July 19, 2010

Review of Past Sorting Algorithms

Bubble Sort

Bubble sort repeatedly finds two numbers which are out of order and swaps their positions. The numbers have to be next to each other. Then it looks for two more pairs which are out of order. This is repeated until the algorithm cannot find any more out of order pairs. The Bubble Sort is considered to be useful only for educational purposes. It's recommended that you never actually use the Bubble Sort.

Gap Sort

The gap sort is an algorithm which is similar to the Bubble Sort in that it is swapping values that are out of order. The difference is that there can exist a gap between the two values. The algorithm starts on the extreme ends of the array and works it's way inward.

More Sorting Algorithms

Last week we discussed the Bubble Sort and the Gap Sort. Today, we'll talk about four more sorting algorithms.

- Counting Sort
- Insertion Sort
- Selection Sort
- Shell Sort

Counting Sort

```
void counting_sort(int array[], int size)
{
   int i, min, max;

   min = max = array[0];
   for(i = 1; i < size; i++)
   {
      if (array[i] < min)
         min = array[i];
      else if (array[i] > max)
         max = array[i];
   }

   int range = max - min + 1;
   int *count = malloc(range * sizeof(int));

   for(i = 0; i < range; i++)
      count[i] = 0;

   for(i = 0; i < size; i++)
      count[ array[i] - min ]++;
   int j, z = 0;

   for(i = min; i <= max; i++)
      for(j = 0; j < count[ i - min ]; j++)
         array[z++] = i;


   free(count);
}
```

Selection Sort

When I'm thinking about sorting numbers, this method is the most intuitive. This is just my opinion. When I need a sorting algorithm, it is usually the one I implement first. It is by no means the fastest.

The Selection Sort works on the following principle:

First, divide the list into two parts:

- The sorted portion: At the beginning, this is a list of size 0 at the front of the array.
- The unsorted potion: At the beginning, this is the entire array.

Repeat until you process n - 1 elements, where n is the number of elements in the array. We say “n - 1” because if you sort n - 1 elements, the last element will be sorted automatically. Also, if there is exactly 1 element, nothing needs to be done.

1. Find the location of the smallest element in the unsorted portion. This can be done with a simple minimum finding algorithm that we have discussed earlier in the semester.
2. Swap the location of the minimum element with the first element in the unsorted portion.

Like the Bubble Sort and Insertion Sort, this is an O(N^2) algorithm.

Let's go over the pseudocode, which you will need to implement for your homework assignment:

```
function selection_sort(int[] a)
      for i = 0 to length(a)-1
            min_position = i


            // Find the min
            for j = i + 1 to length(a)-1
                  if a[j] < a[min_position]
                        min_position = j
                  end for
            end for


            // Swap values!
            temporary = a[i]
            a[i] = a[min_position]
            a[min_position] = temporary
      end for
end function
```

Insertion Sort

The Insertion Sort works on the following principle:

Imagine that there exists a mystery function that is outputting numbers. You know how many numbers the box will output, but you don't know what the values are. All you know is that you grab the next number out of the box and place it in the right spot, moving other numbers as necessary.

Like the Bubble Sort, this is an O(N^2) algorithm.

A list of exactly 1 number is always sorted. If our mystery function only outputs one function, there's not much use for the insertion sort. The insertion sort starts at the section number.

You can divide the unsorted array into three parts:

- Sorted portion: This is the first element in the list.
- Number to be inserted in the sorted potion: This is the second element in the list.
- Unsorted potion: This is the remaining values in the list.

Here are the basic steps which need to be taken to sort values:

1. If the number to be inserted in the sorted portion is larger than every value in the sorted portion (you can tell simply by looking at the last value of the sorted potion), then do nothing. Move to the next value in the unsorted potion.
2. If the number to be inserted in the sorted potion is not, do the following:
    1. Store the number to be inserted in the sorted potion into a temporary storage value.
    2. Starting with the tail end of the sorted portion, move values forward one at a time until a gap is created which house our number.
    3. Take the value stored in the temporary variable and put it into the gap.
    4. Move to the next unsorted element.

Let's go over the pseudocode, which you will need to implement for your homework assignment:

```
function insertion_sort(int[] a)
    for i = 1 to length(a)-1
        temporary = a[i]
        low_position = i - 1

        // Move elements forward
        while j >= 0 and temporary < a[low_position]
            a[low_position+1] = a[low_position]
            low_position = low_position - 1
        end while

        // Drop the unsorted element into the array.
        a[low_position] = temporary
    end for
end function
```

Shell Sort

```
import java.io.*;
import java.util.*;


public class ShellSort extends Sortable {
    private int[] a;

    public ShellSort(int[] a) {
        this.a = new int[a.length];
        for (int i = 0; i < a.length; i++) {
            this.a[i] = a[i];
        }
    }

    public int[] sort() {
        for (int gap = a.length / 2; gap >= 1; gap--) {
            for (int i = 0; i < gap; i++) {
                gapInsertionSort(i, gap);
            }
        }
        return a;
    }

    private void gapInsertionSort(int start, int gap) {
        for (int i = start + gap; i < a.length; i += gap) {
            int low_position = i - gap;
            int value = a[i];
            while (low_position >= start && value < a[low_position]) {
                a[low_position + gap] = a[low_position];
                low_position -= gap;
            }
            a[low_position + gap] = value;
        }
    }
}
```