CSCI 112
Monday, July 19, 2010

Counting Sort

- Very fast sort
  - works in **O(N+S)** time
  - **N**: number of elements to sort
  - **S**: range of elements from smallest to largest
- Works best when the range of values is small.

Code:

```
void countingSort( int[] a ){
        int min = a[0];
        int max = a[0];

        for( int i = 1; i < a.length; i++ ){
                if( a[i] > max )
                        max = a[i];
                if( a[i] < min )
                        min = a[i];
        }
        int range = max - min + 1;

        int[] count = new int[range];

        for( int i = 0; i < a.length; i++ ){
                count[a[i] - min]++;
        }

        int z = 0;

        for( int i = min; i <= max; i++ ){
                for( int j = 0; j <= count[i-min]; j++ ){
                        a[z++]=i;
                }
        }
}
```

Selection Sort

- done in $\mathbf{O(N^2)}$ time

1. Find the location of the smallest value in the unsorted portion.
2. Swap the location of the minimum with the first element in the unused portion.

Pseudo-Code

```
function selection_sort( int[] a )
        for i = 0 to length(a)
                min_position = i
                //find the min

                for j = i+1 to length(a)
                        if a[j] < a[min_position]
                                min_position = j
                end for
                //swap values
                temporary = a[i]
                a[min_position] = temporary
        end for
end function
```

Insertion Sort

- done in **$O(n^2)$** time

Three Parts of Insertion Sort

1. Sorted Portion
   - At the beginning, this is the first element
2. Next Number to Sort
   - At the beginning, this is the second element
3. Unsorted Portion
   - At the beginning, this contains everything after the second element
   -

Basic Steps for the Insertion Sort

- If the value to be inserted is larger than the largest portion of the sorted list, do nothing.
- If not, do the following:
  1. Store the value in a temporary and
  2. Starting with the tail, move values to the end one value at a time until the gap is created which will house our number.
  3. Take the value out of the temporary and add it to the gap.

Pseudo-Code

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

```
            while low_position >= 0 and temporary < a[low_position]
                a[low_position+1] = a[low_position]
                low_position = low_position - 1
            end while
            a[low_position+1] = temporary
        end for
end function
```

Shell Sort

- Improvement of the Insertion Sort
- runs in **O()** time

Steps for Shell Sort

Start with this list:

```
5 3 8 4 5 2
```

1. Divide the list in two

```
5 3 8
4 5 2
```

2. Perform insertion sort on each column

```
4 3 2
5 5 2
```

3. Divide list into three
4. Perform insertion sort on each column
5. Divide the list into four
6. Perform insertion sort on each column

Code

```
void gapInsertionSort(int[] a; int start, int gap){
        for(int i = start+gap; i < a.length; i++){
            int temp = a[i];
            int loaw_position = i - gap;
            while( low_position >= start && temp < a[low_position] ){
                a[low_position+gap] = a[low_position];
                low_position -= gap;
            }
            a[low_position+gap] = temp;
        }
}
void shell_sort( int[] a ){
        for(int gap = a.length/2; gap>=1; gap--){
            for( int i = 0; i <= gap; i++ ){
                gapInsertionSort(a, i, gap);
```

```
                }
            }
}
```