Merge sort

Given array A[1..n].


1) Divide input array into two subarrays of equal size.
2) Recursively merge sort each subarray.
3) Merge the newly sorted subarrays into a single sorted array.
If \( n \leq 1 \), do nothing.
Merging: think recursively

1) Pick first element of A as the smaller of the two subarrays' least elements.

2) "recursively" merge everything else!

might write this iteratively
sorts A

merges A[0..m-1] A[m+1..n]

assuming they are sorted

\[\text{if } n > 1\]
\[m \leftarrow \lfloor n/2 \rfloor\]
\[\text{MERGESORT}(A[0..m])\]
\[\text{MERGESORT}(A[m+1..n])\]

\((\text{Recurse})\)

\(\text{MERGE}(A[0..n], m)\)
Lemma: Consider any iteration \( k \) of the first for loop (including \( k = n+1 \)). Assuming,
\[
1 \leq \omega \leq m+1, \\
m+1 \leq j \leq n+1, \text{ and} \\
m+1 - \omega + n+1 - j = n+1 - k
\]

Then remaining iterations do copy \( AC[\omega \ldots m] + AC[j \ldots n] \) to \( B[k \ldots n] \) in sorted order.
Apply lemma for $k=1$ to prove we merge correctly.

Proof: Assume for iteration $k'$ with $k' > k$, the lemma holds for $k'$.

Note: $(n+1-k') < (n+1-k)$

↑

# iterations remaining
If $k = n+1$, we spend $O$ iterations filling $B[n+1..n]$.

0, W. (otherwise):

if $j > n$, array $A[j..n]$ is empty
(two are no indices between $n+1 \pm n$),
so
$\min(A[i..m] \cup A[j..n])$

is $A[i]$.

We do assign $A[i, j]$ to $B(k)$
t by I.H., iters $k+1$ to $n+1$.
merge $A[k+1..m] \cup B[j..n]$ into $B[k+1..n]$

O.W. if $\omega > m$, we should take $A[j..J] + d$, so, by IH, we merge $A[i..m] \cup A[j+1..n]$
into $B[k+1..n]$


Theorem: MergeSort(A(1...n)) sorts A.

Proof: Assume it works on arrays of size k when k<n. If n=1, nothing happens.

O.W. Recursion Fairy does sort the subarrays by I.H.

Previous lemma implies we merge them. Done.
Quick sort:

1) Choose a pivot element from the array.

2) Partition array into three subarrays
   
   a) elements < pivot
   
   b) pivot
   
   c) elements > pivot

3) Recursively quicksort subarrays a & c.
<table>
<thead>
<tr>
<th>Input:</th>
<th>SORTING EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose a pivot:</td>
<td>SORTING EXAMPLE</td>
</tr>
<tr>
<td>Partition:</td>
<td>AGOELNMLMP TXSR</td>
</tr>
<tr>
<td>Recurse Left:</td>
<td>AEGILMNP TXSR</td>
</tr>
<tr>
<td>Recurse Right:</td>
<td>AEGILMNOPRSTX</td>
</tr>
</tbody>
</table>

**QuickSort**(A[1 .. n]):
- **if** \( n > 1 \)
  - Choose a pivot element \( A[p] \)
  - \( r \leftarrow \text{Partition}(A, p) \)
  - \( \text{QuickSort}(A[1 .. r - 1]) \) (*Recurse!*)
  - \( \text{QuickSort}(A[r + 1 .. n]) \) (*Recurse!*)

**Partition**(A[1 .. n], p):
- swap \( A[p] \leftarrow A[n] \)
- \( \ell \leftarrow 0 \) (*#items < pivot*)
- for \( i \leftarrow 1 \) to \( n - 1 \)
    - \( \ell \leftarrow \ell + 1 \)
    - swap \( A[\ell] \leftarrow A[i] \)
- swap \( A[n] \leftarrow A[\ell + 1] \)
- return \( \ell + 1 \)
Partition: In iteration \( i \), 0 \( \leq l \leq i \). All elements in \( A[1..l] \) are less than pivot \( A[n] \). Elements in \( A[l+1..i] \) are \( \geq A[n] \).

**Theorem:** Quick Sort \((A[1..n])\) sorts \( A \).

**Proof:** Assume Quick Sort works on arrays of size \( k \) when \( k < n \). If \( n \leq 1 \), we do nothing. O.w., we partition. \( TH \Rightarrow \) we sort
first to last subarrays.
Divide-and-Conquer

1) Divide given instance into several smaller, independent instances of the same problem.

2) Delegate each smaller instance to the Recursion Fairy.

3) Combine solutions for smaller instances into one solution for original instance.