**public** **static** **void** sort(Comparable[] a) {

*aux* = **new** Comparable[a.length];

*sort* (a, 0, a.length-1);

}

**static** **void** sort (Comparable[] a, **int** lo, **int** hi) {

**if** (hi <= lo) **return**;

**// Truth:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**int** mid = lo + (hi - lo)/2;

*sort*(a, lo, mid);

*sort*(a, mid+1, hi);

**// Truth:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*merge*(a, lo, mid, hi);

}

// merge sorted results a[lo..mid] with a[mid+1..hi] back into a

**static** **void** merge (Comparable[] a, **int** lo, **int** mid, **int** hi) {

**int** i = lo; // starting index into left sorted sub-array

**int** j = mid+1; // starting index into right sorted sub-array

// copy a[lo..hi] into aux[lo..hi]

**for** (**int** k = lo; k <= hi; k++) {

*aux*[k] = a[k];

}

// now comes the merge. Something you might simulate with flashcards

// drawn from two stack piles. This is the heart of mergesort.

**for** (**int** k = lo; k <= hi; k++) {

**if** (i > mid) { a[k] = *aux*[j++]; }

**else** **if** (j > hi) { a[k] = *aux*[i++]; }

**else** **if** (*less*(*aux*[j], *aux*[i])) { a[k] = *aux*[j++]; }

**else** { a[k] = *aux*[i++]; }

**// Truth:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

}

**// Truth:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

}

In-class Example:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| E | M | R | G | E | S | O | R |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |