Algorithms

Quicksort

Slide credit: David Luebke (Virginia)

Sorting revisited

We have seen algorithms for sorting: INSERTION-SORT, MERGESORT

More generally: given a sequence of items

Each item has a characteristic called sorting key. The values of the sorting key belong to a set on which there exists a total order relationship

Sorting the sequence = arrange its elements such that the sorting keys are in increasing (or decreasing) order

Sorting revisited

Other assumptions:

We shall consider that the sequence is stored in a random access memory (e.g. in an array)

This means that we will discuss about internal sorting

We shall analyze only sorting methods which are in place (the additional space needed for sorting has at most the size of an element/few elements.

Stability: preserves ordering of elements with identical keys

Stability

Example:

- Initial configuration:

  ((Adam,9), (John, 10), (Peter,9), (Victor,8))

- Stable sorting :

  ((John,10),(Adam,9),(Peter,9),(Victor,8))

- Unstable sorting :

  ((John,10), (Peter,9),(Adam,9), (Victor,8))

Insertion sort – stability

8 5 9 8' 1 $\longrightarrow$ 5 8 9 8' 1

5 8 9 8' 1 $\longrightarrow$ 5 8 9 8' 1

5 8 9 8' 1 $\longrightarrow$ 5 8 8' 9 1

5 8 8' 9 1 $\longrightarrow$ 1 5 8 8' 9

The insertion method is stable

Homework 2

- Assigned today, due next Wednesday
- Will be on web page shortly after class
- Thursday's seminar: will recover next week.

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

Quicksort Code

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

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?*

Partition In Words

- Partition(A, p, r):
  - Select an element to act as the “pivot” (*which?*)
  - Grow two regions, A[p..i] and A[j..r]
    - All elements in A[p..i] <= pivot
    - All elements in A[j..r] >= pivot
  - Increment i until A[i] >= pivot
  - Decrement j until A[j] <= pivot
  - Swap A[i] and A[j]
  - Repeat until i >= j
  - Return j

***Note: slightly different from book’s* `partition()`**

Partition Code

```
Partition(A, p, r)
    x = A[p];
    i = p - 1;
    j = r + 1;
    while (TRUE)
        repeat
            j--;
        until A[j] <= x;
        repeat
            i++;
        until A[i] >= x;
        if (i < j)
            Swap(A, i, j);
        else
            return j;
```

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

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

Partition Code

```
Partition(A, p, r)
    x = A[p];
    i = p - 1;
    j = r + 1;
    while (TRUE)
        repeat
            j--;
        until A[j] <= x;
        repeat
            i++;
        until A[i] >= x;
        if (i < j)
            Swap(A, i, j);
        else
            return j;
```

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

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

Analyzing Quicksort

- In the worst case:

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

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

- Works out to

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

Analyzing Quicksort

- In the best case:

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

- What does this work out to?

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

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

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 9-to-1 split. This looks quite unbalanced!
  - The recurrence is thus:

    $T(n) = T(9n/10) + T(n/10) + n$ ← *Use n instead of O(n) for convenience (how?)*
  - *How deep will the recursion go?* (draw it)

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?*

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
    - Combined cost of splits = n + n -1 = 2n -1 = O(n)
    - No worse than if we had good-split the root node!

Analyzing Quicksort: Average Case

- Intuitively, the O(n) cost of a bad split (or 2 or 3 bad splits) can be absorbed into the O(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
- How can we be more rigorous?

Analyzing Quicksort: Average Case

- For simplicity, assume:
  - All inputs distinct (no repeats)
  - Slightly different **`partition()`** procedure
    - partition around a random element, which is not included in subarrays
    - all splits (0:n-1, 1:n-2, 2:n-3, … , n-1:0) equally likely
- *What is the probability of a particular split happening?*
- Answer: 1/n

Analyzing Quicksort: Average Case

- So partition generates splits
  (0:n-1, 1:n-2, 2:n-3, … , n-2:1, n-1:0)
  each with probability 1/n
- If T(n) is the expected running time,

$$T(n) = \frac{1}{n}\sum_{k=0}^{n-1}\left[T(k) + T(n-1-k)\right] + \Theta(n)$$

- *What is each term under the summation for?*
- *What is the Θ(n) term for?*

Analyzing Quicksort: Average Case

- So…

$$T(n) = \frac{1}{n}\sum_{k=0}^{n-1}\left[T(k)+T(n-1-k)\right]+\Theta(n)$$

$$= \frac{2}{n}\sum_{k=0}^{n-1}T(k)+\Theta(n)$$

Write it on the board

- Note: this is just like the book's recurrence (p166), except that the summation starts with k=0
- We'll take care of that in a second

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
  - Assume that the inductive hypothesis holds
  - Substitute it in for some value < n
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - *What's the answer?*
  - Assume that the inductive hypothesis holds
  - Substitute it in for some value < n
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
  - Substitute it in for some value $< n$
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
    - *What's the inductive hypothesis?*
  - Substitute it in for some value < n
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
    - $T(n) \le an \lg n + b$ for some constants a and b
  - Substitute it in for some value < n
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
    - $T(n) \leq an \lg n + b$ for some constants a and b
  - Substitute it in for some value < n
    - *What value?*
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
    - $T(n) \leq an \lg n + b$ for some constants a and b
  - Substitute it in for some value < n
    - The value k in the recurrence
  - Prove that it follows for n

Analyzing Quicksort: Average Case

- We can solve this recurrence using the dreaded substitution method
  - Guess the answer
    - $T(n) = O(n \lg n)$
  - Assume that the inductive hypothesis holds
    - $T(n) \le an \lg n + b$ for some constants a and b
  - Substitute it in for some value < n
    - The value k in the recurrence
  - Prove that it follows for n
    - Grind through it…

Analyzing Quicksort: Average Case

$$T(n) = \text{ㅎ}\ \frac{2}{n}\sum_{k=0}^{n-1} T(k) + \Theta(n)$$

The recurrence to be solved

$$\text{ㅎ}\ \frac{2}{n}\sum_{k=0}^{n-1}(ak\lg k + b) + \Theta(n)$$

Plug in inductive hypothesis

$$\text{ㅎ}\ \frac{2}{n}\left[b + \sum_{k=1}^{n-1}(ak\lg k + b)\right] + \Theta(n)$$

Expand out the k=0 case

$$\text{ㅎ}\ \frac{2}{n}\sum_{k=1}^{n-1}(ak\lg k + b) + \frac{2\mathrm{b}}{n} + \Theta(n)$$

2b/n is just a constant, so fold it into Θ(n)

$$\text{ㅎ}\ \frac{2}{n}\sum_{k=1}^{n-1}(ak\lg k + b) + \Theta(n)$$

Note: leaving the same recurrence as the book

Analyzing Quicksort: Average Case

$$T(n) = \text{ㅎ} \frac{2}{n} \sum_{k=1}^{n-1} (ak \lg k + b) + \Theta(n)$$

The recurrence to be solved

$$\text{ㅎ} \frac{2}{n} \sum_{k=1}^{n-1} ak \lg k + \frac{2}{n} \sum_{k=1}^{n-1} b + \Theta(n)$$

Distribute the summation

$$\text{ㅎ} \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + \frac{2b}{n}(n-1) + \Theta(n)$$

Evaluate the summation:
b+b+…+b = b (n-1)

$$\text{ㅎ} \frac{2a}{n} \sum_{k=1}^{n-1} \boxed{k \lg k +} 2b + \Theta(n)$$

Since n-1<n, 2b(n-1)/n < 2b

This summation gets its own set of slides later

Analyzing Quicksort: Average Case

$$T(n) \le \text{ㅎ} \frac{2\mathrm{a}}{n} \sum_{k=1}^{n-1} k \lg k + 2\mathrm{b} + \Theta(n)$$

The recurrence to be solved

$$\text{ㅎ} \frac{2\mathrm{a}}{n} \left(\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + 2\mathrm{b} + \Theta(n)$$

We'll prove this later

$$\text{ㅎ}\, an \lg n - \frac{a}{4} n + 2\mathrm{b} + \Theta(n)$$

Distribute the (2a/n) term

$$\text{ㅎ}\, an \lg n + b + \left(\Theta(n) + b - \frac{a}{4} n \right)$$

Remember, our goal is to get $T(n) \le an \lg n + b$

$$\text{ㅎ}\, an \lg n + b$$

Pick a large enough that an/4 dominates $\Theta(n)+b$

Analyzing Quicksort: Average Case

- So $T(n) \le an \lg n + b$ for certain a and b
  - Thus the induction holds
  - Thus T(n) = O(n lg n)
  - Thus quicksort runs in O(n lg n) time on average (phew!)
- Oh yeah, the summation…

Tightly Bounding The Key Summation

$$\sum_{k=1}^{n-1} k \lg k = \text{ㅎ} \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k$$

Split the summation for a tighter bound

$$\text{ㅎ} \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg n$$

The* lg *k* *in the second term is bounded by* lg *n

$$\text{ㅎ} \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

Move the* lg *n* *outside the summation

Tightly Bounding The Key Summation

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

The summation bound so far

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

The* lg *k* *in the first term is bounded by* lg *n/2

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

lg *n/2* = lg *n* - 1

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

Move (*lg *n* - 1*) outside the summation

Tightly Bounding The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \leq \text{ㅎ} \left(\lg n - 1\right) \sum_{k=1}^{\left(n/2\right)-1} k + \lg n \sum_{k=\left(n/2\right)}^{n-1} k$$

The summation bound so far

$$\text{ㅎ} \lg n \sum_{k=1}^{\left(n/2\right)-1} k - \sum_{k=1}^{\left(n/2\right)-1} k + \lg n \sum_{k=\left(n/2\right)}^{n-1} k$$

Distribute the (lg n - 1)

$$\text{ㅎ} \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\left(n/2\right)-1} k$$

The summations overlap in range; combine them

$$\text{ㅎ} \lg n \left(\frac{\left(n-1\right)\left(n\right)}{2}\right) - \sum_{k=1}^{\left(n/2\right)-1} k$$

The Gaussian series

Tightly Bounding The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \leq \text{ㅎ} \left(\frac{(n-1)(n)}{2}\right) \lg n - \sum_{k=1}^{(n/2)-1} k$$

The summation bound so far

$$\text{ㅎ} \frac{1}{2}\left[n(n-1)\right] \lg n - \sum_{k=1}^{n/2-1} k$$

Rearrange first term, place upper bound on second

$$\text{ㅎ} \frac{1}{2}\left[n(n-1)\right] \lg n - \frac{1}{2}\left(\frac{n}{2}\right)\left(\frac{n}{2} - 1\right)$$

X Gaussian series

$$\text{ㅎ} \frac{1}{2}\left(n^2 \lg n - n \lg n\right) - \frac{1}{8} n^2 + \frac{n}{4}$$

Multiply it all out

Tightly Bounding The Key Summation

$$\sum_{k=1}^{n-1} k \lg k \leq \text{ᄒ} \frac{1}{2}\left(n^2 \lg n - n \lg n\right) - \frac{1}{8} n^2 + \frac{n}{4}$$

$$\text{ᄒ} \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \text{ when } n \geq 2$$

Done!!!