Analysis of Algorithms

Frank Walsh/Eamonn Deleaster

Agenda

- Introduction
    - Why analyse algorithms
- Observations
- Mathematical Models
- Growth Classification for algorithms
- Theory of Algorithms

Why analyse algorithms

- Programmers need to develop working solutions to problem
- Algorithm analysis helps developers to write programs that:
  - provide an optimal working solution
  - predict resources and time necessary to execute a program
  - give guarantees regarding performance.
- Helps to avoid performance problems
  - Clients get poor performance because programmer didn't understand or investigate performance characteristics of program.

Scientific Method

Approach that scientists use to understand the natural world

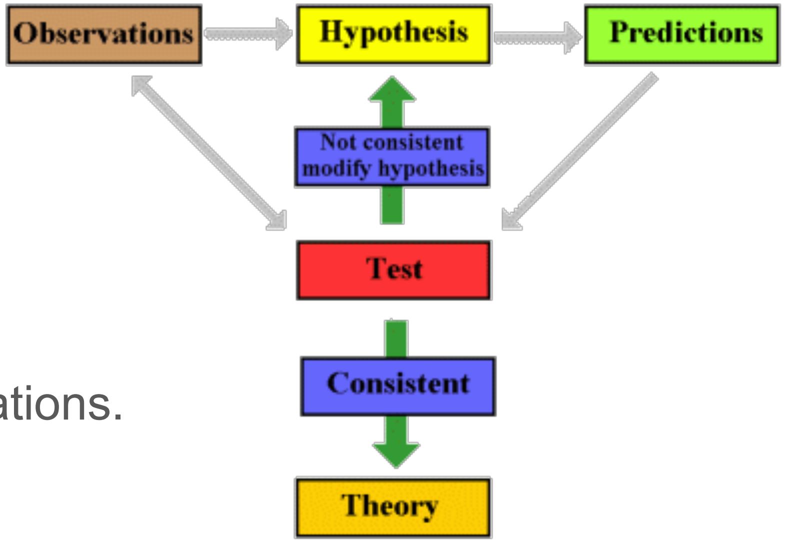
■ Observe some feature of the natural world, generally with precise measurements.

■ Hypothesise a model that is consistent with the observations.

■ Predict events using the hypothesis.

■ Verify the predictions by making further observations.

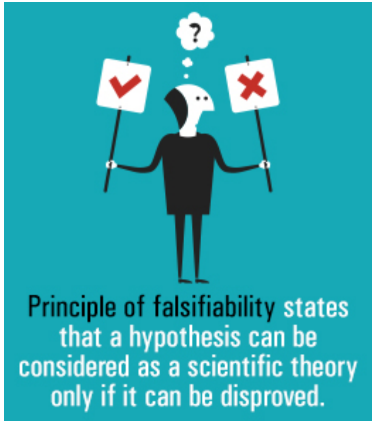
■ Validate by repeating until the hypothesis and observations agree.



Scientific Method

Key features of the scientific method:

- Experiments must be reproducible
  - so that you can convince others.
- Your hypothesis must be falsifiable
  - “No amount of experimentation can ever prove me right; a single experiment can prove me wrong”
- Are these scientific hypothesis?:
  - “There is life on other planets”
  - “Two objects will hit the ground at the same time when dropped from the same height(excluding air resisitance)”



Observations

- We can make quantitative measurements of the running time of our programs.
  - Easy compared to other sciences (don't need to build a hadron collider)
- Answers a core question: How long will my program take?
- Initial observation, the problem size:
  - The problem size can be the size of input or value of input)
  - Most of the time, programming running time is insensitive to the input itself, but IS SENSITIVE to the size of the input.

Observations: Example

ThreeSum: Given N distinct integers, how many triples sum to exactly zero:

```
public class ThreeSum
{
public static int count(int[] a)
{
 int N = a.length;
 int count = 0;
 for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
     for (int k = j+1; k < N; k++)
       if (a[i] + a[j] + a[k] == 0)
          count++;
 return count;
}
public static void main(String[] args)
{
  int[] a = In.readInts(args[0]);
  System.out.println.println(count(a));
}
}
```

Observation: Example

- How do we measure running time
  - Manual (e.g. stopwatch)
  - Use JUnit(look at running times of methods)
  - Automatic (build it into the program). Can use the Stopwatch() class.

```
public static void main(String[] args)
{
int[] a = In.readInts(args[0]);
Stopwatch   stopwatch = new Stopwatch();
System.out.println(ThreeSum.count(a));
double  time = stopwatch.elapsedTime();
System.out.println("elapsed time " + time);
}
```

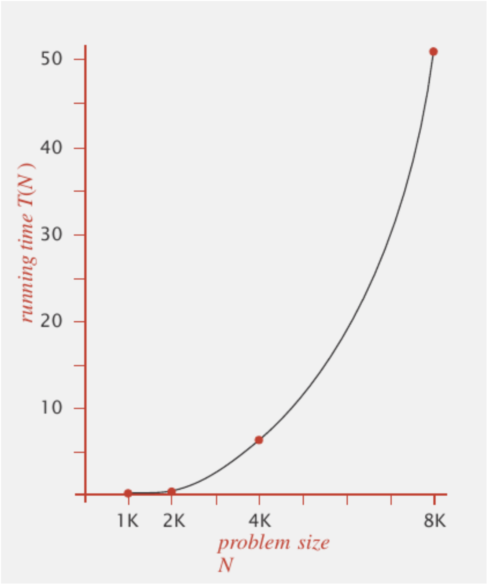
Observation: Empirical Analysis

Running for different size input (N):

N	time (seconds) †
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

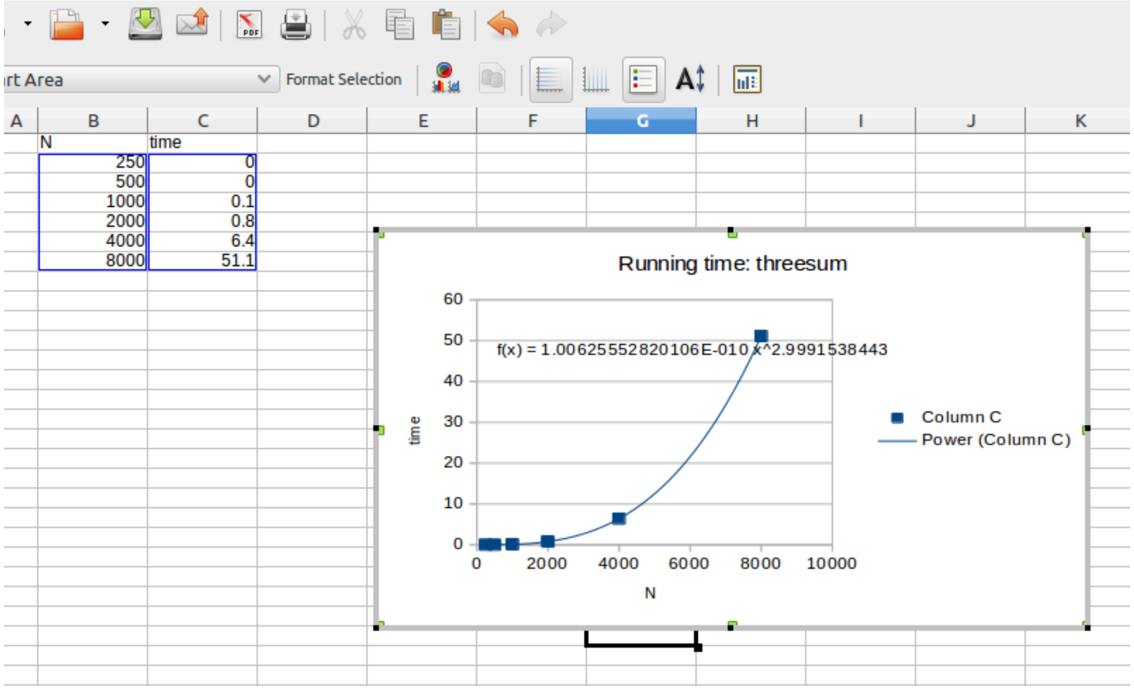
Observation: Data Analysis

- Plot the running time T(N) against input size (N)
- How can we predict values for 16K
  - get an equation for the trendline in the graph
  - Equation can be used to calculate how long will my program take, as a function of the input size.
- One approach:
  - use a tool that can “fit” an equation to the trendline.
  - use the equation to predict other values



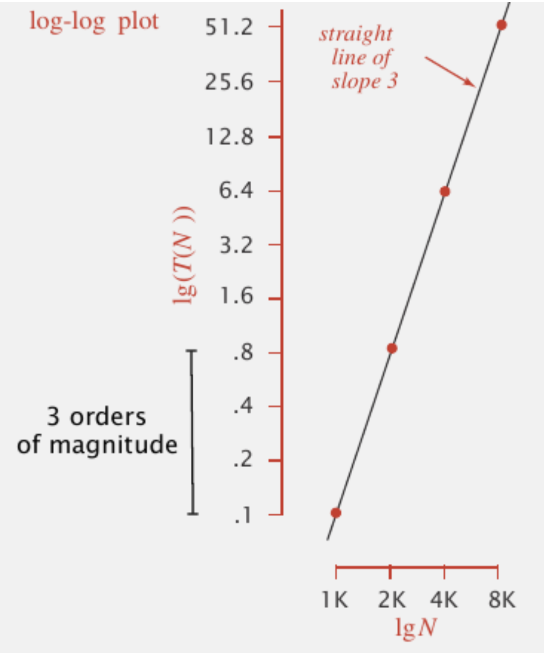
Observation: Data analysis with Spreadsheet

- Chart data as X-Y plot
- Insert Trendline
- More info here: http://www.cpp.edu/~seskandari/documents/Curve_Fitting_William_Lee.pdf
- Use equation for trendline to predict future values:
- Aproximating eqn: $T(N) = 1.006 \times 10^{-10} N^3$



Observation: Data Analysis using logs

- Log-log plot: Plot running time T (N) vs. input size N using log-log scale.
- Get straight line with slope of 3:
  - eqn. of straight line is y=mx + c
  - for this graph: lg(T (N)) = b lg N + c
  - b=2.99, c=-33.2103
- $T(N)=aN^b$, where $a=2^c$ using power law https://en.wikipedia.org/wiki/Power_law
- Now we can make a Hypothesis for running time
  - Running time is approx. $2^{-33.21}N^3$
    $T(N)=1.006 \times 10^{-10} N^3$
- Same as previous slide...



Prediction and Validation

- Hypothesis: Running time is $1.006 \times 10^{-10} N^3$ where N is the size of the input
- Predictions:
  - 51 seconds for N=8000
  - 408.1 seconds for N=16000
- Observations:

N	time (seconds) †
8,000	51.1
8,000	51.0
8,000	51.1
16,000	410.8

Hypothesis validated!

What effects the Running Time

- System independent effects:
  - Algorithm
  - Input Data
- System dependent effects
  - hardware: processor, memory
  - software: compiler, garbage collection etc.
  - System: operating system, network, other apps…

- System independent effects determine the exponent in eqn.
- Both System independent and dependent effects determine the constant
- Difficult to get precise measurement but easier to obtain measurements
  - no animals were harmed in this experiment!
  - Can run large number of experiments.

Mathematical Models for Algorithms

Mathematical Models for Algorithms

- Example: 1-Sum
  - How many instructions are performed in the code:

```
int count = 0;
for (int i = 0; i < N; i++)
      if (a[i] == 0)
            count++;
```

operation	frequency
variable declaration	2
assignment statement	2
less than compare	$N + 1$
equal to compare	N
array access	N
increment	N to $2N$

Mathematical Models for Algorithms

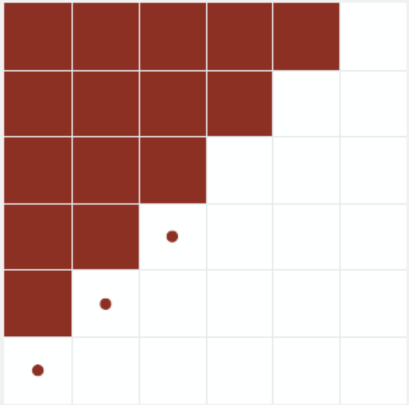
Example: 2-sum

- How many instructions as a function of input size

```
1.  int count = 0;
2.  for (int i = 0; i < N; i++)
3.     for (int j = i+1; j < N; j++)
4.        if (a[i] + a[j] == 0)
5.           count++;
```

- Line 4 is executed (N-1)+(N-2)+(N-3)+...+2+1+0 times

Pf. [n even]



$$0 + 1 + 2 + \ldots + (N-1) = \underbrace{\frac{1}{2}N^2}_{\text{half of square}} - \underbrace{\frac{1}{2}N}_{\text{half of diagonal}}$$

Mathematical Models for Algorithms

Example: 2-sum

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

- NEED TO SIMPLIFY!!!

operation	frequency
variable declaration	$N + 2$
assignment statement	$N + 2$
less than compare	$\frac{1}{2}(N + 1)(N + 2)$
equal to compare	$\frac{1}{2} N (N - 1)$
array access	$N (N - 1)$
increment	$\frac{1}{2} N (N - 1)$ to $N (N - 1)$

tedious to count exactly

Mathematical Cost Models: Simplify

"...we shall therefore only attempt to count the number of multiplications and recordings. " — Alan Turing

- Identify a basic operation
  - usually the operation that executes the most number of times
  - Can ignore other operations
- In 2-sum, the array accesses in the "if" statement is a good choice:

```
int count = 0;
for (int i = 0; i < N; i++)
  for (int j = i+1; j < N; j++)
    if (a[i] + a[j] == 0)
      count++;
```

Mathematical Cost Model: Simplicity

Time efficiency can analysed by determining the number of repetitions of the *basic operation* as a function of *input size.* For big input sizes, N:

$$T(N) \approx c_{op} C(N)$$

Running Time

Execution time of basic operation

Number of times basic operation is executed

Mathematical Cost Model: Simplify

Use “Tilda Notation”

- Estimate Number of Times Basic Operation is executed and use Higher Order term:
- For 2-Sum example:
    - Basic Operation runs N(N-1)

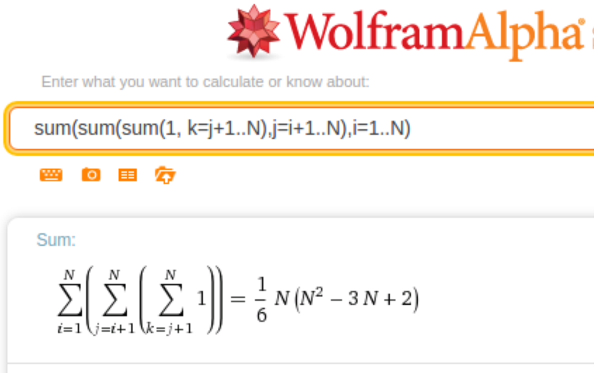
$C(N) = N^2 - N \sim N^2$

operation	frequency
variable declaration	$N + 2$
assignment statement	$N + 2$
less than compare	$\frac{1}{2}(N + 1)(N + 2)$
equal to compare	$\frac{1}{2}N(N - 1)$
array access	$N(N - 1)$
increment	$\frac{1}{2}N(N - 1)$ to $N(N - 1)$

cost model = ar

Mathematical Cost Model

3-Sum Example:




```
1int N = a.length;
2 int count = 0;
3 for (int i = 0; i < N; i++)
4   for (int j = i+1; j < N; j++)
5     for (int k = j+1; k < N; k++)
6       if (a[i] + a[j] + a[k] == 0)
7          count++;
 return count;
```

`Basic Operation (line 6: “touches the array 3 times)`

`Number of times Line 6 executes: N(N-1)(N-2)/6` ~ $N^3/6$ (Can calculate using discrete maths or online tool: http://www.wolframalpha.com/)

Number of times array accessed C(N) ~ $N^3/2$

What does this tell us about how the algorithm running time grows as you increase size?

$T(N) = c_{op}C(N) = c_{op}N^3/3$

Mathematical Cost Model: Summary

Develop a Mathematical model using the following steps

■ Develop an input model, including a definition of the problem size(e.g. size of array)

■ Identify the inner loop.

■ Define a cost model that includes the “basic operation” in the inner loop.

■ Determine the frequency of execution of the basic operation for the given input.

Doing so might require mathematical analysis...