Sorting

Mergesort

- Another **divide-and-conquer** approach
- Divide array near its midpoint, sort the 2 halves by recursive calls, then merge the halves

```
void mergesort(int A[ ], const int n) {
  if (n > 1) {
    int n1 = n / 2;
    int n2 = n – n1;

    A1 = first n1 of elements from A;
    A2 = remaining elements from A;
    mergesort(A1, n1);
    mergesort(A2, n2);

    merge A1 and A2, putting result back into appropriate part of A;
   }
 }
```

```
merge:   // merges A1 and A2 (which are both sorted) into temp[ ]

initialize copied, copied1, and copied2 to zero;   // counters
while (both A1 and A2 have more elements to copy)
  if (A1[ copied1 ] <= A2[ copied2 ]) {
    temp[ copied ] = A1[ copied1 ];
    copied++;
    copied1++;
  }
  else {
        temp[ copied ] = A2[ copied2 ];
        copied++;
        copied2++;
  }

if (any elements still left in A1 or A2)   // can only be one or the other
  copy them into temp;
```

# times mergesort called (i.e., # times you can make 2 halves out of n items) is **O(log n)**

Each time mergesort called, merge step requires **O(n)** time

So this algorithm is **O(n log n)**

Characteristics

- Not as simple to implement as $O(n^2)$ sorting algorithms
- Worst case time is as good as it gets for sorting
- Mergesort is the best for external sorting (i.e., not having all elements of array in memory at one time)