Lecture 20: Analysis of Algorithms

Overview

Analysis of algorithms: framework for comparing algorithms and predicting performance.

Scientific method.

- *Observe* some feature of the universe.
- *Hypothesize* a model that is consistent with observation.
- *Predict* events using the hypothesis.
- *Verify* the predictions by making further observations.
- *Validate* the theory by repeating the previous steps until the hypothesis agrees with the observations.

Algorithmic Successes

N-body Simulation.

- Simulate gravitational interactions among N bodies.
- Brute force: N^2 steps.
- Barnes-Hut: N log N steps, enables new research.

Andrew Appel
PU '81

Discrete Fourier transform.

- Break down waveform of N samples into periodic components. Applications: DVD players, JPEG, analysis of astronomical data, medical imaging, nonlinear Schrödinger equation, ...
- Brute force: N^2 steps.
- FFT algorithm: N log N steps, enables new technology.

Freidrich Gauss
1805

Sorting.

- Rearrange N items in ascending order.
- Fundamental information processing abstraction.

Jon von Neumann
IAS 1945

Case Study: Sorting

Sorting problem:

- Given N items, rearrange them in ascending order.
- Applications: statistics, databases, data compression, computational biology, computer graphics, scientific computing, ...

name
Hauser
Hong
Hsu
Hayes
Haskell
Hanley
Hornet

name
Hanley
Haskell
Hauser
Hayes
Hong
Hornet
Hsu

Insertion Sort

Insertion sort.

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

```
public static void insertionSort(double[] a) {
   int N = a.length;
   for (int i = 0; i < N; i++) {
      for (int j = i; j > 0; j--) {
         if (less(a[j], a[j-1]))
            exch(a, j, j-1);
         else break;
      }
   }
}
```


Helper Functions

Sorting helper functions.

- Is real number `x` strictly less than `y`?

```
public static boolean less(double x, double y) {
   return (x < y);
}
```

- Swap real numbers stored in `a[i]` and `a[j]`.

```
public static void exch(double[] a, int i, int j) {
   double swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```


Insertion Sort: Observation

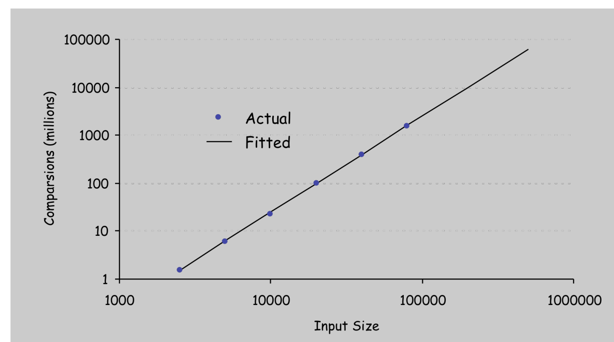
Observe and tabulate running time for various values of N.

- Data source: N random numbers between 0 and 1.
- Machine: Apple G5 1.8GHz with 1.5GB memory running OS X.
- Timing: Skagen wristwatch.

N	*Comparisons*	*Time*
5,000	6.2 million	0.13 seconds
10,000	25 million	0.43 seconds
20,000	99 million	1.5 seconds
40,000	400 million	5.6 seconds
80,000	16 million	23 seconds

Insertion Sort: Experimental Hypothesis

Data analysis. Plot # comparisons vs. input size on log-log scale.



slope

Regression. Fit line through data points $\approx a N^b$.
Hypothesis. # comparisons grows quadratically with input size $\approx N^2/4$.

Insertion Sort: Prediction and Verification

Experimental hypothesis. # comparisons ≈ $N^2/4$.

Prediction. 400 million comparisons for N = 40,000.

Observations.

N	*Comparisons*	*Time*	
40,000	401.3 million	5.595 sec	Agrees.
40,000	399.7 million	5.573 sec	
40,000	401.6 million	5.648 sec	
40,000	400.0 million	5.632 sec	

Prediction. 10 billion comparisons for N = 200,000.

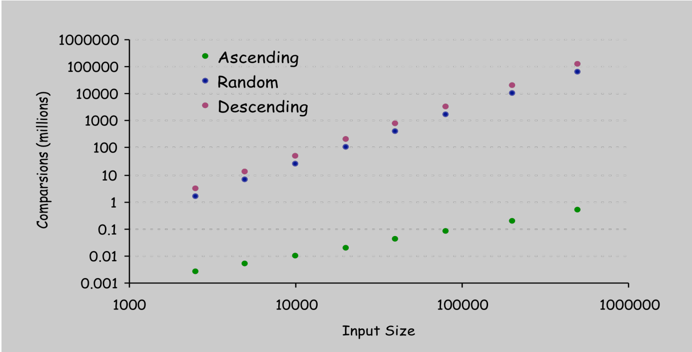
Observation.

N	*Comparisons*	*Time*	
200,000	9.997 billion	145 seconds	Agrees.

Insertion Sort: Validation

Number of comparisons depends on input family.

- Ascending: N.
- Random: $N^2/4$.
- Descending: $N^2/2$.



Insertion Sort: Theoretical Hypothesis

Experimental hypothesis.

- Measure running times, plot, and fit curve.
- Model useful for predicting, but not for explaining.

Theoretical hypothesis.

- Analyze algorithm to estimate # comparisons as a function of:
  - number of elements N to sort
  - average or worst case input
- Model useful for predicting and explaining.
- Model is independent of a particular machine or compiler.

Difference. Theoretical model can apply to machines not yet built.

Insertion Sort: Theoretical Hypothesis

Worst case. (descending)

- Iteration i requires i comparisons.
- Total = 0 + 1 + 2 + . . . + N-2 + N-1 = N (N-1) / 2.

E	F	G	H	I	J	D	C	B	A
						i			

Average case. (random)

- Iteration i requires i/2 comparisons on average.
- Total = 0 + 1/2 + 2/2 + . . . + (N-1)/2 = N (N-1) / 4.

A	C	D	F	H	J	E	B	I	G
						i			

Insertion Sort: Theoretical Hypothesis

Theoretical hypothesis.

Analysis	*Comparisons*	*Stddev*
Worst	$N^2 / 2$	NA
Average	$N^2 / 4$	$1/6\ N^{3/2}$
Best	N	NA

Validation. Theory agrees with observations.

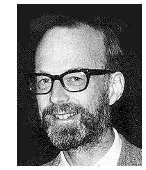
Remark. Supercomputer can't rescue a bad algorithm.

Computer	*Comparisons Per Second*	*Thousand*	*Million*	*Billion*
laptop	10^7	instant	1 day	3 centuries
super	10^{12}	instant	1 second	2 weeks

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`


C. A. R. Hoare, 1960

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

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`

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

≤ L ≥ L

partitioned array

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.

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 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.

```
public static void quicksort(double[] a, int left, int right) {
   if (right <= left) return;
   int i = partition(a, left, right);
   quicksort(a, left, i-1);
   quicksort(a, i+1, right);
}
```


Quicksort : Implementing Partition

Q. How to partition in-place efficiently?


```
public static int partition(double[] a, int left, int right) {
   int i = left - 1;
   int j = right;

   while(true) {
      while (less(a[++i], a[right]))      find item on left to swap
         ;
      while (less(a[right], a[--j]))      find item on right to swap
         if (j == left) break;

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

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


Quicksort: Observation

Observe and tabulate running time for various values of N.

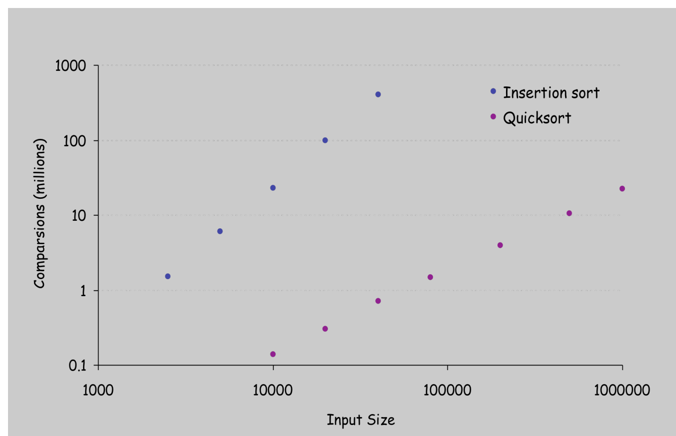
- Data source: first N words of Charles Dicken's life work.
- Machine: Apple G5 1.8GHz with 1.5GB memory running OS X.

N	*Comparisons*	*Time*
200,000	4.5 million	0.10 sec
400,000	9.5 million	0.23 sec
1 million	26 million	0.47 sec
2 million	55 million	0.96 sec
4 million	120 million	2.0 sec
8 million	240 million	4.2 sec

Remark. Takes 1.8 seconds to generate input of size 8 million!

Quicksort: Preliminary Hypothesis

Experimental hypothesis. Number of comparisons ≈ 30N.



Quicksort: Prediction and Verification

Experimental hypothesis. Number of comparisons ≈ 30N.

Prediction. 120 million comparisons for N = 4 million.

Observations.

N	*Comparisons*	*Time*
4 million	112.9 million	2.04 sec
4 million	116.7 million	2.07 sec
4 million	116.8 million	2.02 sec

Agrees.

Prediction. 600 million comparisons for N = 20 million.

Observations.

N	*Comparisons*	*Time*
20 million	638 million	11.1 sec
100 million	3.6 billion	60.6 sec

Not quite.

Quicksort: Theoretical Hypothesis

Average case. (random)

- Number of comparisons ≈ 2 N ln N (stddev ≈ 0.65N).
- Number of exchanges ≈ 1/3 N ln N.

Worst case. Number of comparisons ≈ 1/2 N^2.

Validation.

- Random shuffle before sorting to eliminate worst case.
- Alternate: partition on random element.
- Theory now agrees with observations.

Lesson. Great algorithms can be more powerful than supercomputers.

Computer	*Comparisons Per Second*	*Insertion*	*Quicksort*
laptop	10^7	3 centuries	3 hours
super	10^{12}	2 weeks	instant

N = 1 billion

Order of Growth

Asymptotic running time.

- Estimate time as a function of input size N.
- Ignore lower order terms and leading coefficients.
  - when N is large, terms are negligible
  - when N is small, we don't care
- Ex: $6N^3 + 17N^2 + 56$ is asymptotically proportional to N^3.

Donald Knuth

Complexity	*Description*	*When N doubles, running time*
1	*Constant* algorithm is independent of input size.	does not change
log N	*Logarithmic* algorithm gets slightly slower as N grows.	increases by a constant
N	*Linear* algorithm is optimal if you need to process N inputs.	doubles
N log N	*Linearithmic* algorithm scales to huge problems.	slightly more than doubles
N^2	*Quadratic* algorithm practical for use only on relatively small problems.	quadruples
2^N	*Exponential* algorithm is not usually practical.	squares!

Scientific Method

Scientific method applies to estimate running time.

- Experimental analysis: not difficult to perform experiments.
- Theoretical analysis: may require advanced mathematics.
- Small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

N

```
for (int i = 0; i < N; i++)
   ...
```

N^2

```
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      ...
```

$\log_2 N$

```
while (N > 1) {
   N = N / 2;
   ...
}
```

2^N

```
public static void f(int N) {
   if (N == 0) return;
   f(N-1);
   f(N-1);
   ...
}
```

N $\log_2 N$

```
public static void g(int N) {
   if (N == 0) return;
   g(N/2);
   g(N/2);
   for (int i = 0; i < N; i++)
      ...
}
```

Computational Complexity of Problems

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X.

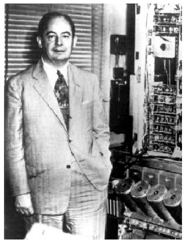
Upper bound. Cost guarantee provided by some algorithm for X.
Lower bound. Proven limit on cost guarantee of any algorithm for X.
Optimal algorithm. Algorithm with best cost guarantee for X.

↑ lower bound ~ upper bound

Example 1: X = sorting.
- Measure costs in terms of comparisons.
- Upper bound = $N \log_2 N$ with mergesort. ← quicksort usually faster, but mergesort never slow
- Lower bound = $N \log_2 N - N \log_2 e$. ← applies to any comparison-based algorithm (see COS 226)
- Optimal algorithm = mergesort.

Sorting Case Study: mergesort

Mergesort.
- 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

Sorting Case Study: mergesort

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

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

Sorting Case Study: mergesort

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

Q. How to merge efficiently?

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	T	S	sort

A	G	H	I	L	M	O	R	S	T	merge

Computational Complexity of Problems

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X.

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.

Example 2: X = Euclidean TSP.

- Measure cost in terms of arithmetic operations.
- Upper bound = 2^N by dynamic programming. ⇐ N! by brute force
- Lower bound = N.
- Optimal algorithm = ask again in 50 years.

Essence of computational complexity: closing the gap.

Summary

How can I evaluate the performance of my algorithm?

- Computational experiments.
- Theoretical analysis.

What if it's not fast enough?

- Understand why.
- Buy a faster computer.
- Find a better algorithm in a textbook.
- Discover a new algorithm.

Attribute	*Better Machine*	*Better Algorithm*
Cost	$$$ or more.	$ or less.
Applicability	Makes "everything" run faster.	May not apply to some problems.
Improvement	Incremental quantitative improvements.	Dramatic quantitative improvements possible.

Summary

Sobering philosophical thoughts.

- In theory, most problems are undecidable.
- In practice, most remaining problems are intractable.
- Analysis of algorithms helps us improve the ones we use.

Announcements

Your Very Last Exam

- Wed April 27, 7:30 PM, right here
- Closed book, but
- You can bring one *cheatsheet*
  - *both sides* of one (8.5 by 11) sheet, handwritten by you
- P.S. No calculators, laptops, Palm Pilots, talking watches, etc.

Helpful review session

- Tuesday April 26, 7:30 PM, COS 105
- Not a canned presentation
- Driven by your questions