Sorting Algorithms

As some more practice with arrays and the beginning of algorithms, here are three basic sorting algorithms. In our case we'll just be sorting arrays of integers, but they could be arrays of anything else. You can visualize many of these algorithms online at http://sorting.at

Bubble Sort

The idea behind bubble sort is to swap consecutive elements from left to right until the largest is at the right. We repeat this except we leave the last element "bubbled" to the right alone. Each pass through the array results in the largest item "bubbled" to the right.

(Demo in class)

```
void swap(int &a, int &b)
{
      int temp = a;
      a = b;
      b = temp;
}

void bubbleSort( int a[] , int n )
{
    for (int i = 0 ; i < n - 1 ; i++)
    {
        for (int j = 0 ; j < n - 1 - i; j++)
        {
            if ( a[j] > a[j + 1] )
            {
                swap(a[j],a[j+1]);
            }
        }
    }
}

// Test driver
int main()
{
  int a[] = {4, 5, 1, 354, 21, 45, 3};
  bubbleSort(a,7);
  for (int i = 0; i < 7; i++)
    cout << a[i] << endl;
}
```

Selection Sort

In selection sort we sweep through the array and find the smallest item and put it into index 0. Then we sweep through the array starting at index 1 and find the smallest item and put it into index 1. Then we sweep through the array starting at index 2 and find the smallest item and put it into index 2. Repeat!

(Demo in class)

```
void selectionSort(int a[], int n)
{
  for (int i = 0; i < n-1; i++)
  {
     int indexOfMin = i;
     for (int j = i+1; j < n; j++)
     {
        if (a[j] < a[indexOfMin])
                indexOfMin = j;
     }
     swap(a[i],a[indexOfMin]);
  }
}
```

Insertion Sort

Insertion sort is a little bit trickier to implement, but fairly simple in concept. The idea is to make sure the left side of the array is sorted as we move from left to right. First, start with one element. One element by itself is sorted, so there is nothing to do. Now, expand to two elements. We take the second element (called the key) then move it to the left if it's smaller than the first element, but leave it alone if it's bigger than the first element.

In general, we move the key to the left until we find a value that is smaller than it. If we ever move all the way to the left past the first element then the key should be inserted onto the front of the list and everything else needs to move over one slot.

(Demo in class)

```
void insertionSort(int a[], int n)
{
   for (int i = 1; i < n; i++)
   {
        int key = a[i];
        int j = i - 1;
        while ((j >= 0) && (a[j] > key))
        {
                a[j+1] = a[j];
                j--;
        }
        a[j+1] = key;
   }
}
```

Count Sort

Count sort is a little different than the above in that it doesn't make comparisons between elements. However, it only works if the items being sorted can be mapped to integers. We make an auxiliary array

that is able to hold the range of data values being sorted, loop through the values, and count how many times we see each one.

(Demo in class)

Simple version of count sort (there are fancier ones that are stable sorts). In this case we have to know that there are no larger values in the array than 500:

```
const int MAX = 500;
void countSort(int a[], int n)
{
  int count[MAX] = {0};

  for (int i = 0; i < n; i++)
     count[a[i]]++;
  for (int i = 0, j = 0; i < MAX; i++)
  while (count[i]>0)
  {
        a[j] = i;
        count[i]--;
        j++;
  }
}
```

There are many other sorting algorithms, including more efficient ones, like Mergesort, Quicksort, Heapsort, Radix sort, and others, which are discussed in more detail in the Algorithms and Data Structures course.