Design and Analysis
of Algorithms I

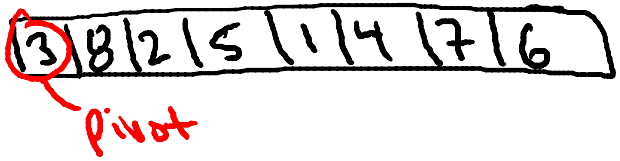
QuickSort

The Partition Subroutine

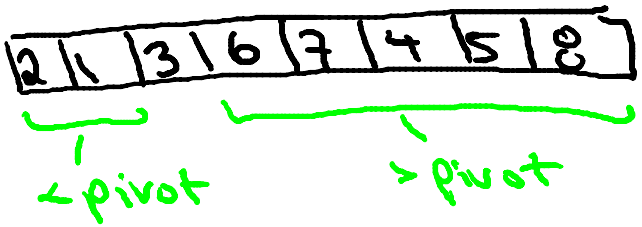
Partitioning Around a Pivot

Key idea: partition array around a pivot element.

- pick element of array



- rearrange array so that:
  - left of pivot ⇒ less than pivot
  - right of pivot ⇒ greater than pivot



Note: puts pivot in its "rightful position".

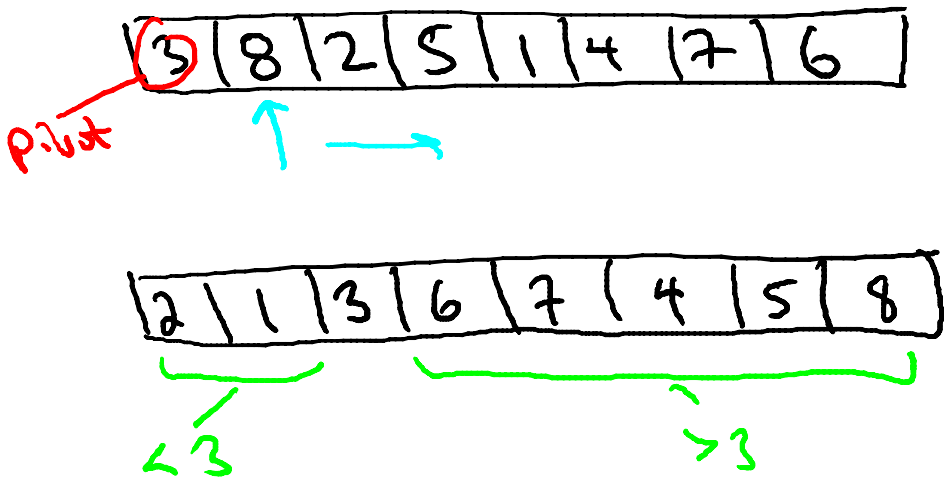
Two Cool Facts About Partition

(1) linear ($O(n)$) time, no extra memory
[see next video]

(2) reduces problem size

The Easy Way Out

Note: using O(n) extra memory, easy to partition around pivot in O(n) time.



In-Place Implementation

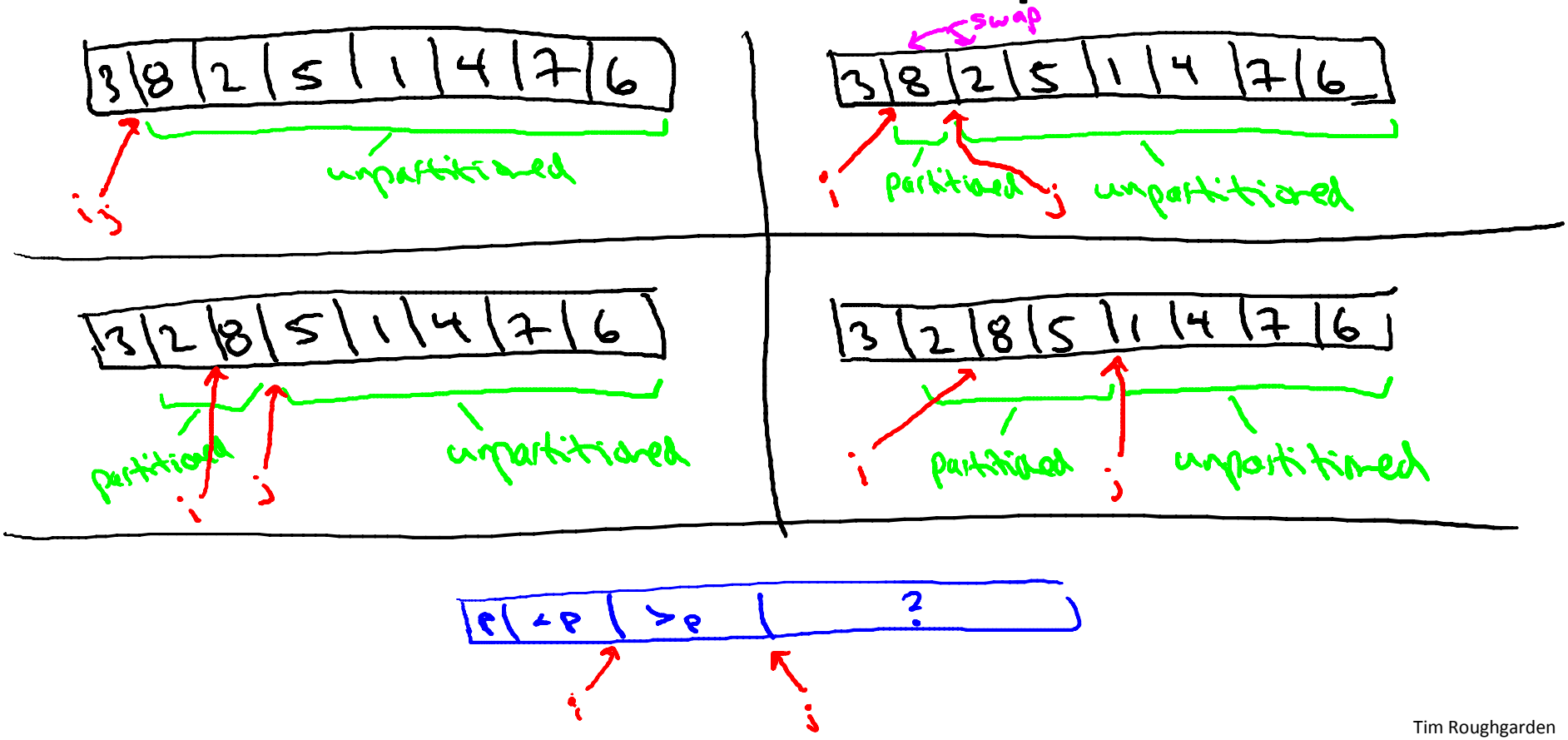
Assume: pivot = 1st element of array.

[if not, swap pivot ↔ 1st element as preprocessing step]

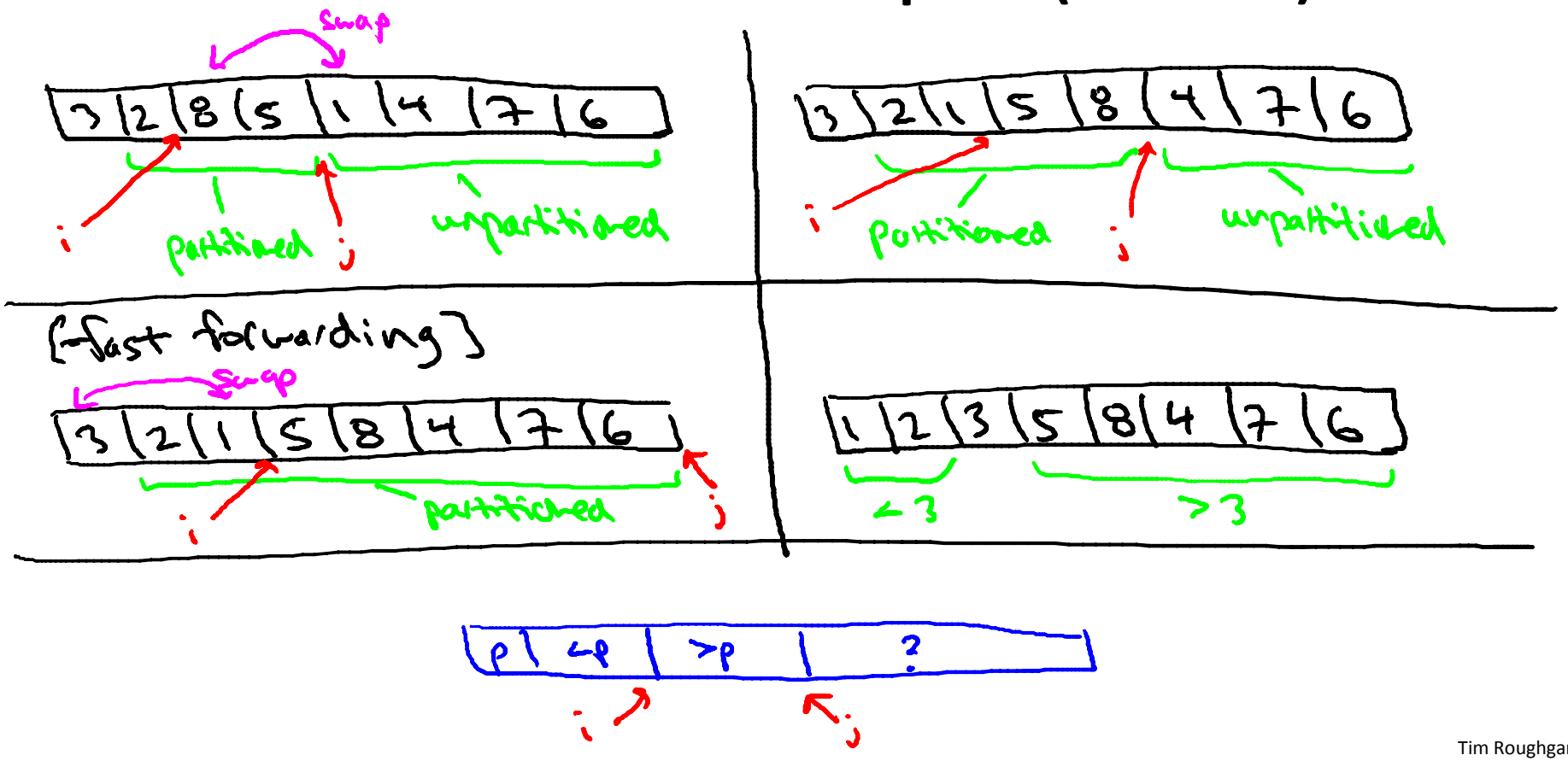
High-level Idea:

- single scan through array
- invariant: everything looked at so far is partitioned

Partition Example



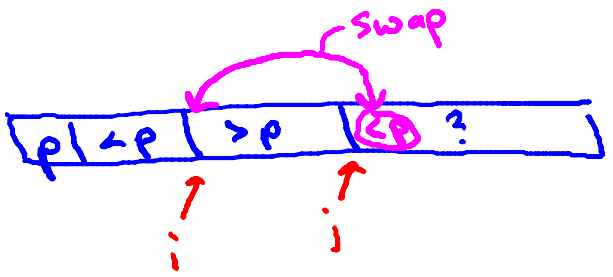
Partition Example (con'd)



Pseudocode for Partition

Partition (A, l, r) [input = A[l ... r]]

- p := A[l]
- i := l+1
- for j = l+1 to r
  - if A[j] < p [if A[j] > p, do nothing]
    - swap A[j] and A[i]
    - i := i+1
- swap A[l] and A[i-1]



Running Time

Running time = $O(n)$, where $n = r - l + 1$ is the length of the input (sub) array.

Reason: $O(1)$ work per array entry.

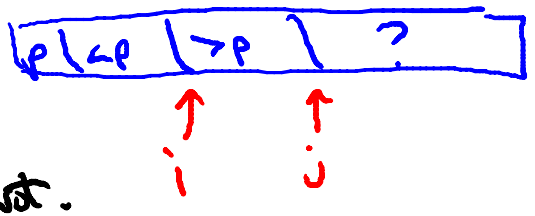
Also: clearly works in place (repeated swaps).

Correctness

Claim: the for loop maintains the invariants:

① A[l+1], ..., A[i-1] are all less than the pivot

② A[i], ..., A[j-1] are all greater than pivot.



[Exercise: check this, by induction.]

Consequence: at end of for loop, have

=> after final swap, array partitioned around pivot. QED!