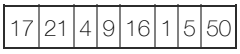
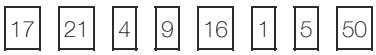
Mergesort Algorithm For Arrays, Reading

Mergesort can be adapted to arrays and linked lists. The latter is a bit more work because it involves tracking many separate lists and merging them, so just the arrayed version is shown here. The principles are the same in both.

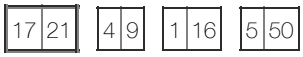
Here's how it works. Break down the size "n" array (or linked list) into n small, 1-element arrays (or linked lists). It's easier with arrays, because the small arrays can actually reside in one big array, and the start and end of each can be calculate as offsets. With linked lists, you actually need that many start pointers! So:



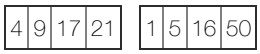
...becomes:



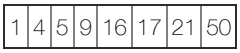
Now with each pair of adjacent "sub-arrays", merge them by looking at the zeroth element of each, choosing the smaller, then copy the remaining element:



Now with each pair of adjacent 2-element "sub-arrays", start the merge by looking at the zeroth element of each, choosing the smaller. Then compare the unchosen zeroth element with the *first* element of the other sub-array. So in our example, considering the 17-21 sub-array and the 4-9 sub-array, the 4's chosen, Then compare the unchosen 17 with *9*. Keep comparing the unselected and remaining values in the 2-element sub-arrays to form 4-element sub-arrays in the next iteration:



Yes, there are details to work out, like how to swap the values in the array and what if the size is not a multiple of 2. And what to do when you run out of values in one of the sub-arrays and theres nothing to compare anymore? We'll get to these, but just focus on the breakdown and merging concepts for now. Here's the last iteration:



We Need A Working Space

The first thing we need is another array. At the start of each iteration, the values in the full n-sized array are copied into this temporary array, then as the merging takes place, they are put back into the original array. That avoids having to work out a complicated swap-in-place scheme.

When One Array Runs Out Of Values

Actually, that one's easy. Remember when we were merging 17-21 and 4-9? we compare 17 vs 4 and choose 4, making 9 the new "first element" of that array. Then 17 vs 9, we choose 9. Now 4-9's empty! All we have to do is copy the remaining values in 17-21 -- that's it.

When One Array Is Smaller Than The Other

That one's easy too, because we only ever compare the "lead values" of two arrays, and one runs out of values, we copy the remaining values of the array that's not empty.

The Mergesort Algorithm

```
the name of the array to be sorted is "a" and its size is "n"
create a dynamic array of size n as a working space, named "temp"
```

```
create an integer to track the size of the sub-arrays in each iteration -- set to 1
start a loop
  if the sub-array size for this iteration is greater than n, break -- nothing more to do!
  declare and set an "index" to 0 -- the next position in the array to fill
  copy the array into the working space -- the whole thing!
  start a loop to visit each adjacent pair of sub-arrays, with a counter i=0
    create these integers to track the lead and end values of the two sub-arrays being merged:
    int left = i
    int leftMax = the minimum of n and (left + sub-array size)
    int right = leftMax
    int rightMax = the minimum of n and (right + sub-array size)
    start another loop to run as long as left < leftMax OR right < rightMax
      if (left == leftMax) a[index++] = temp[right++];
      else if (right == rightMax) a[index++] = temp[left++];
      else if (temp[right] < temp[left]) a[index++] = temp[right++];
      else a[index++] = temp[left++];
    add two sub-array sizes to i for the next cycle of the inner loop
  double the sub-array size for the next iteration
deallocate the working space array
```

Watch This! (https://www.youtube.com/watch?v=GCae1WNvnZM)

(https://www.youtube.com/watch?v=GCae1WNvnZM)