

Lecture 7: Slides 34 - 55

Outline

- QuickSort

Quicksort

- Sorts “in place” like heapsort
- Very practical (perhaps w/ some tuning)
- Divide and Conquer approach

Quicksort – Divide

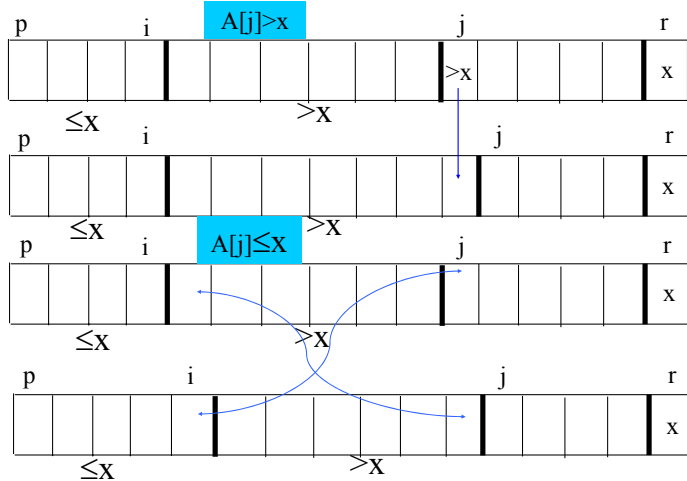
Partition(rearrange) the array $A[p..r]$ into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$

- Ensuring that each element of $A[p..q-1]$ is less than or equal to each element of $A[q+1..r]$.
- Return q as part of the scheme

QuickSort – Conquer & Combine

- Conquer: Sort the subarrays by recursive calls to quicksort
- Combine: Since the subarrays are sorted in place, no work is needed to combine them: the entire array $A[p..r]$ is sorted.

QuickSort –two cases shown



Performance of Quicksort

- Depends on whether the partitioning is balanced or unbalanced
- Depends on what is used for the pivot
- If “balanced”, quicksort runs asymptotically as fast as merge sort.
- If “unbalanced”, quicksort runs asymptotically as fast as insertion sort

Worst – Case partitioning

- This happens when the partitioning produces one subproblem with $n-1$ elements and a second subproblem with 0 elements.

Assume we have unbalanced partitioning. We have shown that the partitioning is $\Theta(n)$ time. The recursive call to the 0 length array just returns and $T(0) = \Theta(1)$.

Worst – Case partitioning (cont)

The recurrence for the running time is then

$$T(n) = T(n-1) + T(0) + \Theta(n) \\ = T(n-1) + \Theta(n)$$

This is an arithmetic series, which is $\Theta(n^2)$

If the partitioning is maximally unbalanced, the running time is $\Theta(n^2)$

This occurs in a common case, i.e., when the data is already sorted.

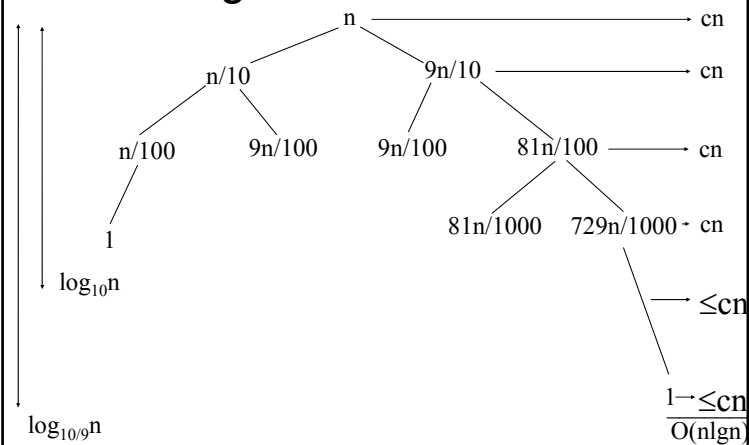
Best-Case partitioning

- Partition produces 2 subprograms, both of size at most $n/2$, since one is of size $\lfloor n/2 \rfloor$ and the other is of size $\lceil n/2 \rceil - 1$.
- So, we have $T(n) \leq 2T(n/2) + \Theta(n)$
- Case 2 of the master theorem gives $T(n) = O(n \lg n)$

How about the average case?

- The average case is fortunately much closer to the best case than the worst case.
- If we assume for example a 9-1 proportional split (such that the partition operation always does quite poorly) we have
- $T(n) \leq T(9n/10) + T(n/10) + cn$ where cn is hidden in the $\Theta(n)$ term.

Average case recursion tree



More quicksort

- Bad splits at the root are worse than bad splits farther down the tree...
 - A bad split at the root costs n and produces two subarrays of $n-1$ and 0 . Then the partitioning of the $n-1$ subarray costs $n-1$ and produces $(n-1)/2 - 1$ and $(n-1)/2$ sizes.
- The running time of quicksort even when alternating between good and bad splits is $O(n \lg n)$ but with a slightly larger constant.

Randomized quicksort

- The next idea is an improvement for quicksort...because the partition procedure is where the breakdown occurs when the data is sorted.
- We have assumed that all permutations of the input data are equally likely...but that is not likely to hold in the real world.
- We could randomize the input to obtain good average-case performance over all inputs by explicitly permuting the input. However, we will use random sampling, meaning that the pivot will be chosen randomly from the range subarray $A[p..r]$

Randomized Quicksort – pseudocode

```
RANDOMIZED-PARTITION(A,p,r)
  i ←RANDOM(p,r)
  exchange A[r] ↔ A[i]
  return Partition(A,p,r)
RANDOMIZED QUICKSORT(A,p,r)
if p<r
  then q ←RANDOMIZED-PARTITION(A,p,r)
      RANDOMIZED-QUICKSORT(A,p,q-1)
      RANDOMIZED-QUICKSORT(A,q+1,r)
```

Exercises...

- Why do we analyze the average-case performance of a randomized algorithm and not its worst-case performance?
- During the running of the procedure RANDOMIZED-QUICKSORT, how many calls are made to the random-number generator RANDOM in the worst case? Answer in terms of Θ -notation.

Worst-Case analysis of quicksort

We now prove the assertion that the worst case running time is $\Theta(n^2)$

For an input of size n and q ranges from 0 to $n-1$ because PARTITION produces two subproblems with total size $n-1$.

$$T(n) = \max_{\text{where } 0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

Guess that $T(n) \leq cn^2$ for some c then we have by substitution into the recurrence

Worst-Case analysis of quicksort (cont)

$$T(n) \leq \max (cq^2 + c(n-q-1)^2) + \Theta(n)$$

where $0 \leq q \leq n-1$

$$= c \cdot \max (q^2 + (n-q-1)^2) + \Theta(n)$$

where $0 \leq q \leq n-1$

We can use a little calculus

here...remember that for a local minimum
the second derivative must be > 0 when
the first derivative is 0.

Worst-Case analysis of quicksort (cont)

We have maximum at both endpoints

$$(q^2 + (n-q-1)^2) \leq (n-1)^2 = n^2 - 2n + 1$$

for $0 \leq q \leq n-1$

$$T(n) \leq cn^2 - c(2n-1) + \Theta(n)$$

$$\leq cn^2$$

The worst case running time is $\Theta(n^2)$