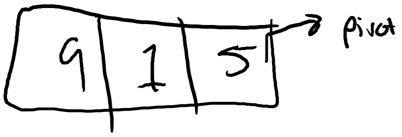
2019-11-21 Quicksort and Such

Thursday, November 21, 2019 8:58 AM

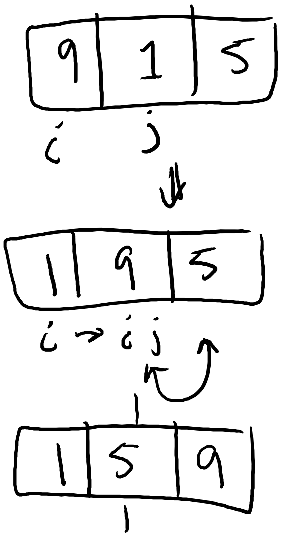
- Key aspect of quicksort is the pivot.
    - A selected reference value that is compared against all other values during a single iteration
    - Items that are smaller than the pivot go to the "left" of the pivot
    - Items that are larger than the pivot go to the "right" of the pivot
- We continue to iterate and find new pivots until the array is sorted
- The select of the pivot greatly determines the runtime characteristic of quicksort
    - The idea pivot is one that equally splits data
        - Unfortunately, it's too costly to continually calculate the exact median **every iteration**
    - Compromise is to find a "good enough" value
    - Richard Weiss suggests a "best of 3" approach
        - Select the middle-most value from array[start], array[midpoint], array[end]
        - This approach guarantees that we don't select the worst case pivot

Most basic 3-element example

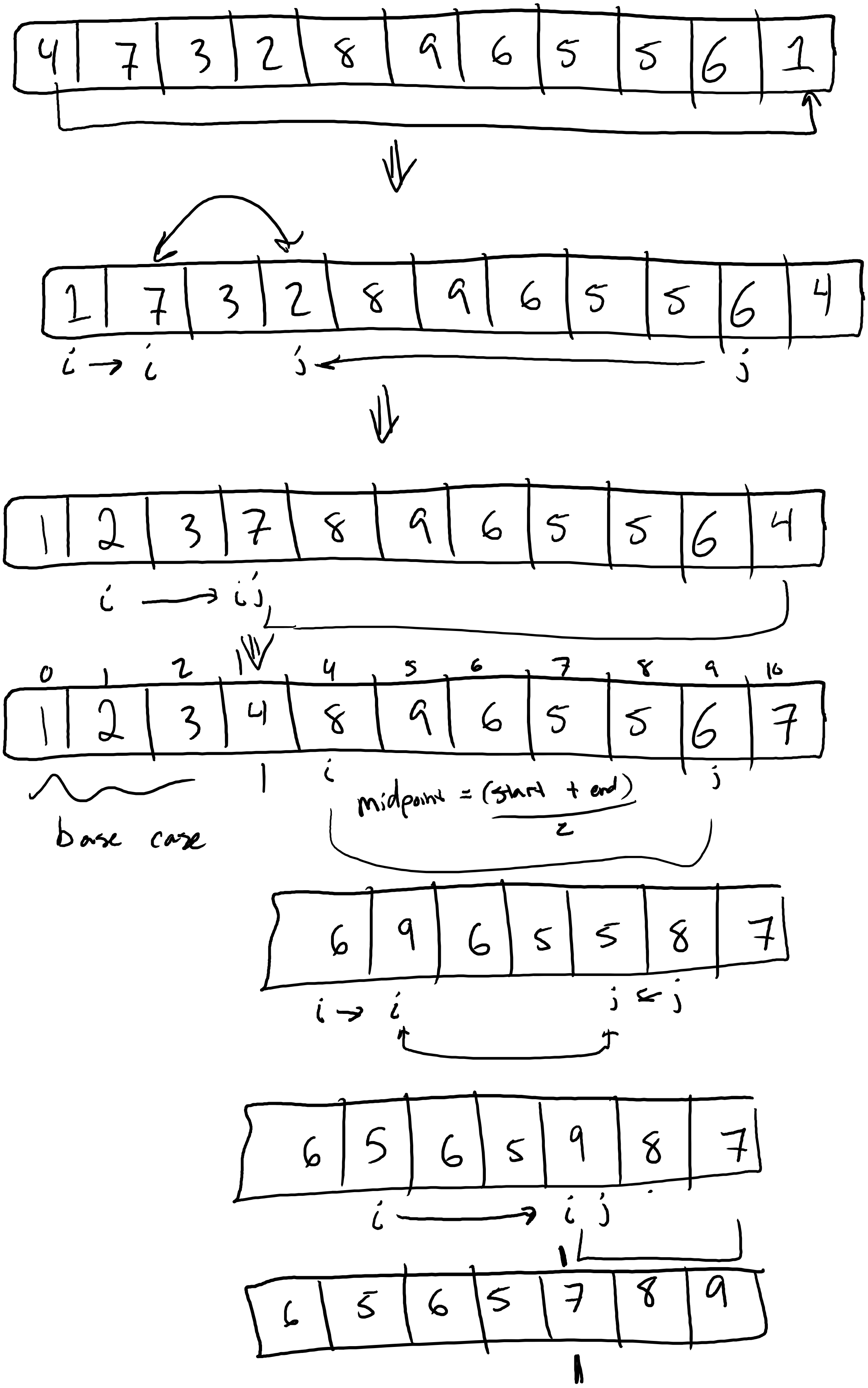


- If the pivot is not at the "end" of the sequence under consideration, swap it with the end element (we're not doing that in this case)

1. Define i = front_index; j = end_index - 1;
2. While data[i] < pivot AND i < j
    a. i++
3. While data[j] > pivot and i < j
    a. j--
4. if i != j
    a. Swap(data[i], data[j])
    b. GOTO #2
5. Iteration is done. Swap(numbers[end], numbers[i])
6. Recursively repeat on two subdivided arrays



Another example


4 7 3 2 8 9 6 5 5 6 1
1 7 3 2 8 9 6 5 5 6 4
i → i
j ← j
1 2 3 7 8 9 6 5 5 6 4
i → i j
0 1 2 3 4 5 6 7 8 9 10
1 2 3 4 8 9 6 5 5 6 7
i
j
base case
midpoint = (start + end) / 2
6 9 6 5 5 8 7
i → i
j ← j
6 5 6 5 9 8 7
i → i j
6 5 6 5 7 8 9