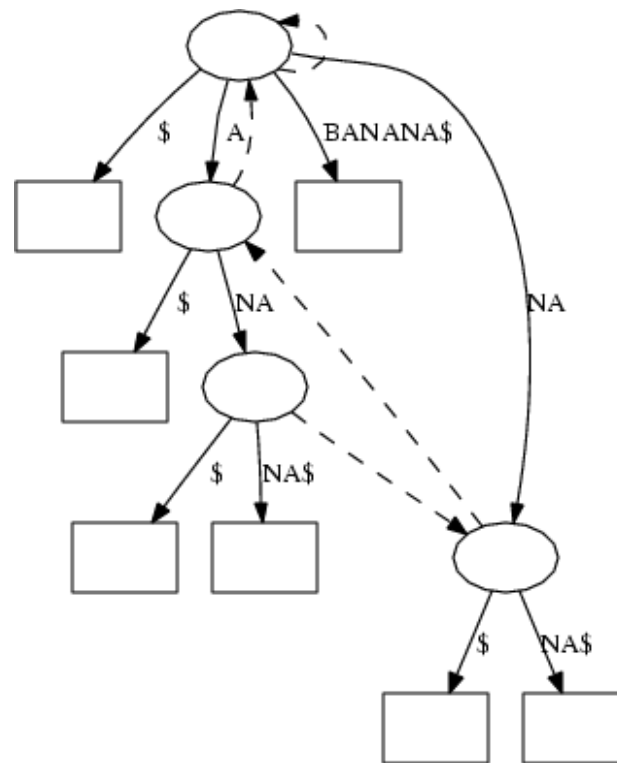


# Further Mathematics and Algorithms

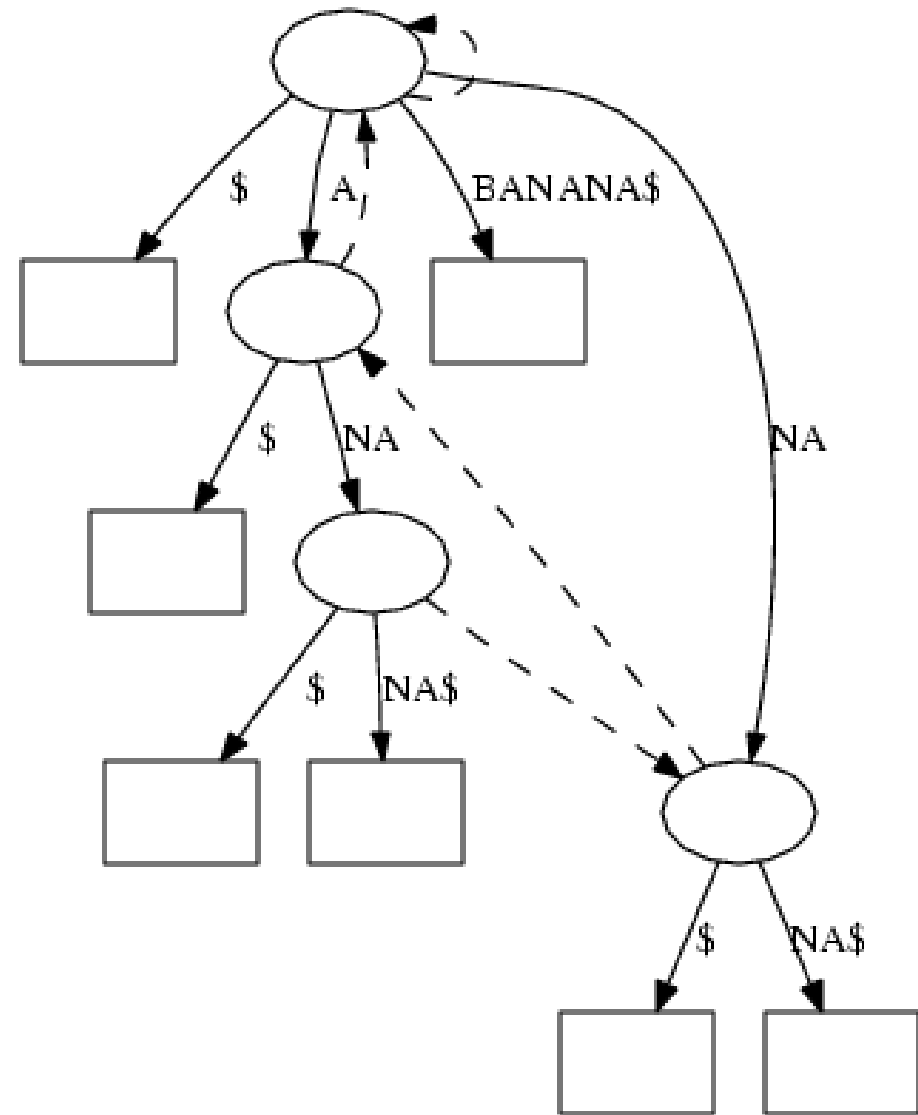
## Lesson 1: *Use Data Structures and Algorithms!*



*Course structure, examples of data structures and algorithms*

# Outline

1. **Course structure**
2. Example of Using DSA
3. Sophisticated Program
4. State-of-the-Art



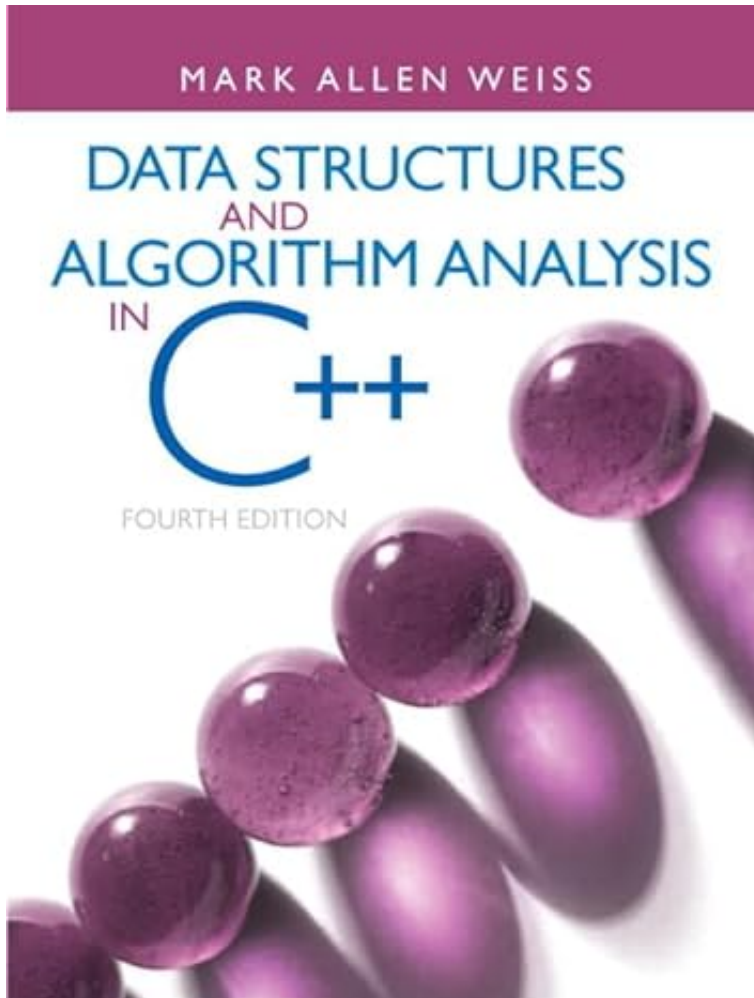
# Welcome to Further Mathematics and Algorithms

- First 7 weeks Daniela and I will be teaching you about algorithms■
- The last 4 weeks you will learn some further maths■
- I'm teaching you algorithms (and data structures) in C++■
- My ambition is not only to teach you data structures and algorithms academically■, but also to get to a new level of coding in C++■

# Quick Survey on C++

- Who considers themselves a competent coder in C++?■
- Who knows what a class constructor is?■
- Who knows what a default constructor is?■
- Who is happy using templates?■
- Who understands pointers?■
- Who knows what the key word `explicit` means?■
- Who has heard of *resource acquisition is initialisation (RAII)*?■

# Recommended Course Text



- *Data Structures and Algorithm Analysis in C++* by M. A. Weiss■
  - ★ Best introduction to Data Structures and Algorithms■
  - ★ Not huge, but covers all the basics■
- Available in the library■

# What is a Data Structure?

*any of various methods of organising data items (as records) in a computer*



- Container for data
- E.g. sets, stacks, lists, trees, graphs
- Clean interface, e.g. push, pop, delete
- Usually designed for fast or convenient access

# What is an Algorithm?

*a sequence of unambiguous instructions for solving a problem, i.e. for obtaining a required output for a legitimate input in a finite amount of time*



- E.g. sort, search, match
- Well defined and generic
- Guarantees on performance

# Exemplary OO-Software

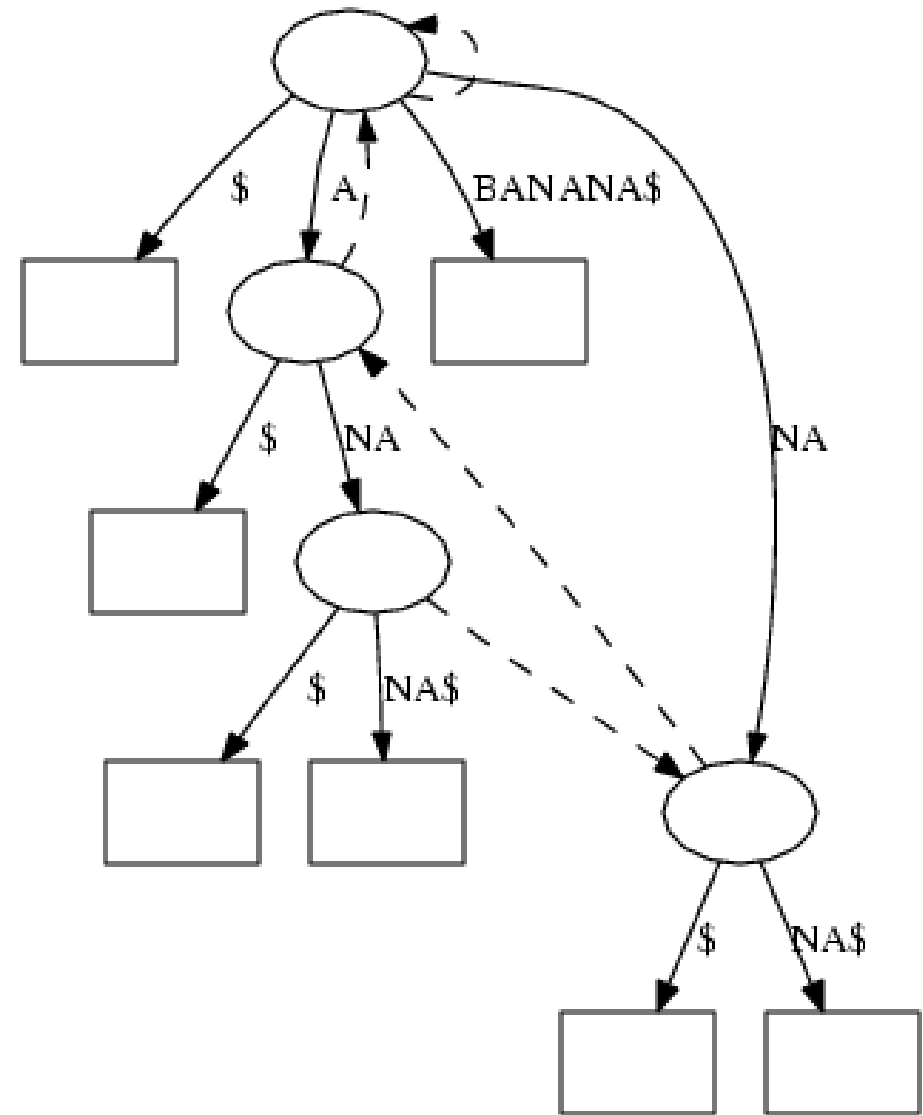
- Abstraction from details of problem■
- Declaration of intention■
- Clean interfaces■
- Hidden implementations■
- Makes programs readable and maintainable■
- Reuse code■—don't even have to write it yourself■

*Thou shall not re-implement common data structures■*



# Outline

1. Course structure
2. **Example of Using DSA**
3. Sophisticated Program
4. State-of-the-Art



# Example: Sort program

- Suppose we want to write a program to
  - ★ read an input file of integers
  - ★ sort the integers
  - ★ write a list of integers to standard out
- In Unix there is a command called `sort` which does just this
- Note that you don't know the number of inputs

# Code for sort

```
#include <iostream>
#include <fstream>

int main(int argc, char** argv) {
    std::ifstream myfile(argv[1]);

    int array_size = 10;
    int* array = new int[array_size];
    int cnt = 0;
    while(myfile.good()) {
        if (cnt==array_size) {
            int* new_array = new int[2*array_size];
            for(int i=0; i<array_size; ++i)
                new_array[i] = array[i];
            delete[] array;
            array = new_array;
            array_size *= 2;
        }
        myfile >> array[cnt++];
    }
}
```

```
for(int i=0; i<cnt; ++i) {  
    int index = 0;  
    for(int j=1; j<cnt-i; ++j) {  
        if (array[j]<array[index])  
            index = j;  
    }  
    std::cout << array[index] << std::endl;  
    array[index] = array[cnt-i-1];  
}
```

}

# Notes on Code

- Details of code don't matter■
- Simple program ( $\sim 20$  lines of code)■
- Uses a simple array■
- Difficult to see what is going on■
- On 100 000 inputs it takes 10 seconds to run■

# Using Data Structures and algorithms

```
#include <iostream>
#include <fstream>
#include <iterator>
#include <vector>
#include <algorithm>
using namespace std;

int main(int argc, char *argv[])
{
    ifstream in(argv[1]);
    vector<int> data;
    copy(istream_iterator<int>(in), istream_iterator<int>(),
        back_inserter(data));
    sort(data.begin(), data.end());
    copy(data.begin(), data.end(), ostream_iterator<int>(cout, "\n"));
}
```



# Sorting Doubles

```
#include <iostream>
#include <fstream>
#include <iterator>
#include <vector>
#include <algorithm>
using namespace std;

int main(int argc, char *argv[])
{
    ifstream in(argv[1]);
    vector<double> data;
    copy(istream_iterator<double>(in), istream_iterator<double>(),
        back_inserter(data));
    sort(data.begin(), data.end());
    copy(data.begin(), data.end(), ostream_iterator<double>(cout, "\n"));
}
```



# Notes on C++

- `vector<int>` is the C++ standard resizable array
- input/output is treated as a copy
- Code is easy to read
  - ★ Declare `vector<int>` or `vector<double>`
  - ★ copy input file into vector
  - ★ sort vector
  - ★ copy sorted vector to standard output stream
- On 100 000 inputs takes 10ms to run



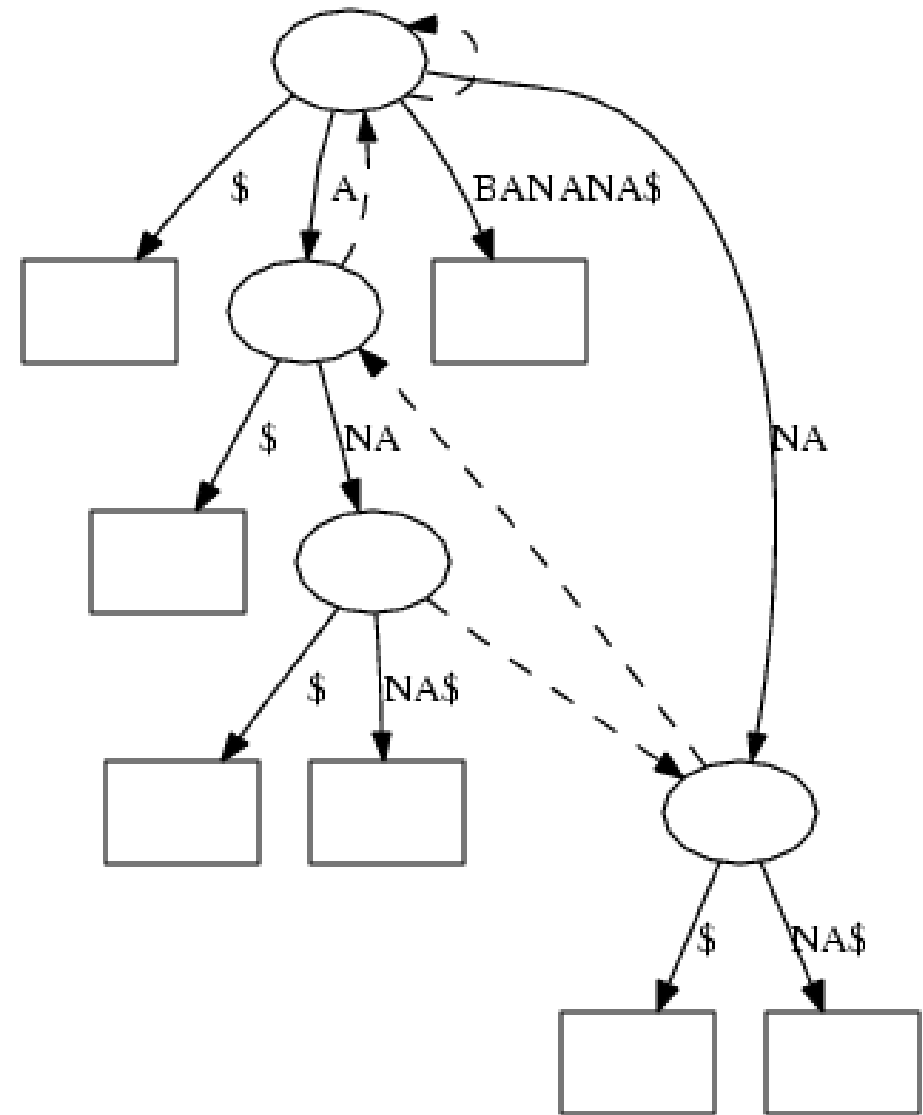
# Summary: Why use Data Structures?

Data structure version is

- Easier/quicker to code■
- More readable (less bugs)■
- Easier to modify and change■
- Easier to port to another language■
- Better (in this case faster)■

# Outline

1. Course structure
2. Example of Using DSA
3. **Sophisticated Program**
4. State-of-the-Art



# Sophisticated Programs

- Data structures and algorithms allow moderately competent programmers to write some very impressive programs■
- E.g. consider a program to count all occurrences of words in a document■
- We want to output the words in sorted order■

# countWords

```
#include <stuff>

int main(int argc, char** argv) {
    ifstream in(argv[1]);
    map<string, int> words;

    string s;
    while(in >> s) {
        ++words[s];
    }

    vector<pair<string,int> > pairs;
    copy(words.begin(), words.end(), back_inserter(pairs));
    sort(pairs.begin(), pairs.end(),
        [](auto& a, auto&b){return a.second>b.second;});

    for(auto w=pairs.begin(); w!=pairs.end(); ++w) {
        cout << w->first << "_occurs_" << w->second << "_times\n";
    }
}
```

# Using countWords

```
> countWords text.dat | more
```

```
the occurs 97 times
```

```
of occurs 96 times
```

```
to occurs 57 times
```

```
and occurs 42 times
```

```
a occurs 36 times
```

```
be occurs 31 times
```

```
will occurs 26 times
```

```
we occurs 23 times
```

```
that occurs 23 times
```

```
is occurs 21 times
```

```
have occurs 19 times
```

```
freedom occurs 18 times
```

# Programming Challenge

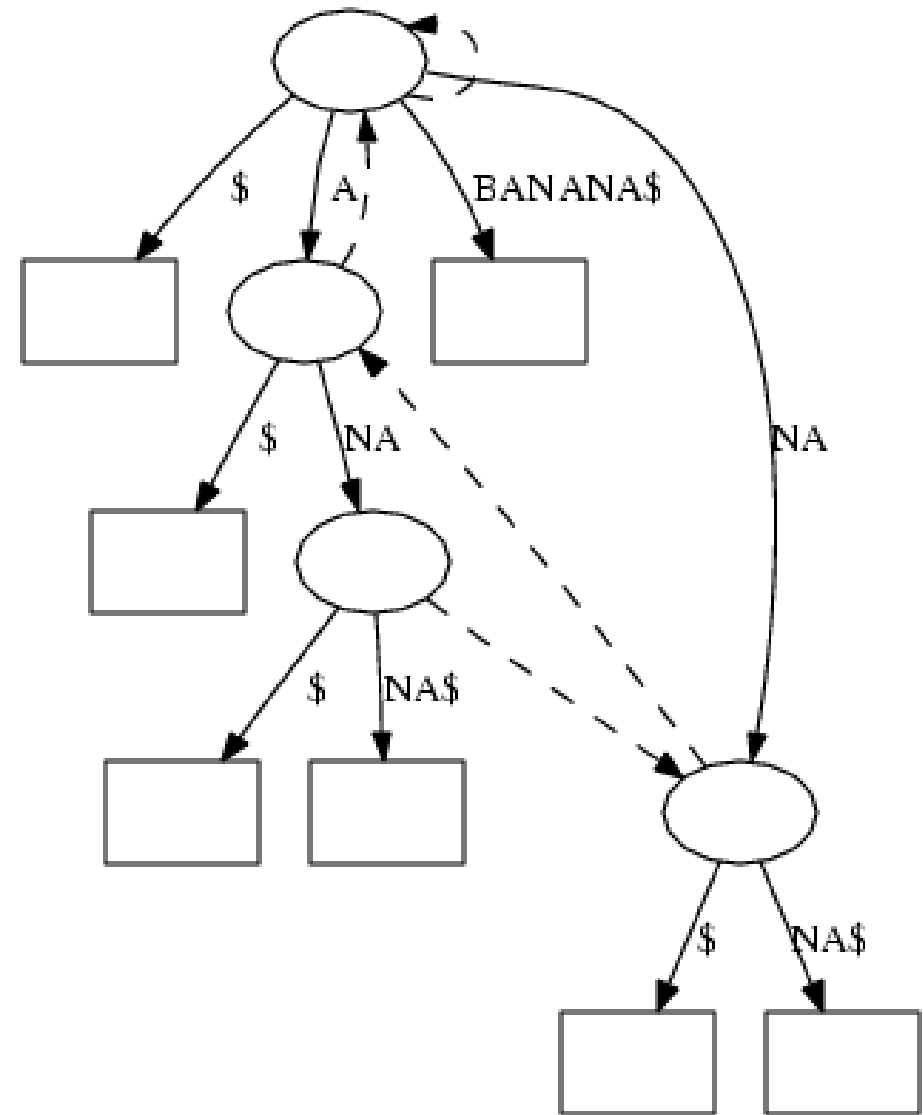
- Run on “I have a dream” speech with 1550 words in 0.02 seconds■
- Challenge for good programmers

*Write a program without use data structures in less than 10 times as much code that runs in less than 10 times as long*

- 
- Probably possible, but certainly not easy■—almost certainly take you 10 times longer to code■

# Outline

1. Course structure
2. Example of Using DSA
3. Sophisticated Program
4. **State-of-the-Art**



# DNA Sequencing

- In modern whole shotgun genome sequencing the full genome is broken into small pieces■
- The pieces are then read by a sequencing machine■
- This reads short sections (around 1000) bases■
- The reads are then assembled to construct the full genome■



# Sequencing and Assembly

A T A C C A C C A T G C C T C C T T G C T C C A A T A T T A A T T C A A G G C G  
A T A C C A C C A T G C C T C C T T G C T C C A A T A T T A A T T C A A G G C G  
A T A C C A C C A T G C C T C C T T G C T C C A A T A T T A A T T C A A G G C G  
A T A C C A C C A T G C C T C C T T G C T C C A A T A T T A A T T C A A G G C G  
A T A C C A C C A T G C C T C C T T G C T C C A A T A T T A A T T C A A G G C G

A T A C C A C C A T G C C T C C T T G C T C C A A T A T T A A T T C A A G G C G

TTGCT	TACCA	CAAGG	TTAAT	TTCAA	TCCTT
TGCCT	AATAT	CCTCC	AGGCG	ATACC	ACCAT
CTCCT	CCTTG	ATGCC	GCTCC	TGCCT	TTGCT
TGCTC	ACCAC	TTGCT	AATAT	CAAGG	TGCCT
TACCA	CACCA	CCTTG	CTCCA	TATTA	AATTC

# New Generation Sequencers

- The estimated cost of sequencing the human genome in 2005 was \$10 000 000■
- To reduce the cost there was and is a drive to produce new sequencing machines■
- These tend to read much shorter sections of DNA (e.g. 20-100nt)■
- Can these be assembled?■

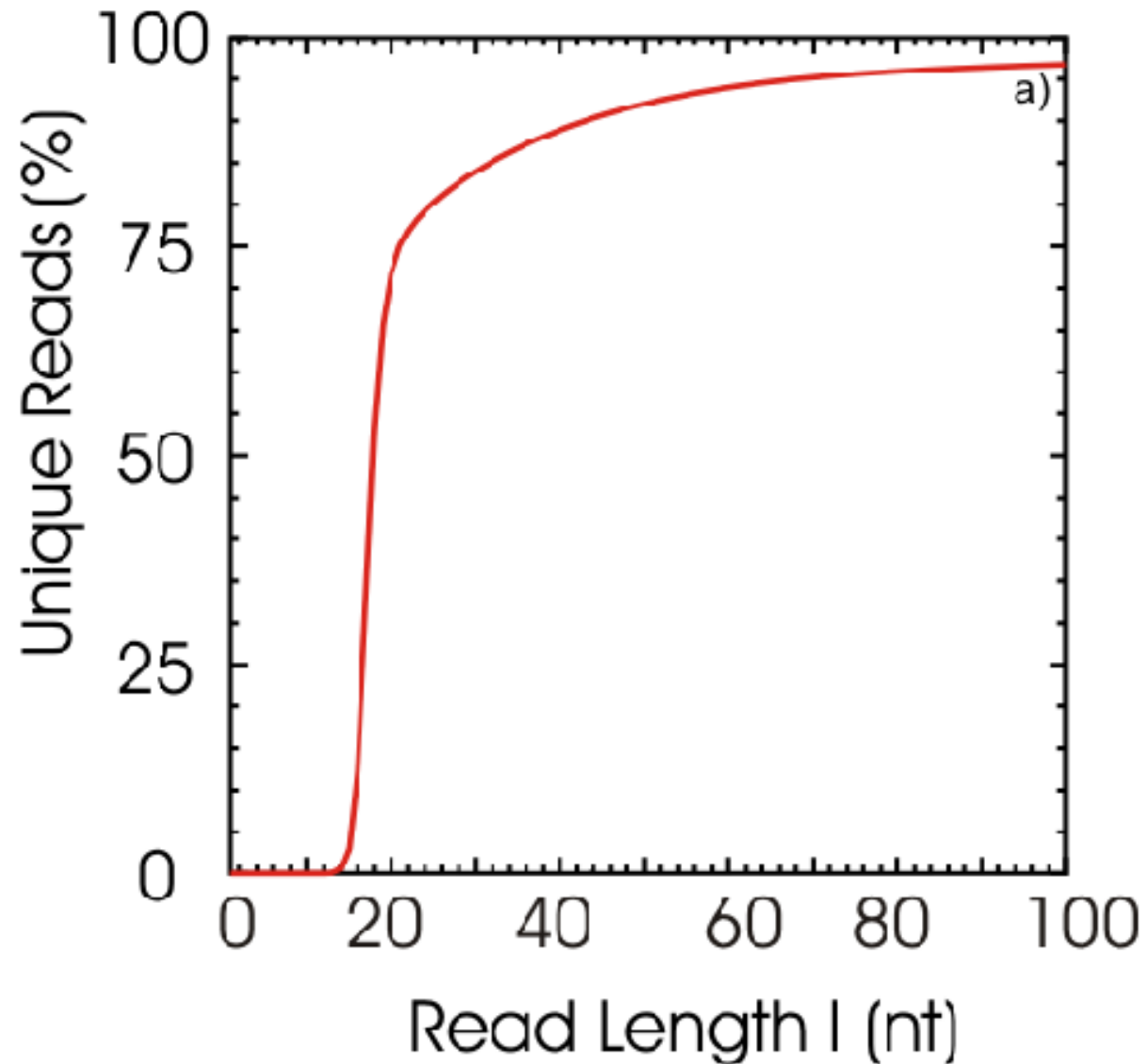
# Repeats

- The difficulty of assembly is caused by repeats■

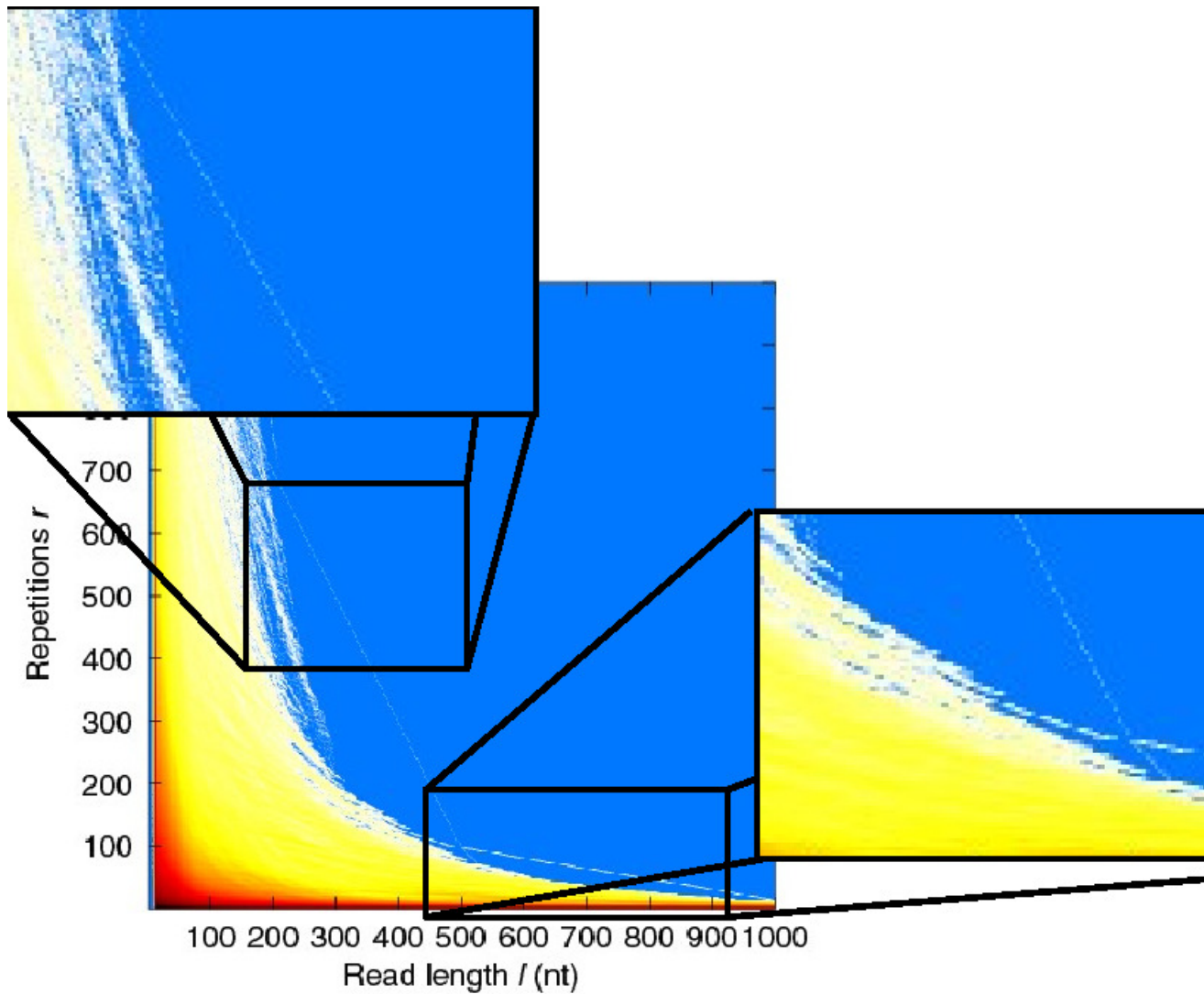
A T A C C A C C A T G C C T C C T T G C T C C A A T C C A C C A T C A A G G C G

- How many repeats are there in the human genome?■ (Incidentally the human genome is 3.2 billion base **pairs**)■
- This is an important question for developing new sequencing technologies■

# Repeats in Human Genome



# Repeats Structure



# Computing Repeats

- A naive program would take  $n^2$  operations where  $n = 6.4 \times 10^9$ ■
- If we used this we would still be waiting for the program to finish■
- Could not answer this question a few years ago■—not because computers weren't powerful, but because the algorithms had not been developed■
- Used state-of-the-art suffix arrays■
- Smart algorithms allow you to do things which you cannot do otherwise■

# To Use DSA You Need To

- Know what common data structures and algorithm do
- Understand the implementations enough to modify existing data structures to be fit for purpose
- Understand time/space complexity to select the right data structure or algorithm
- Understand software interfaces for DSA
- Be able to combine data structures
- The rest of this course teaches you these skills