



Algorithms: Divide and Conquer

Quicksort

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 \leq 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 $A[p..i] \leq \text{pivot}$
 - ◆ All elements in $A[j..r] \geq \text{pivot}$
 - Increment i until $A[i] \geq \text{pivot}$
 - Decrement j until $A[j] \leq \text{pivot}$
 - Swap $A[i]$ and $A[j]$
 - Repeat until $i \geq j$
 - 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 fail English

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,

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

- *What is each term under the summation for?*
- *What is the $\Theta(n)$ term for?*

Analyzing Quicksort: Average Case

- So...

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

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

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
 - ◆ $T(n) \leq an \lg n + b$ 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
 - ◆ $T(n) \leq an \lg n + b$ 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
 - ◆ Grind through it...

Analyzing Quicksort: Average Case

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

The recurrence to be solved

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

Plug in inductive hypothesis

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

Expand out the $k=0$ case

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

$2b/n$ is just a constant, so fold it into $\Theta(n)$

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

Note: leaving the same recurrence as the book

Analyzing Quicksort: Average Case

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

The recurrence to be solved

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

Distribute the summation

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

*Evaluate the summation:
 $b+b+\dots+b = b(n-1)$*

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

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

This summation gets its own set of slides later

Analyzing Quicksort: Average Case

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

The recurrence to be solved

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

We'll prove this later

$$= an \lg n - \frac{a}{4} n + 2b + \Theta(n)$$

Distribute the $(2a/n)$ term

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

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

$$\leq an \lg n + b$$

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

Analyzing Quicksort: Average Case

- So $T(n) \leq an \lg n + b$ 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 (pew!)
- Oh yeah, the summation...

Tightly Bounding The Key Summation

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

Split the summation for a tighter bound

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

The $\lg k$ in the second term is bounded by $\lg n$

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

Move the $\lg n$ outside the summation

Tightly Bounding The Key Summation

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

The summation bound so far

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

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

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

$\lg n/2 = \lg n - 1$

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

Move $(\lg n - 1)$ outside the summation

Tightly Bounding The Key Summation

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

The summation bound so far

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

Distribute the $(\lg n - 1)$

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

The summations overlap in range; combine them

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

The Gaussian series

Tightly Bounding The Key Summation

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

The summation bound so far

$$\leq \frac{1}{2} [n(n-1)] \lg n - \sum_{k=1}^{n/2-1} k$$

Rearrange first term, place upper bound on second

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

X Guassian series

$$\leq \frac{1}{2} (n^2 \lg n - n \lg n) - \frac{1}{8} n^2 + \frac{n}{4}$$

Multiply it all out

Tightly Bounding The Key Summation

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

Done!!!