Mergesort

Given array $A[1..n]$.

Goal: rearrange elements of A so $A[1] \le A[2] \le \dots \le A[n]$

1) Divide input array into two subarrays of equal size.
2) Recursively mergesort each subarray.
3) Merge the newly sorted subarrays into a single sorted array.

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Divide:	S	O	R	T	I	N	G	E	X	A	M	P	L
Recurse Left:	I	N	O	R	S	T	G	E	X	A	M	P	L
Recurse Right:	I	N	O	R	S	T	A	E	G	L	M	P	X
Merge:	A	E	G	I	L	M	N	O	P	R	S	T	X

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 <u>write</u> this iteratively

```
MERGESORT(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])          ⟨⟨Recurse!⟩⟩
      MERGESORT(A[m+1..n])        ⟨⟨Recurse!⟩⟩
      MERGE(A[1..n], m)
```

```
MERGE(A[1..n], m):
  i ← 1; j ← m+1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i+1
      else if i > m
          B[k] ← A[j]; j ← j+1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i+1
      else
          B[k] ← A[j]; j ← j+1

  for k ← 1 to n
      A[k] ← B[k]
```

sorts A

merges A[1...m] & A[m+1..n],
assuming they are sorted

Lemma: Consider any iteration k of the first for loop (including $k=n+1$). Assuming,

$1 \le i \le m+1$,

$m+1 \le j \le n+1$, and

$$\underbrace{m+1-i}_{\text{size of } A[i..m]} + \underbrace{n+1-j}_{\text{sizes of } A[j..n]} = \underbrace{n+1-k}_{\text{siz of } B[k..n]}.$$

Then remaining iterations <u>do</u> copy $A[i..m]$ & $A[j..n]$ to $B[k..n]$ 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 0 iterations filling $B[n+1..n]$.

o.w. (otherwise):

if $j>n$, array $A[j..n]$ is empty (there are no indicies between $n+1$ & n),

so $\min(A[i..m] \cup A[j..n])$ is $A[i]$

We do assign $A[i]$ to $B[k]$, & by I.H., iters $k+1$ to $n+1$

merge $A[i+1..m]$ & $B[j..n]$
into $B[k+1..n]$

o.w. if $i > m$, we should take
$A[j]$ & do so. By IH,
we merge $A[i...m]$, $A[j+1..n]$
to $B[k+1..n]$

o.w., if $A[i] < A[j]$, min is
$A[i]$. We take it & copy
the rest by induction.

o.w. $A[i] \geq A[j]$. We can
take $A[j]$, ... induction.

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

Proof: Assume it works on arrays of size k when $k<n$. If $n\le 1$, nothing happens. o.w. Recursion Fairy does sort the subarrays by I.H. Previous lemma implies we merge them. Done.

Quicksort:

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.

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L
Choose a pivot:	S	O	R	T	I	N	G	E	X	A	M	$\boxed{\text{P}}$	L
Partition:	A	G	O	E	I	N	L	M	$\boxed{\text{P}}$	T	X	S	R
Recurse Left:	A	E	G	I	L	M	N	O	$\boxed{\text{P}}$	T	X	S	R
Recurse Right:	A	E	G	I	L	M	N	O	$\boxed{\text{P}}$	R	S	T	X

$\underline{\text{QuickSort}(A[1..n])\text{:}}$
  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!*⟩⟩

sorts A

$\underline{\text{Partition}(A[1..n], p)\text{:}}$
  swap $A[p] \leftrightarrow A[n]$
  $\ell \leftarrow 0$ ⟨⟨*#items < pivot*⟩⟩
  for $i \leftarrow 1$ to $n-1$
    if $A[i] < A[n]$
      $\ell \leftarrow \ell + 1$
      swap $A[\ell] \leftrightarrow A[i]$
  swap $A[n] \leftrightarrow A[\ell+1]$
  return $\ell + 1$

p: index of pivot.

Returns new index of pivot. (its rank)

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: QuickSort$(A[1..n])$ sorts A.

Proof: Assume QuickSort works on arrays of size k when $k < n$. If $n \leq 1$, we do nothing ✓

o.w. we partion. IH $\Rightarrow$ we sort first & last subarrays.

Divide-and-Conquer

1) <u>Divide</u> given instance into several <u>smaller independent</u> instances of the same problem.

2) <u>Delegate</u> each smaller instance to the Recursion Fairy.

3) <u>Combine</u> solutions for smaller instances into one solution for original instance.