CSE 373: Sorting

(Quicksort, Quickselect, Bucketsort)

Chapter 7

Quicksort

Quicksort:

- Another recursive divide-and-conquer sorting algorithm
- In practice, the fastest known sorting algorithm

Partitioning

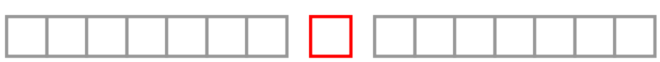
Partitioning: Quicksort's main operation

- given a list…
- choose a pivot element, *p*, from the list
- divide the rest of the values into two sets:
  - those less than *p*
  - those greater than *p*
  - (for now, we'll ignore those that are equal to *p*)

Partitioning Example

(Assume we'll use the first element as a pivot):

7	4	8	6	9	2	5	3



Running time of partition?

Quicksort Overview

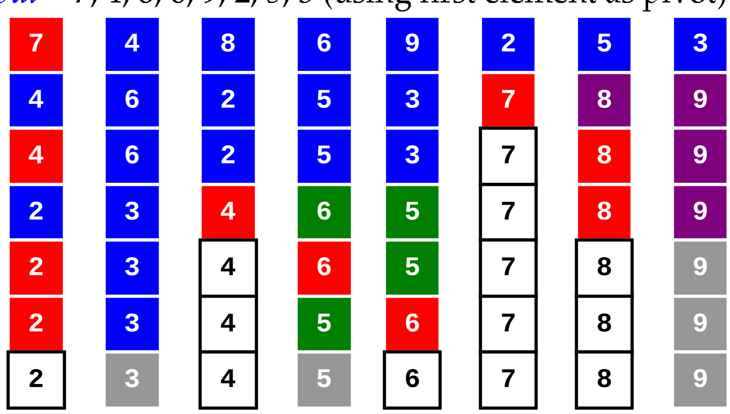
Quicksort: given a list of values…

- if the list contains zero or one elements, return it
- otherwise, *partition* the list
- call Quicksort recursively on each half
- concatenate the results of the recursive calls:

  Quicksort(small values) :: pivot :: Quicksort(big values)

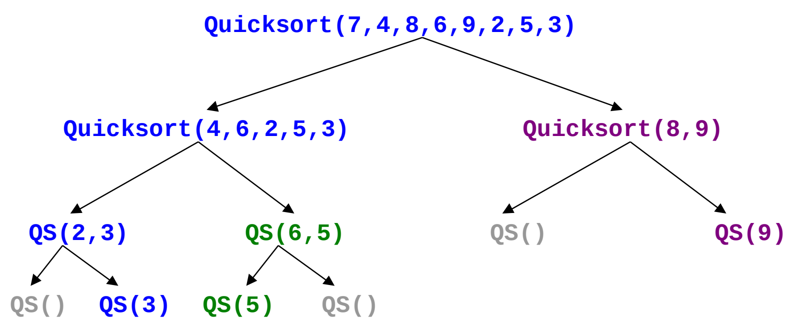
Quicksort Example

input = 7, 4, 8, 6, 9, 2, 5, 3 (using first element as pivot)



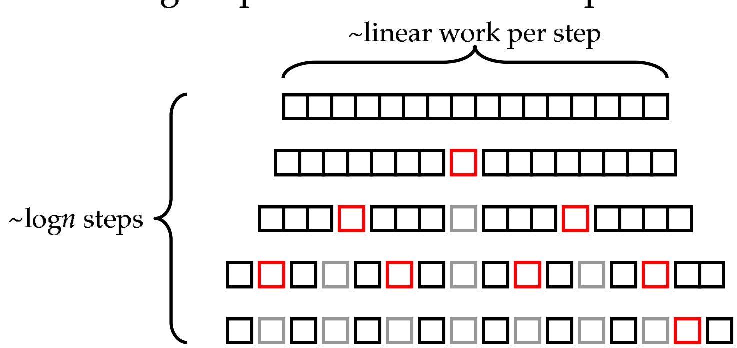
UW, Autumn 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

Quicksort Call Tree



Running Time (Approximate & Optimistic)

Assuming all pivots result in even partitions…



$$\approx O(n) \times O(\log n) \approx O(n \log n)$$

Worst-Case Analysis

- What would be a worst-case partition step?
- What input would cause this worst case at *every* step (assuming pivot is first element)?
- What's the running time of this worst-case?

Design Decision: Choosing Pivot

- first element – should ***never, never*** be used
- random element
- median
- median of three (first, middle, last?)
- middle element

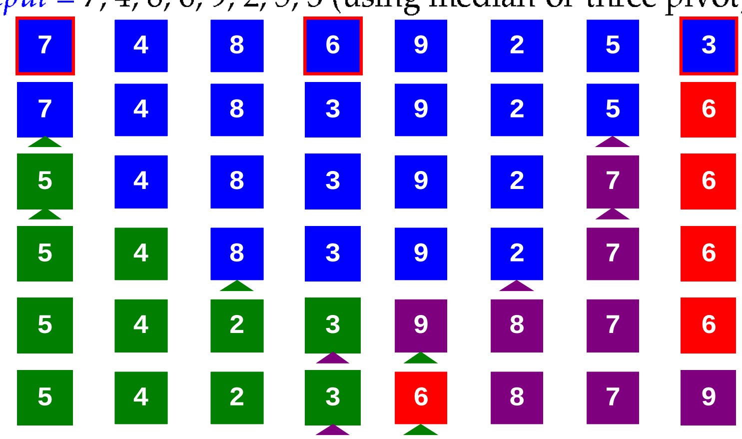
UW, Autumn 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

In-Place Partitioning

1) swap the pivot *p* with the last element
2) set a pointer *i* to the first element
3) set a second pointer *j* to the second-to-last
4) walk *i* up the array until a value > *p* is found
5) walk *j* down the array to a value < *p*
6) swap elements pointed to by *i* and *j*
7) continue until *i* and *j* pass one another
8) when they do, swap *i*'s element with *p*

In-Place Partitioning Example

input = 7, 4, 8, 6, 9, 2, 5, 3 (using median-of-three pivot)



Quicksort Best-Case Analysis

Use a recurrence relation:

$$T(0) = k$$
$$T(1) = k$$
$$T(n) = 2T(n/2) + c \cdot n$$

Solve using repeated substitution:

Quicksort Overview

- Running Times:
  - Best Case: O($n$log$n$)
  - Worst Case: O(n^2) – ***but very unlikely***
  - Average Case: O($n$log$n$) – shown in book
- Space Requirement: sorts in-place

Design Details

- Sort small arrays (n < 20?) using insertion sort
  - insertion sort faster for small problems
  - all Quicksorts on big lists must also sort small lists
- How to handle elements equal to pivot?
  - annoying detail; see book
- *Quickselect* – a modification of Quicksort to do selection in O(n) time (on average)

Bucketsort

Useful for sorting integers of a fixed range:

- Declare an array: `int count[range]`
- Initialize `count[]` to all 0's
- Iterate over the input list
- For each value v, increment `count[`v`]`
- Once done, print out `count[0]` 0's, `count[1]` 1's, …, `count[i]` i's etc.

Running time?