CECS 174 – Lecture 22 – Arrays Continued

Sorting Algorithms – There are many different types of algorithms that will sort an array of items (ints, doubles, strings, objects, etc.). Some algorithms are faster, some are more efficient, and some are just easier to program.

Starting with an array of 10 integers:

```
int [] array = {4, 2, 6, 5, 1, 3, 9, 0, 7, 8};
```

Selection Sort – An intuitive sorting algorithm that is easy to implement.

Algorithm:

1. Find the minimum value in the list.
2. Swap with the element in the first position.
3. Move to the next position and repeat.

```
public static void selectionSort(int [] array){
     for (int i=0; i<array.length; i++){
          int lowest = i;
          for( int j=i+1; j<array.length; j++) {
               if( array[j] < array[lowest]){
                    lowest = j;
               }
          }
          int swap = array[i];
          array[i] = array[lowest];
          array[lowest] = swap;
     }
}
```

On the first pass of the for loop, lowest is initialized to 0, and then the value stored in array[0] (4) is compared with the other elements of the array until a the lowest value is found, array[7] (0). The values of array[0] and array[7] are swapped resulting in:

```
array = {0, 2, 6, 5, 1, 3, 9, 4, 7, 8}
```

On the second pass, lowest is initialized to 1, and then the value stored in array[1](2) is compared with the other elements of the array until the lowest value is found, array[4](1). The values are swapped:

```
array = {0, 1, 6, 5, 2, 3, 9, 4, 7, 8}
```

This process continues until the outer for loop finishes and the list is sorted, it will take 10 passes through the loop to finish sorting.

Bubble Sort – An easy swapping algorithm that moves through the list switching any values that are out of order.

Algorithm –

1. Compare first two values, if first is greater than the second then swap.
2. Compare next set of values, if first is greater than the second, then swap.
3. Repeat step 2 until you reach the end of array.
4. Repeat above until sorted.

```
public static void bubblesort (int [] array){
     boolean swapped = false;
     do{
          swapped = false;
          for( int i=0; i<array.length-1; i++){
               if( array[i] > array[i+1]){
                    int swap = array[i];
                    array[i] = array[i+1];
                    array[i+1] = swap;
                    swapped = true;
               }
          }
     }while(swapped);
}
```

On every pass of the while loop, swapped is reset to false as a flag to signal when the array is sorted. The for loop iterates through the array swapping any items that are greater than the items to their right.

First pass: `array = {2, 4, 5, 1, 3, 6, 0, 7, 8, 9}`

Second pass: `array = {2, 4, 1, 3, 5, 0, 6, 7, 8, 9}`

Third pass: `array = {2, 1, 3, 4, 0, 5, 6, 7, 8, 9}`

Fourth pass: `array = {1, 2, 3, 0, 4, 5, 6, 7, 8, 9}`

Fifth pass: `array = {1, 2, 0, 3, 4, 5, 6, 7, 8, 9}`

Sixth pass: `array = {1, 0, 2, 3, 4, 5, 6, 7, 8, 9}`

Seventh pass: `array = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`

This algorithm only took 7 passes through the array to finish sorting the array. But in the worst case (starting from array = {9,8,7,6,5,4,3,2,1,0}) it would have also taken 10 passes thorough the array to finish sorting.

Gnome Sort – This algorithm is based on the idea that a garden gnome would want to sort different sized flowerpots by swapping them as he moves down the row.

Algorithm –

1. If you are at location 0, then move forward one.
2. Compare previous location with new location, if they are in order, move forward one, otherwise swap them and move back one.
3. Repeat until sorted.

```
public static void gnomeSort (int[] array){
     int loc = 0;
     while (loc<array.length){
          if(loc==0 || (array[loc-1] <= array[loc])){
               loc++;
          }else{
               int swap = array[loc];
               array[loc] = array[loc-1];
               array[loc-1] = swap;
               loc--;
          }
     }
}
```

On every pass of the while loop, the gnome moves back and forth through the array swapping any that are out of order, he will not reach the end of the array until it is sorted.

First pass: loc = 0, `array = {4, 2, 6, 5, 1, 3, 9, 0, 7, 8}`

Second pass: loc = 1, array[0] and array[1] are compared and then swapped
`array = {2, 4, 6, 5, 1, 3, 9, 0, 7, 8}`, loc is decremented.

Third pass: loc = 0

Fourth pass: loc = 1, array[0] and array[1] are compared, loc is incremented.

Fifth pass: loc = 2, array[1] and array[2] are compared, loc is incremented.

Sixth pass: loc = 3, array[2] and array[3] are compared, and then swapped
`array = {2, 4, 5, 6, 1, 3, 9, 0, 7, 8}`, loc is decremented.

Seventh pass: loc = 2, array[1] and array[2] are compared, loc is incremented.

Eighth pass: loc = 3, array[2] and array[3] are compared, loc is incremented.

Ninth pass: loc = 4, array[3] and array[4] are compared, and then swapped.
`array = {2, 4, 5, 1, 6, 3, 9, 0, 7, 8}`, loc is decremented.

Tenth pass: loc = 3, array[2] and array[3] are compared, and then swapped.
`array = {2, 4, 1, 5, 6, 3, 9, 0, 7, 8}`, loc is decremented…