Example: Quick Sort

A: \begin{array}{ccc}
\text{items < } A[i] & \text{items yet to be processed} & \text{items \( \geq \) } A[i] \\
\hline
i & x & y & j
\end{array}

- To sort \( A[i]...A[j] \), partition \( A \) into the two portions that are < and > a \textit{pivot element}, and recursively sort the two halves.
- We choose \( A[i] \) as the pivot element (a random element of \( A \) may be better).
- Work from the outsides towards the middle, exchanging elements when the one on the left is \( \geq A[i] \) and the one on the right is <\( A[i] \) until the variables \( x \) and \( y \) "cross" (i.e., \( x = y + 1 \)).
- Then move \( A[i] \) between the two halves (so the recursive calls will not include \( A[i] \), which is already correctly positioned in the sorted order).
- Quick sort is \textit{in-place}, using only \( O(1) \) space in addition to \( A \) (and space used by the recursion stack).
(quick sort continued)

Sort $A[i]...A[j]$

procedure QSORT($i, j$)
  if $i < j$ then begin
    $a, b := \text{PARTITION}(i, j)$
    QSORT($i, a$)
    QSORT($b, j$)
  end
end
(quick sort continued)

**Partition in-place** $A[i]...A[j]$ with respect to $A[i]$

**Idea:** A sequence of exchanges makes:


**function** PARTITION($i, j$)

```
x := i + 1
y := j
while $x \leq y$ do
    while $x \leq y$ and $A[x] < A[i]$ do $x := x + 1$
    while $x \leq y$ and $A[y] \geq A[i]$ do $y := y − 1$
    if $x < y$ then begin exchange $A[x]$ and $A[y]$; $x := x + 1$; $y := y − 1$ end
end
if $i < y$ then exchange $A[i]$ and $A[y]$.
return $(y − 1, y + 1)$
end```
Theorem: The expected time of arrayQSORT is $O(n\log(n))$, under the assumption that the input items are distinct and randomly ordered.

Proof:

- Time is proportional to the number of comparisons in $A$.
- Partitioning compares $n-1$ items to $A[1]$ and, if $A[1]$ is the $i^{th}$ largest element, $T(i-1)$ comparisons are used to sort elements $< A[1]$ and $T(n-i)$ comparisons are used to sort elements $> A[1]$.
- Since $A[1]$ is equally likely to be anywhere in the sorted order, add up for all $n$ values of $i$ and divide by $n$ to get the same recurrence as for building a binary search tree:

$$T(n) = (n - 1) + \frac{1}{n} \sum_{i=1}^{n} \left( T(i - 1) + T(n - i) \right) = O\left(n \log(n)\right)$$