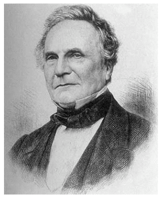
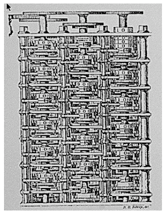
Analysis of Algorithms

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 (schematic)

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.

Universe = computer itself.

Case Study: Sorting

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

Hauser		Hanley
Hong		Haskell
Hsu		Hauser
Hayes	→	Hayes
Haskell		Hong
Hanley		Hornet
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;
      }
   }
}
```

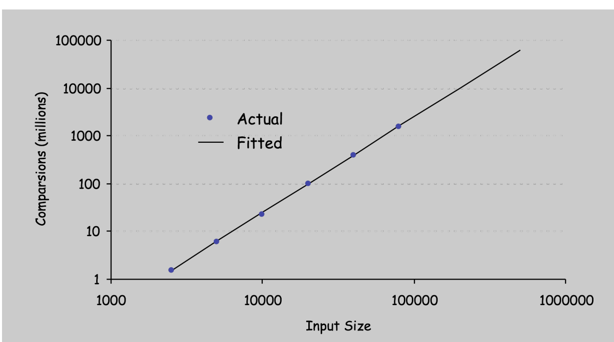

Insertion Sort: Observation

Observe and tabulate running time for various values of N.
- Data source: N random numbers between 0 and 1.

N	Comparisons
5,000	6.2 million
10,000	25 million
20,000	99 million
40,000	400 million
80,000	16 million

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 $\approx N^2/4$.

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

Observations.

N	Comparisons
40,000	401.3 million
40,000	399.7 million
40,000	401.6 million
40,000	400.0 million

Agrees.

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

Observation.

N	Comparisons
200,000	9.997 billion

Agrees.

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.

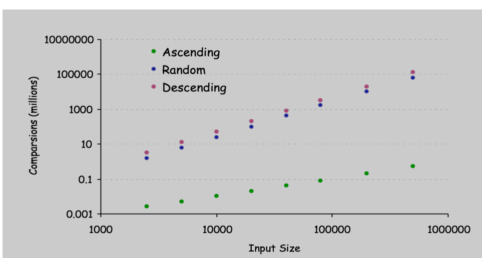
A	C	D	F	H	J	E	B	I	G
						i			

Insertion Sort: Theoretical Hypothesis

Theoretical hypothesis.

Analysis	Input	Comparisons	Stddev
Worst	Descending	$N^2 / 2$	-
Average	Random	$N^2 / 4$	$1/6\ N^{3/2}$
Best	Ascending	N	-

Validation. Theory agrees with observations.



Insertion Sort: Observation

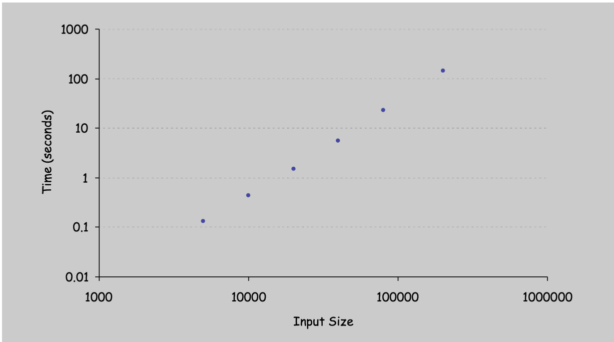
Observe and tabulate running time for various values of N.

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

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
200,000	10 billion	145 seconds

Insertion Sort: Experimental Hypothesis

Data analysis. Plot time vs. input size on log-log scale.



Regression. Fit line through data points $\approx a N^b$.

Hypothesis. Running time grows quadratically with input size.

Timing in Java

Wall clock. Measure time between beginning and end of computation.

- Manual: Skagen wristwatch.
- Automatic: `Stopwatch.java` library.

```
Stopwatch.tic();
. . .
double elapsed = StopWatch.toc();
```

```
public class Stopwatch {
   private static long start;
   public static void tic() {
      start = System.currentTimeMillis();
   }
   public static double toc() {
      long stop = System.currentTimeMillis();
      return (stop - start) / 1000.0;
   }
}
```


Measuring Running Time

Factors that affect running time.

- Machine.
- Compiler.
- Algorithm.
- Input data.

More factors.

- Caching.
- Garbage collection.
- Just-in-time compilation.
- CPU used by other processes.

Bottom line. Often hard to get precise measurements.

Summary

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.

Remaining question. How to formulate a hypothesis?

How To Formulate a Hypothesis

Types of Hypotheses

Worst case running time. Obtain bound on running time of algorithm on any input of a given size N.

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Average case running time. Obtain bound on running time of algorithm on random input as a function of input size N.

- Hard to accurately model real instances by random distributions.
- May perform poorly on other distributions.

Amortized running time. Worst-case bound on running time of any sequence of N operations.

Estimating the Running Time

Total running time: sum of cost × frequency for all of the basic ops.

- Cost depends on machine, compiler.
- Frequency depends on algorithm, input.

Cost for sorting.

- A = # exchanges.
- B = # comparisons.
- Cost on a typical machine = 11A + 4B.

Frequency of sorting ops.

- N = # elements to sort.
- Selection sort: A = N-1, B = N(N-1)/2.

Donald Knuth
1974 Turing Award

Asymptotic 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.

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

Constant Time

Linear time. Running time is $O(1)$.

Elementary operations.

- Function call.
- Boolean operation.
- Arithmetic operation.
- Assignment statement.
- Access array element by index.

Logarithmic Time

Logarithmic time. Running time is O(log N).

Searching in a sorted list. Given a sorted array of items, find index of query item.

O(log N) solution. Binary search.

```
public static int binarySearch(String[] a, String key) {
   int left  = 0;
   int right = a.length - 1;
   while (left <= right) {
      int mid = left + (right - left) / 2;
      int cmp = key.compareTo(a[mid]);
      if      (cmp < 0) right = mid - 1;
      else if (cmp > 0) left  = mid + 1;
      else return mid;
   }
   return -1;
}
```


Linear Time

Linear time. Running time is O(N).

Find the maximum. Find the maximum value of N items in an array.

```
double max = Double.NEGATIVE_INFINITY;
for (int i = 0; i < N; i++) {
   if (a[i] > max)
      max = a[i];
}
```


Linearithmic Time

Linearithmic time. Running time is O(N log N).

Sorting. Given an array of N elements, rearrange in ascending order.

O(N log N) solution. Mergesort. [stay tuned]

Remark. Ω(N log N) comparisons required. [stay tuned]

Quadratic Time

Quadratic time. Running time is O(N^2).

Closest pair of points. Given N points in the plane, find closest pair.

O(N^2) solution. Enumerate all pairs of points.

```
double min = Double.POSITIVE_INFINITY;
for (int i = 0; i < N; i++){
   for (int j = i+1; j < N; j++) {
      double dx = (x[i] - x[j]);
      double dy = (y[i] - y[j]);
      if (dx*dx + dy*dy < min)
         min = dx*dx + dy*dy;
   }
}
```

Remark. Ω(N^2) seems inevitable, but this is just an illusion.

Exponential Time

Exponential time. Running time is $O(a^N)$ for some constant $a > 1$.

Finbonacci sequence: 1 1 2 3 5 8 13 21 34 55 …

$O(\phi^N)$ solution. Spectacularly inefficient!

$\phi = \frac{1}{2}\left(1+\sqrt{5}\right) = 1.618034....$

```
public static int F(int N) {
   if (n == 0 || n == 1) return n;
   else                  return F(n-1) + F(n-2);
}
```

Efficient solution. $F(N) = \left[\frac{\phi^N}{\sqrt{5}}\right]$.

nearest integer function

Summary of Common Hypotheses

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!