MergeSort
Divide and Conquer Algorithms

• A technique for designing algorithms where a solution is found by breaking the problem into smaller (similar) subproblems.

• The subproblems solutions are combined to form the solution to the original problem.

• Often implemented using recursion.
Divide and Conquer: Sorting

• Break the array to be sorted into smaller pieces,
• Process (sort) the pieces, and then
• Put them back together
• This is the idea behind *Mergesort*
Mergesort
Mergesort

• Pseudo code

```java
public static Array mergesort(Array inlist) {
    if (inlist.length <= 1) {
        return inlist;
    }
    Array L1 = half of the items from inlist;
    Array L2 = other half of the items from inlist;
    return merge(mergesort(L1), mergesort(L2));
}
```
Merge function

• Combines two pre-sorted lists into a sorted whole.
• The hardest step about Mergesort
• Algorithm:
  • Examine the first record of each sublist and picks the smaller value as the smallest record overall
  • The smaller value is removed from its sublist and placed into the output list
  • Merging continues in this way, comparing the front records of the sublists and continually appending the smaller to the output list until no more input records remain
public static int[] merge(int[] A, int[] B) {
    int[] tmp = new int[A.length + B.length];
    int i1 = 0; // will iterate through A
    int i2 = 0; // will iterate through B
    for (int i = 0; i < tmp.length; i++) {
        if (i1 >= A.length) { // A exhausted
            tmp[i] = B[i2++];
        } else if (i2 >= B.length) { // B exhausted
            tmp[i] = A[i1++];
        } else if (A[i1] <= B[i2]) { // Get smaller value
            tmp[i] = A[i1++];
        } else {
            tmp[i] = B[i2++];
        }
    }
    return tmp; // the sorted array
}
Mergesort implementation

• Problem: avoid having each merge operation to create a new array
• Solution: Use an auxiliary array
• The initial call
  • `mergesort(arrayToSort, auxiliaryArray, 0, n-1)`
  (n = arrayToSort.length)
Mergesort implementation

```java
static void mergesort(Comparable[] A, Comparable[] temp, int left, int right) {
    if (left == right) return; // List has one record
    int mid = (left+right)/2; // Select midpoint
    mergesort(A, temp, left, mid); // Mergesort first half
    mergesort(A, temp, mid+1, right); // Mergesort second half
    for (int i=left; i<=right; i++) // Copy subarray to temp
        temp[i] = A[i];
    // Do the merge operation back to A
    int i1 = left;
    int i2 = mid + 1;
    for (int curr = left; curr <= right; curr++) {
        if (i1 == mid+1) // Left sublist exhausted
            A[curr] = temp[i2++];
        else if (i2 > right) // Right sublist exhausted
            A[curr] = temp[i1++];
        else if (temp[i1].compareTo(temp[i2]) <= 0) // Get smaller value
            A[curr] = temp[i1++];
        else
            A[curr] = temp[i2++];
    }
}
```
Mergesort running time

• Mergesort runs in $O(n \log n)$