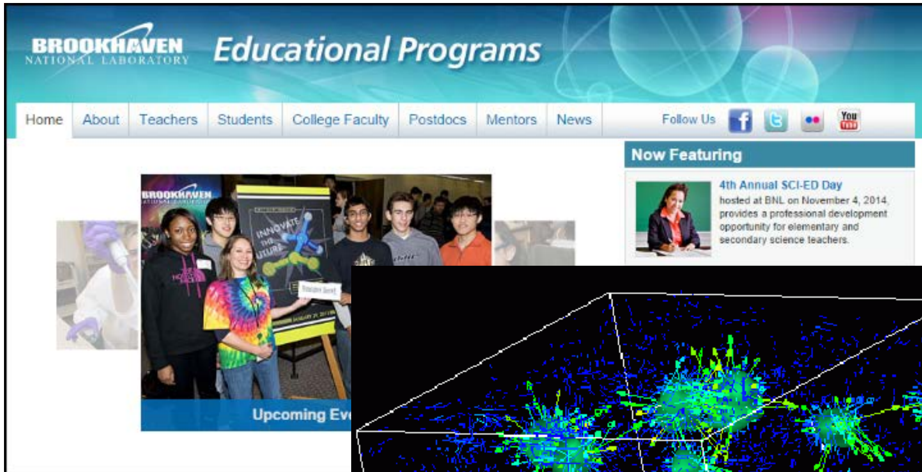
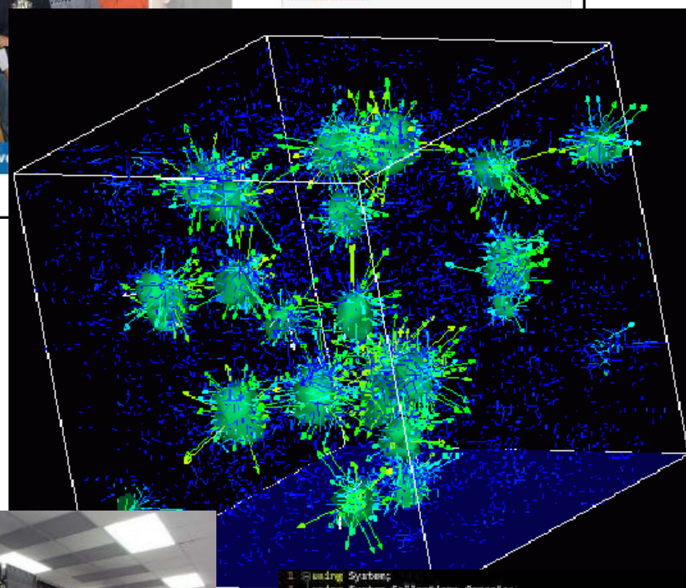
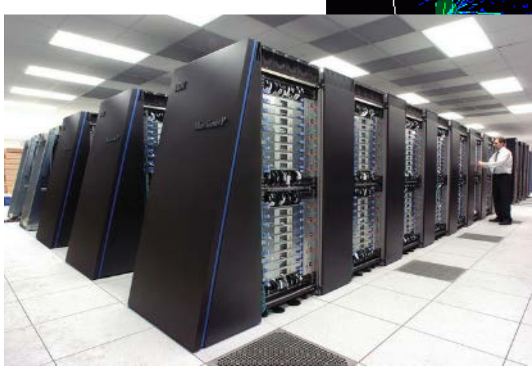






Survey of Scientific Computing
(SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Session 13
CERN ROOT,
Nyquist Sampling,
Collatz Conjecture

Session Goals

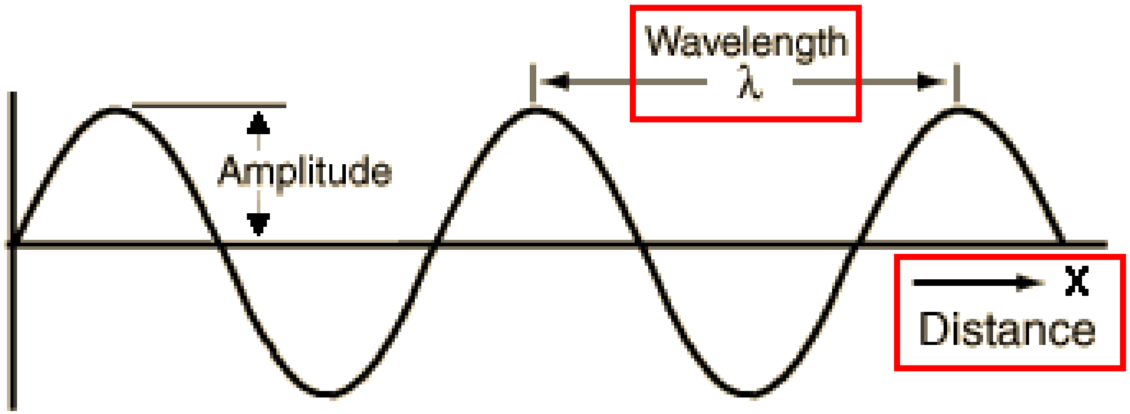
- Review the key parts of a **sinusoidal** transverse wave
- Graph a waveform using the **ROOT** libraries from **CERN**
- Appreciate the affect of sampling rate on **aliasing**
- Understand the **Nyquist Sampling Theorem**
- Perform a computational mathematical experiment to analyze the stopping time of the **Collatz** sequence

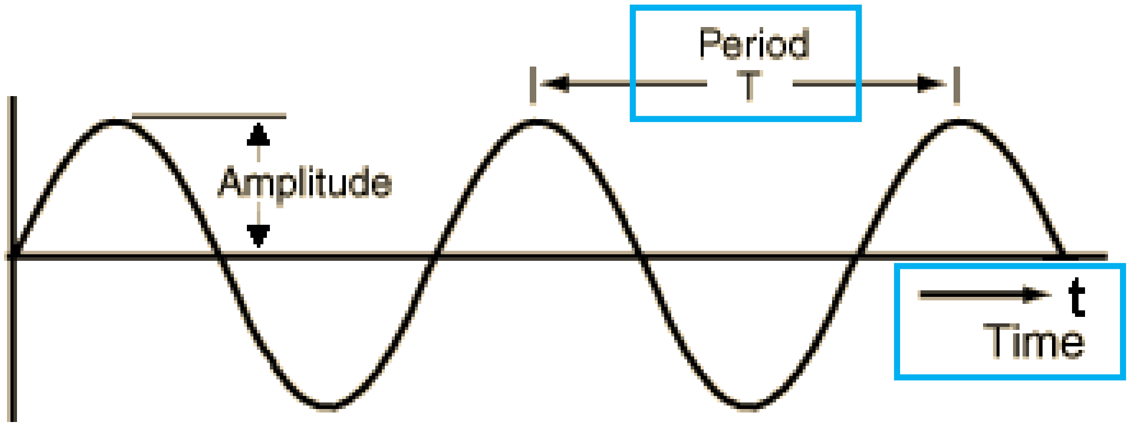
All of Physics is Waves

- Electrical
- Magnetic
- Acoustic
- Heat Flow
- Vibrational
- Torsional
- Nuclear / Quantum
- Gravitational
- Oceanic / Tidal
- Orbital Precession
- Springs
- Pendulums
- Tomography
- Stock Market
- Economics
- Astronomical
- Fluid Dynamics
- Earthquakes
- AC / DC
- AM / FM
- Speech
- Heartbeats

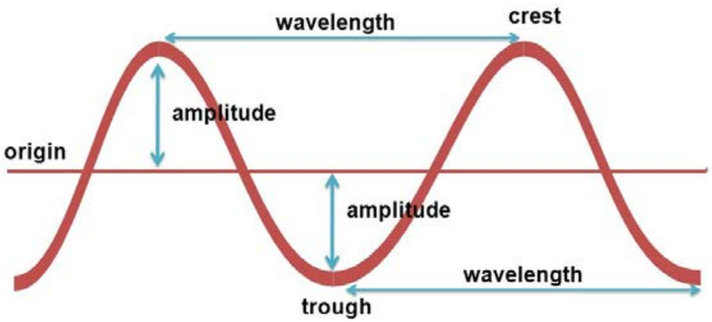
It is really important that you develop a keen understanding of the mathematics of waves!

Transverse Wave Components





Transverse Wave Components



$$\textbf{Period} = \boldsymbol{T} = \frac{time}{crests}$$

$$\textbf{Frequency} = \boldsymbol{f} = \frac{crests}{time} = \frac{1}{\boldsymbol{T}}$$

$$\textbf{Angular Frequency} = \boldsymbol{\omega} = 2\pi f$$

$$\textbf{Wavelength} = \boldsymbol{\lambda} = \frac{distance}{crests}$$

$$\textbf{Wave velocity} = v = \frac{distance}{time}$$

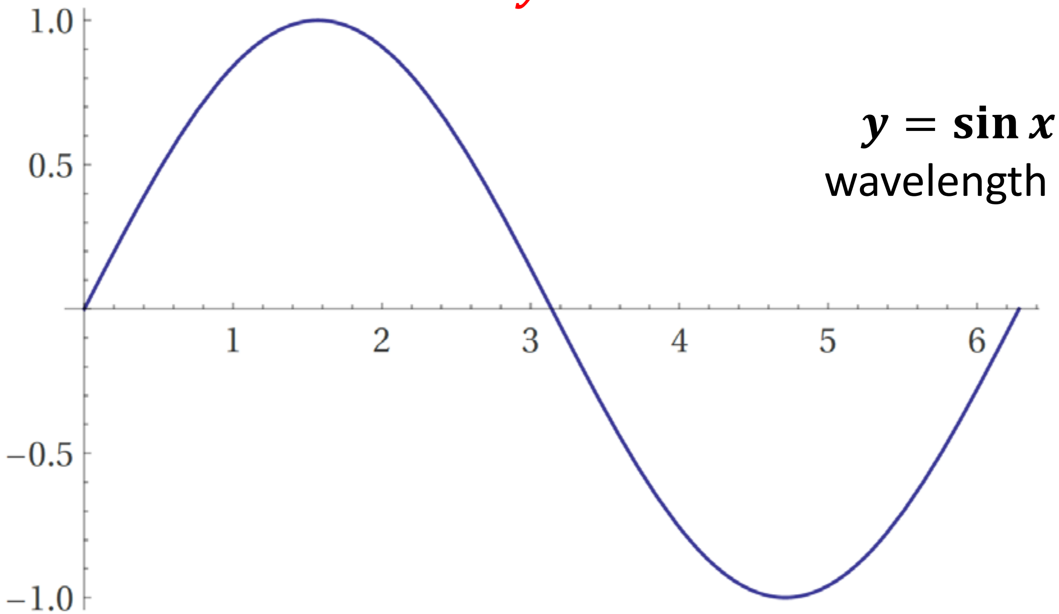
$$v = \left(\frac{distance}{\cancel{crests}}\right)\left(\frac{\cancel{crests}}{time}\right)$$

$$v = \lambda f$$

$$\boxed{\textbf{Wavenumber} = \boldsymbol{k} = \frac{2\pi}{\boldsymbol{\lambda}}}$$

Transverse Wave Components

$\boldsymbol{y = \sin x}$



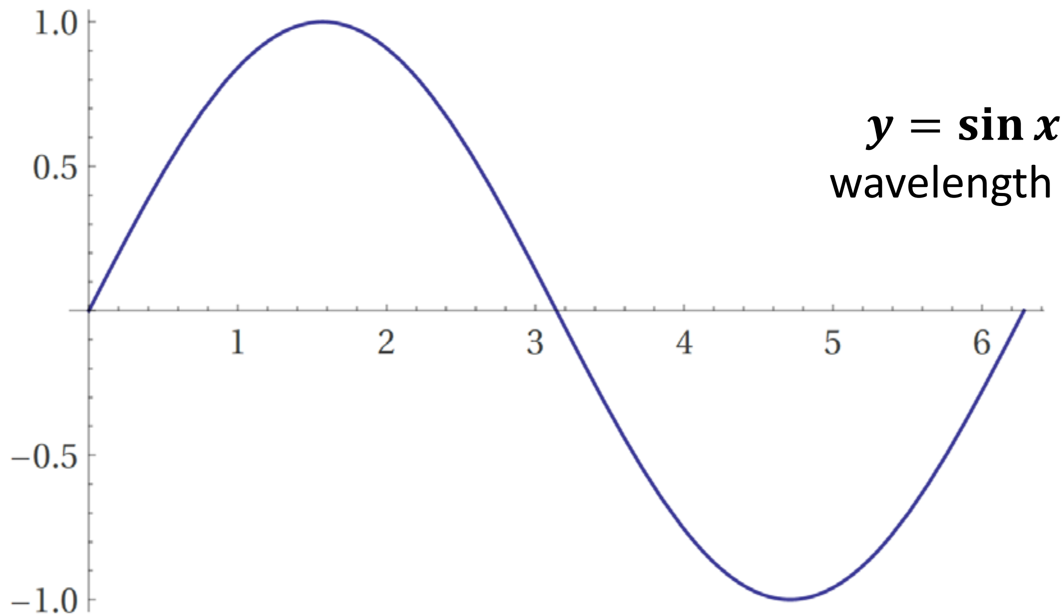
$\boldsymbol{y = \sin x}$ has a wavelength $\lambda = 2\pi$

$$\sin\left(\frac{\pi}{2}\right) = 1$$

$$\sin\left(\frac{\pi}{2} + 2\pi\right) = \sin\frac{\pi}{2}\cos 2\pi + \cos\frac{\pi}{2}\sin 2\pi = 1$$

Crest to Crest

Transverse Wave Components



$\boldsymbol{y} = \mathbf{sin}\,\boldsymbol{x}$ has a wavelength $\lambda = 2\pi$

if k = 1

$$\boxed{Wavenumber = \boldsymbol{k} = \frac{2\pi}{\boldsymbol{\lambda}}}$$

$$y = \sin \boldsymbol{k}x \Rightarrow \lambda = \frac{2\pi}{\boldsymbol{k}}$$

Example: $\lambda = \dfrac{5\,(distance)}{2\,(crests)} \Rightarrow \boxed{y = \sin \dfrac{4\pi}{5} x}$

ROOT

Data Analysis Framework

CERN

ROOT
Data Analysis Framework

ROOT is a software toolkit

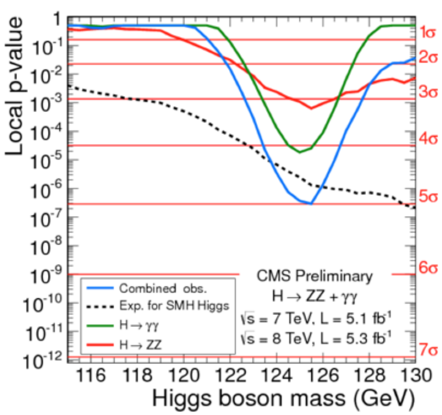
- Data processing
- Data analysis
- Data visualisation
- Data storage

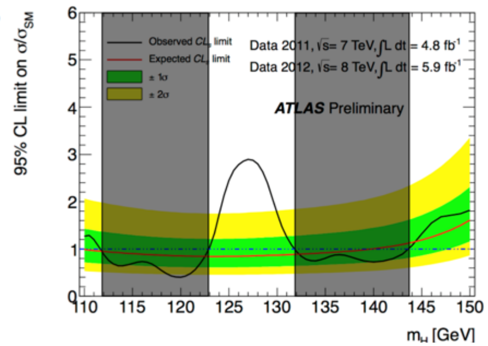
https://root.cern.ch

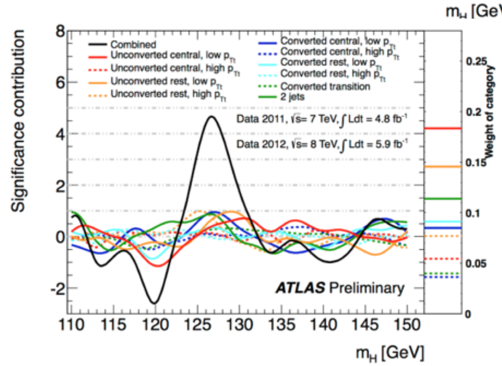
- ROOT is written in **C++**
- Bindings for **Python** are also provided

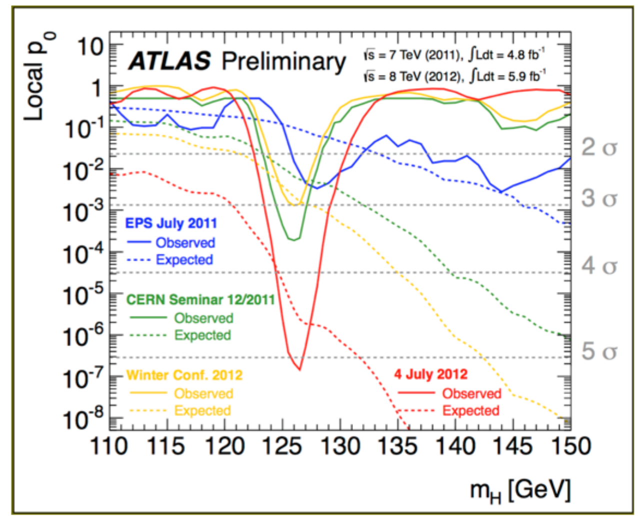
Higgs boson discovery

On July 4th 2012, the plots presented by the ATLAS and CMS collaborations where all produced with ROOT !



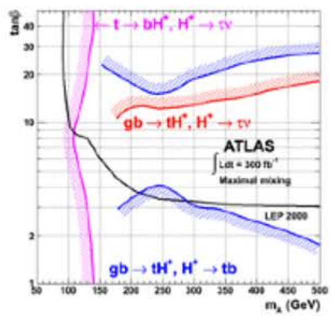


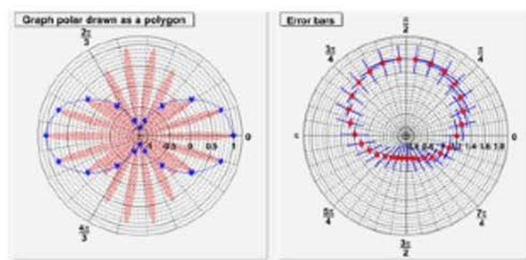


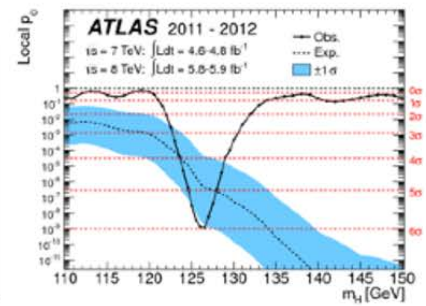


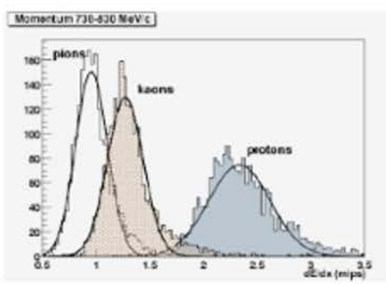
Graphics In ROOT

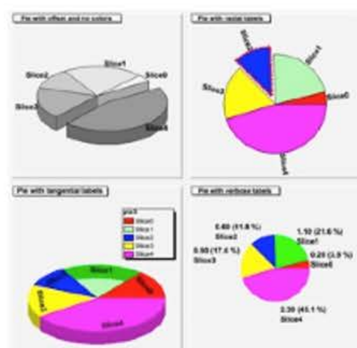
Many formats for data analysis, and not only, plots

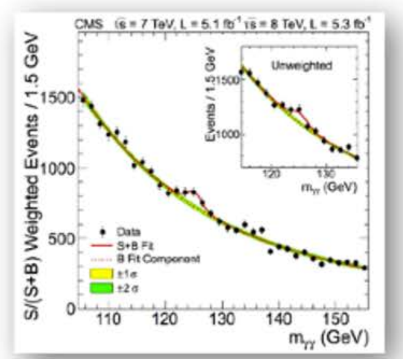












The ROOT C++ Libraries

- ROOT is an **object-oriented** framework
- Core capabilities are organized into **hierarchy**
- ROOT custom data types all start with a capital **T**
- TApplication, TCanvas, TGraph, TFunction, THistogram
- Note: ROOT uses C++ “pointers” and the **new()** operator to construct objects on the heap
  - Pointers have an asterisk ***** after the type name before the variable name
  - Calling methods on the object that a pointer “points to” is done using the **->** operator

Known Wave Aliasing

- Write a ROOT program to display a graph of the function

$$y = \sin\left(\frac{4\pi}{5}x\right)\Bigg|_0^{20}$$

- High-level approach:
  1. Subdivide the specified domain into $\boldsymbol{n} = \mathbf{640}$ intervals
  2. Calculate the range $\boldsymbol{y_i}$ at each domain $\boldsymbol{x_i}$ value
  3. Store the domain and range values in **vectors** of type **double**
  4. Pass the two vectors to ROOT so it can draw a line graph connecting successive $(\boldsymbol{x_i}, \boldsymbol{y_i})$ points in the curve

Open Lab 1 – Known Wave Aliasing

```
// SinusoidAliasing.cpp

#include "stdafx.h"

using namespace std;

const double b = 20;
const int n = 640;

vector<double> x(n + 1);
vector<double> y(n + 1);

void InitData()
{
    double dx = b / n;

    for (int i = 0;i <= n;i++) {
        x.at(i) = i * dx;
        y.at(i) = sin(4.0 / 5.0 * M_PI * x.at(i));
    }
}
```

$$y = \sin\left(\frac{4\pi}{5}x\right)\Bigg|_0^{20}$$

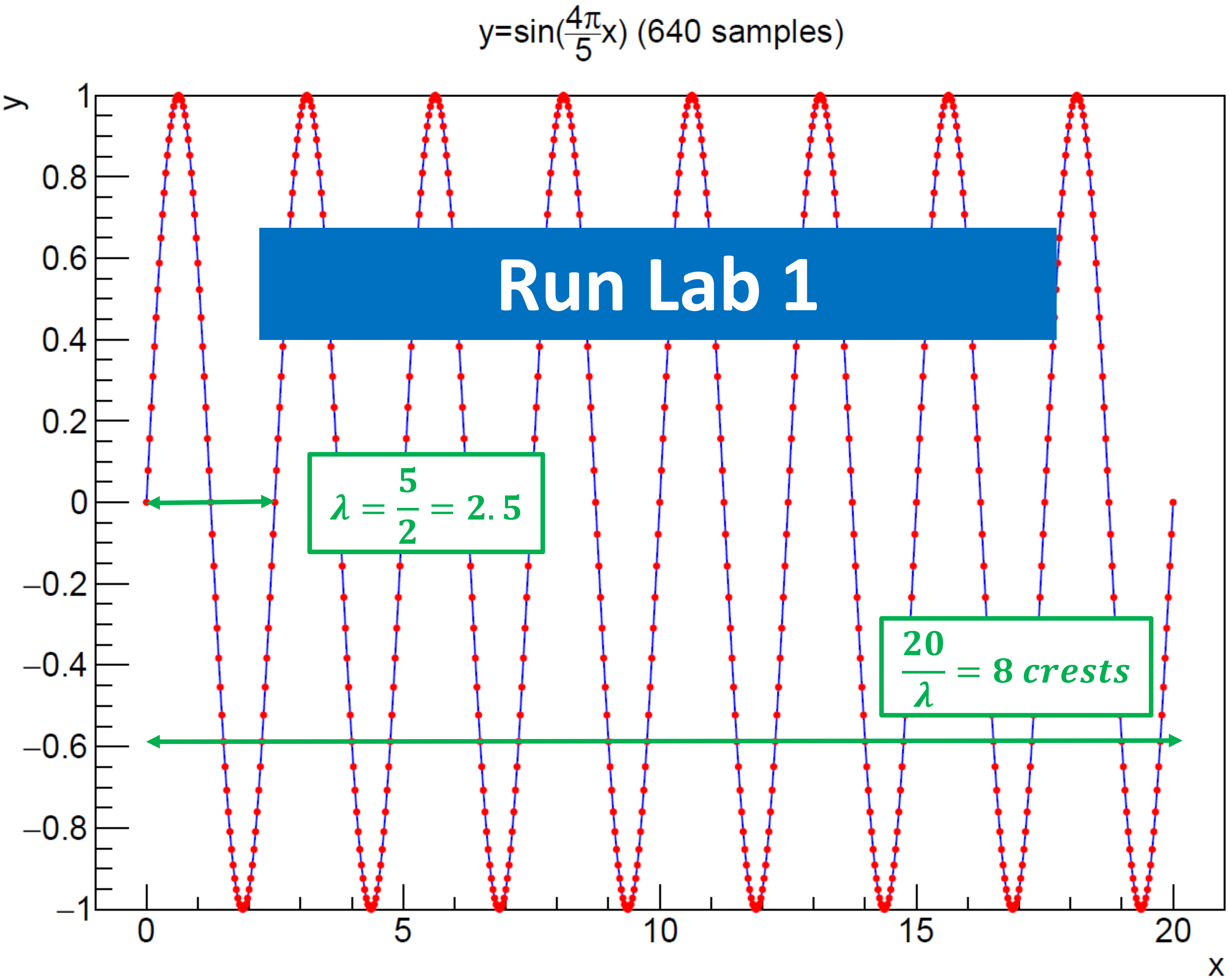
Δx

View Lab 1 – Known Wave Aliasing

```
void main()
{
    InitData();

    TApplication* theApp = new
        TApplication(nullptr, nullptr, nullptr);

    TCanvas* c = new TCanvas(
        (string().append("Sinusoid Aliasing (")
        .append(to_string(n))
        .append(" samples)")).c_str());

    TMultiGraph* mg = new TMultiGraph();
    string title = "y=sin(#frac{4#pi}{5}x) ("
        + to_string(n) + " samples)";
    mg->SetTitle(title.c_str());
    gStyle->SetTitleFontSize(0.03f);

    mg->SetMinimum(-1.);
    mg->SetMaximum(1.);

    TGraph* g = new TGraph(n + 1, &x[0], &y[0]);
    g->SetLineColor(kBlue);
    g->SetMarkerColor(kRed);
    g->SetMarkerStyle(kFullDotLarge);

    mg->Add(g);
    mg->Draw("ALP");

    mg->GetXaxis()->SetTitle("x");
    mg->GetYaxis()->SetTitle("y");

    gPad->Modified();

    theApp->Run();
}
```

- TApplication*
- new(), nullptr
- TCanvas*
- to_string()
- TMultiGraph*
- #frac{} - **LATEX**
- -> operator
- title.c_str()
- TGraph*
- mg->Add(g)
- mg->Draw("ALP")
- gPad->Modified()
- theApp->Run()

Run Lab 1

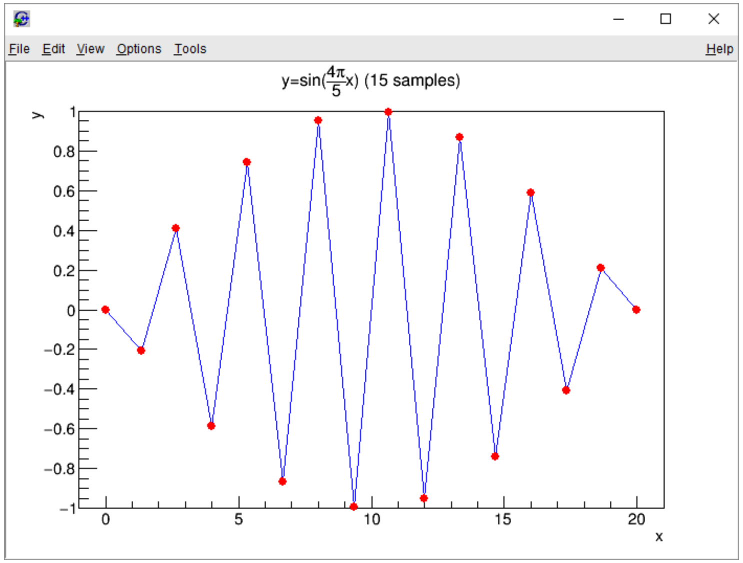


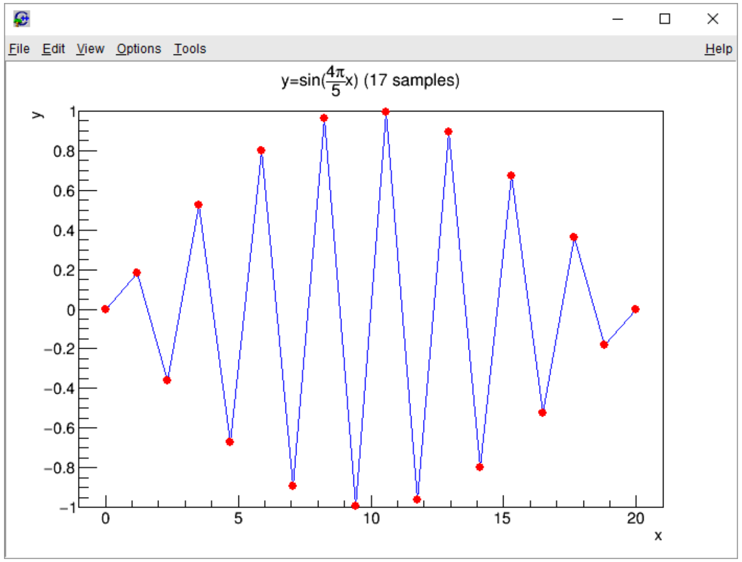
$y=\sin(\frac{4\pi}{5}x)$ (640 samples)
y
x
$\lambda = \frac{5}{2} = 2.5$
$\frac{20}{\lambda} = 8\ crests$

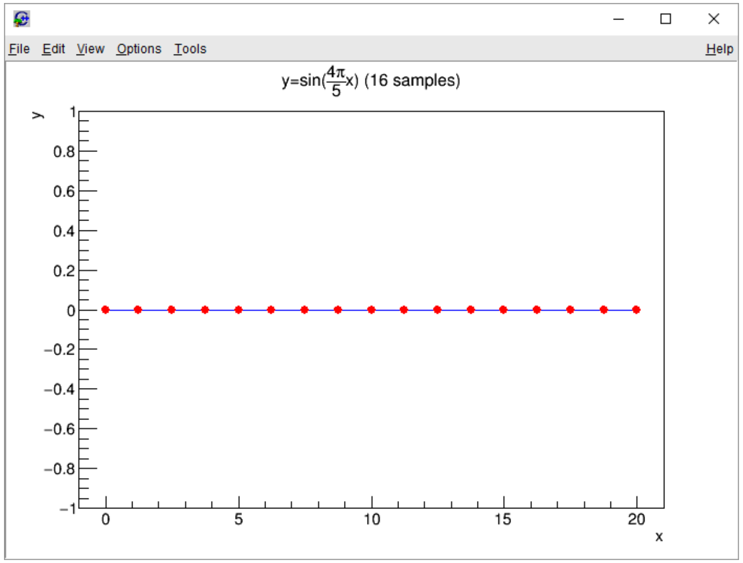
Edit Lab 1 – Known Wave Aliasing

- Edit the Lab 1 code so that **n = 15** and then run it again
- Again edit the code so that **n = 17** and then run it again
- Then set **n = 16** and run it again – *now what happens?*

Run Lab 1 – Known Wave Aliasing



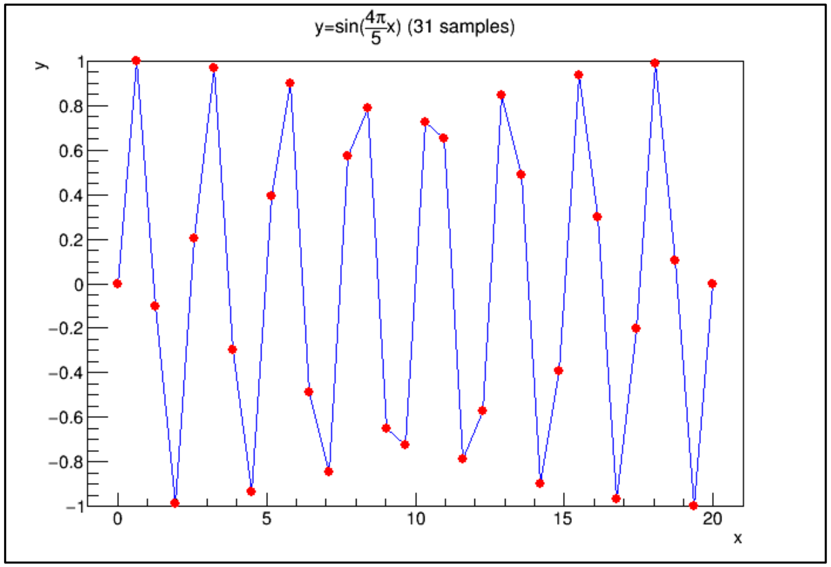


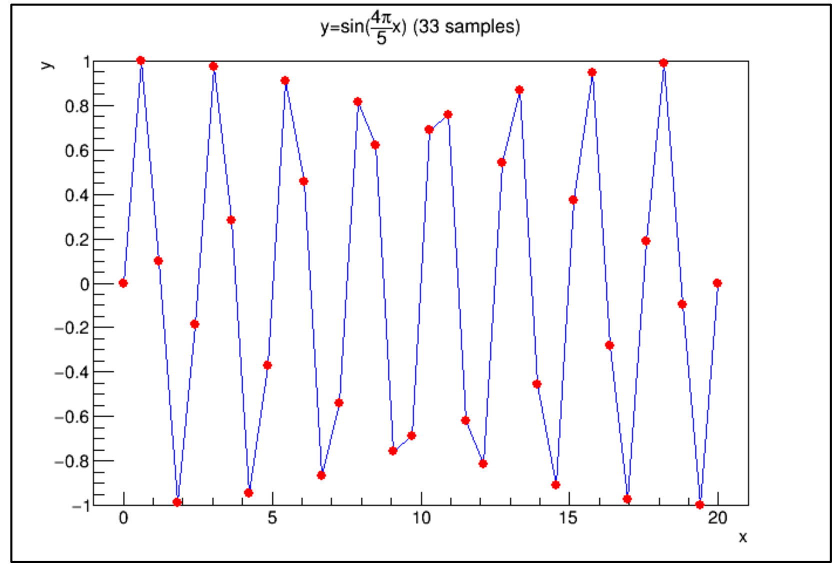


Edit Lab 1 – Known Wave Aliasing

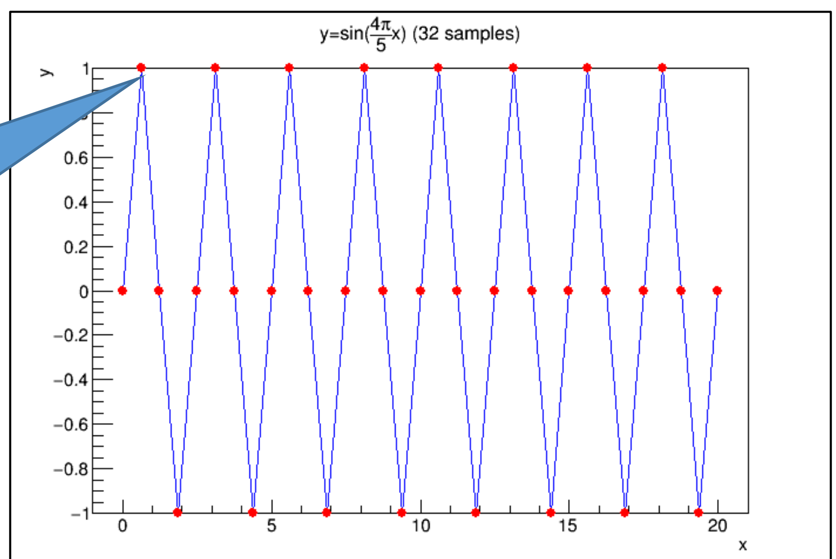
- Run **Lab 1** setting n = 15 and then n = 17
- Then set **n = 16** – *what happens?*
- What is the specific relationship between the sinusoid's $\boldsymbol{\lambda}$ and the graph's $\boldsymbol{\Delta x}$ that causes this ***aliasing***?
- Set **n = 8** – *what happens?*
- Does catastrophic aliasing occur for any value $\mathbf{n \geq 16}$?
- What is the difference in graphs for **n = 31, 32, 33** ?
- How can **n** be chosen to <u>avoid</u> aliasing?

Run Lab 1 – Known Wave Aliasing





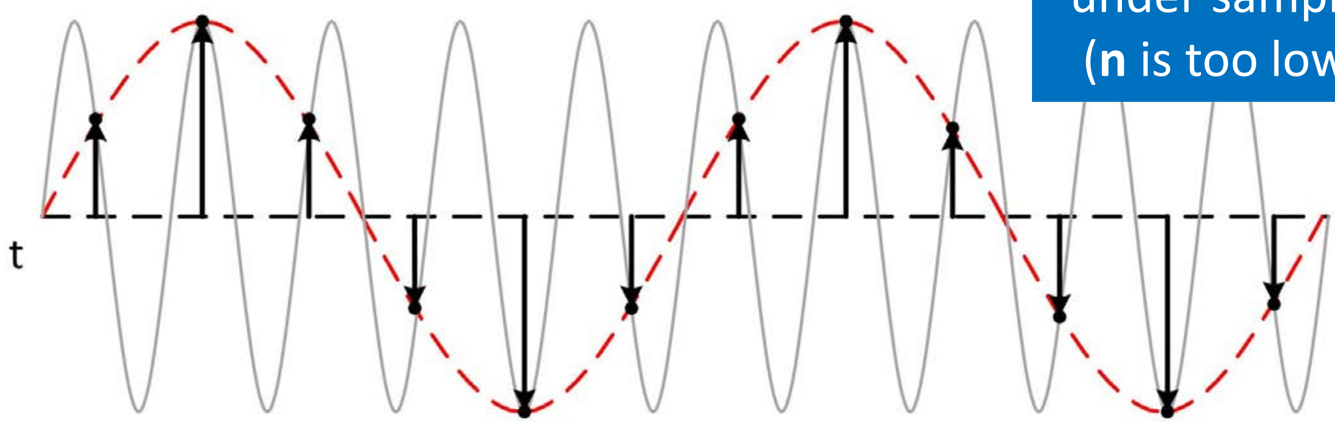
At N=32 the graph is still coarse but at least it now reaches its full ±1 range



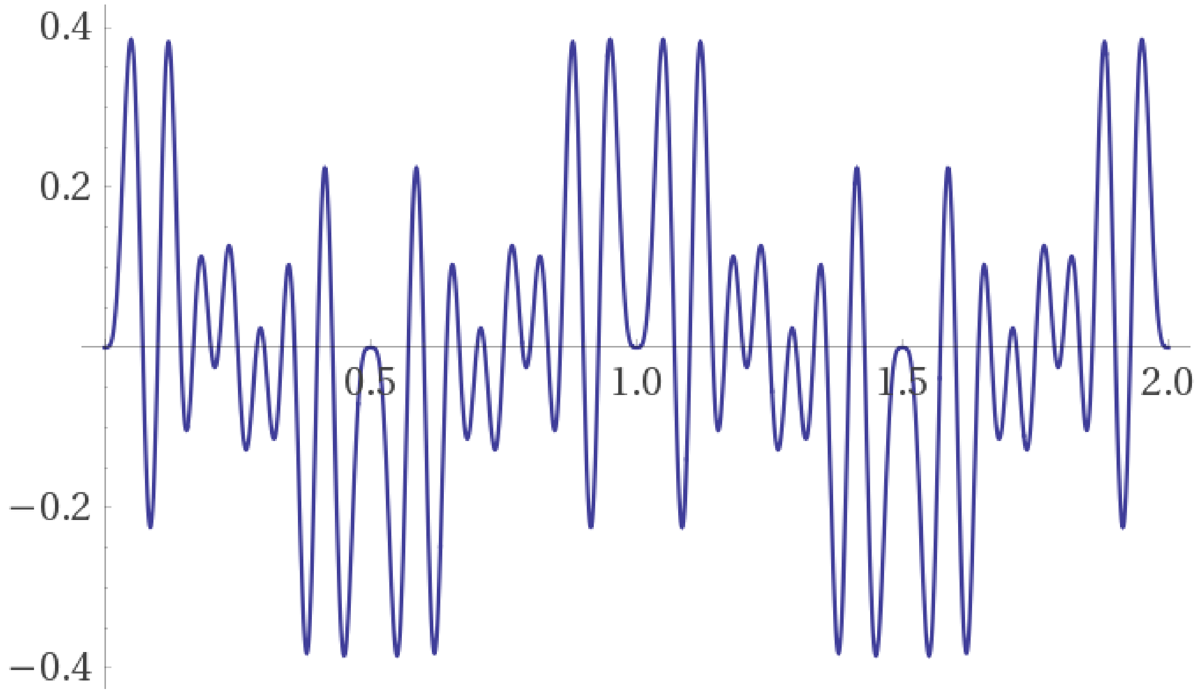
Nyquist Sampling Theorem

- To minimize aliasing (data loss), **<u>if you know λ ahead of time</u>**, set $\boldsymbol{n} = \left(\frac{2}{\lambda}\right)(b-a)(\boldsymbol{2r}), where\ \boldsymbol{r} \in \mathbb{Z}^{+}$
- This rule is the **Nyquist Sampling Theorem**
- You need at least **2x** as many samples as the **highest frequency** you intend to capture

This wave is under sampled (**n** is too low!)



Unknown Wave Aliasing

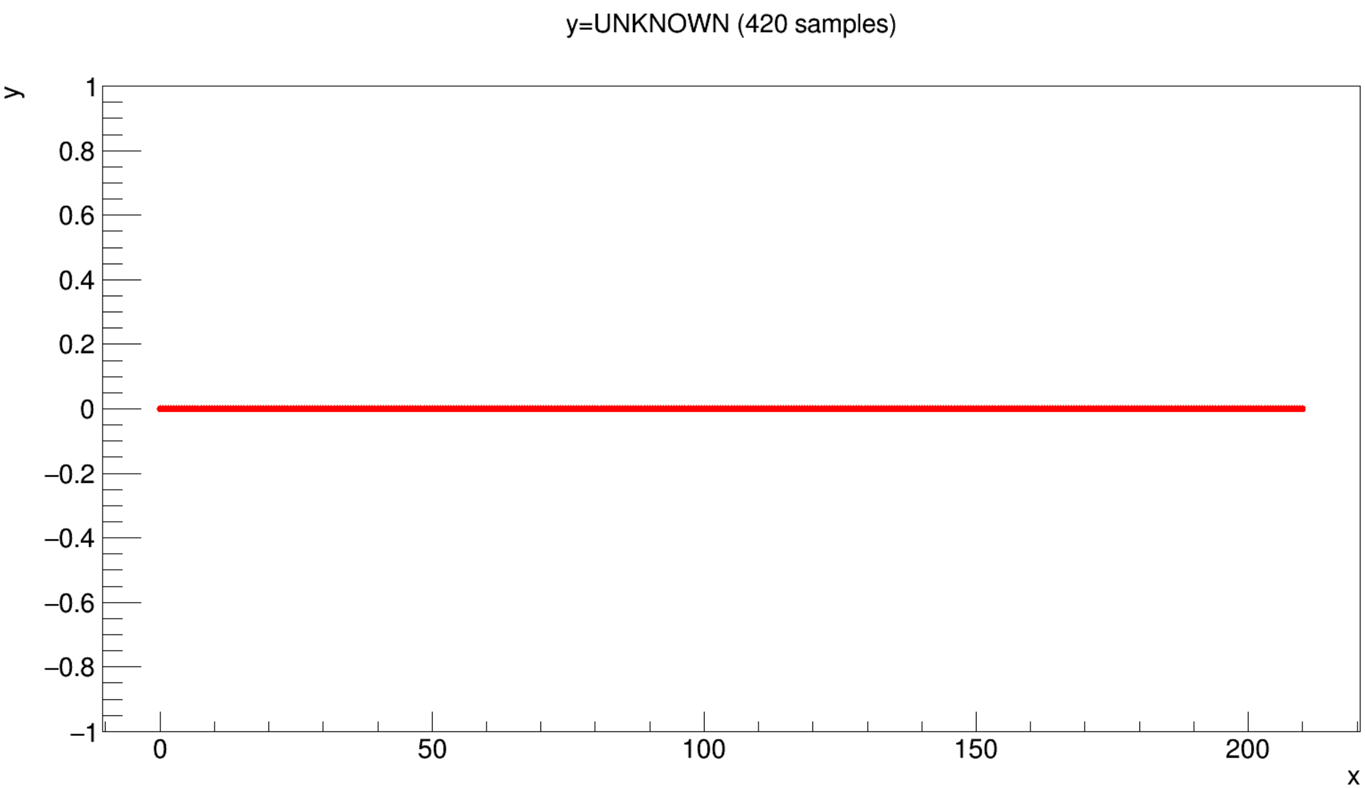


- Imagine you want to sample this wave over a period of $b = 210$ seconds (the average length of a radio song)
- How many intervals ($\boldsymbol{n}$) would you chose?

Sampling **Unknown** Waveforms

```
const double b = 210;
const int n = 420;
```

Run Lab 2



Sampling **Unknown** Waveforms

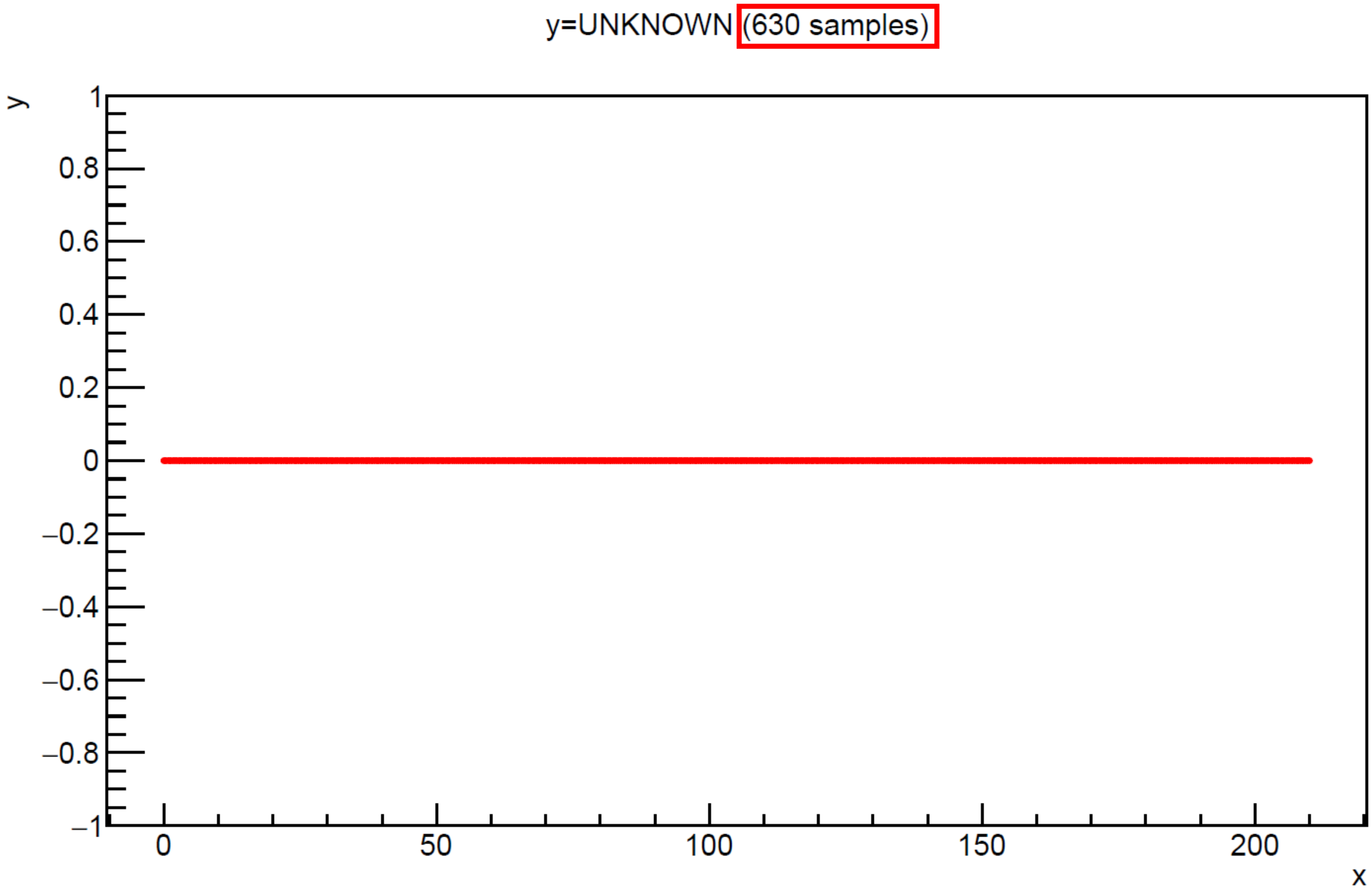
Try these values for *n*

10	63	280
12	70	294
14	75	300
15	84	315
18	90	350
20	98	420
21	100	490
24	105	525
25	120	588
28	126	630
30	140	700
35	147	735
36	150	840
40	168	980
42	175	1050
45	180	1260
49	196	1470
50	210	2100
56	245	2940
60	252	

```
// SinusoidAliasing.cpp

#include "stdafx.h"

using namespace std;

const double b = 210;
const int n = 630;

vector<double> x(n + 1);
vector<double> y(n + 1);

```

Edit Lab 2

Run Lab 2 - Sampling Unknown Waveforms



Edit Lab 2 - Sampling Unknown Waveforms

Try these values for n

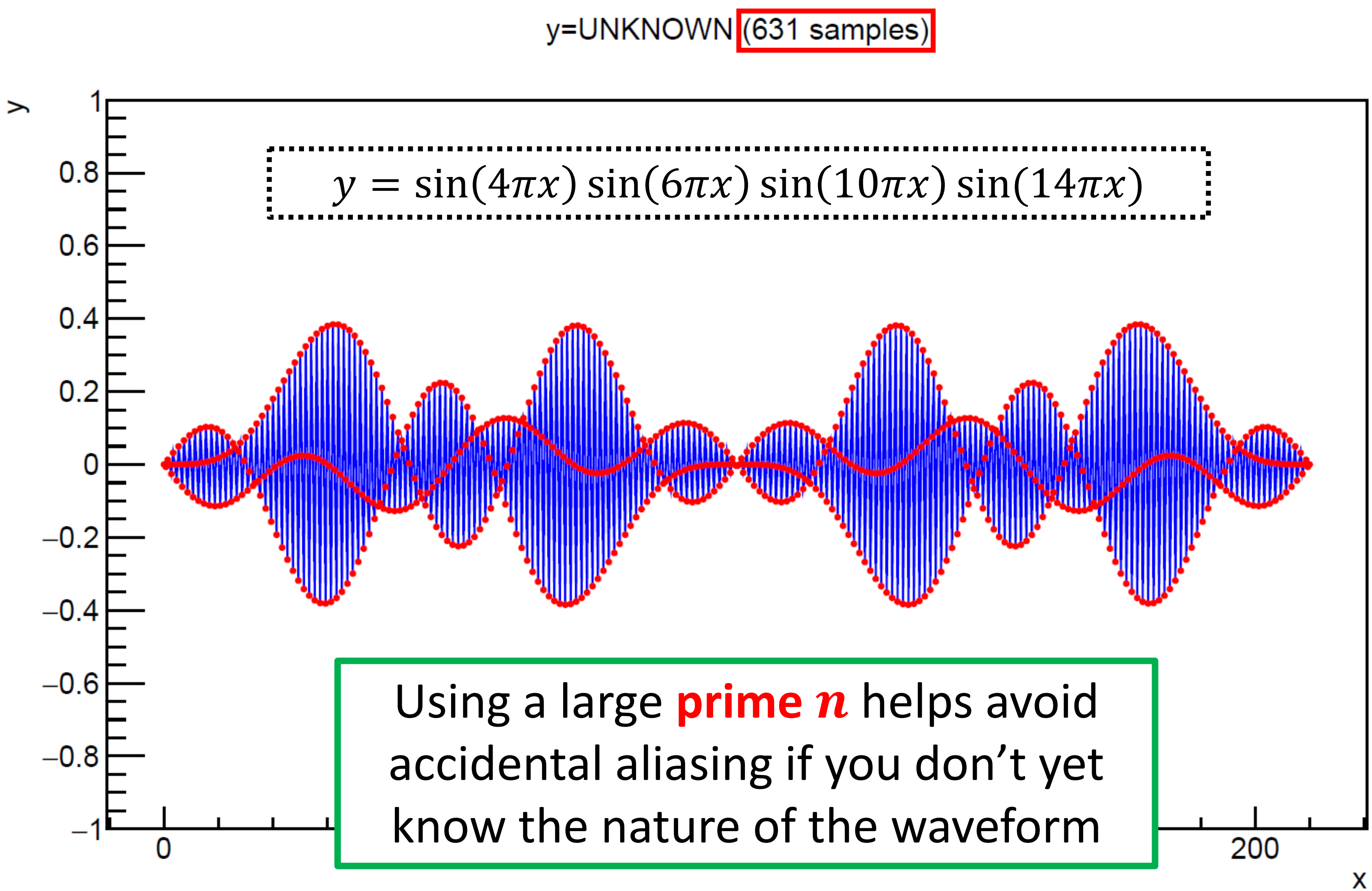
10	63	280
12	70	294
14	75	300
15	84	315
18	90	350
20	98	420
21	100	490
24	105	525
25	120	588
28	126	630
30	140	700
35	147	735
36	150	840
40	168	980
42	175	1050
45	180	1260
49	196	1470
50	210	2100
56	245	2940
60	252	

- What about $n = \boxed{631?}$

Edit Lab 2

```
// SinusoidAliasing.cpp

#include "stdafx.h"

using namespace std;

const double b = 210;
const int n = 631;
```

Run Lab 2 - Sampling Unknown Waveforms



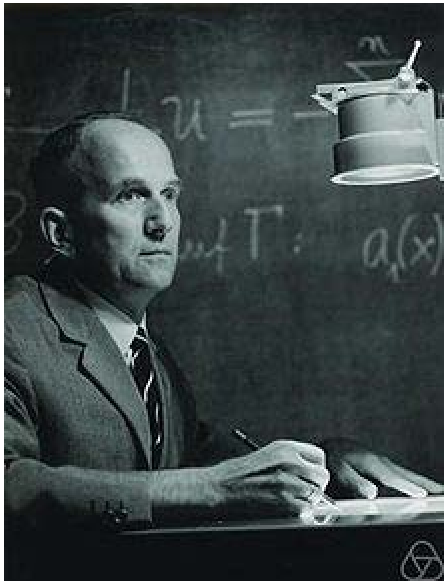
Using a large **prime n** helps avoid accidental aliasing if you don't yet know the nature of the waveform

Research Question

- If you don't know ahead of time the $\boldsymbol{\lambda}$ of a sampled sinusoid, then to minimize the chance of aliasing, ***is it best*** to set $\boldsymbol{n}$ to a **prime number** $\geq 2 \times b$? **Why or why not?**
- What happens if by pure bad luck your sampling rate and the frequency of any of the constituent fundamental harmonics of the sampled wave are ***not*** coprime (GCD > 1)?
- What happens if your sampling rate aligns somehow exactly to the oscillatory period of the sampled wave?
- What can you do to ensure the GCD of two numbers has a higher probability of being equal to one (==1) ?

The Collatz Conjecture

- First proposed in **1945** by Lothar Collatz, it remains **unsolved**
- Take any positive integer **n**
  - If n is **even**, divide it by **2**
  - If n is **odd**, multiply it by **3** and add **1**
  - Repeat the process until **n** reaches **1**
- The conjecture is that ***no matter what integer you start with*** the process will always reach **1**
  - It is maddening simple to state, but no one has been able to prove this claim is *either* true or false!



What is the sequence for 26 ?

The Collatz Conjecture

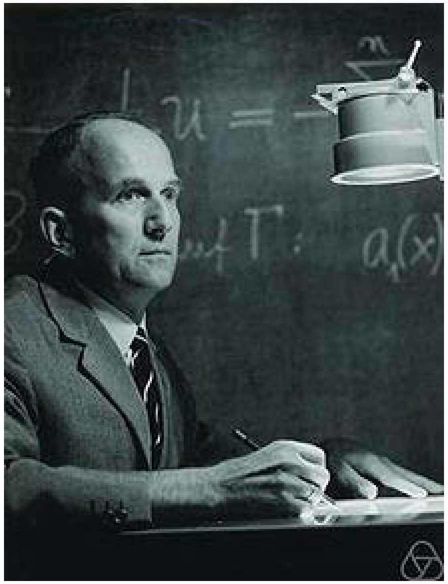
If n is **even**, divide it by **2**

If n is **odd**, multiply it by **3** and add **1**

Repeat the process until **n** reaches **1**

26
13
40
20
10
5
16
8
4
2
1

26 took **10** steps to reach 1



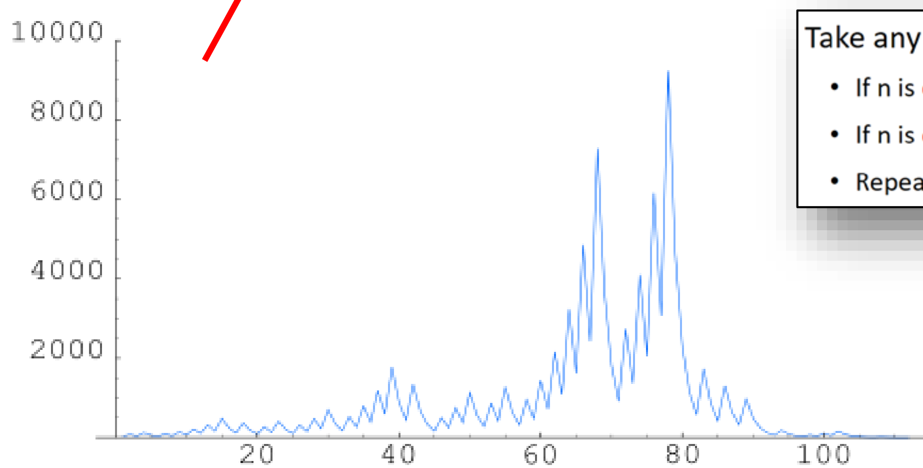
What is the sequence for 26 ?

What is the sequence for 27 ?

The Collatz Conjecture

The sequence for *n* = 27, listed and graphed below, takes 111 steps (41 steps through **odd numbers**), climbing to 9232 before descending to 1.

27, 82, **41**, 124, 62, **31**, 94, **47**, 142, **71**, 214, **107**, 322, **161**, 484, 242, **121**, 364, 182, **91**, 274, **137**, 412, 206, **103**, 310, **155**, 466, **233**, 700, 350, **175**, 526, **263**, 790, **395**, 1186, **593**, 1780, 890, **445**, 1336, 668, 334, **167**, 502, **251**, 754, **377**, 1132, 566, **283**, 850, **425**, 1276, 638, **319**, 958, **479**, 1438, **719**, 2158, **1079**, 3238, **1619**, 4858, **2429**, 7288, 3644, 1822, **911**, 2734, **1367**, 4102, **2051**, 6154, **3077**, 9232, 4616, 2308, 1154, **577**, 1732, 866, **433**, 1300, 650, **325**, 976, 488, 244, 122, **61**, 184, 92, 46, **23**, 70, **35**, 106, **53**, 160, 80, 40, 20, 10, **5**, 16, 8, 4, 2, **1** (sequence A008884 in the OEIS)

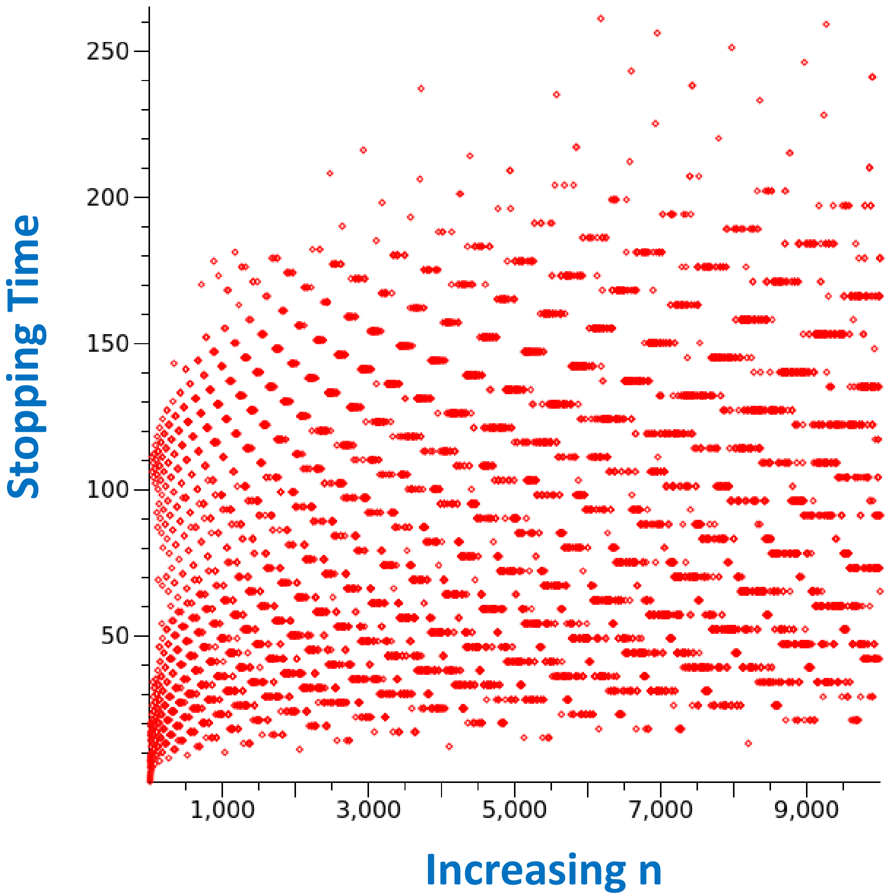


Take any positive integer **n**

- If n is **even**, divide it by **2**
- If n is **odd**, multiply it by **3** and add **1**
- Repeat the process until **n** reaches **1**

Stopping Time

- The **stopping time** (for a given integer **n**) is the **total number of Collatz iterations** before reaching **1**
- This graph shows the stopping times for **n** < **10,000**
- Stopping times appear to exhibit a rough pattern, but ***we have no formula*** to accurately predict it for any **n**



Frequency of Stopping Times

- We are interested in analyzing the *frequency* (count) of each stopping time for n **< 1,000,000**
  - What is the overall *shape* of the distribution of stopping times?
  - We will use a **histogram** to display the **count** of each stopping time
- Your task is to implement the stopping time function:

> Take any positive integer **n**
>
> - If n is **even**, divide it by **2**
> - If n is **odd**, multiply it by **3** and add **1**
> - Repeat the process until **n** reaches **1**

Open Lab 3 – Collatz Conjecture

```
int main()
{
    const uint64_t limit = (int)1e6;
    string title = "Collatz Conjecture (n #LT " + to_string(limit) + ")";
    TApplication* theApp = new TApplication(title.c_str(), nullptr, nullptr);

    TCanvas* c1 = new TCanvas(title.c_str());
    c1->SetTitle(title.c_str());

    TH1I* h1 = new TH1I(nullptr, title.c_str(), 500, 0, 501);
    h1->SetStats(kFALSE);

    TAxis* ya = h1->GetYaxis();
    ya->SetTitle("Count");
    ya->CenterTitle();

    TAxis* xa = h1->GetXaxis();
    xa->SetTitle("Stopping Time");
    xa->CenterTitle();
    xa->SetTickSize(0);

    // Fill the histogram
    for (uint64_t n{ 1 }; n < limit; n++)
        h1->Fill((int)CalcStopTime(n));

    h1->SetFillColor(kBlue);
    h1->Draw("B");

    theApp->Run();
    return 0;
}
```


```
int CalcStopTime(uint64_t n)
{
    int stopTime = 0;

    // TODO:  Insert your code here

    return stopTime;
}
```

Edit Lab 3

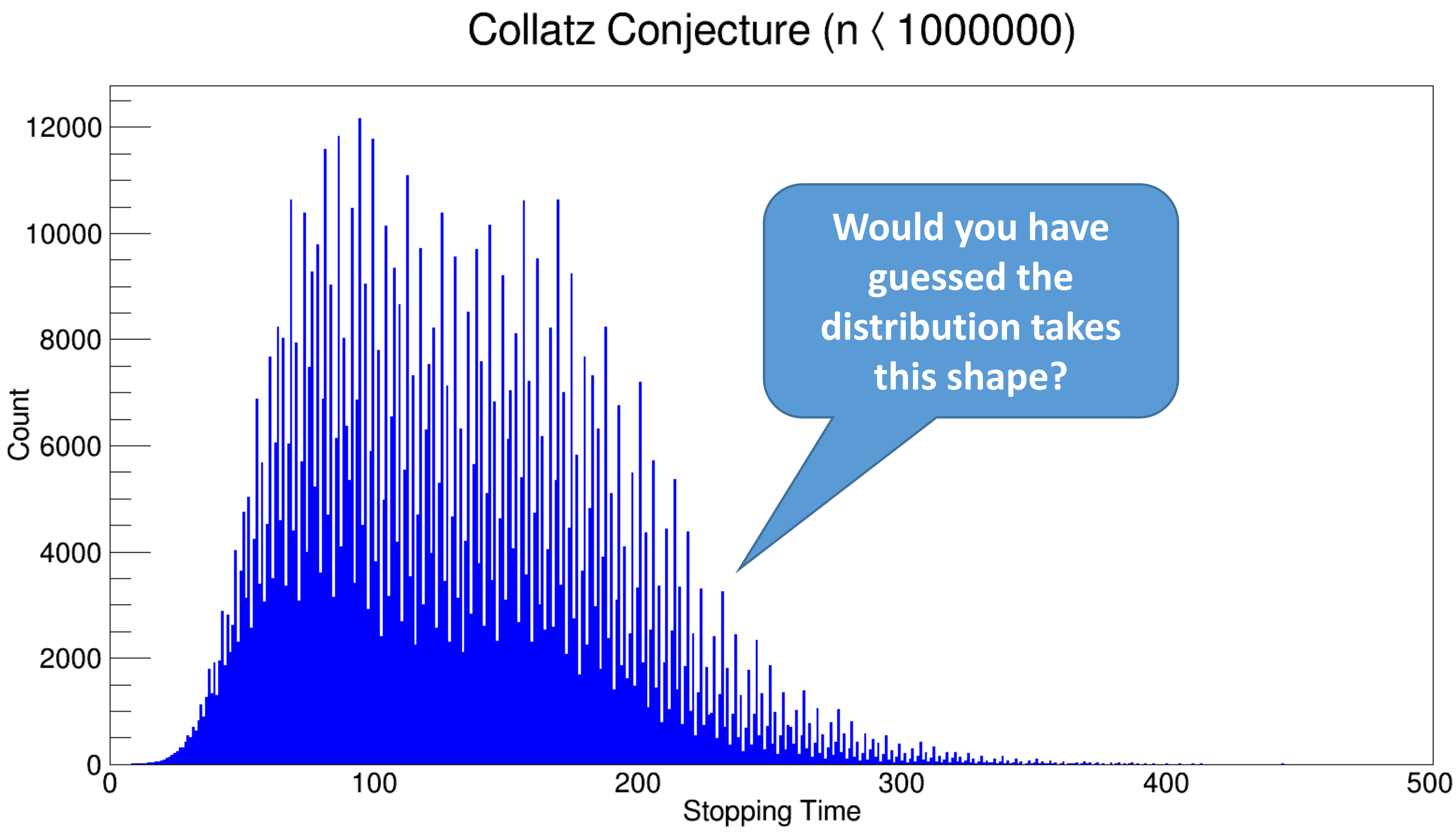
Edit Lab 3 – Collatz Conjecture

```
int CalcStopTime(uint64_t n)
{
    int stopTime = 0;
    while (n != 1)
    {
        if (n % 2 == 0)
            n = n / 2;
        else
            n = 3 * n + 1;
        stopTime++;
    }
    return stopTime;
}
```

Take any positive integer **n**

- If n is **even**, divide it by **2**
- If n is **odd**, multiply it by **3** and add **1**
- Repeat the process until **n** reaches **1**

Stopping Time Histogram



Now you know…

- **CERN ROOT**
  - ROOT is a set of open-source C++ libraries developed by a dedicated team of scientists at **CERN**
  - ROOT is heavily used in many laboratories around the world
  - Most science posters that show **complicated** graphs use **ROOT**
- **Nyquist Sampling Theorem** – to minimize aliasing (data loss):
  - Known wave (you know $\boldsymbol{\lambda}$) set $\boldsymbol{n} = \frac{b}{\lambda}\boldsymbol{r}, where\ \boldsymbol{r} \in \mathbb{Z}^{+} and\ \boldsymbol{r} \geq \mathbf{2}$
  - That you need **at least 2x** as many samples as the highest frequency you want to capture is the essence of the Nyquist Theorem
  - Unknown wave (you don't know $\boldsymbol{\lambda}$) set $\boldsymbol{n}$ to a **prime number** $\geq 2 \times b$ **so as to minimize the chance of aliasing**

Now you know…

- The **Collatz Conjecture** is yet another unsolved problem in number theory which is so easy to state, but so hard to prove
- We know the shape of the distribution changes significantly as you analyze higher values of **n** – it approaches an **exponential** curve but we don't know the power
- This is an excellent example of **experimental computation mathematics** – the computer can find the stopping time for a million integers in just a few seconds
- After extensive calculations, scientists search for patterns in the data, ultimately trying to find **analytic expressions** that describe the underlying natural law