Data Structures and Algorithms

Spring 2009-2010

Outline

Outline

Quicksort

- Fastest known sorting algorithm in practice
- Running time of $O(n \log n)$ in *average*-case
- Has $O(n^2)$ *worst*-case performance but we can make chances of this occurring exponentially small with careful choice of pivot
- Basic algorithm to sort an array, S:
  - If $|S| \leq 1$ return
  - Pick v, a random el. in S and call it the *pivot*
  - Partition S into three sets:
    - S_1: those elements of S smaller than v
    - v
    - S_2: those elements of S larger than v
  - $S = \{S_1, v, S_2\}$
  - return the array { *quicksort*(S_1), v, *quicksort*(S_2)}

Quicksort (contd.)

- Choice of v is key factor in determining running time of quicksort
- We would like v to partition S into S_1, S_2 such that $|S_1| = |S_2| = |S|/2$
- To be exact about choice of v we would need to find the *median* element in order to have $|S_1| = |S_2|$ – too expensive
- Bad v can lead to quadratic running time

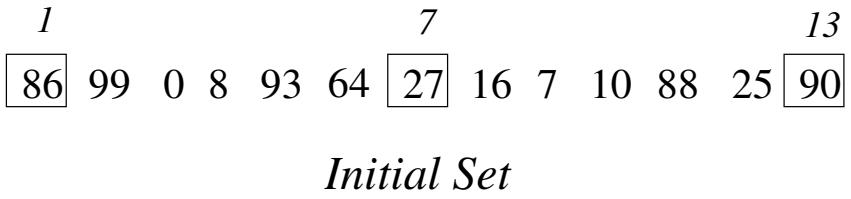
✗ If $n <= 20$ quicksort is beaten by insertion sort!! So...

✓ Incorporate insertion sort in to quicksort routine so that if sub-arrays of size 20 or less need to be sorted, use insertion sort

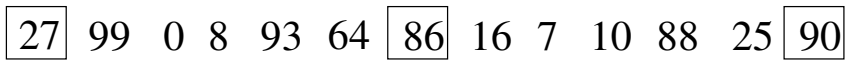
- It is a bad idea to use the first element in the array as the "random" choice of pivot – especially if the data are already sorted to begin with

Quicksort (contd.)

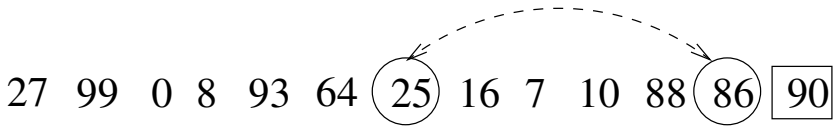
- Usual strategy adopted: "median-of-three" partitioning – not "average-of-three"; code here
- Find the median element of the first, middle and last element of the array and use this as pivot element
- Using the median-of-three guards against $O(n^2)$ running time that using the first element could yield on an already sorted array



Initial Set



Positions l, c, r sorted



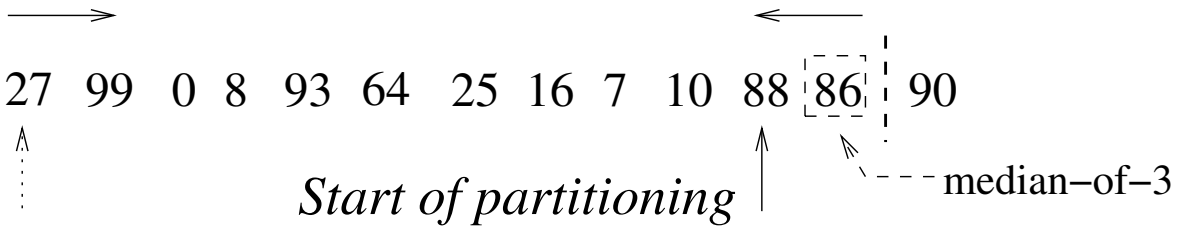
Median "hidden" at posn. **r-1**

Partitioning the Set

- `a[l]` and `a[r]` are in the correct partitions
- Only need to partition remainder now according to pivot
- Want to do this in linear time
- Idea: get pivot out of way by "hiding" at end of set-to-partition (STP)
- Then partition the remainder into those smaller than pivot (left $\leftarrow$) and those larger ($\rightarrow$ right)
- Maintain two indices, `i` and `j`, that start at either end of STP
- Move `i` (`j`, respectively) towards centre until an element is found that is larger (smaller) than pivot

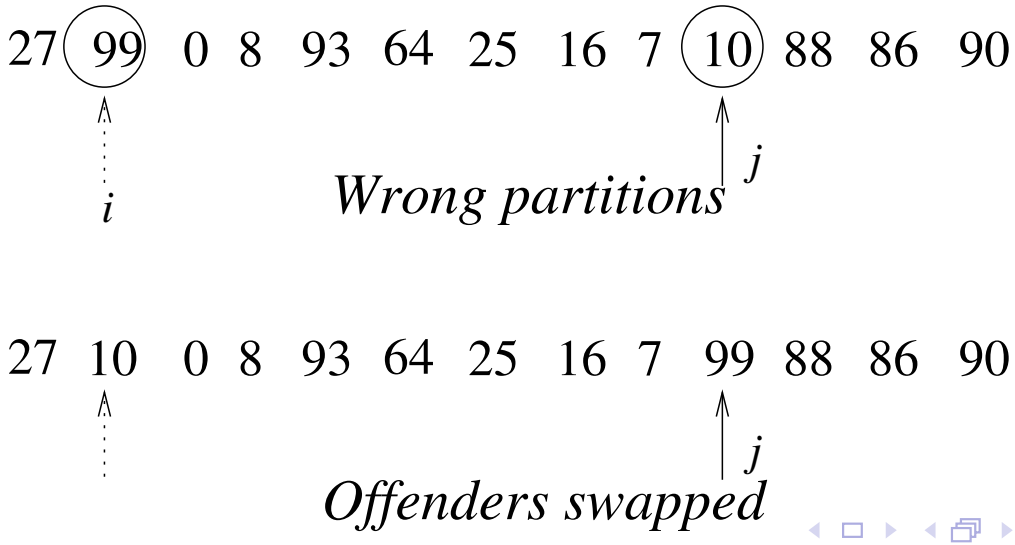
Partitioning the Set (contd.)

- Swap `a[i]` and `a[j]` since they belong in the other "halves"
- Stop when all elements have been looked at: `i == j`
- Since we don't know the ranking of the pivot element in STP, `(i == j == c)` will probably *not* hold
- With all elements smaller than pivot to the left and all elements greater than pivot to the right, put pivot back in its correct and *final* resting place



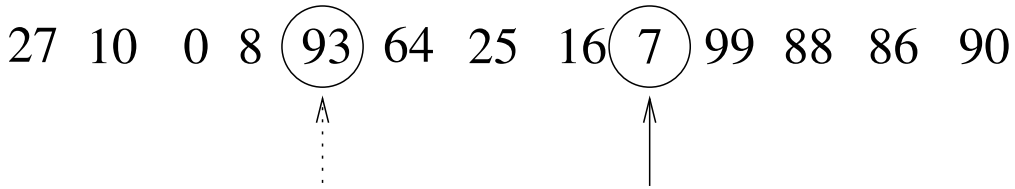
Partitioning the Set (contd.)

- Swap `a[i]` and `a[j]` since they belong in the other "halves"
- Stop when all elements have been looked at: `i == j`
- Since we don't know the ranking of the pivot element in STP, `(i == j == c)` will probably *not* hold
- With all elements smaller than pivot to the left and all elements greater than pivot to the right, put pivot back in its correct and *final* resting place

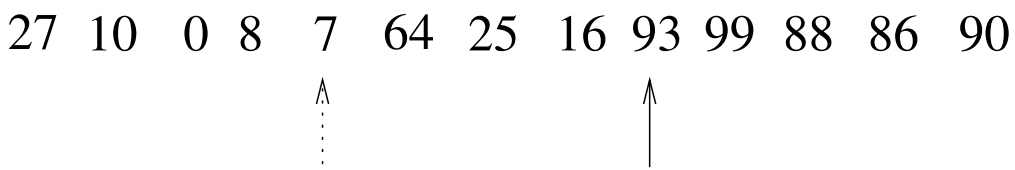


Partitioning the Set (contd.)

- Swap `a[i]` and `a[j]` since they belong in the other "halves"
- Stop when all elements have been looked at: `i == j`
- Since we don't know the ranking of the pivot element in STP, `(i == j == c)` will probably *not* hold
- With all elements smaller than pivot to the left and all elements greater than pivot to the right, put pivot back in its correct and *final* resting place



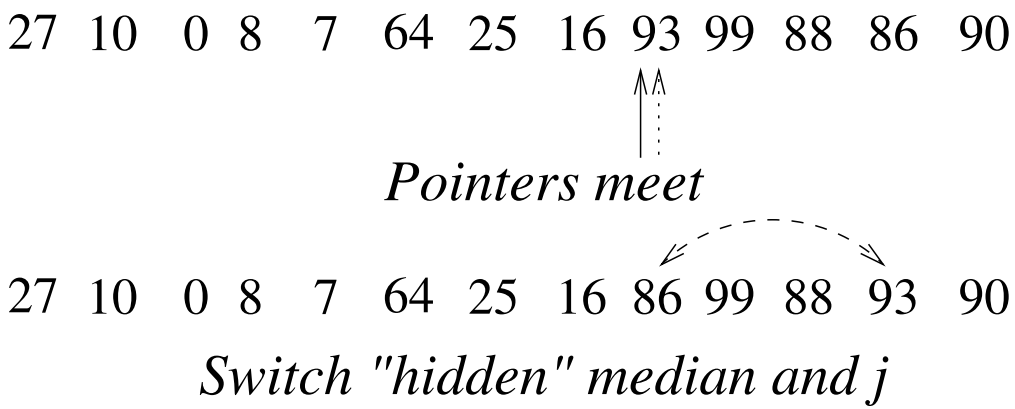
Wrong partitions



Offenders swapped

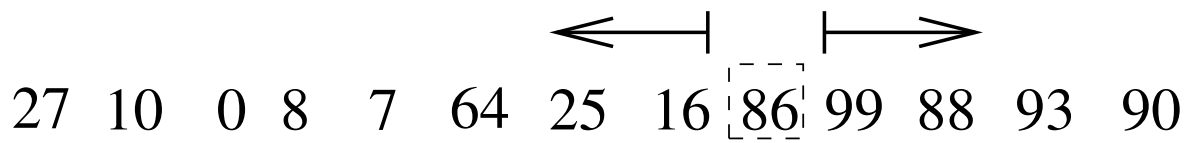
Partitioning the Set (contd.)

- Swap `a[i]` and `a[j]` since they belong in the other "halves"
- Stop when all elements have been looked at: `i == j`
- Since we don't know the ranking of the pivot element in STP, `(i == j == c)` will probably *not* hold
- With all elements smaller than pivot to the left and all elements greater than pivot to the right, put pivot back in its correct and *final* resting place



Partitioning the Set (contd.)

- Swap `a[i]` and `a[j]` since they belong in the other "halves"
- Stop when all elements have been looked at: `i == j`
- Since we don't know the ranking of the pivot element in STP, `(i == j == c)` will probably *not* hold
- With all elements smaller than pivot to the left and all elements greater than pivot to the right, put pivot back in its correct and *final* resting place



Final partition