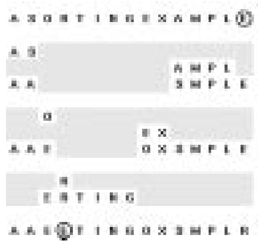
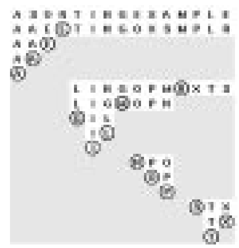
Lecture 3: Quicksort, Mergesort





Two Great Sorting Algorithms

Two great sorting algorithms.
- Full scientific understanding of their properties has enabled us to
- hammer them into practical system sorts.
- Occupies a prominent place in world's computational infrastructure.
  - database search
  - computational geometry
  - finding repetition in DNA sequences
  - Burrows-Wheeler transform

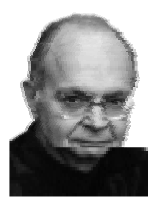
Mergesort.
- Java system sort.

Quicksort.
- Unix system sort.
- C standard library function is even named qsort().

Estimating the Running Time

Total running time is sum of cost * frequency for all of the basic ops.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input.

For sorting.
- A = # recursive calls.
- B = # exchanges.
- C = # comparisons.
- Cost on a typical machine = 35A + 11B + 4C.



Donald Knuth

Estimating the Running Time

An easier alternative.

(i) Analyze asymptotic growth as a function of input size N.

(ii) For medium N, run and measure time.

(iii) For large N, use (i) and (ii) to predict time.

Asymptotic growth rates.
- Estimate as a function of input size N.
  - N, $N \log N$, N^2, N^3, 2^N, $N!$
- Ignore lower order terms and leading coefficients.
  - Ex. $6N^3 + 17N^2 + 56$ is asymptotically proportional to N^3

Insertion sort is quadratic. On arizona: 1 second for N = 10,000.
- How long for N = 100,000? 100 seconds (100 times as long).
- N = 1 million? 2.78 hours (another factor of 100).
- N = 1 billion? 317 years (another factor of 10^6).
- N = 1 trillion?

Why Does It Matter?

Run time in nanoseconds -->		$1.3 N^3$	$10 N^2$	$47 N \log_2 N$	$48 N$
Time to solve a problem of size	1000	1.3 seconds	10 msec	0.4 msec	0.048 msec
	10,000	22 minutes	1 second	6 msec	0.48 msec
	100,000	15 days	1.7 minutes	78 msec	4.8 msec
	million	41 years	2.8 hours	0.94 seconds	48 msec
	10 million	41 millennia	1.7 weeks	11 seconds	0.48 seconds
Max size problem solved in one	second	920	10,000	1 million	21 million
	minute	3,600	77,000	49 million	1.3 billion
	hour	14,000	600,000	2.4 trillion	76 trillion
	day	41,000	2.9 million	50 trillion	1,800 trillion
N multiplied by 10, time multiplied by		1,000	100	10+	10

Orders of Magnitude

Seconds	Equivalent
1	1 second
10	10 seconds
10^2	1.7 minutes
10^3	17 minutes
10^4	2.8 hours
10^5	1.1 days
10^6	1.6 weeks
10^7	3.8 months
10^8	3.1 years
10^9	3.1 decades
10^{10}	3.1 centuries
. . .	forever
10^{21}	age of universe

Meters Per Second	Imperial Units	Example
10^{-10}	1.2 in / decade	Continental drift
10^{-8}	1 ft / year	Hair growing
10^{-6}	3.4 in / day	Glacier
10^{-4}	1.2 ft / hour	Gastro-intestinal tract
10^{-2}	2 ft / minute	Ant
1	2.2 mi / hour	Human walk
10^2	220 mi / hour	Propeller airplane
10^4	370 mi / min	Space shuttle
10^6	620 mi / sec	Earth in galactic orbit
10^8	62,000 mi / sec	1/3 speed of light

Powers of 2		
	2^{10}	thousand
	2^{20}	million
	2^{30}	billion

Big Oh Notation

$\Theta()$, $O()$, and $\Omega()$ notation.

- $\Theta(N^2)$ means $\{ N^2, 17N^2, N^2 + 17N^{1.5} + 3N, \ldots \}$
  - ignore lower order terms and leading coefficients
- $O(N^2)$ means $\{ N^2, 17N^2, N^2 + 17N^{1.5} + 3N, N^{1.5}, 100N, \ldots \}$
  - $\Theta(N^2)$ and faster
  - use for upper bounds
- $\Omega(N^2)$ means $\{ N^2, 17N^2, N^2 + 17N^{1.5} + 3N, N^3, 100N^5, \ldots \}$
  - $\Theta(N^2)$ and slower
  - use for lower bounds

Mergesort

Mergesort (divide-and-conquer)

- Divide array into two halves.

Jon von Neumann (1945)

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R		I	T	H	M	S

divide

Mergesort

Mergesort (divide-and-conquer)

- Divide array into two halves.
- Recursively sort each half.

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R	I	T	H	M	S	divide
A	G	L	O	R	H	I	M	S	T	sort

Mergesort

Mergesort (divide-and-conquer)

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R	I	T	H	M	S	divide
A	G	L	O	R	H	I	M	S	T	sort
A	G	H	I	L	M	O	R	S	T	merge

Mergesort Analysis

How long does mergesort take?

- Bottleneck = merging (and copying).
  - merging two files of size N/2 requires N comparisons
- T(N) = comparisons to mergesort N elements.
  - to make analysis cleaner, assume N is a power of 2

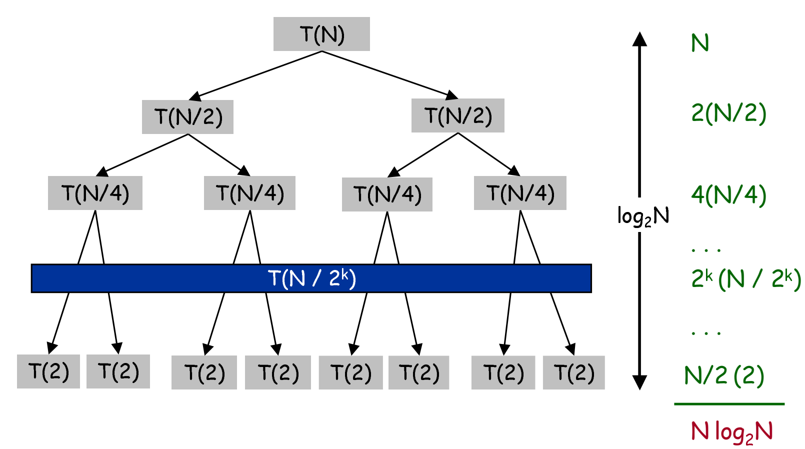
$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{2T(N/2)}_{\text{sorting both halves}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$

Claim. $T(N) = N \log_2 N$.

- Note: same number of comparisons for ANY file.
  - even already sorted
- We'll give several proofs to illustrate standard techniques.

Proof by Picture of Recursion Tree

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{2T(N/2)}_{\text{sorting both halves}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$



Proof by Telescoping

Claim. $T(N) = N \log_2 N$ (when N is a power of 2).

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{2T(N/2)}_{\text{sorting both halves}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$

Proof. For N > 1:

$$\begin{aligned} \frac{T(N)}{N} &= \frac{2T(N/2)}{N} + 1 \\ &= \frac{T(N/2)}{N/2} + 1 \\ &= \frac{T(N/4)}{N/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(N/N)}{N/N} + \underbrace{1 + \cdots + 1}_{\log_2 N} \\ &= \log_2 N \end{aligned}$$

Mathematical Induction

Mathematical induction.

- Powerful and general proof technique in discrete mathematics.
- To prove a theorem true for all integers $k \geq 0$:
  - Base case: prove it to be true for N = 0.
  - Induction hypothesis: assuming it is true for arbitrary N
  - Induction step: show it is true for N + 1

Claim: $0 + 1 + 2 + 3 + \ldots + N = N(N+1) / 2$ for all $N \geq 0$.

Proof: (by mathematical induction)

- Base case (N = 0).
  - $0 = 0(0+1) / 2$.
- Induction hypothesis: assume $0 + 1 + 2 + \ldots + N = N(N+1) / 2$
- Induction step: $0 + 1 + \ldots + N + N + 1 = (0 + 1 + \ldots + N) + N+1$
  $= N (N+1) /2 + N+1$
  $= (N+2)(N+1) / 2$

Proof by Induction

Claim. $T(N) = N \log_2 N$ (assuming N is a power of 2).

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{2T(N/2)}_{\text{sorting both halves}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$

Proof. (by induction on N)

- Base case: N = 1.
- Inductive hypothesis: $T(N) = N \log_2 N$.
- Goal: show that $T(2N) = 2N \log_2 (2N)$.

$$\begin{aligned} T(2N) &= 2T(N) + 2N \\ &= 2N \log_2 N + 2N \\ &= 2N(\log_2(2N) - 1) + 2N \\ &= 2N \log_2(2N) \end{aligned}$$

Proof by Induction

What if N is not a power of 2?

- T(N) satisfies following recurrence.

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{T(\lceil N/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor N/2 \rfloor)}_{\text{solve right half}} + \underbrace{N}_{\text{merging}} & \text{otherwise} \end{cases}$$

Claim. $T(N) \leq N \lceil \log_2 N \rceil$.

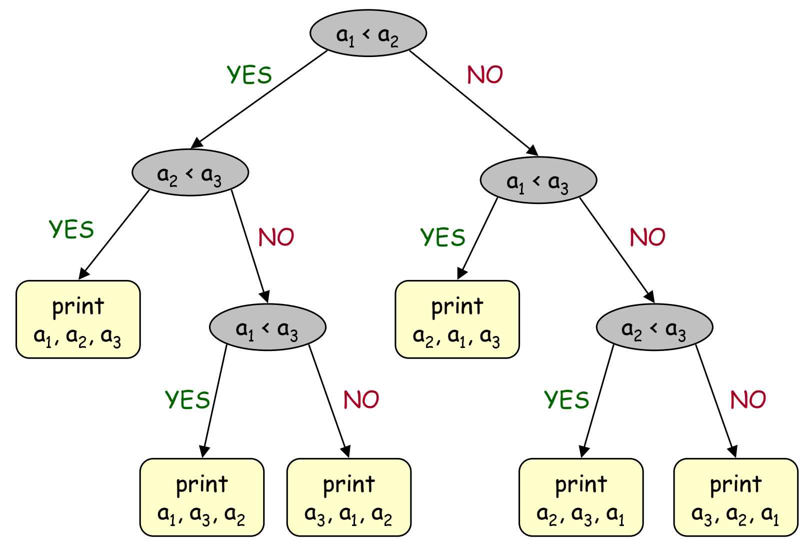
Proof. See supplemental slides.

Computational Complexity

Framework to study efficiency of algorithms. Example = sorting.

- MACHINE MODEL = count fundamental operations.
  - count number of comparisons
- UPPER BOUND = algorithm to solve the problem (worst-case).
  - $N \log_2 N$ from mergesort
- LOWER BOUND = proof that no algorithm can do better.
  - $N \log_2 N - N \log_2 e$
- OPTIMAL ALGORITHM: lower bound ~ upper bound.
  - mergesort

Decision Tree



Comparison Based Sorting Lower Bound

Theorem. Any comparison based sorting algorithm must use $\Omega(N \log_2 N)$ comparisons.

Proof. Worst case dictated by tree height h.

- N! different orderings.
- One (or more) leaves corresponding to each ordering.
- Binary tree with N! leaves must have height

$$\begin{aligned} h &\geq \log_2 (N!) \\ &\geq \log_2 (N/e)^N \\ &= N \log_2 N - N \log_2 e \end{aligned}$$

Stirling's formula

Food for thought. What if we don't use comparisons?

✎ Stay tuned for radix sort.

Sorting Analysis Summary

Running time estimates:

- Home pc executes 10^8 comparisons/second.
- Supercomputer executes 10^{12} comparisons/second.

Insertion Sort (N^2)

computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort (N log N)

thousand	million	billion
instant	1 sec	18 min
instant	instant	instant

Lesson 1: good algorithms are better than supercomputers.

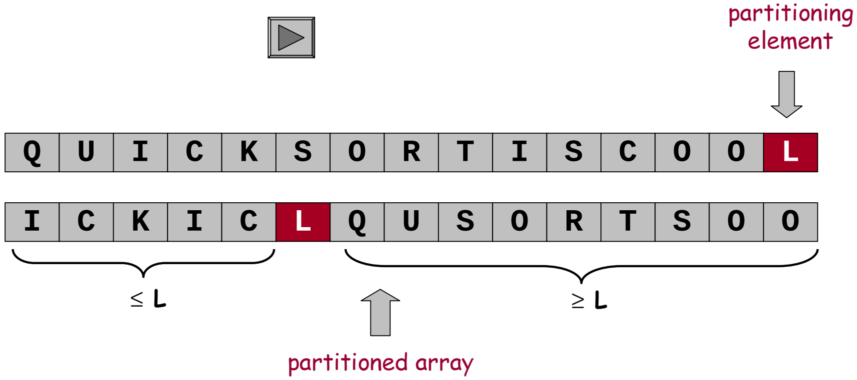
How does quicksort fit into the picture?

Quicksort

C. A. R. Hoare

Quicksort.

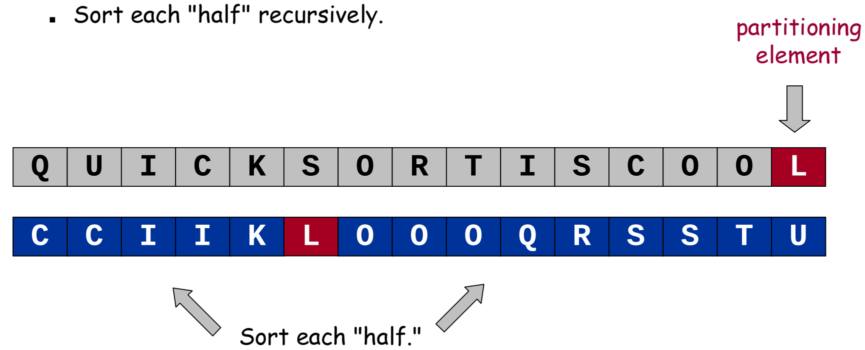
- Partition array so that:
  - some partitioning element `a[m]` is in its final position
  - no larger element to the left of `m`
  - no smaller element to the right of `m`



Quicksort

Quicksort.

- Partition array so that:
  - some partitioning element `a[m]` is in its final position
  - no larger element to the left of `m`
  - no smaller element to the right of `m`
- Sort each "half" recursively.



Quicksort: Worst Case

Number of comparisons in worst case is quadratic.

- $N + (N-1) + (N-2) + \ldots + 1 = N(N-1)/2$

Worst-case inputs.

- Already sorted!
- Reverse sorted.
- All equal. (Stay tuned.)

Fix.

- Pick partitioning element at random.
- Guarantees good performance.

Quicksort: Average Case

Precondition: file is randomly shuffled beforehand.

- Or, partition on RANDOM element.

Expected number of comparisons.

- Roughly $2 N \ln N \approx 1.39 N \log_2 N$.
  - see next slide for proof
- 39% more than mergesort but faster in practice.
  - lower cost of other high-frequency instructions
- Worst case still proportional to N^2.
  - more likely that machine struck by lightning

Quicksort: Average Case

Theorem. The average number of comparisons C_N to quicksort a random file of N elements is about 2N ln N.

The precise recurrence satisfies $C_0 = C_1 = 0$ and for $N \ge 2$:

$$\begin{aligned} C_N &= N+1+\frac{1}{N}\sum_{k=1}^{N}\left(C_k + C_{N-k}\right) \\ &= N+1+\frac{2}{N}\sum_{k=1}^{N} C_{k-1} \end{aligned}$$

Multiply both sides by N and subtract the same formula for N-1:

$$NC_N - (N-1)C_{N-1} = N(N+1) - (N-1)N + 2C_{N-1}$$

Simplify to:

$$NC_N = (N+1)\,C_{N-1} + 2N$$

Quicksort: Average Case

Divide both sides by N(N+1) to yield a telescoping sum:

$$\begin{aligned} \frac{C_N}{N+1} &= \frac{C_{N-1}}{N} + \frac{2}{N+1} \\ &= \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\ &= \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\ &= \vdots \\ &= \frac{C_2}{3} + \sum_{k=3}^{N} \frac{2}{k+1} \end{aligned}$$

Approximate the exact answer by an integral:

$$\frac{C_{N+1}}{N} \approx \sum_{k=1}^{N}\frac{2}{k} \approx \int_{k=1}^{N}\frac{2}{k} = 2\ln N \approx 1.39\log_2 N$$

Quicksort: Improvements

Median of sample.

- Best choice of partitioning element = median.
- Estimate true median by taking median of sample.
- Number of comparisons close to N $\log_2$N.
- FEWER large files.
- Slightly more exchanges, overhead.

Insertion sort small files.

- Even quicksort has too much overhead for tiny files.
- Can delay insertion sort until end.

Dealing with equal keys. Stay tuned for 3-way partitioning.

Optimize parameters.

- Median of 3 elements.
- Cutoff to insertion sort for < 10 elements.

Sorting Analysis Summary

Running time estimates:

- Home pc executes 10^8 comparisons/second.
- Supercomputer executes 10^{12} comparisons/second.

Insertion Sort (N^2)

computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort (N log N)

thousand	million	billion
instant	1 sec	18 min
instant	instant	instant

Quicksort (N log N)

thousand	million	billion
instant	0.3 sec	6 min
instant	instant	instant

Lesson 1: good algorithms are better than supercomputers.
Lesson 2: great algorithms are better than good ones.