Some Sorting Algorithms

Bubble Sort

Let X be an array. In the last exercise we looked at the loop structure:

```
for(j=0; j<X.Length-1; j++)
    if(X[j]>X[j+1])
        MySwap(ref X[j],ref X[j+1]);
```

It was discovered that when executed, this loop moved the largest element to the last position in the array.

If it is executed twice, the two largest elements are at the end. Every time it is executed, we are assured that the next largest element is moved to its correct position in the array.

How many times would guarantee that the array is sorted in increasing order?

X.Length-1

So we can do the sort as follows:

```
for(i=1; i<=X.Length-1; i++)
    for(j=0; j<X.Length-1; j++)
        if(X[j]>X[j+1])
                MySwap(ref X[j],ref
                    X[j+1]);
```

The assumption here is that MySwap is a declared method that will swap element X[j] with X[j+1].

Note that once the largest element is moved to the end, there really is no need to ever consider it again. So we can make the following change.

```
for(i=1; i<=X.Length-1; i++)
   for(j=0; j<X.Length-i; j++)
      if(X[j]>X[j+1])
         MySwap(ref X[j],ref
           X[j+1]);
```

Cost of Bubble Sort

Let's count the total number of times that elements of the array are compared.

Suppose the length of the array is N.

When i is one, N-1 comparisons are done.

When i is two, N-2 comparisons are done.

When i is three, N-2 comparisons are done.

When i is k, N-k comparisons are done.

Adding all these up, we get

(N-1) + (N-2) + (N-3) + … + 1 and this is equal to

(N-1)*N/2 which is $N^2/2$ - N/2.

As N gets larger and larger, how does $N^2/2$ compare to N/2?

Selection Sort

Method: If N is the length of the array set some index variable i = N-1, the last index of the array. Find the index of the largest value from X[0] to X[i], and put that index in some variable k. Swap X[i] and X[k].

Now reduce i by one and repeat. Keep doing it for all i from N-1 to 1. We'll look at the more formal algorithm in class.

Cost of Selection Sort

Again we will count element comparisons. How many comparisons are required to find the largest element from X[0] to X[i]?

Now add them up, from i=N-1 down to i=1.

Doesn’t it look similar to bubble sort?

Insertion Sort (Algorithm)

This sort is inductive. Suppose you are trying to sort an array of length N. If you know that elements in X[0] through X[k] are already in increasing order, you can extend it to X[k+1] by saving X[k+1] and then moving those elements on the left that are larger than X[k+1] one place to the right. Then insert the saved element into the resulting “hole” in the array.

Example:

10 15 20 13 17 5 11

The first element not in increasing order is 13 which is in X[3]. So we save it. Then note that 20 is greater so it moves one to the right and 15 is greater and it moves one to the right. 10 is not greater. So with the moving we get:

10 15 15 20 17 5 11

Then we put 13 in the place where we stopped, to get:

10 13 15 20 17 5 11,

and the array is sorted to one more place.

So, how do you start? An array of length 1 is sorted. So we begin with insertion of the second element, then the third, fourth, and so forth.

The algorithm is on the next slide.

```
for(i=1; i<=X.Length-1; i++)
{
  key = X[i];
  j = i-1; /start one to the left
  while(j>=0 && X[j]>key){
     X[j+1]=X[j];
     j--;
  }
  X[j+1]=key;
}
```

Cost of Insertion Sort

The cost here is not quite as straight forward as the previous two, since the cost depends on the initial condition of the array. We will do this in class, and see that it can be as bad as some constant times N^2 but as good as some constant times N.

Cost Comparison

All three of these algorithms have one thing in common. Their worst case cost is some constant times N^2, where N is the length of the array.

In essence, if you double the size of the array, you quadruple the amount of work required to sort it in worst case.

Would these be good for sorting an array of length 1,000,000? Fortunately, there are more efficient sorting algorithms than these three.

Multi-subscripted Arrays

We will restrict ourselves to “two” for “multi”.

There are two kinds of double subscripted arrays in C#.

- Rectangular
- Jagged

Rectangular

All the rows have the same number of elements, and all the columns have the same number of elements; although, the column and row lengths may be different.

Declaring a rectangular array:

```
int[,] X;
X = new int[15,20];
```

X is now a rectangular array with 15 rows and 20 columns. X is an object.

GetLength(int) Method

A rectangular array object can call the method GetLength(int) to determine the number of rows and columns.

GetLength(0) is the first dimension, so for the previous example X.GetLength(0) is 15. X.GetLength(1) is 20.

These quantities are extremely useful in manipulating a rectangular array.

Passing to a Method

Passing a rectangular array to a method is much like passing a singly subscripted array.

Problem: Write a method that will receive a rectangular array and initialize it so that all the elements on row k have value k. In solving such a problem, we will make use of nested for loops, where the outer loop controls the row number and the inner loop controls the column number. You must become familiar with this technique.

```
static void Initialize(int[,] X)
{
   int i, j;
   for (i = 0; i < X.GetLength(0);i++)
   {
     for (j = 0; j < X.GetLength(1); j++)
        X[i,j] = i;
   }
}
```

Jagged Arrays

These are a bit more involved. Essentially each row can have a different length. They "are maintained as arrays of arrays".

Declaration:

```
int[][] c = new int[2][];
```

This will allow c to have two rows and the lengths can be different. Note: The 2 is supplied and is used to tell how many rows there are.

Jagged Array Example

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

The first line declares that Z is an array with 2 rows. The next two declare each row as one-dimensional arrays with length 5 and 3 respectively.

Array Information

In the previous example, the number of rows can be obtained with GetLength; for example:

Console.WriteLine(Z.GetLength(0));

would print 2.

However, GetLength(1) makes no sense, and an attempt to use it will result in a runtime error.

Row Lengths

Row lengths can be obtained individually by the following:

Z[0].Length (length of first row)

Z[1].Length (length of second row)

Accessing the Elements of a Jagged Array

The jth element in the ith row in a jagged array, say Z, is accessed by Z[i][j] and NOT by Z[i , j] as with rectangular arrays.

Keep in mind that a jagged array Z is an array of arrays.