CS171Z: Sorting

October 21, 2013

`theArray` is an array of `long` integers

Insertion Sort

```
public static void sort() {
    insertion(theArray.length);  // sort the entire array
}

public static void insertion(int n){
    for (int i = 1; i < n; i++){
        int j = i;
        long next = theArray[i];
        while ((j > 0) && (theArray[j-1] > next)){
           // look for the right place to put next
           theArray[j] = theArray[j-1];
           j--;
        }
        theArray[j] = next;
    }
}
```

Bubble Sort

```
public static void bubble(int n){
    for (int i = 0; i < n; i++){
       for (int j = 1; j < (n-i); j++) {
           if (theArray[j-1] > theArray[j]) {
              // out of order
              long tmp = theArray[j-1];
              theArray[j-1] = theArray[j];
              theArray[j] = tmp;
          }
       }
    }
}
```

Recursive Sorting

Quick Sort

```
public static void sort() {
    sort(0, theArray.length - 1);  // sort the entire array
}

public static void sort(int left, int right) {
    // pre: left and right between 0 and theArray.length - 1
    // post: the portion of array between left and right sorted
    if(right-left <= 0)              // if size <= 1,
        return;                      //    already sorted
    else {                           // size is 2 or larger
        long pivot = theArray[right];// choose the rightmost item as pivot
        int partition = partitionIt(left, right, pivot);
        sort(left, partition-1);   // sort left side
        sort(partition+1, right);  // sort right side
    }
}

public static int partitionIt(int left, int right, long pivot) {
    // pre: left <= pivot <= right
    // post: left <= partition <= right, and
    //       all elements between left and partition <= pivot
    //       all elements between partition and right >= pivot
    int leftPtr = left - 1;
    int rightPtr = right;
    while(true) {
        while( theArray[++leftPtr] < pivot ) ; // keep looking
        // after the loop, a bigger or equal element found

        while(rightPtr > 0 && theArray[--rightPtr] > pivot) ; // keep looking
        // after the loop, a smaller or equal element found

        if(leftPtr >= rightPtr)       // if pointers cross,
            break;                    //    partition done
        else                          // not crossed, so
            swap(leftPtr, rightPtr);  //    swap elements
    }
    swap(leftPtr, right);           // restore pivot
    return leftPtr;                 // return pivot location
}  // end partitionIt()

public static void swap(int dex1, int dex2) { // swap two array elements
    long temp = theArray[dex1];        // A into temp
    theArray[dex1] = theArray[dex2];   // B into A
    theArray[dex2] = temp;             // temp into B
}
```

Merge Sort

```
public static void sort() {
    sort(0, theArray.length);
}

// Sort a[lo, hi).
public static void sort(int lo, int hi) {
    int N = hi - lo;        // number of elements to sort

    // 0- or 1-element file, so we're done
    if (N <= 1) return;

    // recursively sort left and right halves
    int mid = lo + N/2;
    sort(lo, mid);
    sort(mid, hi);

    // merge two sorted subarrays
    long [] aux = new long[N];
    int i = lo, j = mid;
    for (int k = 0; k < N; k++) {
        if (i == mid)
             aux[k] = theArray[j++];
        else if (j == hi)
             aux[k] = theArray[i++];
        else if (theArray[j] < theArray[i])
             aux[k] = theArray[j++];
        else
             aux[k] = theArray[i++];
    }

    // copy back to theArray
    for (int k = 0; k < N; k++) {
        theArray[lo + k] = aux[k];
    }
}
```