Lecture 3: Efficient Sorts

Mergesort
Quicksort
Analysis of Algorithms

Mergesort and Quicksort

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.
- Quicksort honored as one of top 10 algorithms for science and engineering of 20^{th} century.

Mergesort.
- Java `Arrays` sort for type `Object`.
- Java `Collections` sort.
- Perl stable, Python stable.

Quicksort.
- Java `Arrays` sort for primitive types.
- C `qsort`, Unix, g++, Visual C++, Perl, Python.

Sorting Applications

Applications.
- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.

⇐ obvious applications

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

⇐ problems become easy once items are in sorted order

- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

. . .

⇐ non-obvious applications

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.

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

Frequency of sorting ops.
- N = # elements to sort.
- Selection sort: $A = 1$, $B = N-1$, $C = N(N-1) / 2$.

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

Big Oh Notation

Big Theta, Oh, and Omega notation.

- $\Theta(N^2)$ means { N^2, $17N^2$, $N^2 + 17N^{1.5} + 3N$, . . . }
  - 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, . . . }
  - $\Theta(N^2)$ and smaller
  - use for upper bounds
- $\Omega(N^2)$ means { N^2, $17N^2$, $N^2 + 17N^{1.5} + 3N$, N^3, $100N^5$, . . . }
  - $\Theta(N^2)$ and larger
  - use for lower bounds

Never say: insertion sort makes at least $O(N^2)$ comparisons.

Estimating the Running Time

Insertion sort is quadratic.

- $N^2 / 4 - N / 4$ comparisons on average.
- $\Theta(N^2)$.

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 It Matters

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

Reference: *More Programming Pearls* by Jon Bentley

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^{17}	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

Reference: *More Programming Pearls* by Jon Bentley

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 Implementation in Java

```
public static void mergesort(Comparable[] a, int low, int high) {
   Comparable temp[] = new Comparable[a.length];
   for (int i = 0; i < a.length; i++) temp[i] = a[i];
    mergesort(temp, a, low, high);
}

private static void mergesort(Comparable[] from, Comparable[] to,
                              int low, int high) {
   if (high <= low) return;
   int mid = (low + high) / 2;
   mergesort(to, from, low,   mid);
   mergesort(to, from, mid+1, high);

   int p = low, q = mid+1;
   for(int i = low; i <= high; i++) {
      if      (q > high)               to[i] = from[p++];
      else if (p > mid)                to[i] = from[q++];
      else if (less(from[q], from[p])) to[i] = from[q++];
      else                             to[i] = from[p++];
   }
}
```


Mergesort Analysis

Stability? Yes, if underlying merge is stable.

How much memory does array implementation of mergesort require?

- Original input = N.
- Auxiliary array for merging = N.
- Local variables: constant.
- Function call stack: $\log_2 N$.
- Total = 2N + O(log N).

How much memory do other sorting algorithms require?

- N + O(1) for insertion sort, selection sort, bubble sort.
- In-place = N + O(log N).

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.
  - assume N is a power of 2
  - assume merging requires exactly N comparisons

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

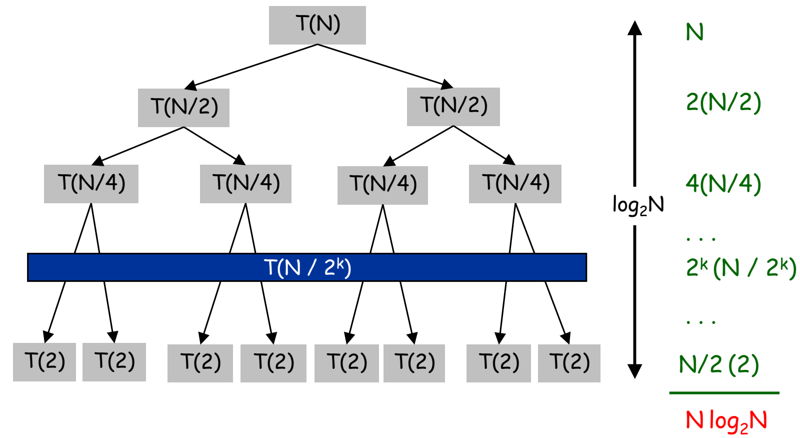
including already sorted

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

- Note: same number of comparisons for ANY file.
- 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. If T(N) satisfies this recurrence, then T(N) = N $\log_2$ N.

assumes 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 ≥ 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 + . . . + N = N(N+1) / 2 for all N ≥ 0.

Proof: (by mathematical induction)

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

Proof by Induction

Claim. If T(N) satisfies this recurrence, then $T(N) = N \log_2 N$.

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

assumes N is a power of 2

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

Q. What if N is not a power of 2?
Q. What if merging takes at most N comparisons instead of exactly N?

A. T(N) satisfies following recurrence.

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

Claim. $T(N) \le N \lceil \log_2 N \rceil$.
Proof. Challenge for the bored.

Mergesort: Practical Improvements

Eliminate recursion. Bottom-up mergesort. Sedgewick Program 8.5

Stop if already sorted.

- Is biggest element in first half ≤ smallest element in second half?
- Helps for nearly ordered lists.

Insertion sort small files.

- Mergesort has too much overhead for tiny files.
- Cutoff to insertion sort for < 7 elements.

Use sentinels.

- Two of four statements in inner loop are bounds checking.
- "Superoptimization requires mindbending recursive switchery."

Sorting By Different Fields

Design challenge: enable sorting students by email or section.

```
// sort by email
Student.setSortKey(Student.EMAIL);
ArraySort.mergesort(students, 0, N-1);

// then by precept
Student.setSortKey(Student.SECTION);
ArraySort.mergesort(students, 0, N-1);
```

```
1 Anand Dharan adharan
1 Ashley Evans amevans
1 Alicia Myers amyers
1 Arthur Shum ashum
1 Amy Trangsrud atrangsr
1 Bryant Chen bryantc
1 Charles Alden calden
1 Cole Deforest cde
1 David Astle dastle
1 Elinor Keith ekeith
1 Kira Hohensee hohensee

. . .
5 Tom Brennan tpbrenna
5 Timothy Ruse truse
5 Yiting Jin ycjin
```

Mergesort is stable

Sorting By Different Fields

```java
public class Student implements Comparable {
   private String first, last, email;
   private int section;

   public final static int FIRST   = 0;
   public final static int LAST    = 1;
   public final static int EMAIL   = 2;
   public final static int SECTION = 3;
   private static int sortKey = SECTION;

   public static void setSortKey(int k) { sortKey = k; }

   public int compareTo(Object x) {
      Student a = this;
      Student b = (Student) x;
      if      (sortKey == FIRST)  return a.first.compareTo(b.first);
      else if (sortKey == LAST)   return a.last.compareTo(b.last);
      else if (sortKey == EMAIL)  return a.email.compareTo(b.email);
      else                        return a.section - b.section;
   }
   ...
}
```

data members (one for each student)

classwide variables (shared by all students)

compare using chosen key

Computational Complexity

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X.
Machine model. Count fundamental operations.

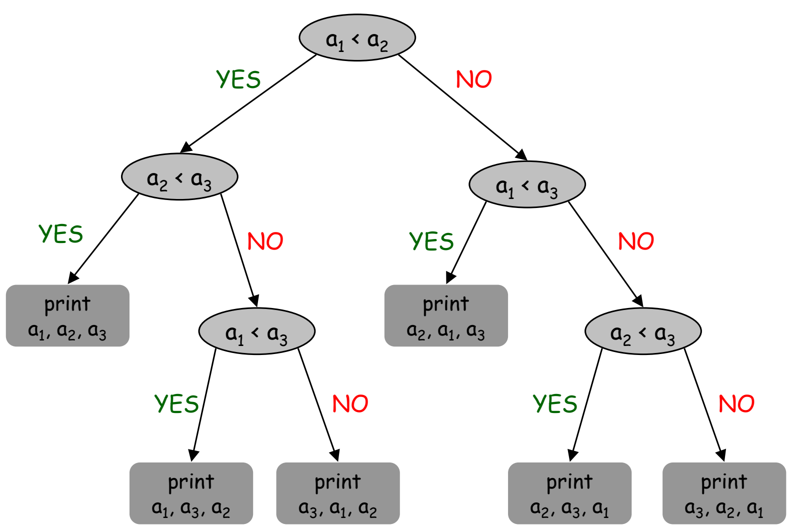
Upper bound. Cost guarantee provided by some algorithm for X.
Lower bound. Proven limit on cost guarantee of any algorithm for X.
Optimal algorithm. Algorithm with best cost guarantee for X.

lower bound ~ upper bound

Example: sorting.
- Machine model = # comparisons on random access machine.
- Upper bound = $N \log_2 N$ from mergesort.
- Lower bound = $N \log_2 N - N \log_2 e$
- Optimal algorithm = mergesort.

applies to any comparison-based algorithm (see COS 226)

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

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)

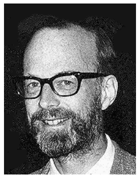
thousand	million	billion
instant	1 sec	18 min
instant	instant	instant

Lesson 1: good algorithms are better than supercomputers.

Quicksort

Quicksort.

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



Sir Charles Antony Richard Hoare, 1960

Q	U	I	C	K	S	O	R	T	I	S	C	O	O	L

Quicksort

Quicksort.

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

partitioning element

Q	U	I	C	K	S	O	R	T	I	S	C	O	O	L

I	C	K	I	C	L	Q	U	S	O	R	T	S	O	O

$\leq$ L

$\geq$ L

partitioned array

Quicksort

Quicksort.

- Partition array so that:
  - some pivot 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.

partitioning element

Q	U	I	C	K	S	O	R	T	I	S	C	O	O	L

C	C	I	I	K	L	O	O	O	Q	R	S	S	T	U

sort each piece

Quicksort: Java Implementation

Quicksort.

- Partition array so that:
  - some pivot 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.

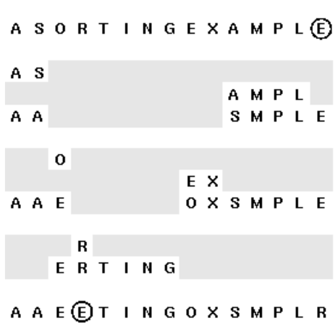
```
public static void quicksort(Comparable[] a, int L, int R) {
   if (R <= L) return;
   int m = partition(a, L, R);
   quicksort(a, L, m-1);
   quicksort(a, m+1, R);
}
```


Quicksort : Implementing Partition

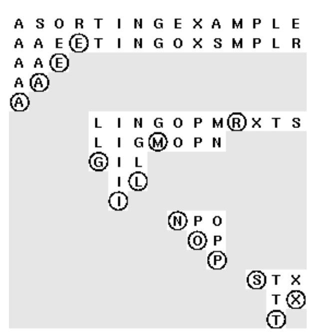
How do we partition in-place efficiently?

```
static int partition(Comparable[] a, int L, int R) {
   int i = L - 1;
   int j = R;

   while(true) {
      while (less(a[++i], a[R]))     ← find item on left to swap
         ;

      while (less(a[R], a[--j]))     ← find item on right to swap
         if (j == L) break;

      if (i >= j) break;             ← check if pointers cross
      exch(a, i, j);                 ← swap
   }

   exch(a, i, R);                    ← swap with partitioning element
   return i;                         ← return index where crossing occurs
}
```


Quicksort Example



Partitioning



Quicksort

Quicksort: Worst Case

Number of comparisons in worst case is quadratic.

- N + (N-1) + (N-2) + . . . + 1 = N(N+1)/2

Worst-case inputs.

- Already sorted!
- Reverse sorted.

What about all equal keys or only two distinct keys?

- Many textbook implementations go quadratic.
- Sedgewick partitioning algorithm stops on equal keys.
- Stay tuned for 3-way quicksort.

Quicksort: Average Case

Average case running time.

- Roughly 2 N ln N comparisons. ← proof on next slide
- Assumption: file is randomly shuffled.
- Equivalent assumption: pivot on random element.

Remarks.

- 39% more comparisons than mergesort.
- Faster than mergesort in practice because of lower cost of other high-frequency instructions.
- Worst case still proportional to N^2 but more likely that you are struck by lightning and meteor at same time.
- Caveat: many textbook implementations have best case N^2 if duplicates, even if randomized!

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 $N \geq 2$:

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

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

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

- Simplify to:

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

Quicksort: Average Case

- Divide both sides by N(N+1) to get 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:

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

- Finally, what we want: $C_N \approx 2(N+1)\ln N \approx 1.39\,N\log_2 N.$

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.

Quicksort: Practical Improvements

Median of sample.
- Best choice of pivot element = median.
- But how would you compute the median?
- Estimate true median by taking median of sample.

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

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

Non-recursive version.
- Use explicit stack.
- Always sort smaller half first.

guarantees O(log N) stack size

Engineering a System Sort

Samplesort.
- Basic algorithm = quicksort.
- Sort a relatively large random sample from the array.
- Use sorted elements as pivots.
- Pivots are (probabilistically) good estimates of true medians.

Bentley-McIlroy.
- Original motivation: improve `qsort` function in *C*.
- Basic algorithm = quicksort.
- Partition on Tukey's *ninther*: Approximate median-of-9.
  - used median-of-3 elements, each of which is median-of-3
  - idea borrowed from statistics, useful in many disciplines
- 3-way quicksort to deal with equal keys.

stay tuned

Reference: *Engineering a Sort Function* by Jon L. Bentley and M. Douglas McIlroy.

System Sorts

Java's `Arrays.sort` library function for arrays.
- Uses Bentley-McIlroy quicksort implementation for objects.
- Uses mergesort for primitive types.

```
Arrays.sort(students, 0, N);
```

starting index is inclusive,
ending index is exclusive

http://java.sun.com/j2se/1.4.2/docs/api/

- To access library, need following line at beginning of program.

```
import java.util.Arrays;
```

Why the difference for objects and primitive types?

Breaking Java's System Sort

Is it possible to make system sort go quadratic?
- No, for mergesort.
- Yes, for deterministic quicksort.

so, why are most system sorts deterministic?

McIlroy's devious idea.
- Construct malicious input WHILE running system quicksort in response to elements compared.
- If p is partition element, commit to x < p, y < p, but don't commit to any order on x, y until x and y are compared.

Consequences.
- Confirms theoretical possibility.
- Algorithmic complexity attack: you enter linear amount of data; server performs quadratic amount of work.
- Blows function call stack and crashes program.

more disastrous possibilities in *C*

Reference: McIlory. *A Killer Adversary for Quicksort.*

Lots of Sorting Algorithms

Internal sorts.
- Insertion sort, selection sort, bubblesort, shellsort, shaker sort.
- Quicksort, mergesort, heapsort.
- Samplesort, introsort.
- Solitaire sort, red-black sort, splaysort, psort, ...

External sorts. Poly-phase mergesort, cascade-merge, oscillating sort.

Radix sorts.
- Distribution, MSD, LSD.
- 3-way radix quicksort.

Parallel sorts.
- Bitonic sort, Batcher even-odd sort.
- Smooth sort, cube sort, column sort.