Njala University
School of Technology

Course Title: Data Structures & Algorithm
Course Code: COMP 216

A.J. Fofanah

Basic Requirements: Fundamentals of Programming

COMPS 216 Will attempt to address the following:

- The history of computing / objects / types / console I/O
- Operators / loops / methods / parameter passing
- Selection statements / arrays / strings
- Exceptions / debugging
- File input / file output
- Pointers / unsafe code / linked lists
- Collections / multi-dimensional arrays / search algorithms
- Sorting algorithms
- Object-oriented design / polymorphism / interfaces / inheritance
- Abstract class

Basic Requirements: *Fundamentals of Programming*

These were the basics of programming

- The ability to manipulate the computer to perform the required tasks

You saw data storage techniques:

- Arrays, and
- Linked lists (collections were discussed)

You saw array accessing/manipulation techniques:

- Searching, and
- Sorting

In this course, we will look at:

- *Algorithms* for solving problems efficiently
- *Data structures* for efficiently storing, accessing, and modifying data

We will see that all data structures have trade-offs

- There is no *ultimate* data structure...
- The choice depends on our requirements

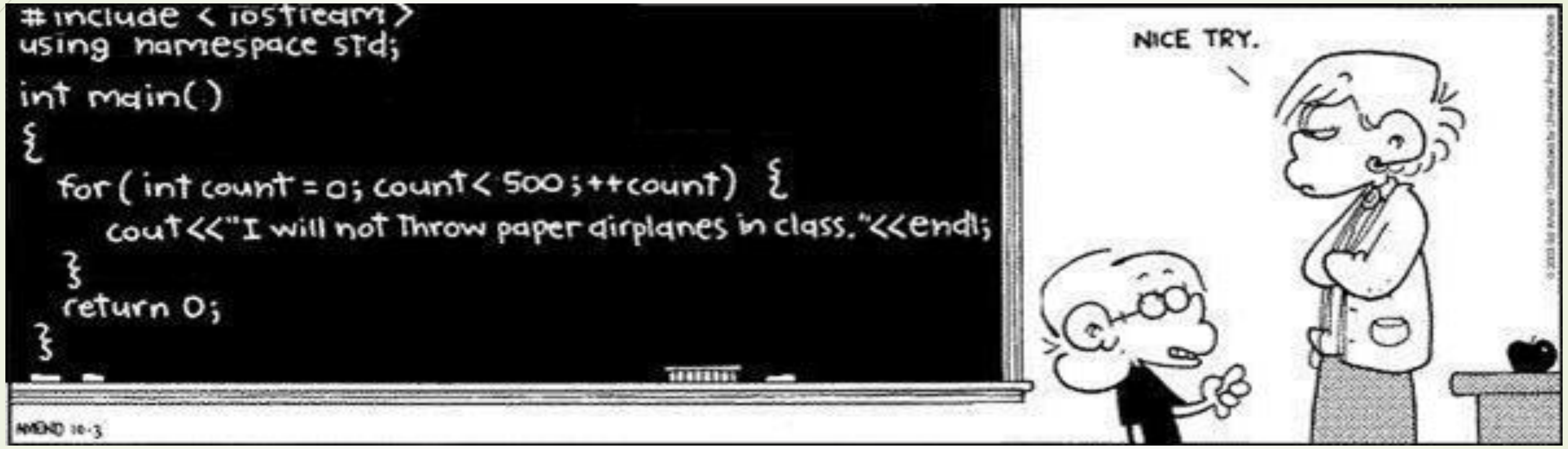
Classroom Etiquette

All laptop computers, cell phones, tablet computers must be closed during all classroom hours

- If you wish to use a computer, you are welcome to step outside
- Computers distract the most people behind and around the user
- You require a Verification of Illness form to use a computer in class
- The classroom is not for watching the next football match—even if your country is playing—but you are welcome to sit outside

Basic Requirements: Fundamentals of Programming

You will be using the C++ programming language in this course



```
#include <iostream>
using namespace std;

int main()
{
  for(int count=0; count<500;++count) {
     cout<<"I will not Throw paper airplanes in class."<<endl;
  }
  return 0;
}
```

This course does not teach C++ programming

- You will use C++ to demonstrate your knowledge in this course

One lecture covers:

- Features of C++
- It assumes minimal knowledge of programming

Evaluation

The course is divided into numerous topics

- Storing ordered and sorted objects
- Storing an arbitrary collection of data
- Sorting objects
- Graphs
- Algorithm Design Techniques

There are two self-study topics:

- Splay trees, and
- Disjoint sets

The self study topics will appear on the final examination as bonus questions

- A question for each will appear but you may only answer one

Evaluation (Cont..)

Your evaluation in this course is based on three components:

- Equally weighted projects
- One mid-semester test
- One final examination
- A commenting bonus (up to 5%)

You must pass **both** the examination component and the project component separately in order to pass the course

- If you fail either component, your grade is the lesser of the two
- Handing in no projects will result in a grade of zero

References

1.Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw Hill, 1990.

2.Weiss, Data Structures and Algorithm Analysis in C++, 3rd Ed., Addison Wesley.

3.Wikipedia, *Mathematical induction*, http://en.wikipedia.org/wiki/Mathematical_induction

4.Donald E. Knuth, *The Art of Computer Programming,Vol 1*, *Fundamental Algorithms*, 3rd Ed., Addison Wesley,1997

5.http://www.tutorialspoint.com/discrete_mathematics/discrete_mathematics_recurrence_relation.htm

6.https://en.wikipedia.org/wiki/Recurrence_relation

7.https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/

8.Online tutorial on C++ : http://www.cplusplus.com/

9.http://www.dummies.com/how-to/content/how-to-use-lhopitals-rule-to-solve-limit-problems.html

10.http://dl.uncw.edu/digilib/mathematics/algebra/mat111hb/eandl/logprop/logprop.html

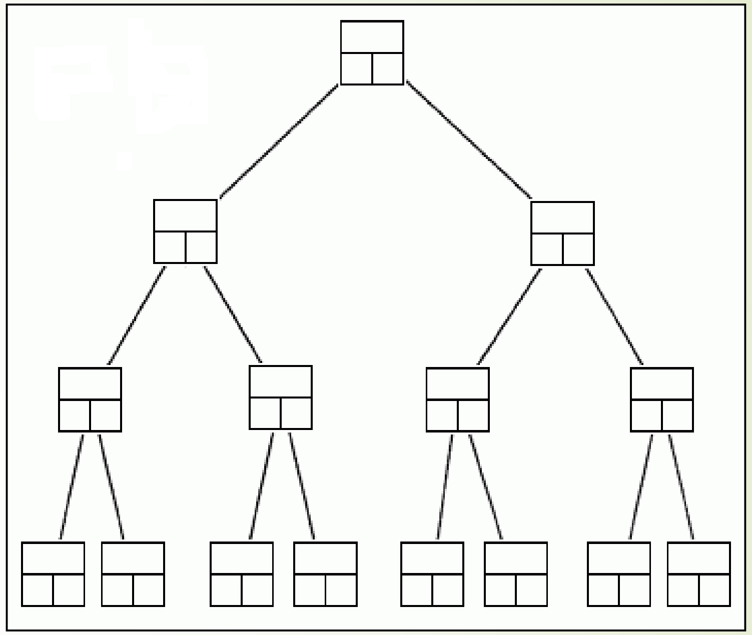
Data Structures

A data structure is a scheme for organizing data in the memory of a computer.

Some of the more commonly used data structures include lists, arrays, stacks, queues, heaps, trees, and graphs.

The way in which the data is organized affects the performance of a program for different tasks.

Computer programmers decide which data structures to use based on the nature of the data and the processes that need to be performed on that data.



Binary Tree

Example: A Queue

A *queue* is an example of commonly used simple data structure. A queue has beginning and end, called the *front* and *back* of the queue.

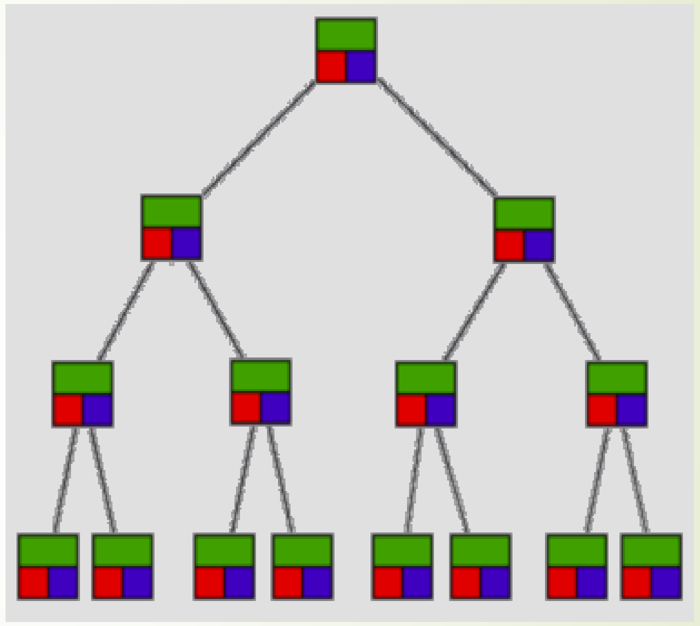
Data enters the queue at one end and leaves at the other. Because of this, data exits the queue in the same order in which it enters the queue, like people in a checkout line at a supermarket.

Example: A Binary Tree

A *binary tree* is another commonly used data structure. It is organized like an upside down tree.

Each spot on the tree, called a *node*, holds an item of data along with a left pointer and a right pointer.

The pointers are lined up so that the structure forms the upside down tree, with a single node at the top, called the root node, and branches increasing on the left and right as you go down the tree.



Binary Tree

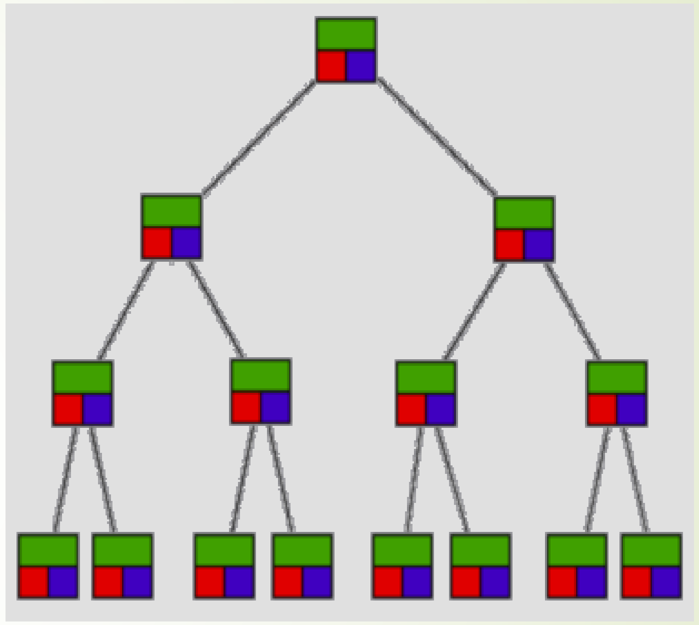
Choosing Data Structures

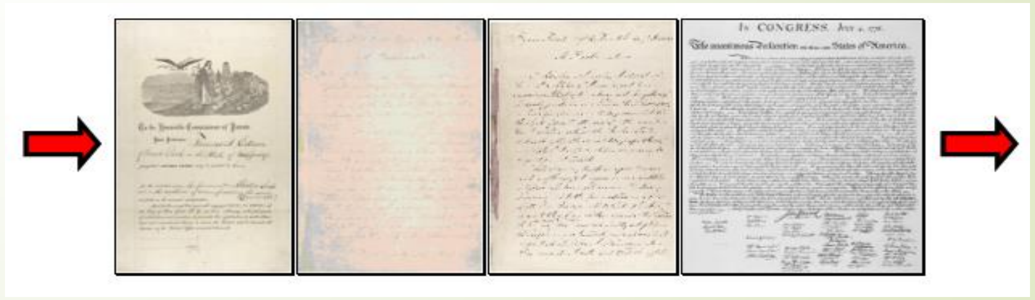
By comparing the queue with the binary tree, you can see how the structure of the data affects what can be done efficiently with the data.

A queue is a good data structure to use for storing things that need to be kept in order, such as a set of documents waiting to be printed on a network printer.

The jobs will be printed in the order in which they are received.

Most network print servers maintain such a *print queue*.


Binary Tree



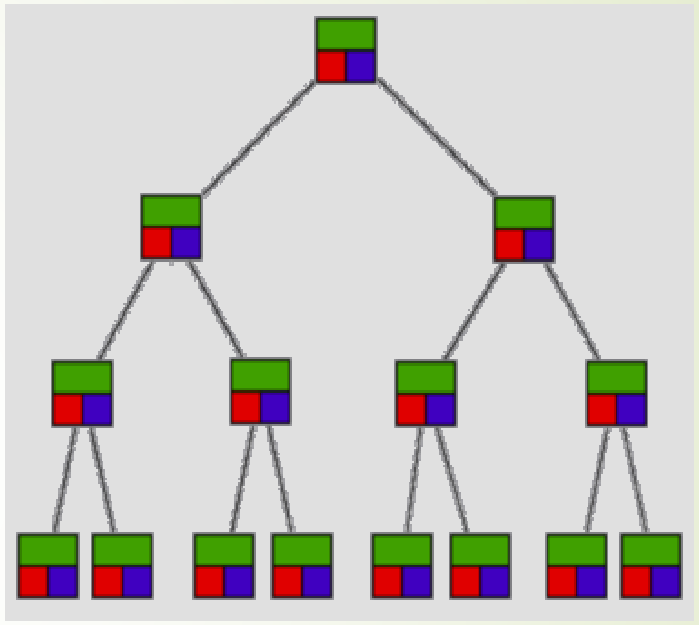
Choosing Data Structures

A binary tree is a good data structure to use for searching sorted data.

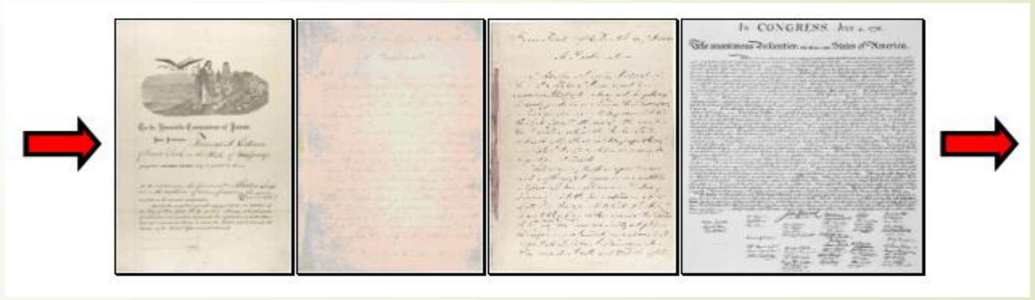
The middle item from the list is stored in the root node, with lesser items to the left and greater items to the right.

A search begins at the root. The computer either find the data, or moves left or right, depending on the value for which you are searching.

Each move down the tree cuts the remaining data in half.



Binary Tree



Choosing Data Structures

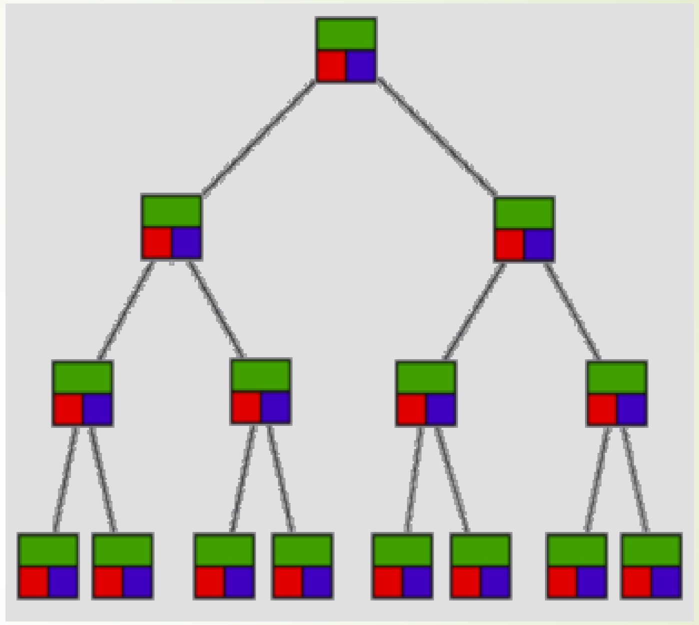
Items can be located very quickly in a tree.

Telephone directory assistance information is stored in a tree, so that a name and phone number can be found quickly.

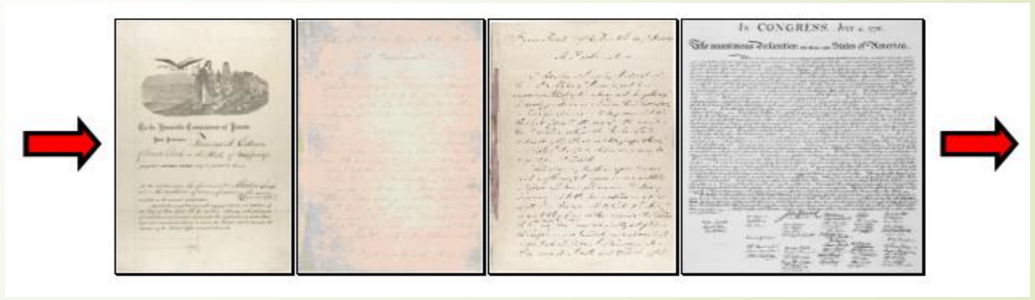
For some applications, a queue is the best data structure to use.

For others, a binary tree is better.

Programmers choose from among many data structures based on how the data will be used by the program.



Binary Tree



Data Structures in Alice

Alice has two built-in data structures that can be used to organize data, or to create other data structures:

- Lists
- Arrays

Lists

A list is an ordered set of data. It is often used to store objects that are to be processed sequentially.

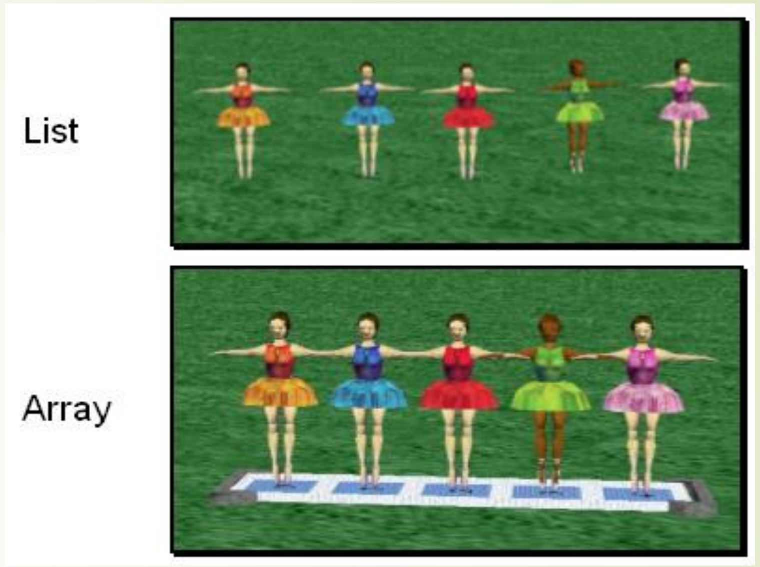
A list can be used to create a queue.

Arrays

An array is an indexed set of variables, such as dancer$_{[1]}$, dancer$_{[2]}$, dancer$_{[3]}$,… It is like a set of boxes that hold things.

A list is a set of items.

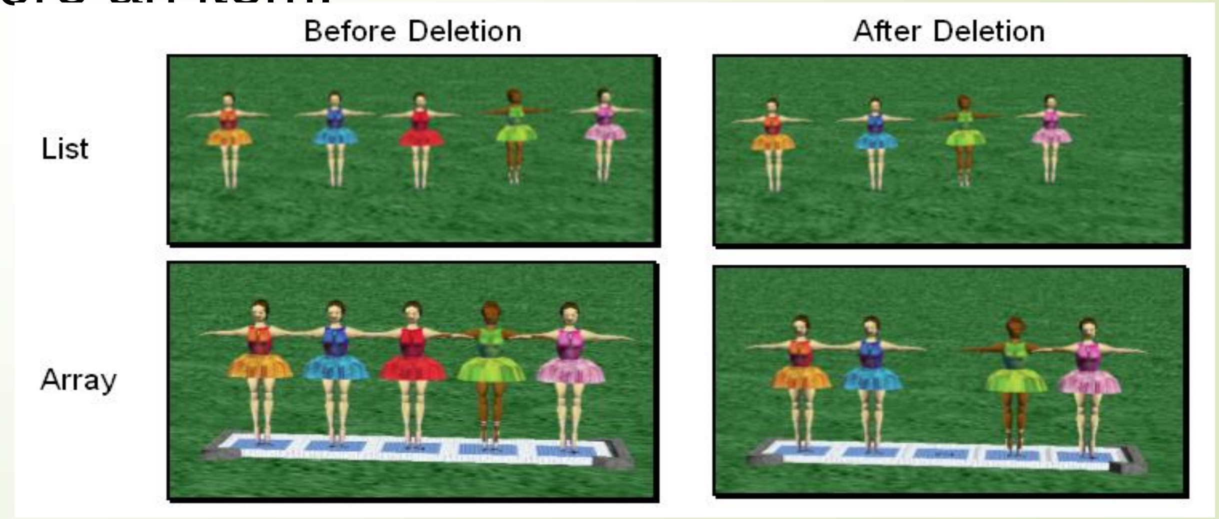
An array is a set of variables that each store an item.



Arrays and Lists

You can see the difference between arrays and lists when you delete items.

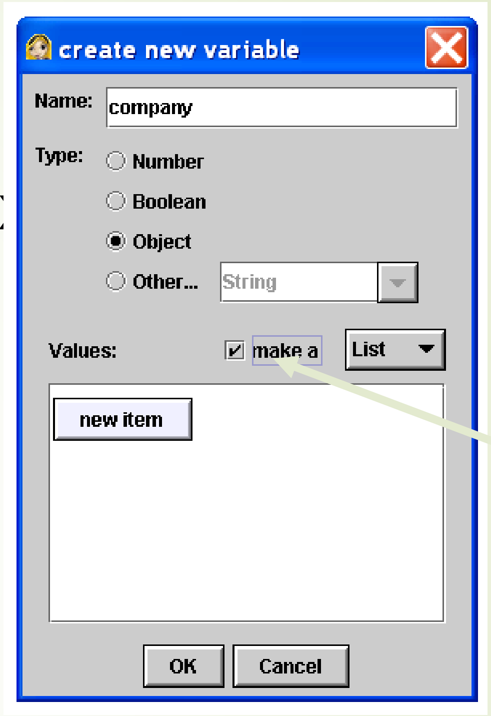
In an array, an empty variable is left behind when something is deleted.



Lists

A list is created in Alice by checking the make a list bo
when creating a new variable.

The *For all in order* and *For all together* tiles can be used to work with lists. They are at the bottom of the editor area.



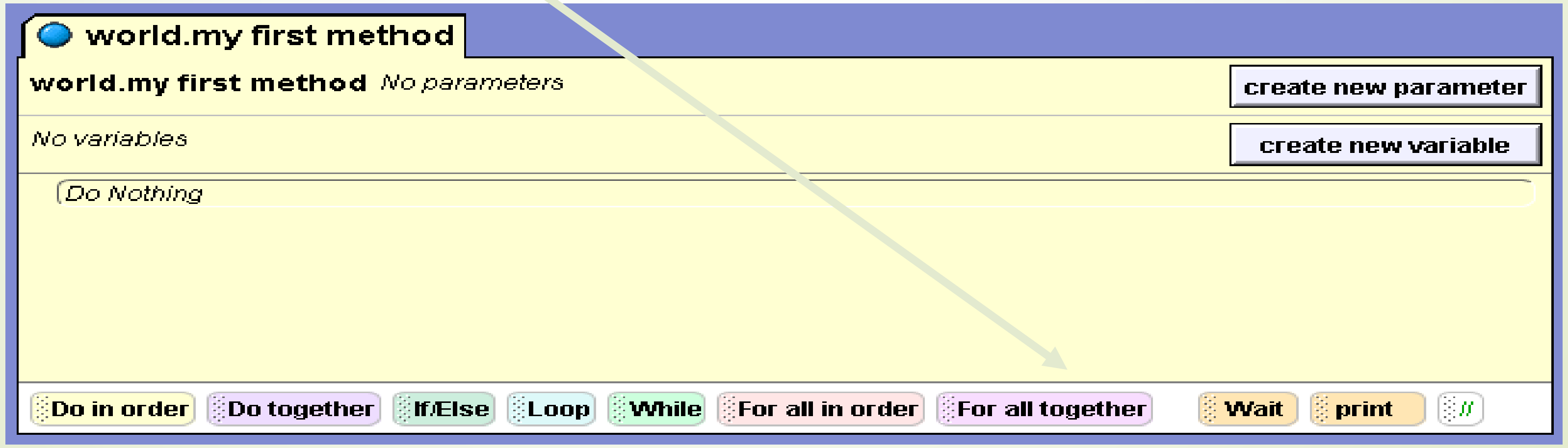
create new variable

Name: company

Type: Number / Boolean / [✓ Object] / Other... String

Values: [✓] make a List

Make a list **box**



world.my first method

world.my first method *No parameters*

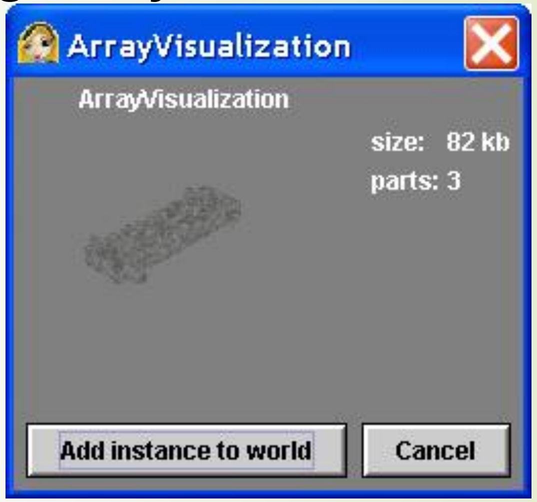
No variables

Do Nothing

Arrays

Arrays can be created in a similar manner, but more often they are created using the array visualization object from the Alice local gallery.

The Array Visualization object has special properties and methods for manipulating the elements in an array.



Alice has a set of built-in functions that can be performed on arrays.

Mathematical Background: Outline

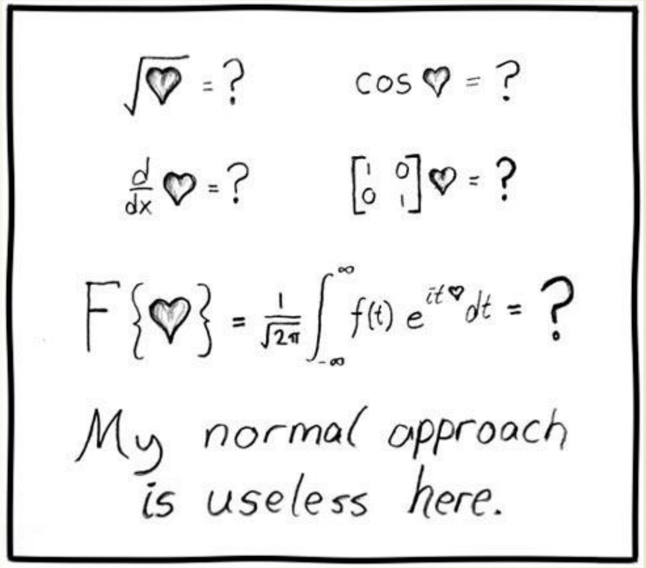
This topic reviews the basic mathematics required in this course:

- A justification for a mathematical framework
- The ceiling and floor functions
- L’Hôpital’s rule
- Logarithms
- Arithmetic and other polynomial series
  - Mathematical induction
- Geometric series
- Recurrence relations
- Weighted averages
- Combinations

Mathematics and Engineering

For engineers, mathematics is a tool:

- Of course, that doesn't mean it always works...


√♡ = ?
cos ♡ = ?
d/dx ♡ = ?
[1 0; 0 1] ♡ = ?
F{♡} = 1/√2π ∫ f(t) e^{it♡} dt = ?
My normal approach is useless here.

Justification

However, as engineers, you will not be paid to say:

Method A is *better* than Method B

or

Algorithm A is *faster* than Algorithm B

Such comparisons are said to be *qualitative*:

qualitative, *a.* Relating to, connected or concerned with, quality or qualities.

Now usually in implied or expressed opposition to quantitative.

OED

Justification (Cont.)

Qualitative statements cannot guide engineering design decisions:

- Algorithm A could be *better* than Algorithm B, but Algorithm A would require three person weeks to implement, test, and integrate while Algorithm B has already been implemented and has been used for the past year
- There are circumstances where it may beneficial to use Algorithm A, but not based on the word *better*

Thus, we will look at a *quantitative* means of describing data structures and algorithms:

quantitative, *a.* Relating to, concerned with, quantity or its measurement; ascertaining or expressing quantity. **OED**

This will be based on mathematics, and therefore we will look at a number of properties which will be used again and again throughout this course

Floor and Ceiling Functions

- The floor function maps any real number x onto the greatest integer less than or equal to x.

$$x = \max \{ m \in Z \mid m \leq x \}$$

- The ceiling function maps x onto the least integer greater than or equal to x

$$x = \min \{ n \in Z \mid n \geq x \}$$

What is the Floor & Ceiling of the following:

I. -2.344
II. 5.000
III. 12.45
IV. 4.57
V. 1- 3
VI. 1+ 3

Floor and Ceiling Functions

The *floor* function maps any real number x onto the greatest integer less than or equal to x:

$$\lfloor 3.2 \rfloor = \lfloor 3 \rfloor = 3$$

$$\lfloor -5.2 \rfloor = \lfloor -6 \rfloor = -6$$

- Consider it *rounding towards negative infinity*

The *ceiling* function maps x onto the least integer greater than or equal to x:

$$\lceil 3.2 \rceil = \lceil 4 \rceil = 4$$

$$\lceil -5.2 \rceil = \lceil -5 \rceil = -5$$

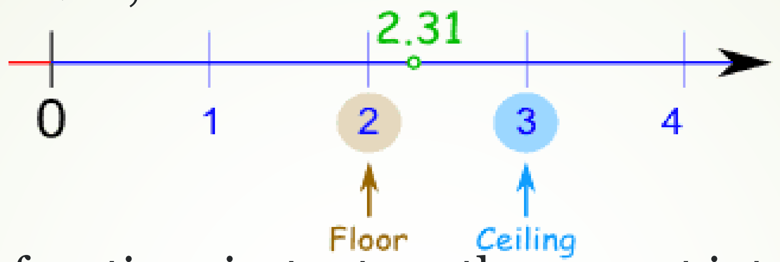
- Consider it *rounding towards positive infinity*
- The `cmath` library implements these as

```
double floor( double );    double ceil( double );
```

Necessary because `double` have a range just under 2^{1024} `long` can only represent numbers as large as $2^{63} - 1$

Floor and Ceiling Functions

It's easy to think about floor and ceil from the perspective of the number line. Let's say you have some decimal number, 2.31



So, as you can see, the functions just return the nearest integer values.

floor returns the nearest lowest integer and ceil returns the nearest highest integer.

All real numbers are made of a **characteristic** (an integer part) and **mantissa** (a fractional part)

Number = Characteristic + Mantissa

$$2.31=2+0.31$$

When floor a number, you can think of it as replacing the Mantissa with 0

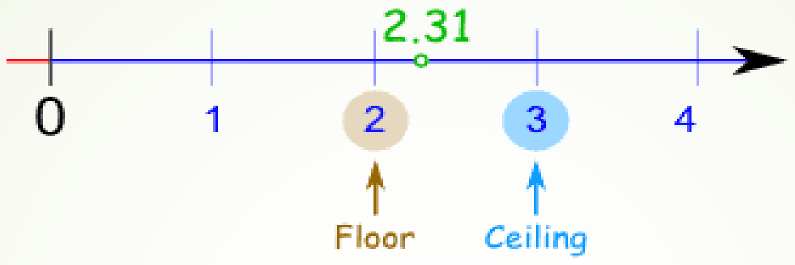
$$\lfloor 2.31\rfloor=2+0=2$$

and ceil can be thought of as replacing the mantissa with 1.

$$\lceil 2.31\rceil=2+1=3$$

Floor and Ceiling Functions (Cont.)

Example: What is the floor and ceiling of 2.31?



The Floor of 2.31 is **2**
The Ceiling of 2.31 is **3**

Floor and Ceiling of Integers

What if we want the floor or ceiling of a number that is already an integer?

That's easy: no change!

Example: What is the floor and ceiling of 5?

The Floor of 5 is **5**
The Ceiling of 5 is **5**

The L’Hôpital Rule

L’Hôpital Rule

Suppose that we have one of the following cases:

$$\lim_{n \to a} \frac{f(n)}{g(n)} = \frac{0}{0} \quad \text{or} \quad \lim_{n \to a} \frac{f(n)}{g(n)} = \frac{\pm\infty}{\pm\infty}$$

where a can be any real number, +ve/-ve infinity, in these case we have:

$$\lim_{n \to a} \frac{f(n)}{g(n)} = \lim_{n \to a} \frac{f'(n)}{g'(n)}$$

- Exercise: Evaluate each of the following limits:

1) $\lim$ $\sin s$

2) $\lim_{t \to 1} \frac{5t^5-4t^2-1}{10-t-9t^3}$

3) $\lim_{n \to œ} \frac{e^x}{s^2}$

Logarithms

Review - Logarithms

- If $n = e^m$, we define $m = \ln(n)$
- It is always true that $e^{\ln(n)} = n$; however, $\ln(e^n) = n$ requires that n is real
- **e** is the **Euler Number** which is 2.71828182845904523536028747135266249775724709369995….

Review – Logarithms (Properties)

1) $\log_b(xy) = \log_b x + \log_b y$

2) $\log_b(x/y) = \log_b x - \log_b y$

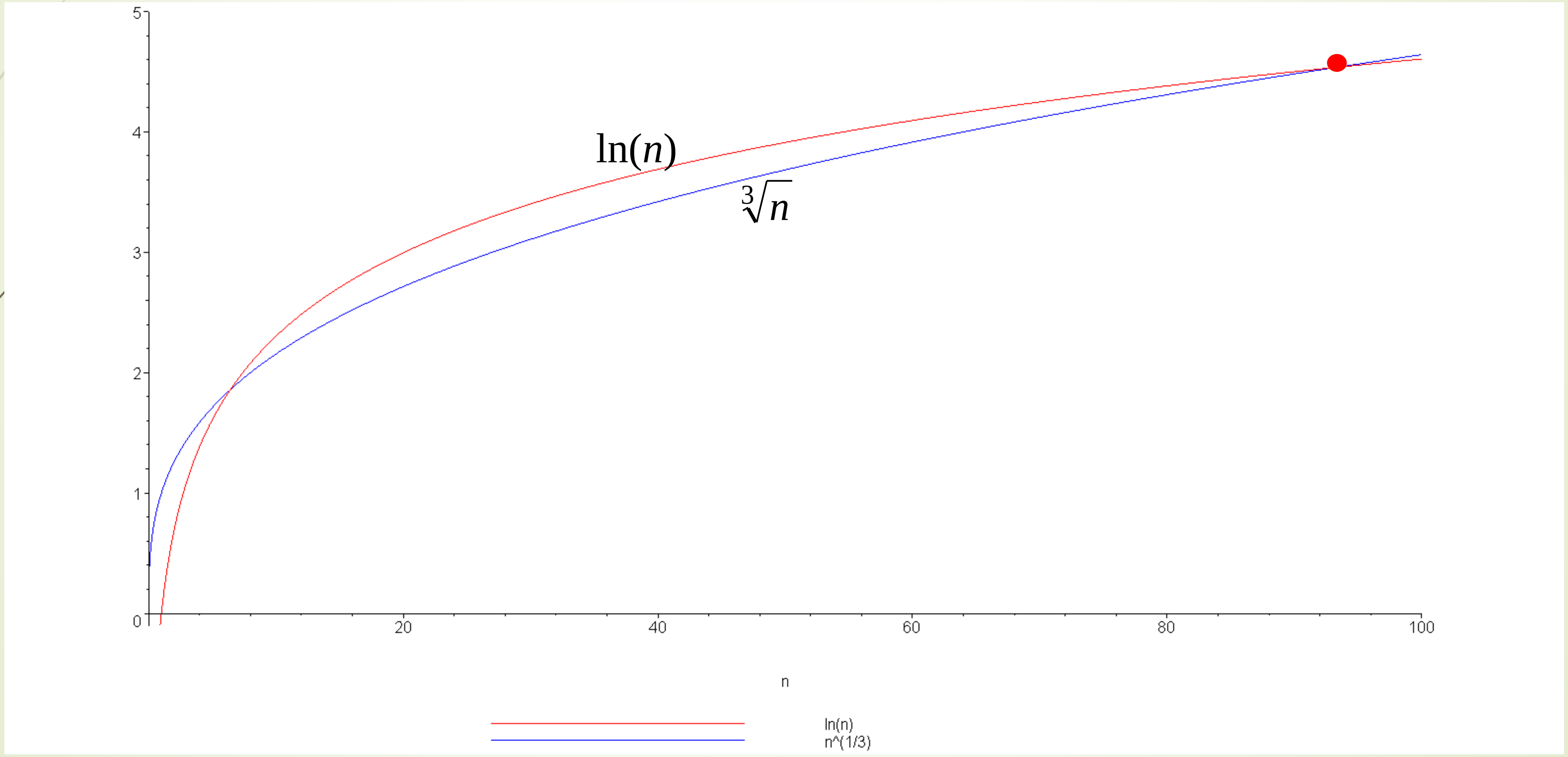
3) $\log_b x^n = n\log_b x$

- Specific Bases & Notations: Base 10, base 2 and base e (Euler Number)

• $\log_{10} x$	or $\log x$	logarithm of x to base 10
• Lg x	logarithm of x to base 2	
• Ln x	logarithm of x to base e	

Logarithms

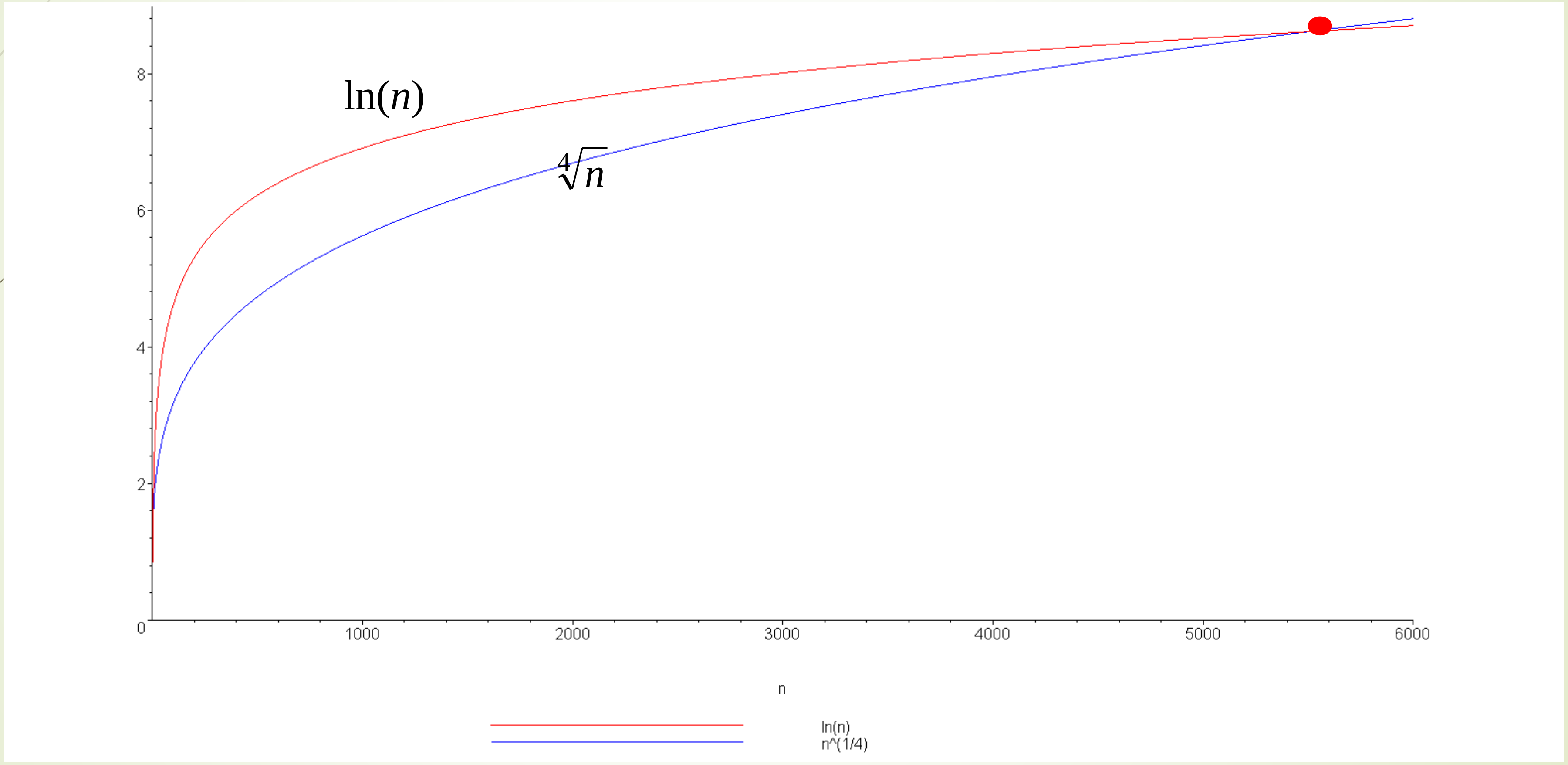
$f(n) = n^{1/3} = \sqrt[3]{n}$ grows slower but only up to $n = 93$



Tutor: A.J. Fofanah

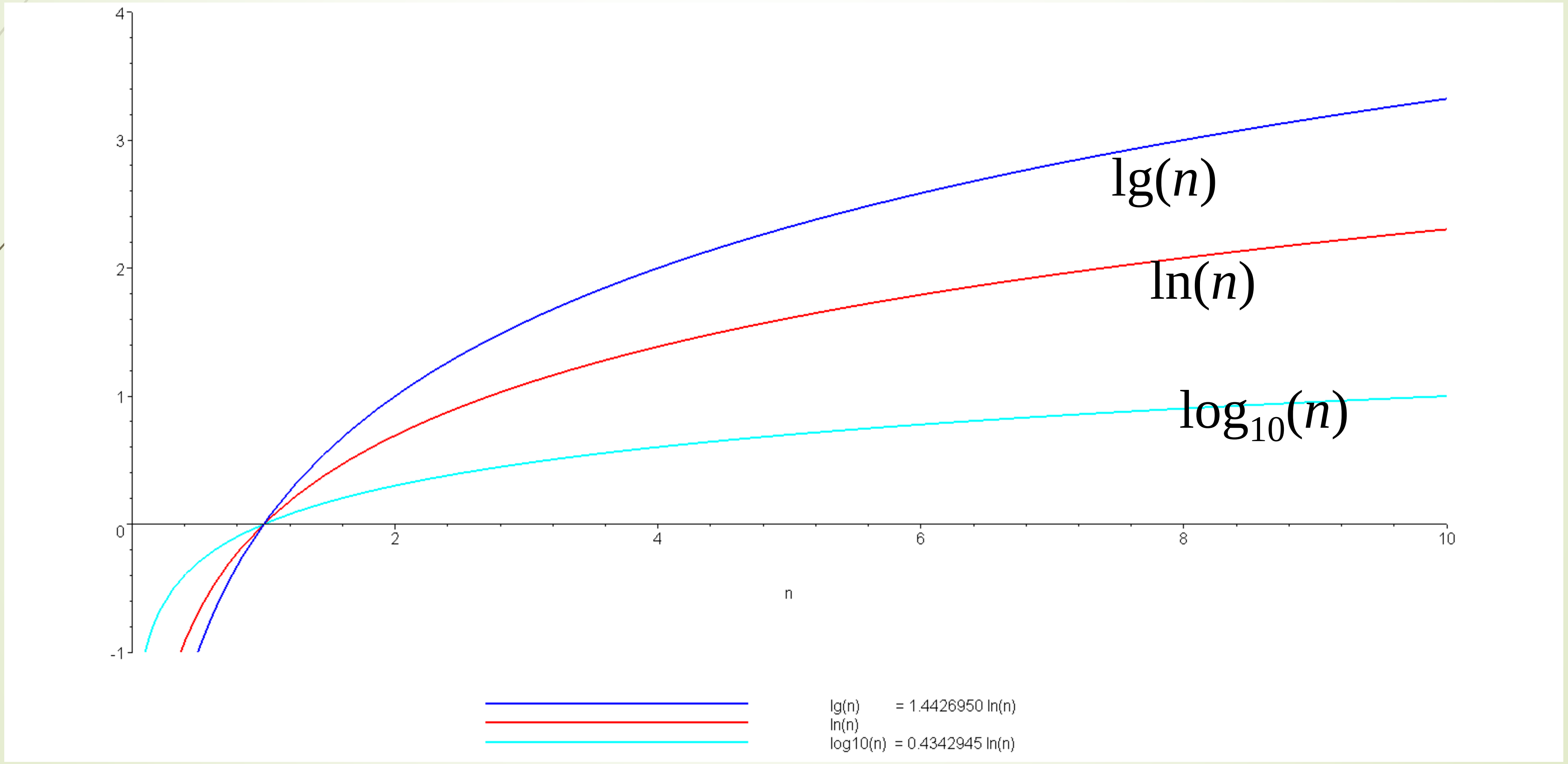
Logarithms

You can view this with any polynomial



Logarithms

A plot of $\log_2(n) = \lg(n)$, $\ln(n)$, and $\log_{10}(n)$



Logarithms

Review – Logarithms (Exercises)

1. Expand the expression:

$$\ln \frac{x}{\sqrt{x^2}+1}$$

2. Condense the expression:

$$3logx + 2logy - \left(\frac{1}{2}\right)\text{logz}$$

Logarithms

You should also, as electrical or computer engineers be aware of the relationship:

$\lg(2^{10}) = \lg(1024) \quad = 10$

$\lg(2^{20}) = \lg(1\ 048\ 576) \quad = 20$

and consequently:

$\lg(10^{3}) = \lg(1000) \quad \approx 10$ kilo

$\lg(10^{6}) = \lg(1\ 000\ 000) \quad \approx 20$ mega

$\lg(10^{9}) \quad \approx 30$ giga

$\lg(10^{12}) \quad \approx 40$ tera

Arithmetic Series

Next we will look various series

Each term in an arithmetic series is increased by a constant value (usually 1) :

$$0+1+2+3+\cdots+n=\sum_{k=0}^{n} k=\frac{n(n+1)}{2}$$

Proof 1: write out the series twice and add each column

$$\begin{array}{cccccccccccccc} & 1 & + & 2 & + & 3 & +\cdots+ & n-2 & + & n-1 & + & n \\ + & \underline{n} & \underline{+} & \underline{n-1} & \underline{+} & \underline{n-2} & \underline{+\cdots+} & \underline{3} & \underline{+} & \underline{2} & \underline{+} & \underline{1} \end{array}$$

$$(n+1)+(n+1)+(n+1)+\cdots+(n+1)+(n+1)+(n+1)$$

$$= n\,(n+1)$$

Since we added the series twice, we must divide the result by 2

Arithmetic Series

Review – Arithmetic Series

- To find any term of an arithmetic sequence:

$$a_n = a_1 + n - 1\ d$$

where n is the number of terns, d the common di$ff$erence $a_1$is the $f$irst term

- To find the sum of a certain number of terms of an arithmetic sequence:

$$S_n = \frac{n(a_1 + a_n)}{2}$$

Arithmetic Series: Other polynomial series

Proof 1: write out the series twice and add each column

$$\begin{array}{ccccccccccccc} & 1 & + & 2 & + & 3 & + \cdots + & n-2 & + & n-1 & + & n \\ + & \underline{n} & \underline{+} & \underline{n-1} & \underline{+} & \underline{n-2} & \underline{+ \cdots +} & \underline{3} & \underline{+} & \underline{2} & \underline{+} & \underline{1} \\ & (n+1) & + & (n+1) & + & (n+1) & + \cdots + & (n+1) & + & (n+1) & + & (n+1) \end{array}$$

$$= n\,(n+1)$$

Since we added the series twice, we must divide the result by 2

Arithmetic Series: Other polynomial series

Proof 2 (by induction):

The statement is true for $n = 0$:

$$\sum_{i=0}^{0} k = 0 = \frac{0 \cdot 1}{2} = \frac{0(0+1)}{2}$$

Assume that the statement is true for an arbitrary n:

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

Using the assumption that

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

for n, we must show that

$$\sum_{k=0}^{n+1} k = \frac{(n+1)(n+2)}{2}$$

Arithmetic Series: Other polynomial series

Then, for $n + 1$, we have:

$$\sum_{k=0}^{n+1} k = (n+1) + \sum_{i=0}^{n} k$$

By assumption, the second sum is known:

$$= (n+1) + \frac{n(n+1)}{2}$$

$$= \frac{(n+1)2 + (n+1)n}{2}$$

$$= \frac{(n+1)(n+2)}{2}$$

The statement is true for $n = 0$ and
the truth of the statement for n implies
the truth of the statement for $n + 1$.

Therefore, by the process of mathematical induction, the statement is true for all values of $n \geq 0$.

Arithmetic Series: Other polynomial series

We could repeat this process, after all:

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \qquad \sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$$

however, it is easier to see the pattern:

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2} \approx \frac{n^2}{2} \qquad \sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3}$$

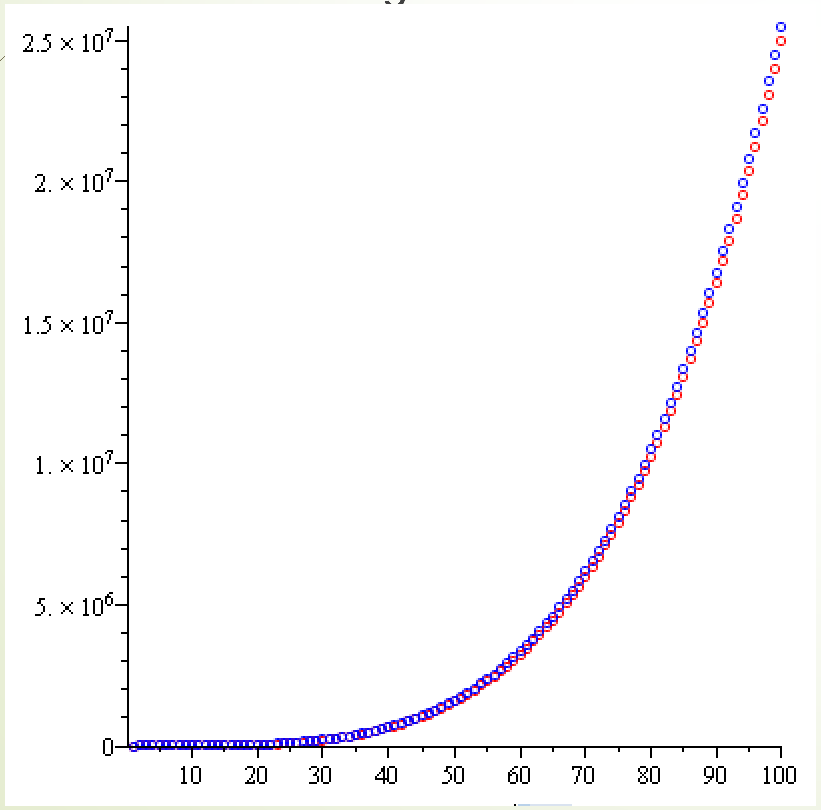
$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4} \approx \frac{n^4}{4}$$

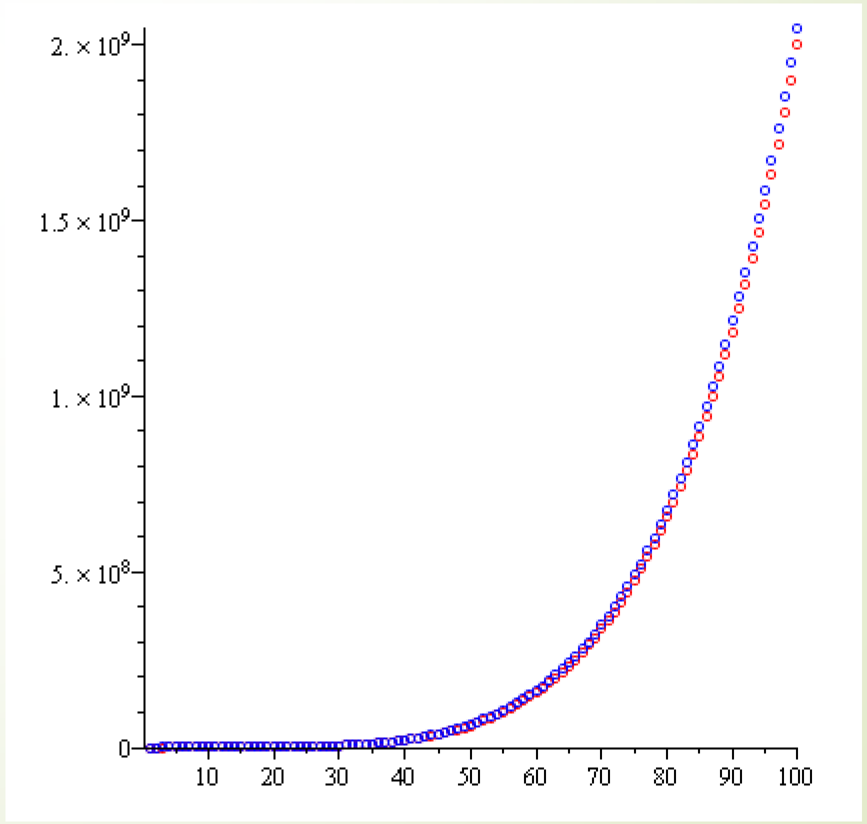
Arithmetic Series: Other polynomial series

We can generalize this formula

$$\sum_{k=0}^{n} k^d \approx \frac{n^{d+1}}{d+1}$$

Demonstrating with $d = 3$ and $d = 4$:



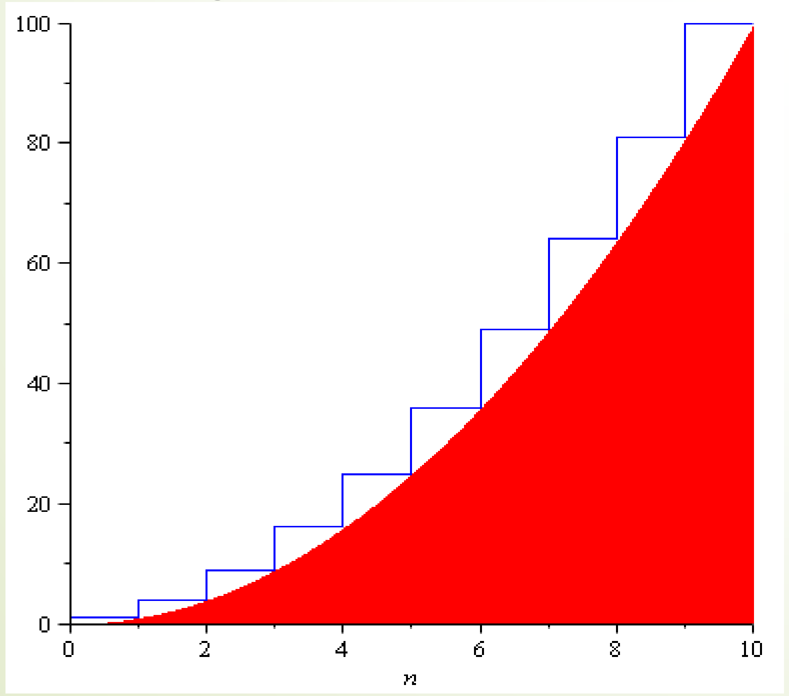


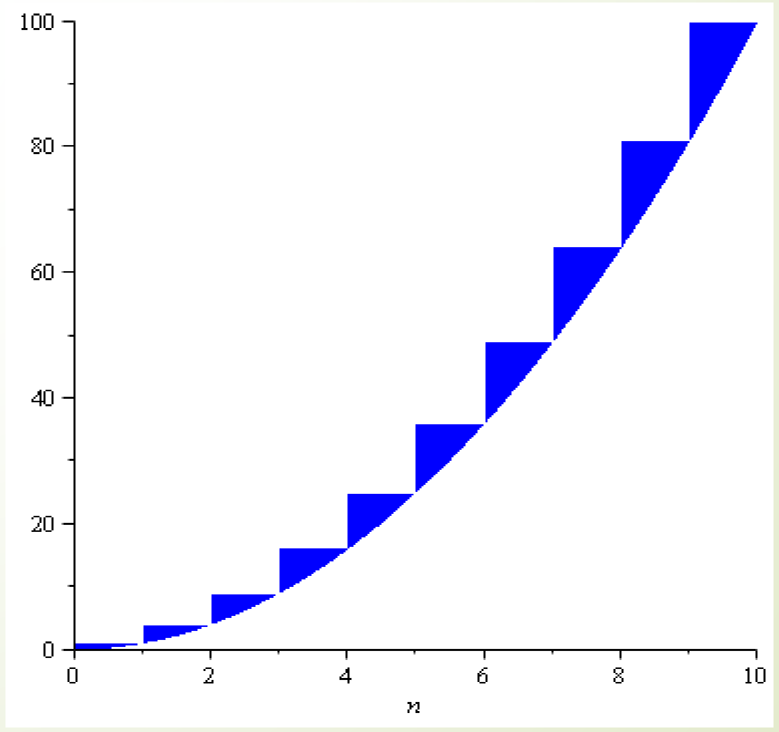
Arithmetic Series: Other polynomial series

The justification for the approximation is that we are approximating the sum with an integral:

$$\sum_{k=0}^{n} k^d \approx \int_0^n x^d\, dx = \left.\frac{x^{d+1}}{d+1}\right|_{x=0}^{n} = \frac{n^{d+1}}{d+1} - 0$$

However, there is an accumulating error:



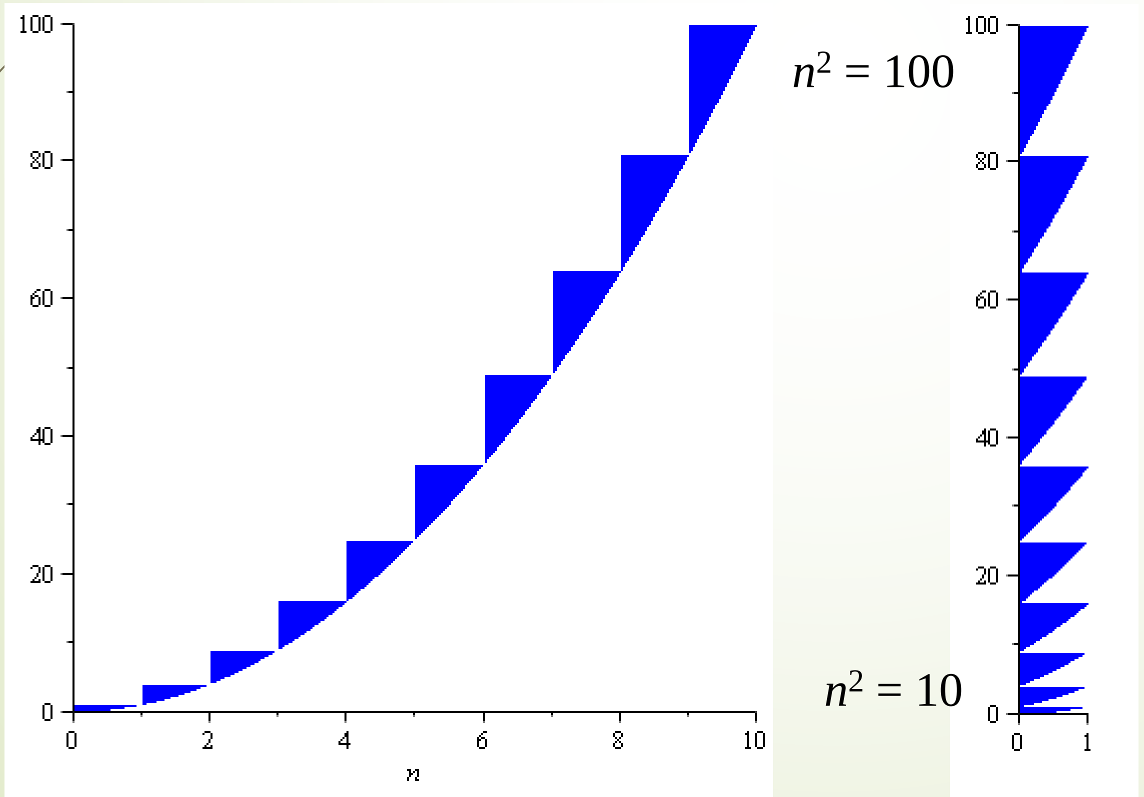


Arithmetic Series: Other polynomial series

How large is the error?

- Assuming $d > 1$, shifting the errors, we see that they would be

$$\frac{n^d}{2} \le \sum_{k=0}^{n} k^d - \frac{n^{d+1}}{d+1} < n^d \ll n^{d+1}$$



Geometric Series

The next series we will look at is the geometric series with common ratio r:

$$\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}$$

and if $|r| < 1$ then it is also true that

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$$

Geometric Series

Elegant proof: multiply by $1 = \frac{1-r}{1-r}$

$$\sum_{k=0}^{n} r^k = \frac{(1-r)\sum_{k=0}^{n} r^k}{1-r}$$

$$= \frac{\sum_{k=0}^{n} r^k - r\sum_{k=0}^{n} r^k}{1-r}$$

$$= \frac{(\textcircled{1} + r + r^2 + \cdots + r^n) - (r + r^2 + \cdots + r^n + \textcircled{r^{n+1}})}{1-r}$$

$$= \frac{1-r^{n+1}}{1-r}$$

Telescoping series:
all but the first and last terms cancel

Geometric Series

Proof by induction:

The formula is correct for $n = 0$: $\sum_{k=0}^{0} r^k = r^0 = 1 = \frac{1-r^{0+1}}{1-r}$

Assume the formula $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$ is true for an arbitrary n; then

$$\sum_{k=0}^{n+1} r^k = r^{n+1} + \sum_{k=0}^{n} r^k = r^{n+1} + \frac{1-r^{n+1}}{1-r} = \frac{(1-r)r^{n+1} + 1 - r^{n+1}}{1-r}$$

$$= \frac{r^{n+1} - r^{n+2} + 1 - r^{n+1}}{1-r} = \frac{1-r^{n+2}}{1-r} = \frac{1-r^{(n+1)+1}}{1-r}$$

and therefore, by the process of mathematical induction, the statement is true for all $n \geq 0$.

Recurrence relations