Quick sort

Carlos Moreno
cmoreno@uwaterloo.ca
EIT-4103

Photo courtesy of ellenmc / Flickr:
http://www.flickr.com/photos/ellenmc/4508741746/

https://ece.uwaterloo.ca/~cmoreno/ece250

These slides, the course material, and course web site are based on work by Douglas W. Harder

Quick sort

Standard reminder to set phones to silent/vibrate mode, please!

Quick sort

- Previously, on ECE-250...
  - Introduced sorting and related concepts
  - Discussed some of the important sorting algorithms.
  - In particular, merge sort and heap sort

Quick sort

- Previously, on ECE-250...
  - Introduced sorting and related concepts
  - Discussed some of the important sorting algorithms.
  - In particular, merge sort and heap sort
    - With merge sort, we split in two halves, sort each one, then merge the sorted halves
      - Divide-and-conquer !

Quick sort

- During today's class:
  - Look into quick sort, the most widely used sorting algorithms (at least in general-purpose libraries and software out there)
    - Discuss the basic idea behind the algorithm
    - Look into its run time, with emphasis on average-case vs. Worst-case (interesting plot twist in this area!)
    - Discuss some of the strategies to avoid the worst-case
  - Work through some examples.

Quick sort

- To introduce the basic idea, let's first recall what the main problem was with merge sort...

Quick sort

- To introduce the basic idea, let's first recall what the main problem was with merge sort...
  - The merge operation can not be done in-place
    - This not only has the immediate implication that we require the extra space, but it also has an implication on the performance — the data is being moved a lot, and so this has a performance penalty!
      - Even more critical if the data is expensive to copy (e.g., strings, data that is really on disk or some extra-slow hardware devices)

Quick sort

- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:

Sequence A: 11, 23, 40, 57, 68, 93
Sequence B: 5, 16, 35, 46, 79, 96

Output sequence:

Quick sort

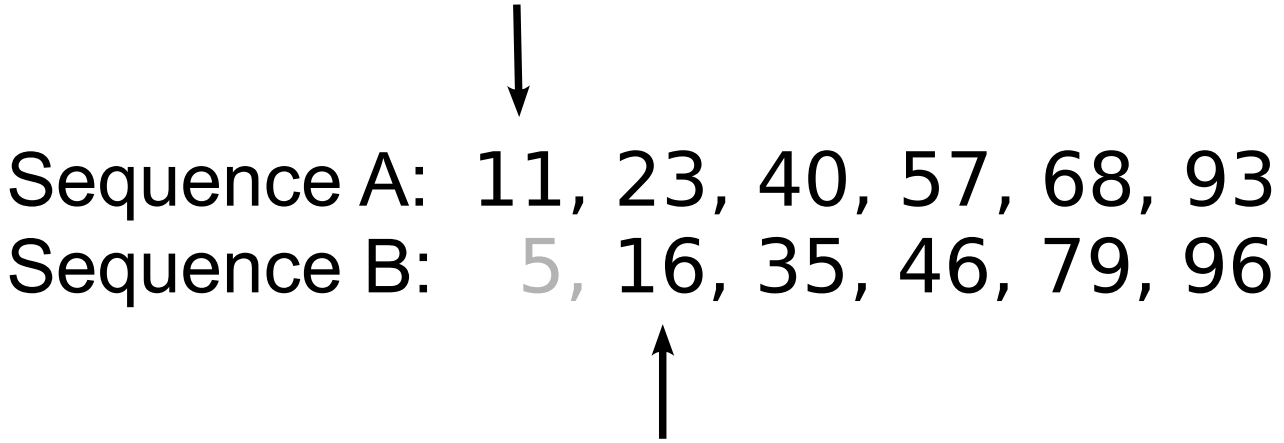
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:

Sequence A: 11, 23, 40, 57, 68, 93
Sequence B: 5, 16, 35, 46, 79, 96

Output sequence:

Quick sort

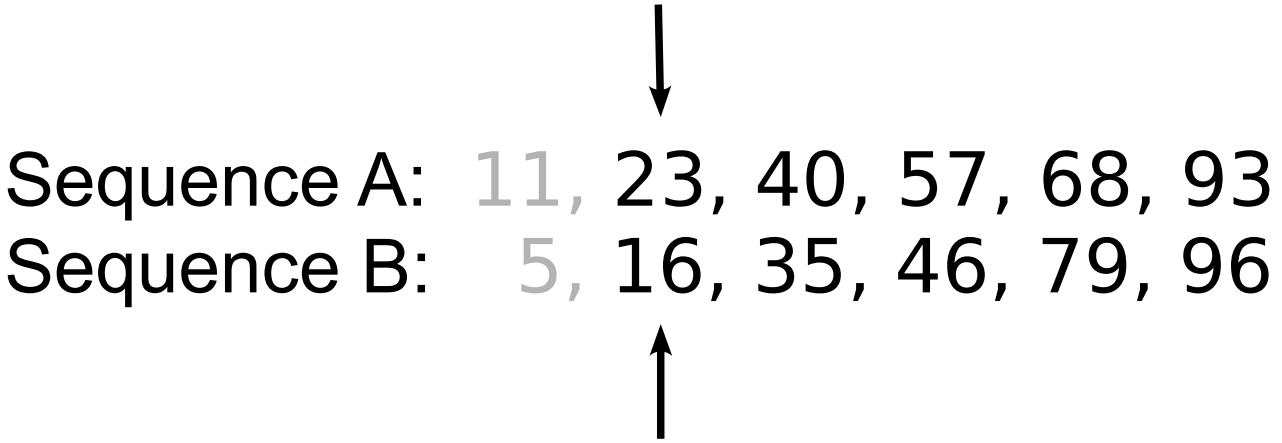
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Output sequence: 5

Quick sort

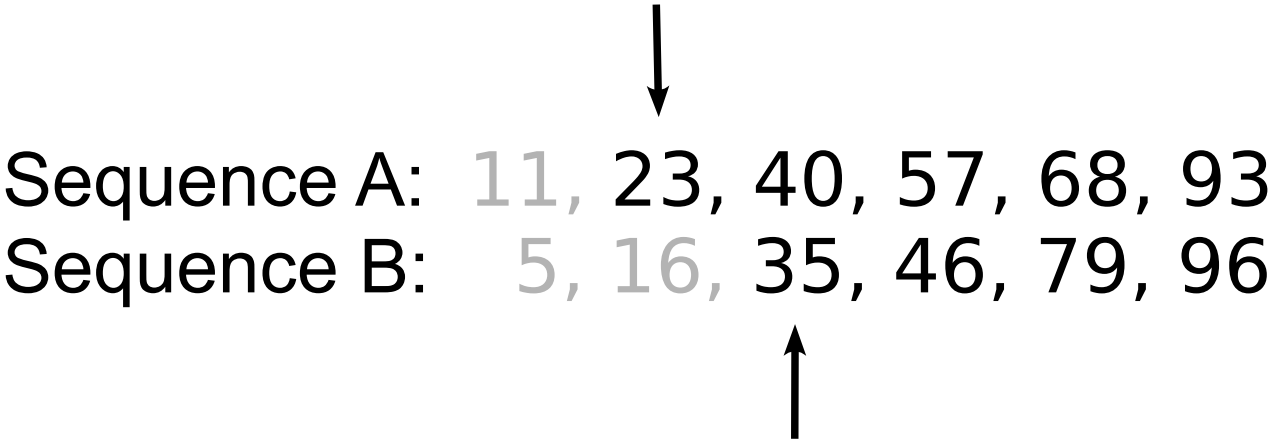
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Output sequence: 5, 11

Quick sort

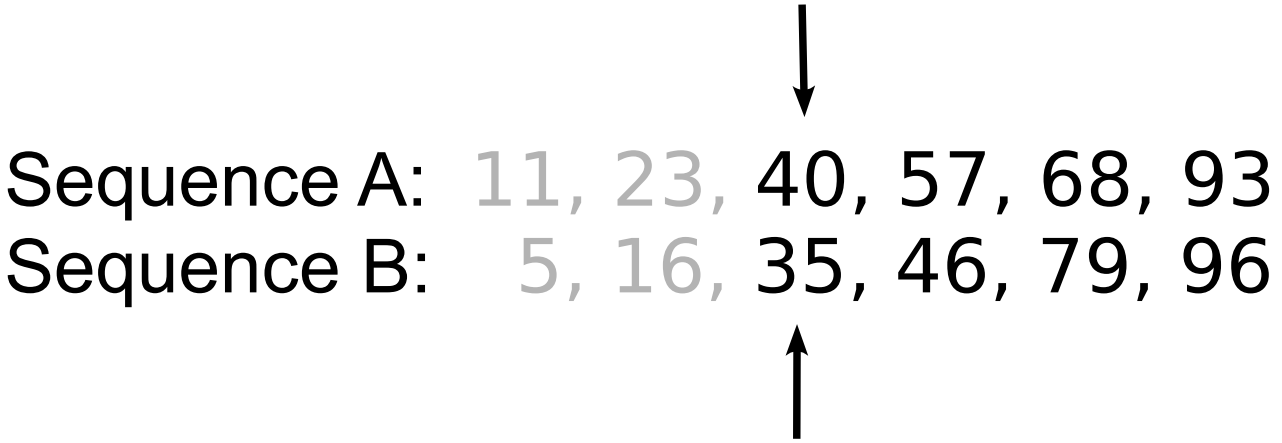
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Output sequence: 5, 11, 16

Quick sort

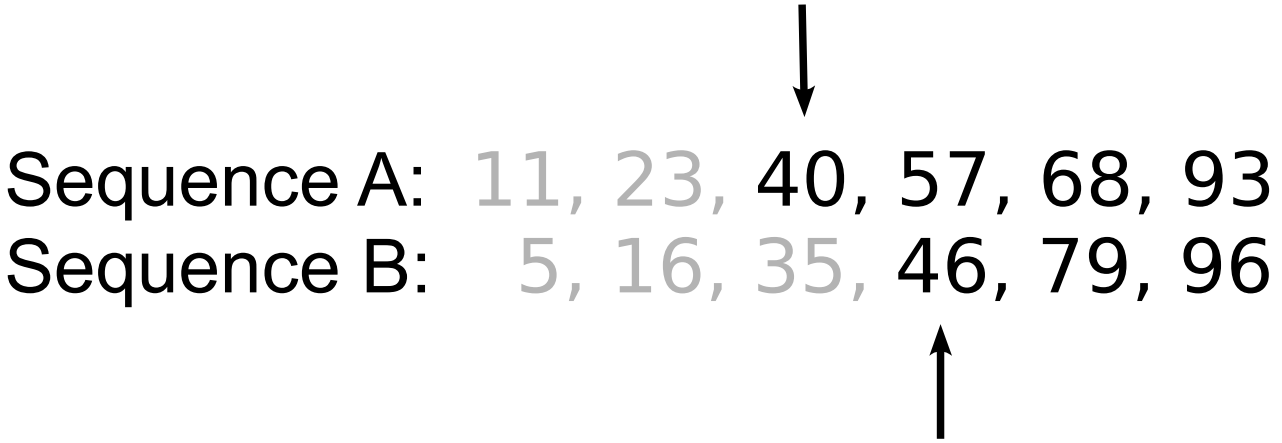
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Output sequence: 5, 11, 16, 23

Quick sort

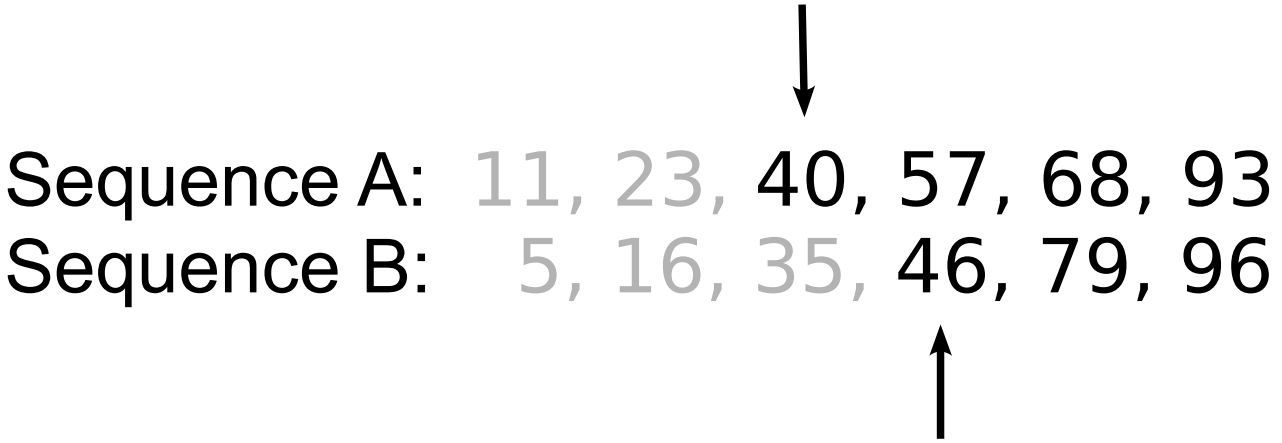
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Output sequence: 5, 11, 16, 23, 35

Quick sort

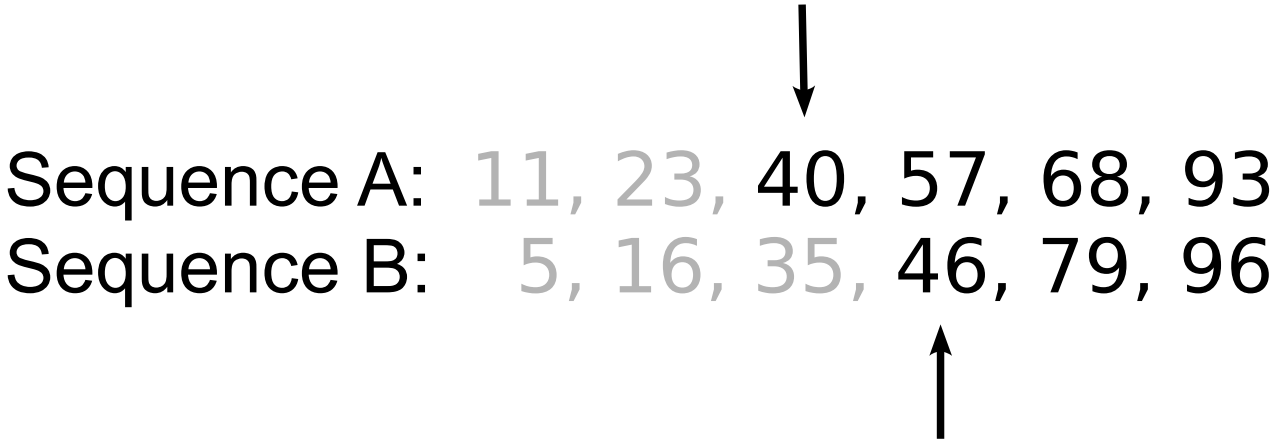
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Output sequence: 5, 11, 16, 23, 35 · · · etc.

Quick sort

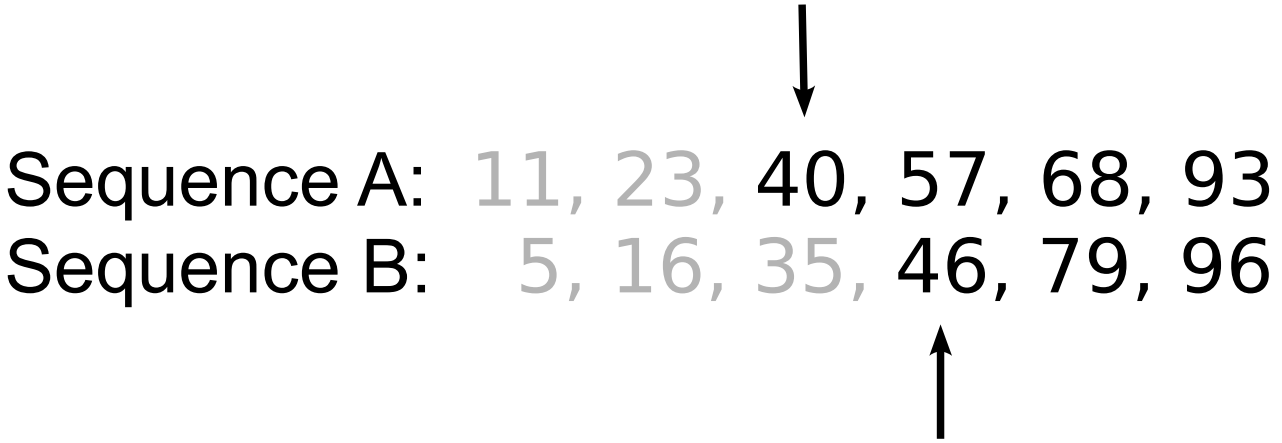
- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



We might be tempted to think of the items at the left of the arrows as available space ... But ...

Quick sort

- The obvious question is: can we do a merge sort without requiring extra storage?
  - Let's try merging the following two sorted sequences:



Do you see why that's not going to work? (we'll discuss this detail during class)

Quick sort

- The problem seems to be due to the fact that we first sort each half, then combine the results.

Quick sort

- The problem seems to be due to the fact that we first sort each half, then combine the results.
  - And yes, one might think "*well, duh! of course we do that — what else are we going to do?*"

Quick sort

- The problem seems to be due to the fact that we first sort each half, then combine the results.
  - And yes, one might think "*well, duh! of course we do that — what else are we going to do?*"
  - And that's actually an excellent question! Is there something else we could do?
    - Perhaps going the other way around — could we do some processing first such that the elements end up in a way that when we recursively sort each half we're done?

Quick sort

- Let's reformulate that question:
- Suppose that I give you sequence A and sequence B, one after the other one, as shown below:

$$\underbrace{a_1, a_2, \cdots, a_{n-1}, a_n}_{\text{Sequence } A} \quad \underbrace{b_1, b_2, \cdots, b_{n-1}, b_n}_{\text{Sequence } B}$$

Quick sort

- Let's reformulate that question:
- Suppose that I give you sequence A and sequence B, one after the other one, as shown below:

$$\underbrace{a_1, a_2, \cdots, a_{n-1}, a_n}_{\text{Sequence } A} \quad \underbrace{b_1, b_2, \cdots, b_{n-1}, b_n}_{\text{Sequence } B}$$

- And suppose I told you that if we sort sequence A and sort sequence B, then the complete sequence will be sorted — what does that tell you about the values in both sequences?

Quick sort

- That's the central aspect behind quicksort's functionality — and no, I won't write the answer in this slide set; we'll complete this discussion in class.

Quick sort

- That's the central aspect behind quicksort's functionality — and no, I won't write the answer in this slide set; we'll complete this discussion in class.
- However, I will discuss another aspect that plays a rather important role, and maybe that will give you some ideas about what the answer to the previous issue would be?

Quick sort

- Let's look at the notion of the *median* of a set of values.
  - The *median* is a statistical measure of a set of values, somewhat similar to the *mean*, or average, but quite interesting:
  - Given a set of values $\{x_1, x_2, \cdots, x_{n-1}, x_n\}$ the median is one of the values in the sequence, x_m, such that there are as many elements $x_i < x_m$ as there are elements $x_k > x_m$

Quick sort

- Let's look at the notion of the *median* of a set of values.
  - The *median* is a statistical measure of a set of values, somewhat similar to the *mean*, or average, but quite interesting:
  - Given a set of values $\{x_1, x_2, \cdots, x_{n-1}, x_n\}$ the median is one of the values in the sequence, x_m, such that there are as many elements $x_i < x_m$ as there are elements $x_k > x_m$
    - Visually, this corresponds to the following: place the elements in order; then, pick the middle element.

Quick sort

- The problem is, algorithms to find the median are not very efficient — in many cases, one simply sorts and then picks the middle element.
  - So, if we're needing the median as part of a sorting procedure, we're kind of stuck, like a dog chasing its tail !!

Quick sort

- What about an approximation of the median??
  - Something that, on average, is close to the median.
  - As it turns out, this is quite useful for quicksort !!
    - We'll investigate at least a couple of techniques and their advantages and disadvantages.