CS 372: Algorithms

Quicksort

(Based on slides by David Luebke)

Review: Quicksort

- Sorts in place
- Sorts O(n lg n) in the average case
- Sorts O(n^2) in the worst case
  - But in practice, it's quick
  - And the worst case doesn't happen often (but more on this later…)

Quicksort

- Another divide-and-conquer algorithm
  - The array A[p..r] is *partitioned* into two non-empty subarrays A[p..q] and A[q+1..r]
    - Invariant: All elements in A[p..q] are less than all elements in A[q+1..r]
  - The subarrays are recursively sorted by calls to quicksort
  - Unlike merge sort, no combining step: two subarrays form an already-sorted array

3

Quicksort Code

```
Quicksort(A, p, r)
{
    if (p < r)
    {
        q = Partition(A, p, r);
        Quicksort(A, p, q);
        Quicksort(A, q+1, r);
    }
}
```

4

Partition

- Clearly, all the action takes place in the **`partition()`** function
  - Rearranges the subarray in place
  - End result:
    - Two subarrays
    - All values in first subarray ≤ all values in second
  - Returns the index of the "pivot" element separating the two subarrays
- *How do you suppose we implement this?*

5

Partition In Words

- Partition(A, p, r):
  - Select an element to act as the "pivot" (*which?*)
  - Grow two regions, A[p..i] and A[i+1..j-1]
    - All elements in A[p..i] <= pivot
    - All elements in A[i+1..j-1] > pivot

6

Partition Code

```
Partition(A, p, r)
    x = A[r];
    i = p - 1;
    for j=p to r-1
      do if A[j] <= x
           then i = i+1;
                exchange A[i] and A[j];
    exchange A[i+1] and A[r]
    return i+1;
```

Illustrate on
A = {5, 3, 2, 6, 4, 1, 7, 3};

What is the running time of ***partition()****?*

7

Partition Code

```
Partition(A, p, r)
    x = A[r];
    i = p - 1;
    for j=p to r-1
      do if A[j] <= x
           then i = i+1;
                exchange A[i] and A[j];
    exchange A[i+1] and A[r]
    return i+1;
```

partition() *runs in O(n) time*

8

Analyzing Quicksort

- *What will be the worst case for the algorithm?*
  - Partition is always unbalanced
- *What will be the best case for the algorithm?*
  - Partition is perfectly balanced
- *Which is more likely?*
  - The latter, by far, except...
- *Will any particular input elicit the worst case?*
  - Yes: Already-sorted input

9

Analyzing Quicksort

- In the worst case:

  $T(1) = \Theta(1)$

  $T(n) = T(n - 1) + \Theta(n)$
- Works out to

  $T(n) = \Theta(n^2)$

10

Analyzing Quicksort

- In the best case:

  $T(n) = 2T(n/2) + \Theta(n)$

- What does this work out to?

  $T(n) = \Theta(n \lg n)$

11

Improving Quicksort

- The real liability of quicksort is that it runs in $O(n^2)$ on already-sorted input
- Book discusses two solutions:
  - Randomize the input array, OR
  - *Pick a random pivot element*
- *How will these solve the problem?*
  - By insuring that no particular input can be chosen to make quicksort run in $O(n^2)$ time

12

Analyzing Quicksort: Average Case

- Assuming random input, average-case running time is much closer to O(n lg n) than $O(n^2)$
- First, a more intuitive explanation/example:
  - Suppose that partition() always produces a 10-to-1 split. This looks quite unbalanced!
  - The recurrence is thus:

    T(n) = T(10n/11) + T(n/11) + cn
  - *How deep will the recursion go?* (draw it)

13

Analyzing Quicksort: Average Case

- Intuitively, a real-life run of quicksort will produce a mix of "bad" and "good" splits
  - Randomly distributed among the recursion tree
  - Pretend for intuition that they alternate between best-case (n/2 : n/2) and worst-case (n-1 : 1)
  - *What happens if we bad-split root node, then good-split the resulting size (n-1) node?*

14

Analyzing Quicksort: Average Case

- Intuitively, a real-life run of quicksort will produce a mix of "bad" and "good" splits
  - Randomly distributed among the recursion tree
  - Pretend for intuition that they alternate between best-case (n/2 : n/2) and worst-case (n-1 : 1)
  - *What happens if we bad-split root node, then good-split the resulting size (n-1) node?*
    - We end up with three subarrays, size 1, (n-1)/2, (n-1)/2-1
    - Combined cost of splits = $\Theta(n) + \Theta(n-1) = \Theta(n)$
    - No worse than if we had good-split the root node!

15

Analyzing Quicksort: Average Case

- Intuitively, the $\Theta(n)$ cost of a bad split (or 2 or 3 bad splits) can be absorbed into the $\Theta(n)$ cost of each good split
- Thus running time of alternating bad and good splits is still $O(n \lg n)$, with slightly higher constants

16

Analyzing Quicksort: Average Case

- Quicksort runs in O(n lg n) time on average (proved in the book)