare 38 to 22. Swap them. So, now list is 22, 38, 12, 41, 67
• compare 38 to 12. Swap them. So, now list is 22, 12, 38, 41, 67
• compare 38 to 41. Leave them.
• (don't need to compare the 4th and 5th positions) – why???
• After second pass: 22, 12, 38, 41, 67
Sorting (BubbleSort)

• After second pass: 22, 12, 38, 41, 67

• Now start at first position again.

• compare 22 to 12. Swap them. So, now list is 12, 22, 38, 41, 67

• compare 22 to 38. Leave them.

• After third pass: 12, 22, 38, 41, 67
• After third pass: 12, 22, 38, 41, 67

• Now start at first position again.

• compare 12 to 22. Leave them.

• Done.

• After fourth pass: 12, 22, 38, 41, 67
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.
• 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?