Parma University, Math. Dept.

Algorithms and Data Structures
Quick-sort

Marco Pellegrini

3/18/03 Spring 2002, lecture 6 1

Summary of lecture 6

- Quicksort
  Analysis (best, worst, average cases)
  Randomized Quicksort

3/18/03 2

Quicksort

- It is the 4th method for sorting numbers that we will survey (after insertion-sort, merge-sort, heap-sort).
- Quick-sort sorts in place, i.e. using only the array of input values.
- Quicksort has a good AVERAGE time O(n log n).
- The constant factors in the O(.) are smaller than, say, heap-sort.
- It is a good example for analysis.

3/18/03 3

Quicksort
(main idea)

- Quicksort takes as input a segment of an array of numbers A[p..r] between indices p and r.
- Divide step. A[p..r] is split into A[p..q] and A[q+1..r]. The elements in A[p..q] are smaller than those in A[q+1..r].
- Conquer step. Recursively sort A[p..q] and A[q+1..r].
- Merge Step. Since we sort in place A[p..r] is now sorted without any further action.

3/18/03 4

Quicksort
the code

```
Quick-sort(A: array;p,r: indices)
  IF p < r
   THEN q := Partition(A,p,r)
        Quick-sort(A,p,q)
        Quick-sort(A,q+1,r)
  END IF
```

- To sort A the initial call is Quick-sort(A,1,length(A)).
- The only part left to explain is how the procedure Partition works.

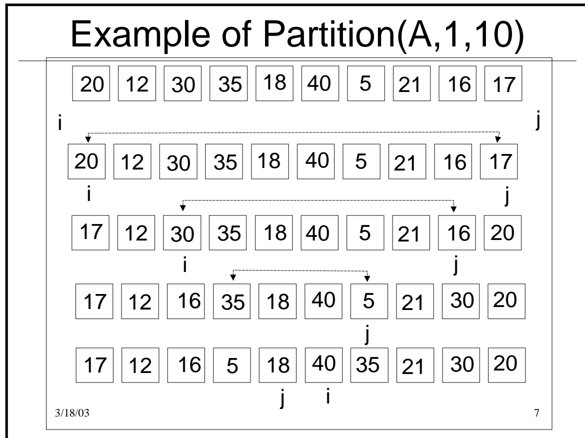
3/18/03 5

Partition
the code

```
Partition(A,p,r)
 x := A[p]      % x is the pivot
 i := p-1, j := r+1
 WHILE True
  DO REPEAT j := j -1
     UNTIL  A[j] ≤ x
     REPEAT i := i+1
     UNTIL  A[i] ≥ x
     IF i < j
       THEN  Exchange A[i] and A[j]
       ELSE  RETURN j
     END IF
 END WHILE
```

3/18/03 6

Example of Partition(A,1,10)



Exercise

- Simulate Partition on the array:
  A = 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21
- Repeat the simulation on the two sub-arrays obtained when i and j cross each other.
- Repeat until the array is sorted.

Performance of Quicksort

- Partition visits each element of the array A[p..r] only once and perform at most constant number of operations on each element.
- Therefore partition takes time LINEAR in the size of the input array.
- Qicksort calls Partition and recursively itself on the two sub-arrays produced by Partition.
- The time of Quicksort depends *CRITICALLY* on the relative SIZE of the two sub-arrays A[p..q] and A[q+1..r].

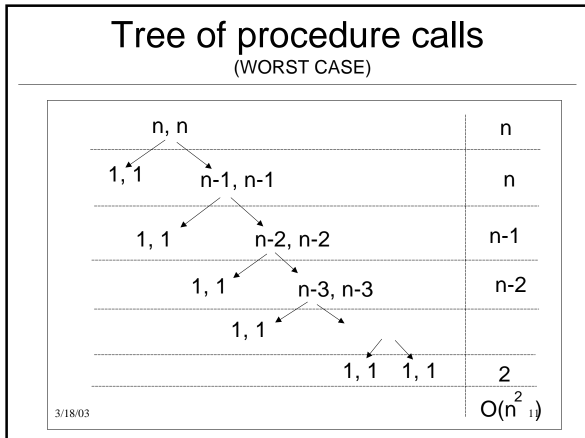
Performance of Qicksort

(WORST CASE)

- The worst case is when the array A[p..r] of n elements is split in a very unbalanced way, with one sub-array of size 1 and one of size n-1.
- If we call T(n) the worst case time we have:
  T(1) = O(1).
  T(n) = O(n) + T(n-1).
- Expanding the above equation we get:
  T(n) = O(n) + O(n-1) + O(n-2) +...+O(1) =
  = $O(n^2)$.
- A second way to derive this result is to use the tree of procedure calls.

Tree of procedure calls

(WORST CASE)



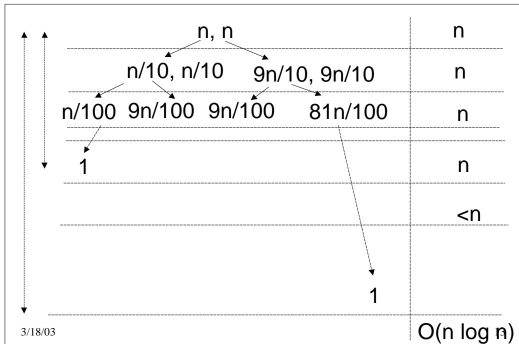
Performance of Quicksort 3

(BEST CASE)

- The best case is when the procedure Partition returns two sub arrays of size approximately n/2 and n/2.
- In this case calling U(n) the time of the best case we have:
  U(1) = 1.
  U(n) = O(n) + 2U(n/2).
- This is the same equation we had in the case of MERGE-SORT. We can solve it using Master's theorem or using the tree of procedure calls.
  The best running time is O(n log n).

How Balanced should the partition be?

(case of 9-to-1 split)



Performance of Quicksort, 4.

(random choice of pivots)

- We saw that a 1-to-1 split gives time O(n log n) and a 9-to-1 split gives still time O(n log n), for a larger constant in the big-Oh.
- Instead a split without fixed ratio (1/n)-to-((n-1)/n) gives us time O(n^2).
- If we choose the pivot at random we are more likely to get a split with bounded ratio rather than one with unbounded ration.
- For example the chances to get a 9-to-1 ratio are (9/10), the chances not to get it is only (1/10).
- Thus we rewrite Quick-sort with a random choice of the pivot.

Randomized Quick-sort

We suppose that we have at our disposal a generator of random numbers: RANDOM(a,b) which returns an integer between a and b, each with the same probability (1/(b-a+1)).

```
RANDOMIZED-PARTITION(A,p,r)
    i := RANDOM(p,r)
    exchange A[p] with A[i]
    Partition(A,p,r).
```

```
RANDOMIZED-Quick-sort(A,p,r)
    IF p < r
      THEN q := RANDOMIZED-PARTITION(A,p,r)
          RANDOMIZED-Quick-sort(A,p,q)
          RANDOMIZED-Quick-sort(A,q+1,r)
    END IF
```


Average running time.

(rank of an element)

- The RANK of x in A[p..r] is the number of values in A[p..r] that are smaller or equal to x.
- Example: RANK(7) in 30,12,24,7,6,8,2 is: 4.
- The index returned by Partition depends only on the rank of A[p], the first element of the array A[p..r].
- If RANK(A[p]) is 1, then A[p] is never moved and the final value of j is 1.
- If RANK(A[p]) is greater than 1, then A[p] is moved to the right side of the array and the value j returned is RANK[A(p)] -1.

- After exchanging A[i] with A[p], with i chosen randomly, the first element can assume every rank with probability 1/(r-p+1).
- So we have as output j=p with probability 2/(r-p+1) and every other value j=p+1,..,r with probability 1/(r-p+1).
- We can now write a recursion for the average time Y(n):

$$Y(n)=O(n)+(2/n)[Y(1)+Y(n-1)] +(1/n)\sum_{q=2}^{q=n-1} [Y(q)+Y(n-q)]$$

- Simplifications: We know the average time is less than the worst case time, so (1/n)[Y(1)+Y(n-1)] <= (1/n)[O(1)+O(n^2)]= O(n).
- We can take the other term (1/n)[Y(1)+Y(n-1)] within the summation by starting from q=1.
- In the summation for q=k we get Y(k)+Y(n-k), for q=n-k we get Y(n-k)+Y(k), thus each term is repeated twice. Finally:

$$Y(n) = O(n) + (2/n)\sum_{k=1}^{k=n-1} Y(k)$$

Lemma

$$\sum_{k=1}^{n-1} k \log k \le \tfrac{1}{2} n^2 \log n - \tfrac{1}{8} n^2$$

Proof: $\sum_{k=1}^{n-1} k \log k = \sum_{k=1}^{\lceil n/2 \rceil - 1} k \log k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \log k$ so:

$$\sum_{k=1}^{n-1} k \log k \le (\log n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + (\log n) \sum_{k=\lceil n/2 \rceil}^{n-1} k =$$

$$= (\log n) \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \le (\log n) \frac{n(n-1)}{2} - \frac{n/2(n/2-1)}{2} \le$$

$$\le \tfrac{1}{2} n^2 \log n - \tfrac{1}{8} n^2$$

3/18/03 19

Proof by induction of a bound on Y(n)

Thesis: Y(n) ≤an(log n)+bn, for constants a and b.

Derivation:

$$Y(n) \le \frac{2}{n} \sum_{k=1}^{n-1} Y(k) + O(n) \le \frac{2a}{n} \sum_{k=1}^{n-1} k \log k + \frac{2b(n-1)}{n} + O(n) \le$$

$$\le \frac{2a}{n} \left(\tfrac{1}{2} n^2 \log n - \tfrac{1}{8} n^2 \right) + \frac{2b(n-1)}{n} + O(n) \le$$

$$\le an \log n + b + \underbrace{\left(O(n) + b - \frac{an}{4} \right)}_{} \le an \log n + b$$

Negative for a large enough.

3/18/03 20

Conclusions

- We have seen the algorithm Quicksort to sort an array of numbers *in place*.
- We have derived the worst case running time and the best case running time.
- A randomized version of Quicksort has a good average running time.
- Example of writing and manipulation of recursive equation.

3/18/03 21