Analysis of Algorithms

Main factor considered is running time.

““As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise — by what course of calculation can these results be arrived at by the machine in the shortest time?” Charles Babbage

Reasons for analysis:

**AVOID PERFORMANCE BUGS**

Our focus is on the below:

1. Predict performance
2. Compare algorithms
3. Provide guarantees

Fast algorithms enable technological innovation

We use the scientific method to analyze algorithms:

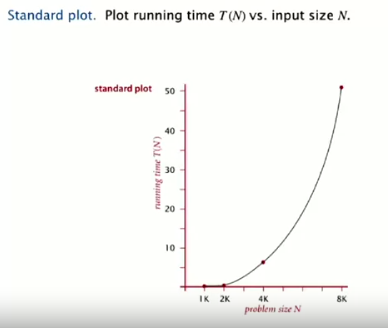
1. Observe
2. Hypothesize
3. Predict
4. Verify
5. Validate

Important principles:

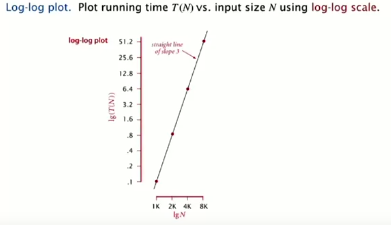
* Experiments MUST be reproducible
* Hypothesis must be falsifiable

Observations

**Empirical analysis** is running the program for different inputs, doubling the time and extrapolating run time from the results. **Data analysis** is plotting running time on the x axis and the input size on the y axis



Can also use a log-log plot; the slope of the line is key (it will be ‘b’). Run a regression (straight line through data points), then find a slope. The function (power law) is a Nb (where b is the slope of the line).



We plot running time T(N) vs input size N with the log-log scale. Since this is computer science the log-log plot is done with powers of two.

Here lg(T(N)) = b lg N + c

B = 2.999

C = -33.2103

T(N) = a Nb, where a = 2c

---

1) Regression formula:

Y = b\*X + c

2) Same for log-log plot:

lg(T(N)) = b\*lg(N) + c

NOTE: here the T(N) its just a runtime value

3) Start from finding 'ratio' from 2 runtimes:

ratio = T2 /T1

4) Find 'b' from ratio (If N doubles from T1 to T2 use log base 2):

b = lg(ratio)

5) Find 'a':

T(N) = a \* N^b

a = T(N) / N^b

6) Create run time formula, e.g. for a = 3\*10^-9, b = 4:

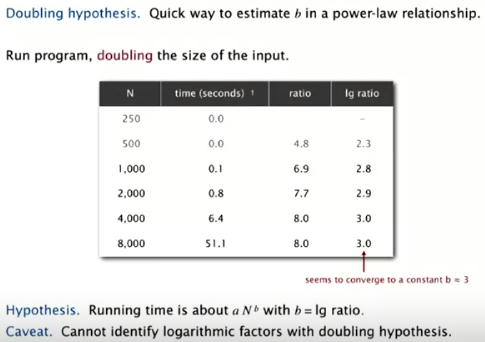
Logarithm complexity = 3\*10^-9 \* N^4;

---

Doubling hypothesis

Double the size of the input each time and take the ratio of the running times for N and 2N. That ratio will eventually converge to a constant. The log of the ratio will converge to a constant, which is b (the exponent of N in the running time).

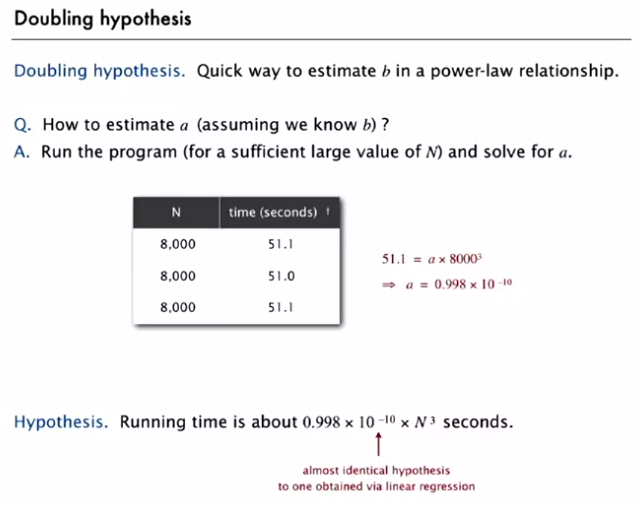
A Nb is the running time where b = lg ratio.



Caveat is that you cannot identify logarithmic factors with the doubling hypothesis, but it is a much faster way to calculate running times.

How to estimate for a?

Run the program and solve for a.



System independent effects are:

* Algorithm
* And input data

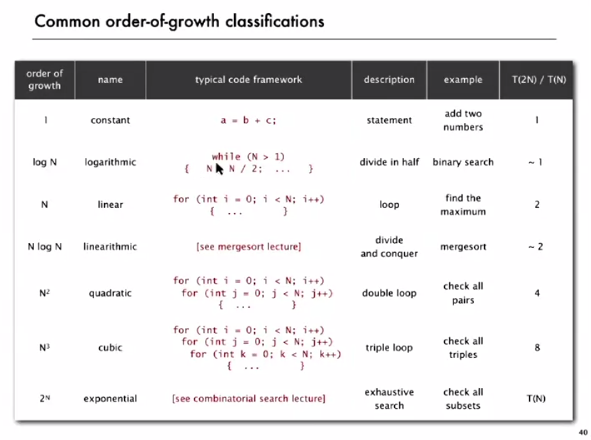
These determine exponent b in power law.

System dependent effects are:

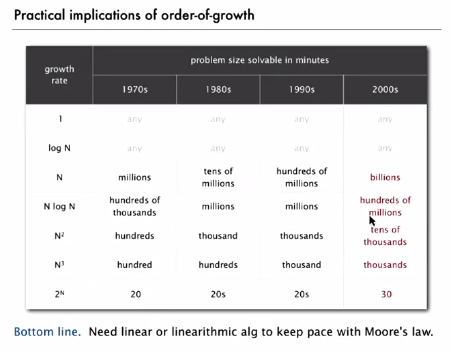
* Hardware (e.g. CPU)
* Software (e.g. compiler)
* System (e.g. OS, network)

Everything above determines constant a in power law.

Order of growth



* Constant time = no loops
* Logarithmic = loops that divide in half
* Linear = loops (none nested)
* Linearithmic = divide and conquer algorithms
* Quadratic = double nested for loop
* Cubic = triple nested for loop



Theory of algorithms

Big Theta is for classification of the asymptotic order of growth

Big Oh is to define (develop) the upper bound longest running time:

Big Omega is to define (develop) the lower bound shortest running time

Algorithm Design approach

* Start
  + Develop an algorithm
  + Prove the lower bound
* Gap?
  + Lower the upper bound (with a new algorithm)
  + Raise the lower bound (more difficult)
* Caveats
  + May be too pessimistic to focus on worst case (not all data is worst case)
  + Need better than “to within a constant factor” to predict performance