Quicksort

Prepared by Suk Jin Lee

Overview

- 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

Quicksort

- Quicksort
  - Uses a Divide and conquer strategy
  - Sorts “in place” (cf. Mergesort needs extra space)
  - Very practical (with tuning)
  - The original problem partitioned into simpler sub-problems
  - Each sub problem considered independently
  - Unlike merge sort, no combining step: two subarrays form an already-sorted array

Quicksort

- Divide and conquer
  - **Divide**: partition the array $A[p \mathinner{..} r]$ into two subarrays $A[p \mathinner{..} q-1]$ and $A[q+1 \mathinner{..} r]$ such that each element of $A[p \mathinner{..} q-1] \leq A[q] \leq A[q+1 \mathinner{..} r]$
  - **Conquer**: sort the two subarrays $A[p \mathinner{..} q-1]$ and $A[q+1 \mathinner{..} r]$ by recursive calls to quicksort
  - **Combine**: subarrays already sorted. No work needed

```
QUICKSORT(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q – 1)
        QUICKSORT(A, q + 1, r)
```

Initial call is QUICKSORT$(A, 1, n)$

Partition

- Clearly, all the action takes place in the PARTITION() function
  - Rearrange 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

Partition

- Partition procedure

```
PARTITION(A, p, r)
    x = A[r]              // select the last element in A[] as the pivot
    i = p – 1
    for j = p to r – 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```

Partition

- Partition procedure

```
PARTITION(A, p, r)
    x = A[r]                 // select the last element in A[] as the pivot
    i = p – 1
    for j = p to r – 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```

What is the running time of PARTITION()?

Partition

- Partition procedure

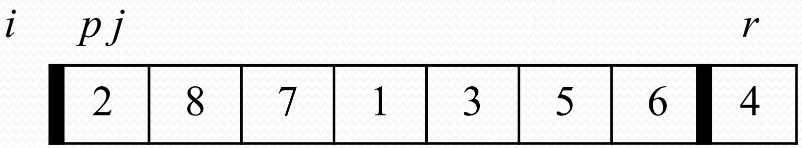
```
PARTITION(A, p, r)
    x = A[r]                  // select the last element in A[] as the pivot
    i = p – 1
    for j = p to r – 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```

What is the running time of PARTITION()?

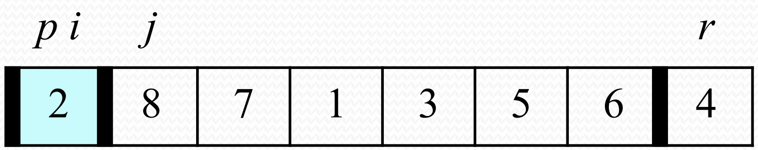
PARTITION() runs in $\Theta(n)$

Quicksort

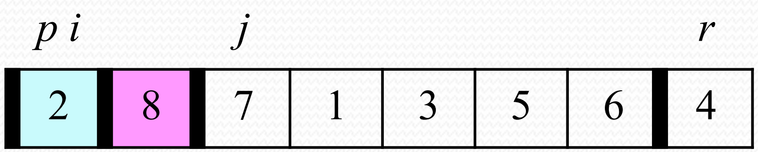
- Operation of Partition



$A[j] = 2 \leq$ Pivot $4 = A[r]$. $i = i + 1$
then exchange $A[i]$ with $A[j]$



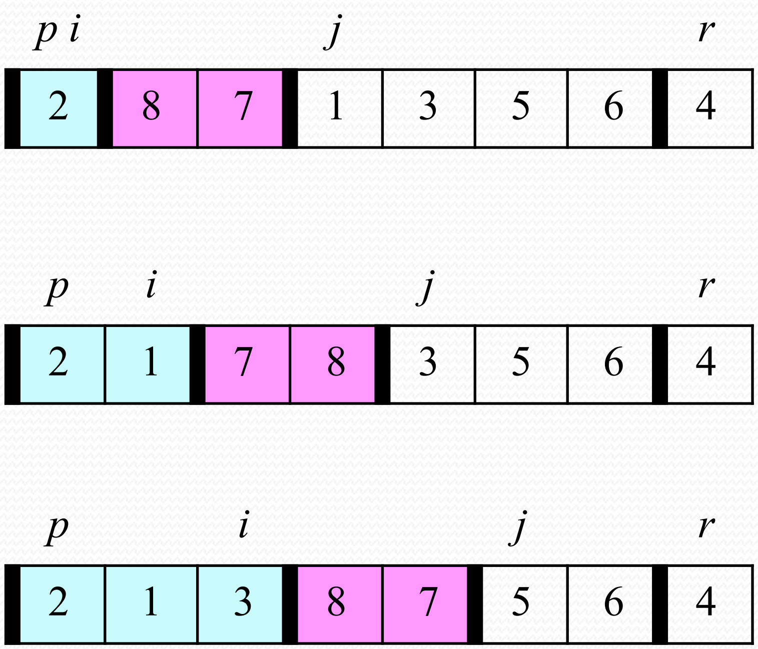
$A[j] >$ Pivot 4, no increase of i



$A[j] >$ Pivot 4, no increase of i

Quicksort

- Operation of Partition



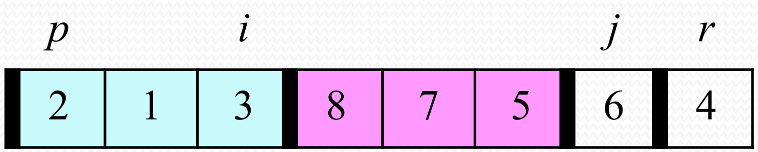
$A[j] \le$ Pivot 4. $i = i + 1$ then exchange $A[i]$ with $A[j]$

$A[j] \le$ Pivot 4. $i = i + 1$ then exchange $A[i]$ with $A[j]$

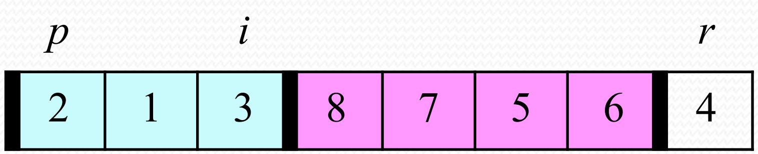
$A[j] >$ Pivot 4, no increase of i

Quicksort

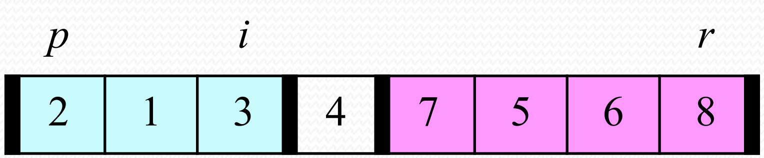
- Operation of Partition



$A[j]$ > Pivot 4, no increase of i



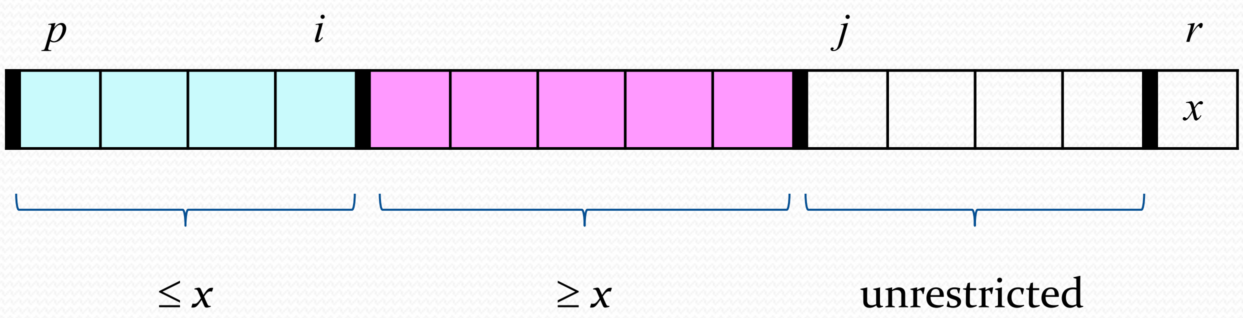
$j > r - 1$, escape the Loop
Exchange $A[i + 1]$ with $A[r]$



The pivot lies between the two partitions

Quicksort

- Four regions maintained by the procedure PARTITION on a subarray $A[p \dots r]$



The running time of PARTITION on subarray $A[p \,..\, r]$ is $\Theta(n)$

Quiz 1

- Using the previous figure model, illustrate the operation of PARTITION on the array

  $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$

Performance of Quicksort

Prepared by Suk Jin Lee

Quicksort Analysis

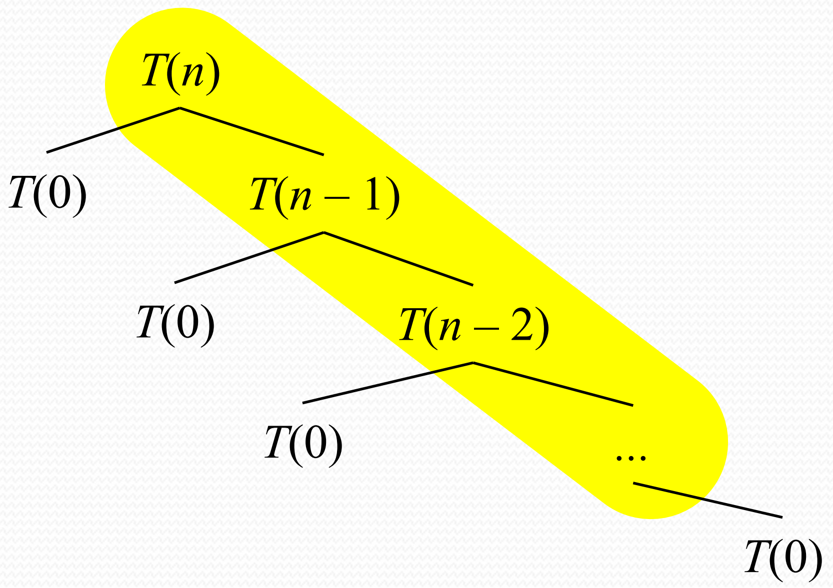
- *What will be the worst case for the algorithm?*
  - Partition is always unbalanced
- *What will be the best case for the algorithm?*
  - Partition is perfectly balanced
- *Which is more likely?*
  - The latter
- *Will any particular input elicit the worst case?*
  - Yes: Already-sorted input

Worst-case partitioning

- Worst-case partitioning
  - Produces one subproblem with $n - 1$ elements and one with 0 elements

    $T(n) \quad = T(n-1) + T(0) + \Theta(n)$ // $\Theta(n)$: partitioning cost

  - Partitioning costs $\Theta(n)$



Worst-case partitioning

- Worst-case partitioning
  - Produces one subproblem with $n - 1$ elements and one with 0 elements

$$
\begin{aligned}
T(n) &= T(n-1) + T(0) + \Theta(n) && \text{// } \Theta(n)\text{: partitioning cost} \\
&= T(n-1) + \Theta(n) && \text{// } T(0) = \Theta(1) \\
&= T(n-2) + \Theta(n) + \Theta(n) = T(n-2) + 2\cdot\Theta(n) \\
&\ldots\ldots \\
&= T(n-k) + \Theta(n) + (k-1)\cdot\Theta(n) = T(n-k) + k\cdot\Theta(n) \\
&= \Theta(n^2)
\end{aligned}
$$

Worst-case partitioning

- Worst-case partitioning
  - Produces one subproblem with $n - 1$ elements and one with 0 elements

    $T(n) = T(n-1) + T(0) + \Theta(n)$ // $\Theta(n)$: partitioning cost

    $= \Theta(n^2)$

    - Same running time as insertion sort
  - In fact, the worst-case running time occur when quicksort takes a sorted array as input, but insertion sort runs in $O(n)$ time in this case

Best-case partitioning

- Best-case partitioning
  - If we're really lucky, produces two subproblem each with $n/2$

$$T(n) = 2T(n/2) + \Theta(n) \quad , \Theta(n)\text{: partitioning cost}$$

Best-case partitioning

- Best-case partitioning
  - If we're really lucky, produces two subproblem each with $n/2$

$$T(n) = 2T(n/2) + \Theta(n)$$, $\Theta(n)$: partitioning cost

  - Using master theorem
    - $a = 2,\ b = 2,\ f(n) = \Theta(n)$
    - $f(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_2 2}\right) = \Theta(n)$
    - Case 2 applies:

$$T(n) = \Theta\left(n^{\log_2 2} \lg n\right) = \Theta(n \lg n)$$

Best-case partitioning

- Best-case partitioning
  - If we're really lucky, produces two subproblem each with $n/2$

$$T(n) = 2T(n/2) + \Theta(n) \quad , \Theta(n)\text{: partitioning cost}$$
$$= \Theta(n \lg n)$$

  - By equally balancing the two sides of the partition at every level of the recursion, we get an asymptotically faster algorithm

Balanced partitioning

- Balanced partitioning
  - Quicksort's average running time is much closer to the best case than to the worst case
  - Imagine that PARTITION always produces a 9-to-1 split

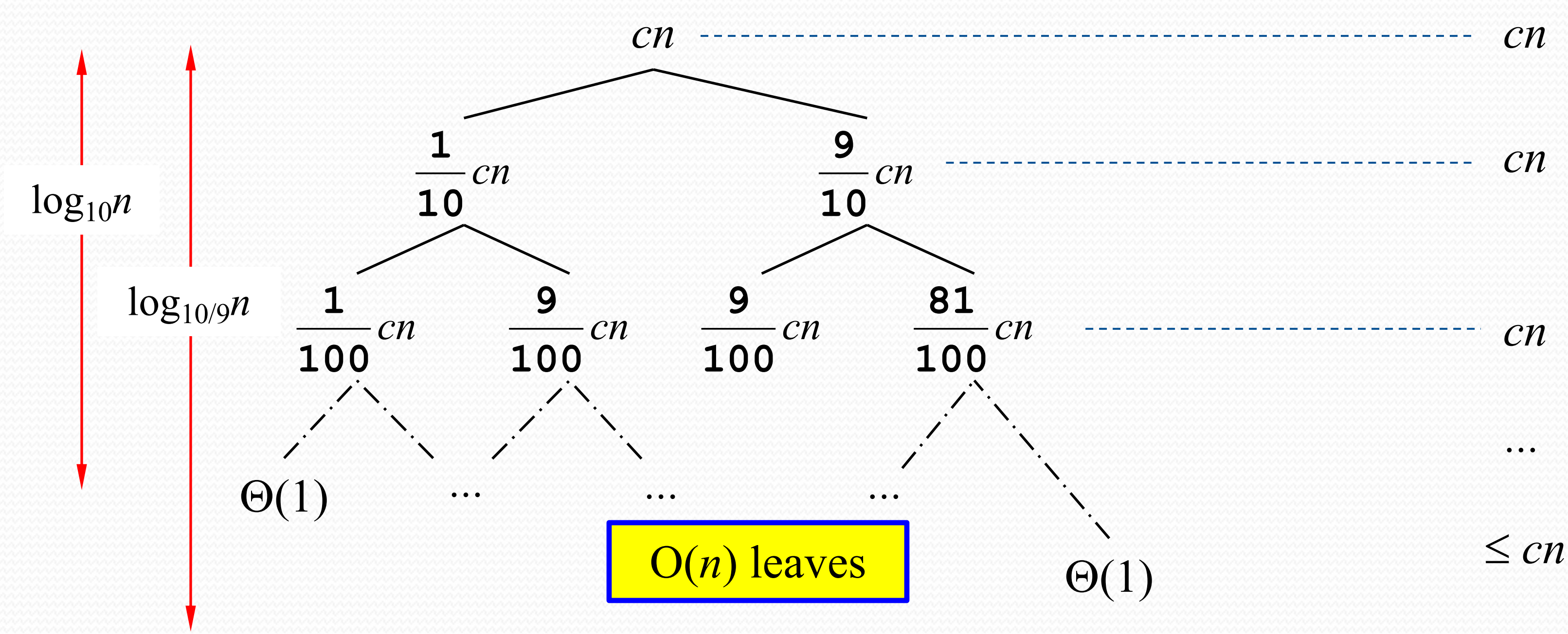
    We obtain the recurrence:

$$T(n) = T(9n / 10) + T(n / 10) + \Theta(n)$$

Balanced partitioning

- Balanced partitioning
  - What if the split is always 9-to-1?

$$T(n) = T(9n/10) + T(n/10) + \Theta(n)$$



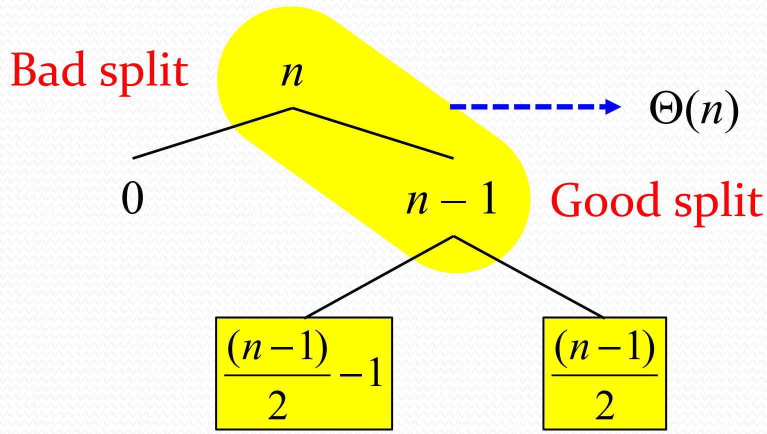
$$cn\log_{10}n \le T(n) \le cn\log_{10/9}n + O(n) = \mathbf{O(n \log n)}$$

Intuition for the average case

- Intuition for the average case
  - Splits in the recursion tree will not always be constant
  - There will usually be a mix of good and bad splits throughout the recursion tree
  - To see that this doesn't affect the asymptotic running time of quicksort, assume that levels alternate between best-case (good) and worst-case (bad) splits

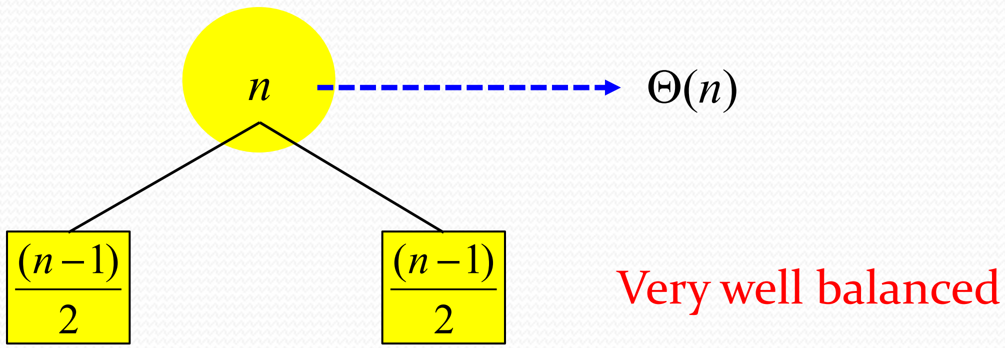
Intuition for the average case

- Intuition for the average case
  - Two levels of a recursion tree for quicksort
    - Partitioning cost: $\Theta(n) + \Theta(n-1) = \Theta(n)$



Intuition for the average case

- Intuition for the average case
  - A single level of a recursion tree for quicksort
    - Partitioning cost: $\Theta(n)$



Intuition for the average case

- Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ...

$L(n) = 2U(n/2) + \Theta(n)$ lucky

$U(n) = L(n-1) + \Theta(n)$ unlucky

- Solving:

$$L(n) = 2(L(n/2 - 1) + \Theta(n/2)) + \Theta(n)$$
$$= 2L(n/2 - 1) + \Theta(n)$$
$$= \Theta(n \lg n)$$: Master theorem case 2 applies

- *How can we make sure we are usually lucky?*

Quiz 2 (1)

- Use the substitution method to prove that the recurrence $T(n) = T(n - 1) + \Theta(n)$ has the solution $T(n) = \Theta(n^2)$

Quiz 2 (2)

- Use the substitution method to prove that the recurrence $T(n) = T(n-1) + \Theta(n)$ has the solution $T(n) = \Theta(n^2)$
  - We guess that $T(n) \leq O(n^2)$

$$
\begin{aligned}
T(n) \quad & \leq c_1(n-1)^2 + \Theta(n) \\
& \leq c_1(n-1)^2 + c_0 n \\
& \leq c_1 n^2 - (2c_1 - c_0)n + c_1 \\
& \leq c_1 n^2 \text{ for } n_0 \geq 1 \text{ and } c_0 > c_1
\end{aligned}
$$

Thus $T(n) \in O(n^2)$. Similarly, we can prove that $T(n) \in \Omega(n^2)$.

Quiz 3 (1)

- What is the running time of Quicksort when all elements of array *A* have the same value?

Quiz 3 (2)

- What is the running time of Quicksort when all elements of array A have the same value?
  - If all elements are the same, the quick sort partition return index $q = r$.
  - The problem with size n is reduced to one sub-problem with size $n - 1$:
    $T(n) = T(n - 1) + \Theta(n)$, $\Theta(n)$ is a partitioning cost

A Randomized version of Quicksort

Prepared by Suk Jin Lee

Randomized Quicksort

- Randomized version of quicksort
  - We have assumed that all input permutations are equally likely.
  - This is not always true.
  - To correct this, we add randomization to quicksort.

Randomized Quicksort

- ***Idea***: instead of always $A[r]$ as the pivot, we will select a randomly chosen element from the subarray $A[p \, . \, . \, r]$
  - Running time is independent of the input order.
  - No assumptions need to be made about the input distribution.
  - No specific input elicits the worst-case behavior.
  - The worst case is determined only by the output of a random-number generator.

Randomized Quicksort

RANDOMIZED-PARTITION(A, p, r)

1. i = RANDOM(p, r)
2. exchange $A[r]$ with $A[i]$
3. **return** PARTITION(A, p, r)

Randomly selecting the pivot element will, on average, cause the split of the input array to be reasonably well balanced

RANDOMIZED-QUICKSORT(A, p, r)

1. **if** $p < r$
2. q = RANDOMIZED-PARTITION(A, p, r)
3. RANDOMIZED-QUICKSORT($A, p, q - 1$)
4. RANDOMIZED-QUICKSORT($A, q + 1, r$)

Analysis of Quicksort

Prepared by Suk Jin Lee

Analysis of 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

Analysis of 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: $1/n$

Analysis of Quicksort: Average Case

- So partition generates splits

  ($0{:}n$-1, 1:n-2, 2:n-3, … , 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 $\Theta(n)$ term for?*

Analysis of Quicksort: Average Case

- So…

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

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

- Note: this is just like the recurrence expect that the summation starts with $k = 0$

Analysis of Quicksort: Average Case

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

Analysis of Quicksort: Average Case

- We can solve this recurrence using the 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

Analysis of Quicksort: Average Case

- We can solve this recurrence using the 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

Analysis of Quicksort: Average Case

- We can solve this recurrence using the 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

Analysis of Quicksort: Average Case

- We can solve this recurrence using the 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 constant a and b
  - Substitute it in for some value $< n$
  - Prove that it follows for n

Analysis of Quicksort: Average Case

- We can solve this recurrence using the 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 constant a and b
  - Substitute it in for some value $< n$
    - *What value?*
  - Prove that it follows for n

Analysis of Quicksort: Average Case

- We can solve this recurrence using the 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 constant a and b
  - Substitute it in for some value $< n$
    - The value k in the recurrence
  - Prove that it follows for n

Analysis of Quicksort: Average Case

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

The recurrence to be solved

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

Plug in inductive hypothesis

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

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 recurrence

Analysis of Quicksort: Average Case

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

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:
$b + b + \ldots + b = b\,(n - 1)$

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

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

This summation gets its own set of slides later

Analysis of Quicksort: Average Case

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

The recurrence to be solved

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

We'll prove this later

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

Distribute the (2*a*/*n*) term

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

Remember, our goal is to get T(*n*) ≤ *an* lg *n* + *b*

$$\le an \lg n + b$$

Pick a large enough that *an*/4 dominates Θ(*n*)+*b*

Analysis of 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!)

Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from code tuning.
- Quicksort behaves well even with caching and virtual memory.

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

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

The summation bound so far

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

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

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

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 Guassian series

Tightly Bounding The Key Summation

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

The summation bound so far

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

Rearrange first term, place upper bound on second

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

X Guassian series

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

Multiply it all out

Tightly Bounding The Key Summation

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

$$\le \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2 \text{ when } n \ge 2$$

Done!!!