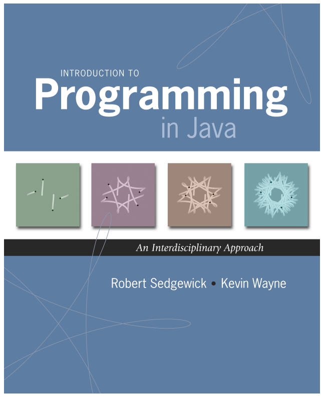
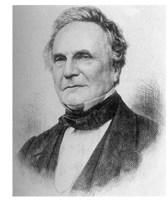
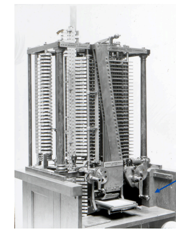
4.1 Performance, 4.2 Sorting and Searching



Running Time

> *"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?"* – *Charles Babbage*


Charles Babbage (1864)


Analytic Engine

how many times do you have to turn the crank?

Scientific Method

Analysis of algorithms. Framework for comparing algorithms and predicting performance.

Scientific method.

- Observe some feature of the natural world.
- Hypothesize a model that is consistent with the observations.
- Predict events using the hypothesis.
- Verify the predictions by making further observations.
- Validate by repeating until the hypothesis and observations agree.

Principles. Experiments we design must be reproducible; hypothesis must be falsifiable.

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, JPEG, MRI, astrophysics, ….
- Brute force: N^2 steps.
- FFT algorithm: N log N steps, enables new technology.


Freidrich Gauss
1805

Sorting.

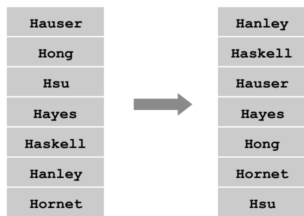
- Rearrange N items in ascending order.
- Fundamental information processing abstraction.


Jon von Neumann
IAS 1945

Case Study: Sorting

Sorting problem. Rearrange N items in ascending order.

Applications. Statistics, databases, data compression, bioinformatics, computer graphics, scientific computing, ...



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.

i	j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
6	6	and	had	him	his	was	you	the	but
6	5	and	had	him	his	was	the	you	but
6	4	and	had	him	his	the	was	you	but
		and	had	him	his	the	was	you	but

Inserting `a[6]` *into position by exchanging with larger entries to its left*

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.

i	j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
		was	had	him	and	you	his	the	but
1	0	had	was	him	and	you	his	the	but
2	1	had	him	was	and	you	his	the	but
3	0	and	had	him	was	you	his	the	but
4	4	and	had	him	was	you	his	the	but
5	3	and	had	him	his	was	you	the	but
6	4	and	had	him	his	the	was	you	but
7	1	and	but	had	him	his	the	was	you
		and	but	had	him	his	the	was	you

Inserting `a[1]` *through* `a[N-1]` *into position (insertion sort)*

Insertion Sort: Java Implementation

```
public class Insertion {

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

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

Insertion Sort: Observation

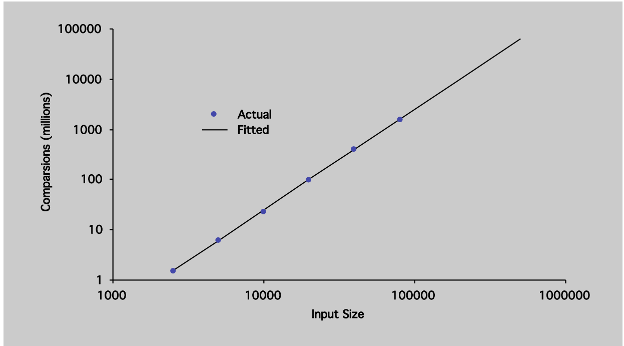
Observe and tabulate running time for various values of N.

- 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	1600 million	23 seconds

Insertion Sort: Empirical Analysis

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



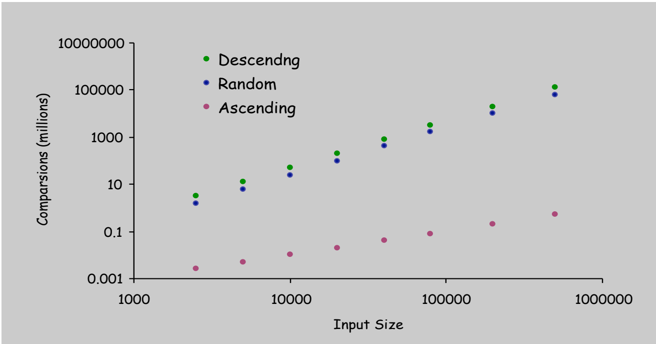
slope

Hypothesis. # comparisons grows quadratically with input size $\sim N^2 / 4$.

Insertion Sort: Empirical Analysis

Observation. Number of comparisons depends on input family.

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



Analysis: Empirical vs. Mathematical

Empirical analysis.

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

Mathematical analysis.

- Analyze algorithm to estimate # ops as a function of input size.
- May require advanced mathematics.
- Model useful for predicting and explaining.

Critical difference. Mathematical analysis is independent of a particular machine or compiler; applies to machines not yet built.

Insertion Sort: Mathematical Analysis

Worst case. [descending]

- Iteration i requires i comparisons.
- Total = $(0 + 1 + 2 + ... + N\text{-}1) \sim N^2 / 2$ compares.

E	F	G	H	I	J	D	C	B	A
						i			

Average case. [random]

- Iteration i requires $i / 2$ comparisons on average.
- Total = $(0 + 1 + 2 + ... + N\text{-}1) / 2 \sim N^2 / 4$ compares

A	C	D	F	H	J	E	B	I	G
						i			

Insertion Sort: Lesson

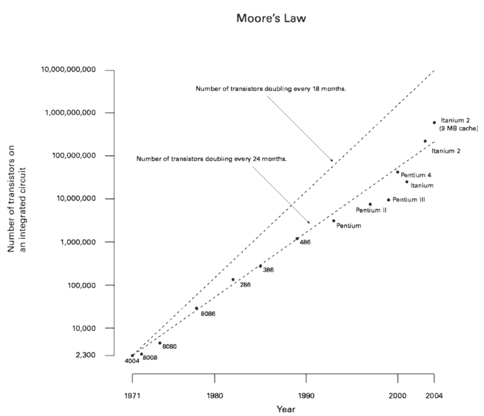
Lesson. Supercomputer can't rescue a bad algorithm.

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

Moore's Law

Moore's law. Transistor density on a chip doubles every 2 years.

Variants. Memory, disk space, bandwidth, computing power per $.



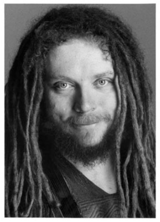
http://en.wikipedia.org/wiki/Moore's_law

Moore's Law and Algorithms

Quadratic algorithms do not scale with technology.

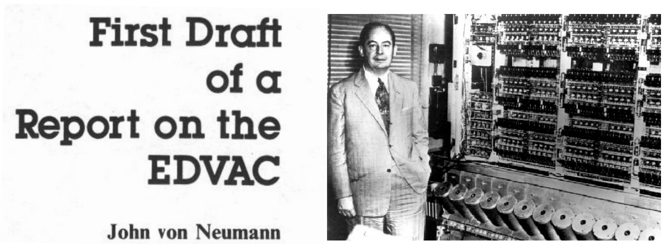
- New computer may be 10x as fast.
- But, has 10x as much memory so problem may be 10x bigger.
- With quadratic algorithm, takes 10x as long!

> *"Software inefficiency can always outpace Moore's Law. Moore's Law isn't a match for our bad coding."*
>
> *– Jaron Lanier*



Lesson. Need linear (or linearithmic) algorithm to keep pace with Moore's law.

Mergesort



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

input
was had him and you his the but

sort left
and had him was you his the but

sort right
and had him was but his the you

merge
and but had him his the was you

Mergesort: Example

M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

i	j	k	aux[k]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
				and	had	him	was	but	his	the	you
0	4	0	and	and	had	him	was	but	his	the	you
1	4	1	but	and	had	him	was	but	his	the	you
1	5	2	had	and	had	him	but	but	his	the	you
2	5	3	him	and	had	him	but	but	his	the	you
3	5	4	his	and	had	him	was	but	his	the	you
3	6	5	the	and	had	him	was	but	his	the	you
3	6	6	was	and	had	him	was	but	his	the	you
4	7	7	you	and	had	him	but	but	his	the	you

Merging

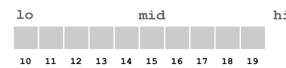
Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

```
String[] aux = new String[N];
// merge into auxiliary array
int i = lo, j = mid;
for (int k = 0; k < N; k++) {
   if      (i == mid)  aux[k] = a[j++];
   else if (j == hi)   aux[k] = a[i++];
   else if (a[j].compareTo(a[i]) < 0)  aux[k] = a[j++];
   else                                aux[k] = a[i++];
}

// copy back
for (int k = 0; k < N; k++) {
   a[lo + k] = aux[k];
}
```

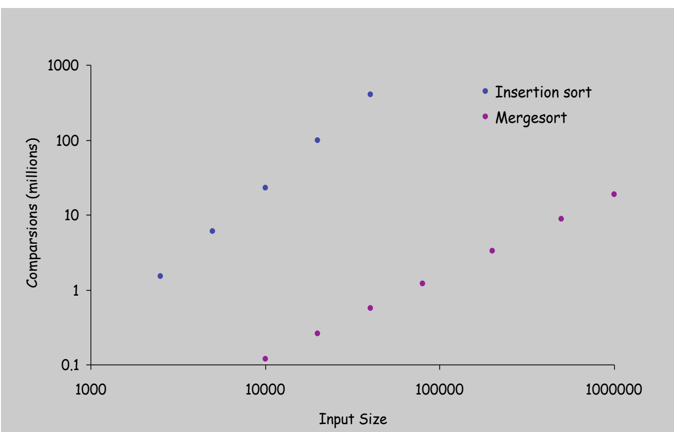

Mergesort: Java Implementation

```
public class Merge {

   public static void sort(String[] a) {
      sort(a, 0, a.length);
   }

   // Sort a[lo, hi).
   public static void sort(String[] a, int lo, int hi) {
      int N = hi - lo;
      if (N <= 1) return;

      // recursively sort left and right halves
      int mid = lo + N/2;
      sort(a, lo, mid);
      sort(a, mid, hi);

      // merge (see previous slide)
   }

}
```



Mergesort: Empirical Analysis

Experimental hypothesis. Number of comparisons ≈ 20N.



Mergesort: Prediction and Verification

Experimental hypothesis. Number of comparisons ≈ 20N.

Prediction. 80 million comparisons for N = 4 million.

Observations.

N	Comparisons	Time
4 million	82.7 million	3.13 sec
4 million	82.7 million	3.25 sec
4 million	82.7 million	3.22 sec

Agrees.

Prediction. 400 million comparisons for N = 20 million.

Observations.

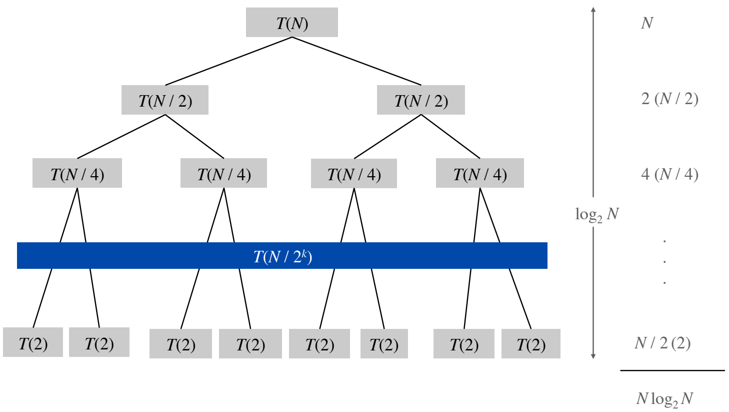
N	Comparisons	Time
20 million	460 million	17.5 sec
50 million	1216 million	45.9 sec

Not quite.

Mergesort: Mathematical Analysis

Analysis. To mergesort array of size N, mergesort two subarrays of size $N / 2$, and merge them together using $\leq N$ comparisons.

we assume N is a power of 2



Mergesort: Mathematical Analysis

Mathematical analysis.

analysis	comparisons
worst	$N \log_2 N$
average	$N \log_2 N$
best	$1/2\ N \log_2 N$

Validation. Theory agrees with observations.

N	actual	predicted
10,000	120 thousand	133 thousand
20 million	460 million	485 million
50 million	1,216 million	1,279 million

Mergesort: Lesson

Lesson. Great algorithms can be more powerful than supercomputers.

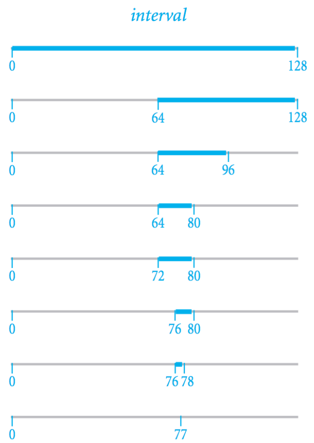
Computer	Comparisons Per Second	Insertion	Mergesort
laptop	10^7	3 centuries	3 hours
super	10^{12}	2 weeks	instant

N = 1 billion

Binary Search

Twenty Questions

Intuition. Find a hidden integer.



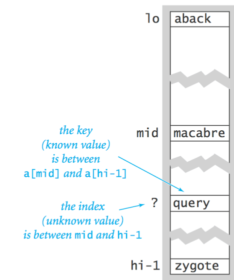
size	Q	A
128	< 64 ?	*false*
64	< 96 ?	*true*
32	< 80 ?	*true*
16	< 72 ?	*false*
8	< 76 ?	*false*
4	< 78 ?	*true*
2	< 77 ?	*false*
1	= 77	

Searching a Sorted Array

Searching a sorted array. Given a sorted array, determine the index associated with a given key.

Ex. Dictionary, phone book, book index, credit card numbers, …

Binary search.
- Examine the middle key.
- If it matches, return its index.
- Otherwise, search either the left or right half.



Binary search in an array (one step)

Binary Search: Java Implementation

Invariant. Algorithm maintains `a[lo] ≤ key ≤ a[hi-1]`.

```
public static int search(String key, String[] a) {
   return search(key, a, 0, a.length);
}

public static int search(String key, String[] a, int lo, int hi) {
   if (hi <= lo) return -1;
   int mid = lo + (hi - lo) / 2;
   int cmp = a[mid].compareTo(key);
   if      (cmp > 0) return search(key, a, lo, mid);
   else if (cmp < 0) return search(key, a, mid+1, hi);
   else              return mid;
}
```

Java library implementation: `Arrays.binarySearch()`

Binary Search: Mathematical Analysis

Analysis. To binary search in an array of size N: do one comparison, then binary search in an array of size $N / 2$.

$$N \to N/2 \to N/4 \to N/8 \to \dots \to 1$$

Q. How many times can you divide a number by 2 until you reach 1?
A. $\log_2 N$.

$$1$$
$$2 \to 1$$
$$4 \to 2 \to 1$$
$$8 \to 4 \to 2 \to 1$$
$$16 \to 8 \to 4 \to 2 \to 1$$
$$32 \to 16 \to 8 \to 4 \to 2 \to 1$$
$$64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$$
$$128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$$
$$256 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$$
$$512 \to 256 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$$
$$1024 \to 512 \to 256 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$$

Order of Growth Classifications

Observation. A small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

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

$\lg N$

$\lg = \log_2$

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

N

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

N^2

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

$N \lg N$

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

2^N

Summary

Q. How can I evaluate the performance of my program?
A. Computational experiments, mathematical analysis.

Q. What if it's not fast enough? Not enough memory?
- 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.	Does not apply to some problems.
Improvement	Quantitative improvements.	Dramatic qualitative improvements possible.