# Recursion & Sorting

CS 62 - Spring 2013
Michael J Bannister

# Lab & Assignment

- Build Frequency Lists:
  Draw pictures of the lists.
- Significantly harder than last lab. Use early mentor hours!
- Lab is to be done in pairs today.

# The Sorting Problem

- **Input:** An array of n numbers
- **Output:** An array of the same numbers as the input, but in non-decreasing order.

# Recursion

- Describe your problem in terms of smaller subproblems and a base case.
- Use this description to construct a recursive algorithm to solve the problem.

# Selection Sort Helper

private int indexOfLargest(int[] array, int endIndex)

- Finds and returns the index of the largest element in array with index less than or equal to endIndex.
- Uses a linear number of comparisons. O(n)

# Selection Sort

```
void selectionSort(int[] array, int endIndex) {
  if (endIndex > 0 ) {
      // find largest element in array[0...endIndex]
      int largest = indexOfLargest(array, endIndex);
      // move largest element to index endIndex
      swap(array, largest, endIndex);
      // sort everything in the array before endIndex
      selectionSort(array, endIndex - 1);
  }
}
```

- Uses $O(n^2)$ comparisons to sort an array of n elts.

# Selection Sort Analysis

- Need to prove correctness and time complexity.
- Use mathematical induction on length of the array.
- Consider the base cases of 0 and 1 element.
- Use the Gauss sum to compute the runtime.

# InsertionSort

- Similar: To sort array of n elements:
  - Sort first n-1 elements
  - Insert last element in correct position
- How long to insert new element into sorted list of n elements?

# Fast Exponentiation

```
public static int fastPower(int base, int exponent) {
  if(exponent == 0) {
    return 1;
  } else if(exponent % 2 == 1) {
    return base * fastPower(base, exponent - 1);
  } else {
    return fastPower(base * base, exponent/2);
  }
}
```