SACRAMENTO STATE

CSc 28
Discrete Structures

Chapter 5
Math for Computer Science

Herbert G. Mayer, CSU CSC

Status 2/1/2021

Syllabus

- Mathematics
- Sequence
- Cauchy Sequence
- Arithmetic Series
- Geometric Series
- Summation
- Recursive Sets
- References

Math and CS

Math

- Merriam Webster: Mathematics is the science of numbers and their operations, interrelations, combinations, generalizations, and abstractions and of space configurations and their structure, measurement, transformations, and generalizations
- Mathematics is a science of numbers, quantity, computability, arrangement, and space
- Mathematics may be studied in its own right, i.e. pure mathematics
- Or as it applies to other disciplines such as physics and engineering, i.e. applied mathematics
- Some call Math ☺ an Experimental Science [5]

Computer Science

- Computer Science (CS) is the study of algorithms, computational machines, and computation itself
- As a discipline, CS spans a range of topics from theoretical studies of algorithms, computation, and information to the practical issues of implementing computational systems in HW and SW
- CS uses methods of thinking and problem solving as practices in the field of Mathematics
- But may be viewed as a field separate from Math
- CS is also the key subject of some Sac State students in Spring 2021 ☺

Computer Science

- In this chapter we discuss sequences, strings, summation formulae, arithmetic series, additions
- And topics than can be discussed in Mathematics as well as in the Computing Sciences

Sequence

Sequence

Dictionary definition:

Sequence is a particular order in which related events, movements, or things follow each other

Sequence (Source Wiki)

- **Informally: Sequence is an enumerated, ordered collection of elements; repetition being allowed and significant**
- **Like sets: a sequence contains members, AKA *elements*, *terms,* or *objects***
- **The number of elements (possibly infinite) is called the *length* of a sequence**
- **Unlike sets: Elements can appear multiple times at different positions in a sequence; order does matter!**
- **Formally: Sequence is a function whose domain is the enumerated set of natural numbers N (may be infinite), or of some other objects**
- **If finite, a sequence with n elements is said to have length *n***

Sequence

- The position of an element in a sequence is its *rank* or *index*
- It is a matter of language convention, whether the first element has index 0 or 1
- The n^{th} element of a sequence is denoted by a position with *n* as the subscript; for example, the n^{th} element of the Fibonacci Sequence is denoted: F_n
- And sequence name1 = { M, A, R, Y } is a sequence of letters with the letter M as the first and Y as the last element
- Somehow we must express that the object of interest be a sequence, not a set or other entity!

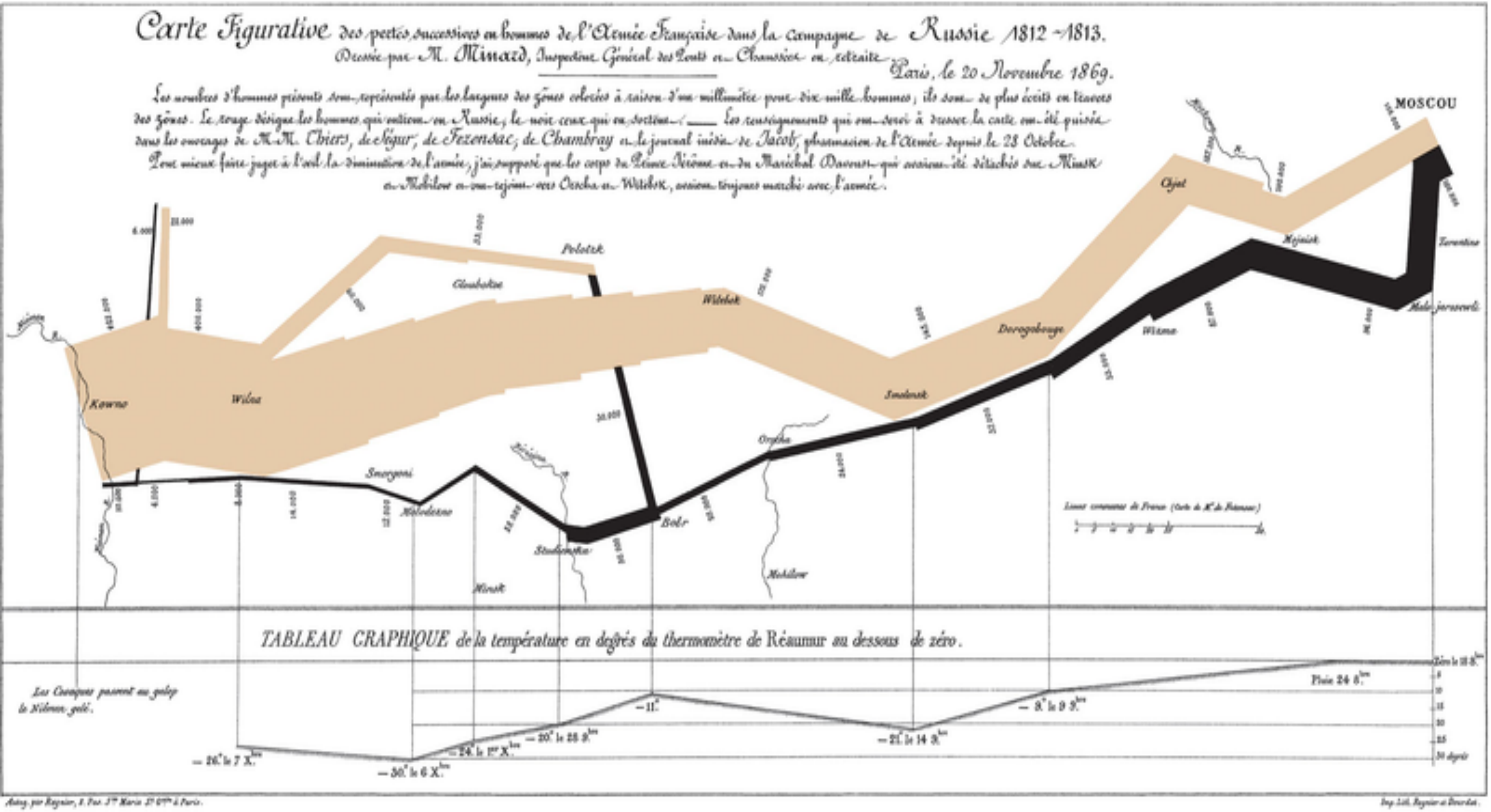
Sequence

- Sequence name1 differs from another sequence name2 = { A, R, M, Y }, as order matters!
- Also the anonymous sequence { 1, 2, 3, 1, 5, 8 }, which contains element 1 more than once, clearly at different positions, is a valid sequence
- Sequences can be finite; some examples here show
- Or infinite, e.g. the sequence of even positive integers { 2, 4, 6, ... ∞ }
- In CS, finite sequences are sometimes called strings, words, or lists
    - different names commonly correspond to different ways to represent them in data structures

Sequence

- Infinite sequences are AKA streams
- The empty sequence { } is included in most representations for sequence
- Specify a priori, whether empty is included or not!

Another Sequence



Battle Sequence in Napoleon’s Russia Campaign

Sequence

- Sequences represent ordered lists of elements –as opposed to sets, which are unordered
- A sequence can be defined as a function from a subset of N to a set S, or a mapping of N → S
- Conventional to use notation a_n for the image of the n^{th} sequence element; a_n is some term of the sequence, for example at position n

Example:

S subset of N: 1 2 3 4 5 index of even #

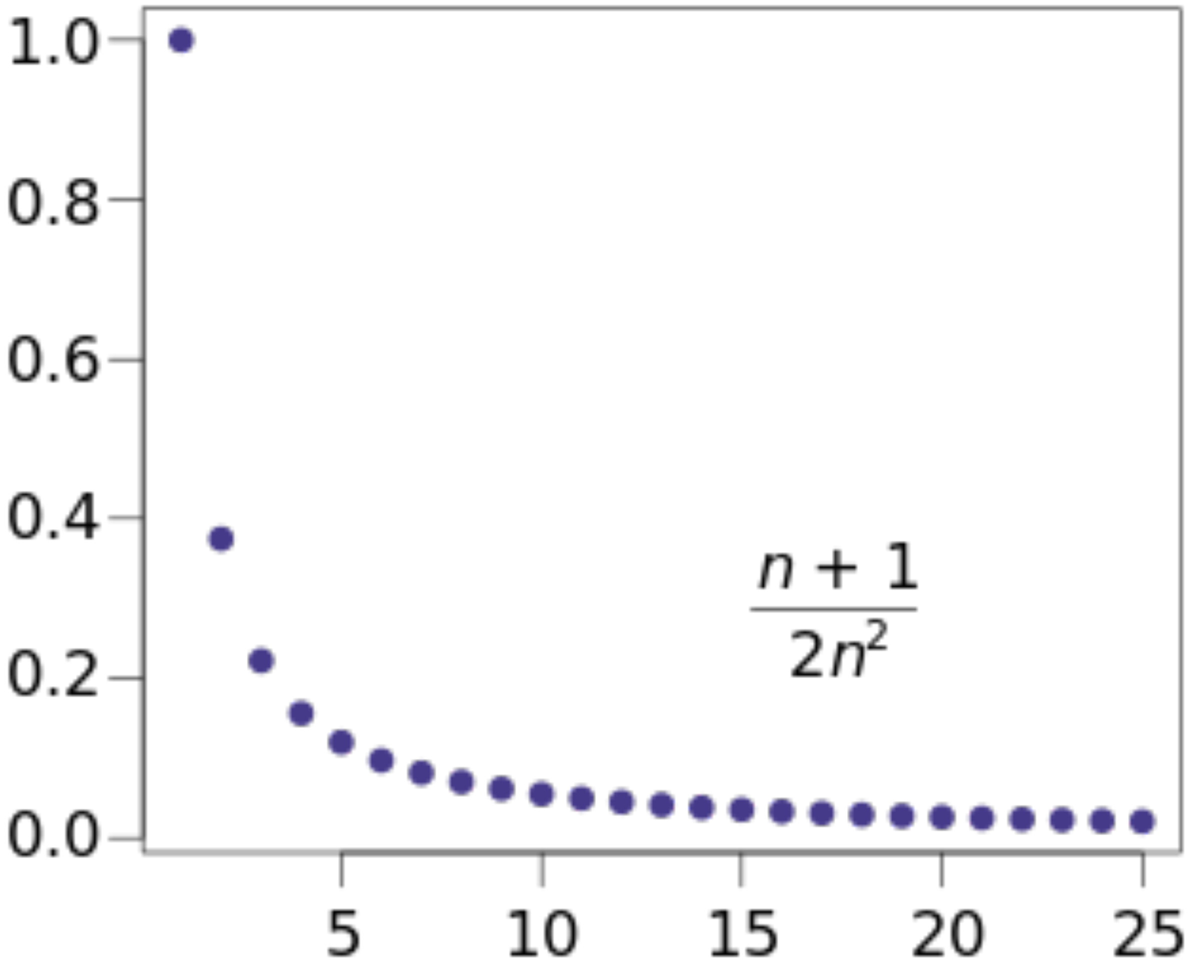
S: 2 4 6 8 10 S: first 5 even int

Sequence

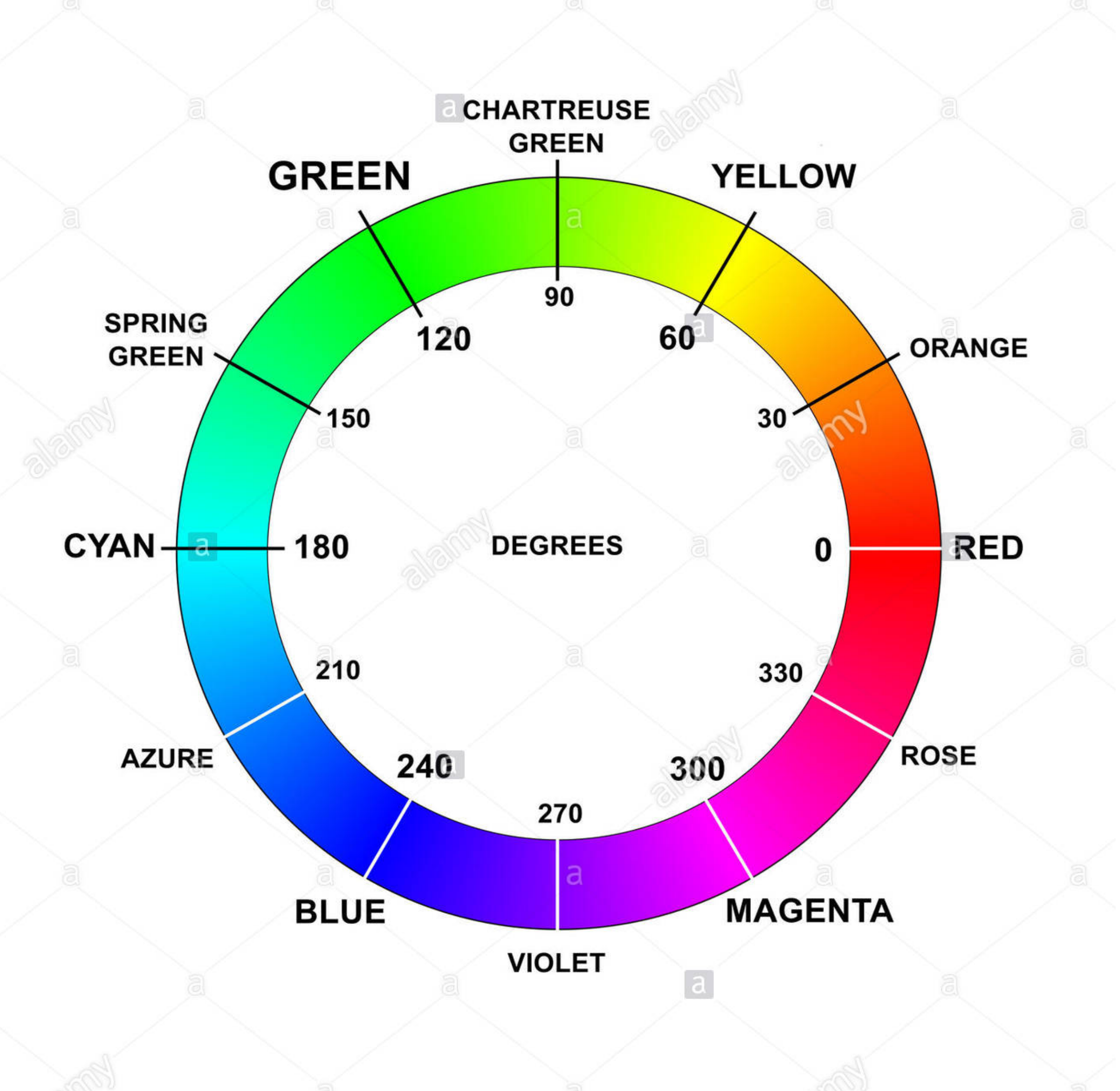
- **Conventional to use notation $\{ a_n \}$ to describe a sequence**
- Important: 1. Symbols { } not to be confused with the { } symbols used in set notation. 2. In sequences repetition is allowed, and order matters; in sets repetition is not permitted –or else duplicates are simply ignored
- **Convenient to describe a sequence with a formula, especially for infinite sequences!**
- **For example, the sequence S on the previous slide can be specified as $\{ a_n \}$, where $a_n = 2*n$, for n=1..5**

Sequence

Plot of sequence converging toward 0; start at position 1



Color Sequence



Sequence Formulae

Which formulae for a_n describe the sequences below, here named: $a_1, a_2, a_3, \ldots$?

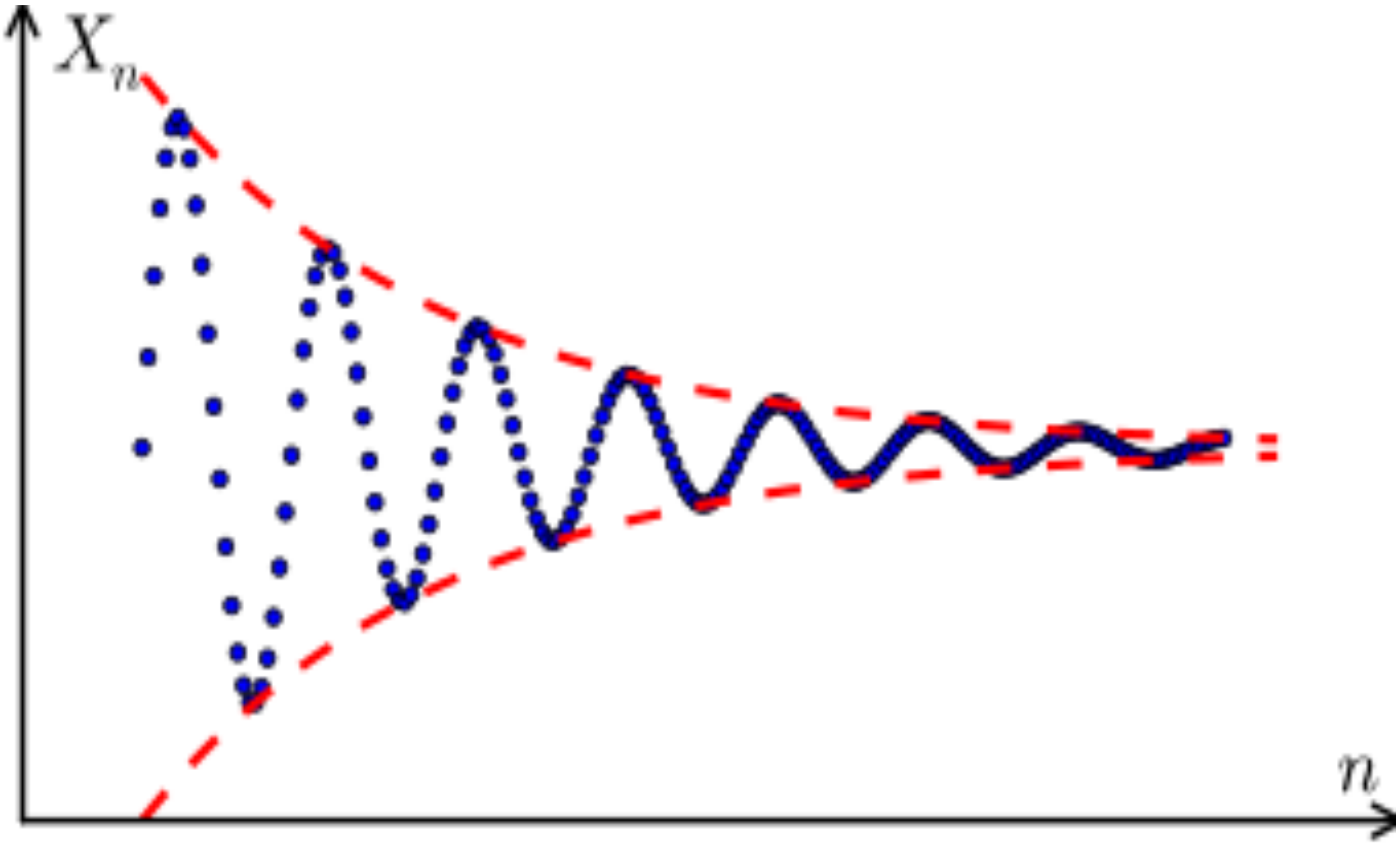
Sequence:	Formula:
$a_1 = 1, 3, 5, 7, 9, \ldots$	$a_n = 2n - 1, n = 1 .. \infty$
$a_2 = -1, 1, -1, 1, -1, \ldots$	$a_n = (-1)^n, n = 1 .. \infty$
$a_3 = 2, 5, 10, 17, 26, \ldots$	$a_n = n^2 + 1, n = 1 .. \infty$
$a_4 = 0.25, 0.5, 0.75, 1, 1.25 \ldots$	$a_n = 0.25n, n = 1 .. \infty$
$a_5 = 3, 9, 27, 81, 243, \ldots$	$a_n = 3^n, n = 1 .. \infty$

Cauchy Sequence

- **Terms of a Cauchy sequence X_n** (see next page) **converge progressively closer together as n increases**
- **Cauchy sequence: part of study of sequences**
- **One particularly important result in real analysis is *Cauchy characterization of convergence* for sequences:**
  - **A sequence of real numbers is convergent if and only if it is Cauchy**
- **In contrast, there are Cauchy sequences of rational numbers that are not convergent in the rationals; not covered here in CSc 28**

Cauchy Sequence

Plot of Sample Cauchy Sequence X_n



Strings

- **Finite sequences** are also called **strings**, often denoted: $a_1\, a_2\, a_3 \ldots a_n$
- **Length** of a string S is the number of terms contained in string S
- **Empty string** contains no terms at all; it has length zero
- Other strings ☺ here:

Game “Sequence”

Summation

Summations

What does $\sum_{j=m}^{n} a_j$ stand for?

- Formula represents sum: $a_m + a_{m+1} + a_{m+2} + \ldots + a_n$
- Where variable j is called **index of summation**, running from its **lower limit m** to its **upper limit n**
- Could have used any other letter to denote this index
- Start index used here is neither 0 nor 1 ☺

Summations

1. How to express the sum of first 1000 terms of the sequence { a_n } with $a_n = n^2$ for n = 1, 2, 3, … , 1000 ?

We write it as $\sum_{j=1}^{1000} j^2$

2. What will be the final value V - sum of j? $\sum_{j=1}^{6} j$

Values of j added up: 1 + 2 + 3 + 4 + 5 + 6 = 21

3. What is the value of $\sum_{j=1}^{100} j$?

Tedious to calculate manually … ☺ see Gauss next:

Summations

Urban legend tells us: Friedrich Gauss, sitting in class, assigned to waste some time by adding the first n = 100 integers, came up instead with a clever formula:

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

Result of such a summation can be calculated more easily, for example for the first 100 elements:

$$\sum_{j=1}^{100} j = \frac{100(100+1)}{2} = \frac{10100}{2} = 5050$$

Sequence vs. Series

- What is the difference between sequence and series?
- A list of numbers written in a definite order is called a sequence; not necessary to combined via operators
- Yet the sum of terms of an infinite sequence is called an infinite series
- A sequence can be defined as a function whose domain is the set of Natural Numbers
- Therefore sequence is an ordered list of numbers and series is the sum of such a list of numbers
- Example of a sequence: 2, 4, 6, 8, 10 … Now if we add them up: 2+4+6+8+10+ … The result is a series

Arithmetic Series

Consider: $$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

Observe that:

1 + 2 + 3 +…+ n/2 + (n/2 + 1) +…+ (n - 2) + (n - 1) + n

= (1 + n) + (2 + (n - 1)) + (3 + (n - 2)) +…+ (n/2 + (n/2 + 1))

= (n + 1) + (n + 1) + (n + 1) + … + (n + 1) *(with n/2 terms)*

= (n + 1) n / 2

Geometric Series

Goal to compute S = $$\sum_{j=0}^{n} a^j = \frac{a^{(n+1)} - 1}{(a-1)}$$

Observe that: $S = 1 + a + a^2 + a^3 + \ldots + a^n$

$aS = a + a^2 + a^3 + \ldots + a^n + a^{n+1}$

$(aS - S) = (a - 1)\, S = a^{n+1} - 1$

$S = (a^{n+1} - 1) / (a - 1)$

Proved that: $1 + a + a^2 + \ldots + a^n = S = (a^{n+1} - 1) / (a - 1)$

For example: $1 + 2 + 4 + 8 + \ldots + 2^{10} = 2047$ Math beauty!

Useful Series

1. $$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

2. $$\sum_{j=0}^{n} a^{j} = \frac{a^{(n+1)} - 1}{(a-1)}$$

3. $$\sum_{j=1}^{n} j^{2} = \frac{n(n+1)(2n+1)}{6}$$

4. $$\sum_{j=1}^{n} j^{3} = \frac{n^{2}(n+1)^{2}}{4}$$

Double Summations

Corresponding to nested loops in C++ or Java, there is ample use of double, triple, etc. summation in Math:

Example:

$$\sum_{i=1}^{5}\sum_{j=1}^{2} ij$$

$$= \sum_{i=1}^{5}(i + 2i)$$

$$= \sum_{i=1}^{5} 3i$$

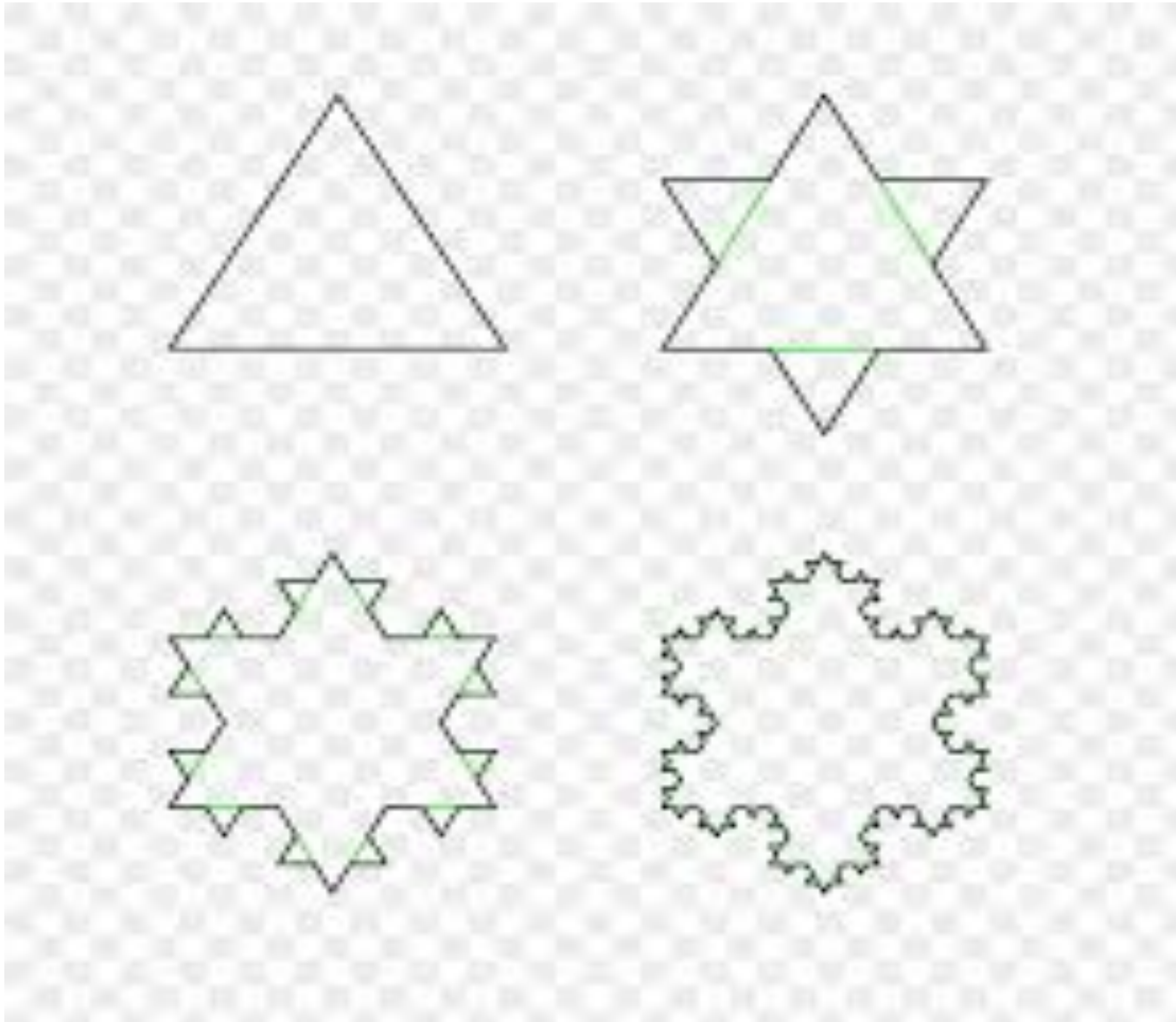
$$= 3 + 6 + 9 + 12 + 15 = 45$$

Recursive Sets

Recursion

- Recursion focus here is:
  - Recursive sets
- A later chapter, more advanced, will cover:
  - Recursion implementation in programming languages
  - Recursion vs. iteration
  - HLL programming uses of recursion (e.g. in Java, C, C++)

Recursion in Geometry



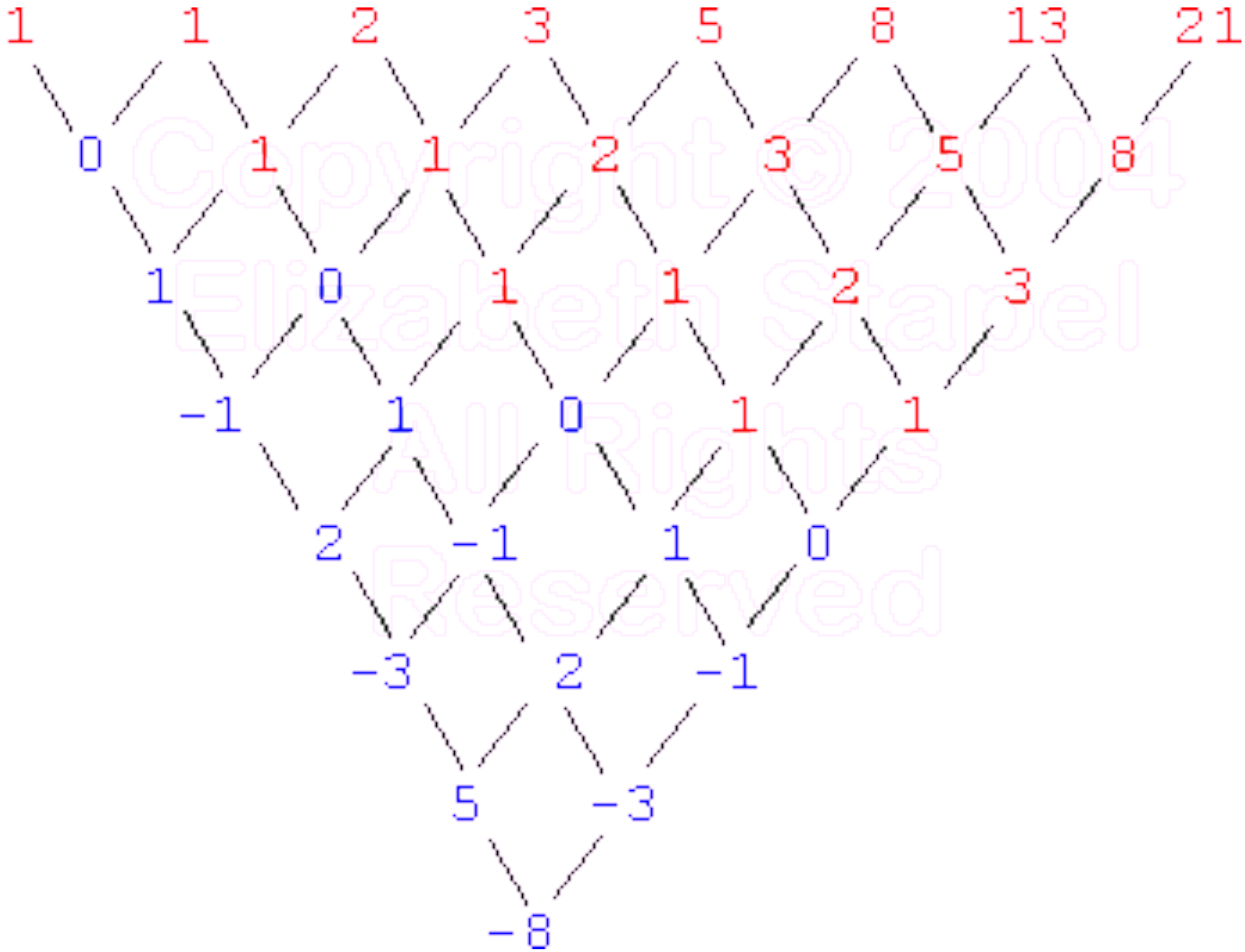
Recursion

- Recursion: Mathematical principle to express algorithmic steps; related to mathematical induction
- Intuitively: In a recursive definition, an algorithm is defined in terms of itself!? –Careful about intuition! ☺
- More accurately: A definition is recursive if it is partly defined via simpler versions of itself
  - Simpler: the total number of steps is reduced; perhaps some parameter is reduced in value
  - Partly: there are other steps, e.g. initial steps, aside form the recursively used portion of such an algorithm
  - Itself: yes the function name occurs as a call inside the function body –directly or possibly indirectly, via intermediary
- We'll recursively define sequences, functions, sets, . . .

Recursively Defined Sequences

- Example: Sequence $\{ a_n \}$ of powers of 2 is given by: $a_n = 2^n$ for $n = 0, 1, 2, 3, \ldots$
- The same sequence can also be defined recursively:
- $a_0 = 1$ for $n = 0$
- $a_{n+1} = 2a_n$ for $n = 0, 1, 2, 3, \ldots$
- Mathematical Induction and Recursion are strongly related!

Recursive Sequence



Recursively Defined Functions

Use the following generic method to define some function f with the natural numbers as its domain:

- Base case: Specify the value of f at index zero, AKA f_0
- Recursion: Express a formula (e.g. math equation) for finding the value of f at any higher index, referring to f's earlier values at lower indices
- Such a definition is called recursive
- AKA inductive definition

Recursive Function fact()

Recursively define the factorial function fact(n) = n! more completely:

fact(0) = 1

fact(n + 1) = (n + 1) * fact(n)   – for n > 0

So:

fact(0) = 1

fact(1) = 1 * fact(0)   = 1 * 1 =   1

fact(2) = 2 * fact(1)   = 2 * 1 =   2

fact(3) = 3 * fact(2)   = 3 * 2 =   6

fact(4) = 4 * fact(3)   = 4 * 6 = 24

Recursive Function fact()

```
#include . . .

unsigned calls = 0;      // track # of calls

// Recursive fact() function
// includes tracking # of calls

unsigned fact( unsigned arg ) // unsigned?
{ // fact
   calls++;                  // global to fact()
   if( 0 == arg ) {          // why strange order?
      return 1;
   }else{
      return fact( arg - 1 ) * arg;
   } // end if
   // Should an assertion be here?
} // end fact
```

Recursive Function fact()

```
r_fact( 0) =          1, calls =  1
r_fact( 1) =          1, calls =  2
r_fact( 2) =          2, calls =  3
r_fact( 3) =          6, calls =  4
r_fact( 4) =         24, calls =  5
r_fact( 5) =        120, calls =  6
r_fact( 6) =        720, calls =  7
r_fact( 7) =       5040, calls =  8
r_fact( 8) =      40320, calls =  9
r_fact( 9) =     362880, calls = 10
r_fact(10) =    3628800, calls = 11
r_fact(11) =   39916800, calls = 12
r_fact(12) =  479001600, calls = 13
r_fact(13) = 1932053504, calls = 14

 . . .
```

Recursive Function fibo()

Fibonacci numbers, AKA Fibonacci Sequence

fibo(0) = 0, fibo(1) = 1

fibo(n) = fibo(n - 1) + fibo(n - 2)   – for n > 1

fibo(0) = 0

fibo(1) = 1

fibo(2) = fibo(1) + fibo(0) = 1 + 0 = 1

fibo(3) = fibo(2) + fibo(1) = 1 + 1 = 2

fibo(4) = fibo(3) + fibo(2) = 2 + 1 = 3

fibo(5) = fibo(4) + fibo(3) = 3 + 2 = 5

fibo(6) = fibo(5) + fibo(4) = 5 + 3 = 8   . . .

Recursive Function fibo()

```
#define MAX 30     // > 30 not computable here

unsigned calls;    // track # of calls

// recursive function fibo()
unsigned fibo( unsigned arg )
{ // fibo
  calls++;                // OK I am sinning
  if( arg <= 1 ) {        // base case?
     return arg;          // if so: done!
  }else{
     return fibo( arg-1 ) + fibo( arg-2 );
  } // end if
  // Should an assertion be here?
} // end fibo
```

Recursive Function fibo()

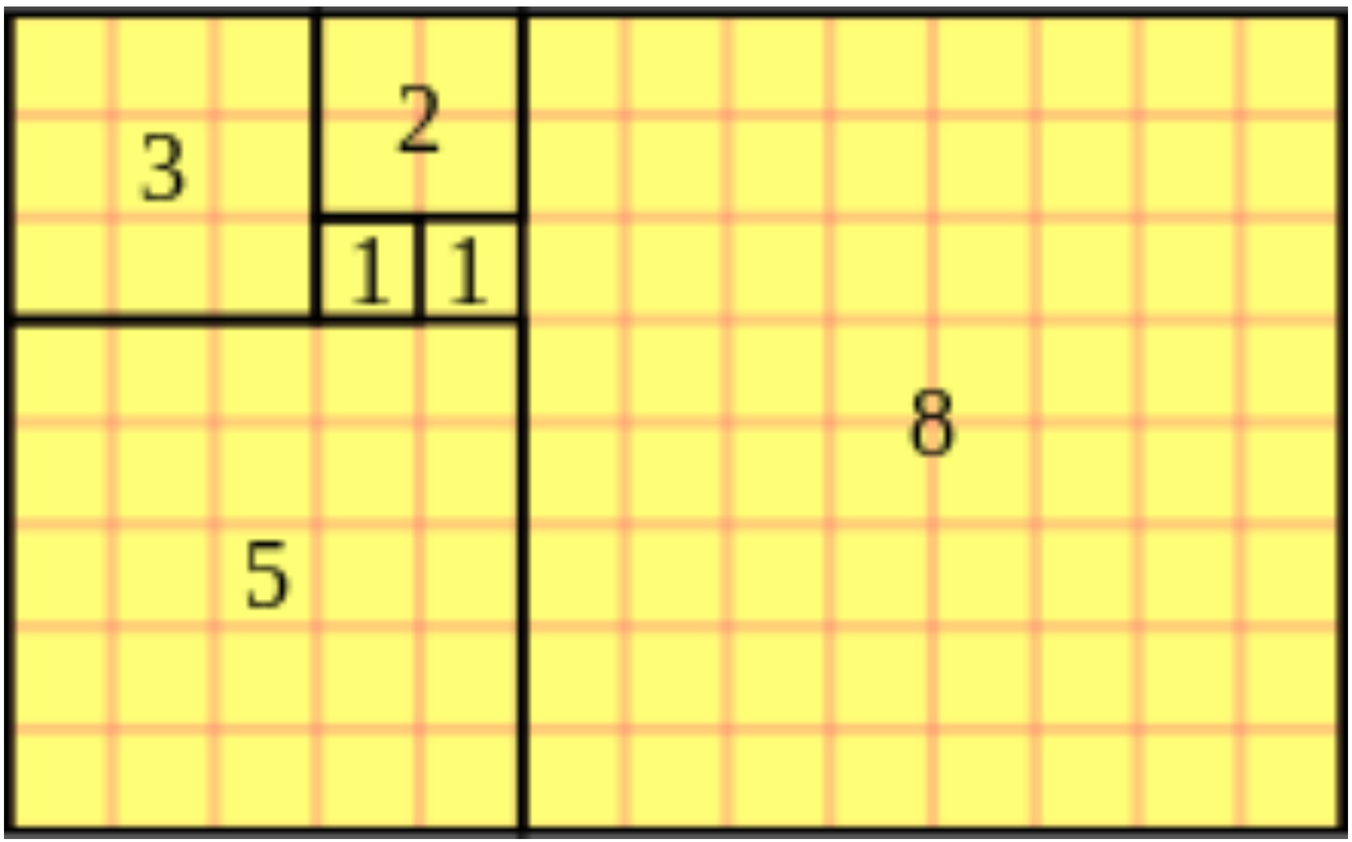
```
 r_fibo( 0) =      0, calls =       1
 r_fibo( 1) =      1, calls =       1
 r_fibo( 2) =      1, calls =       3
 r_fibo( 3) =      2, calls =       5
 r_fibo( 4) =      3, calls =       9

. . .

 r_fibo(22) =  17711, calls =   57313
 r_fibo(23) =  28657, calls =   92735
 r_fibo(24) =  46368, calls =  150049
 r_fibo(25) =  75025, calls =  242785
 r_fibo(26) = 121393, calls =  392835
 r_fibo(27) = 196418, calls =  635621
 r_fibo(28) = 317811, calls = 1028457
 r_fibo(29) = 514229, calls = 1664079
```

Recursive Functions

Interesting squares whose sides are Fibonacci numbers



Recursively Defined Sets

Composing arithmetic formulae:

- Well-formed formulae (AKA formulas) include variables, literals and operators, e.g. +, -, *, /, ^
- Use symbolic names x, y, f, g, . . . for variables
- Use (and) for grouping operators and operands:
    - (f + g)
    - (f - g)
    - (f * g)
    - (f / g)
    - (f ^ g) ^ for "power of" operator
- Consider those formulae, all well-formed

Recursively Defined Sets

With this convention, compose progressively more complex formulae, such as:

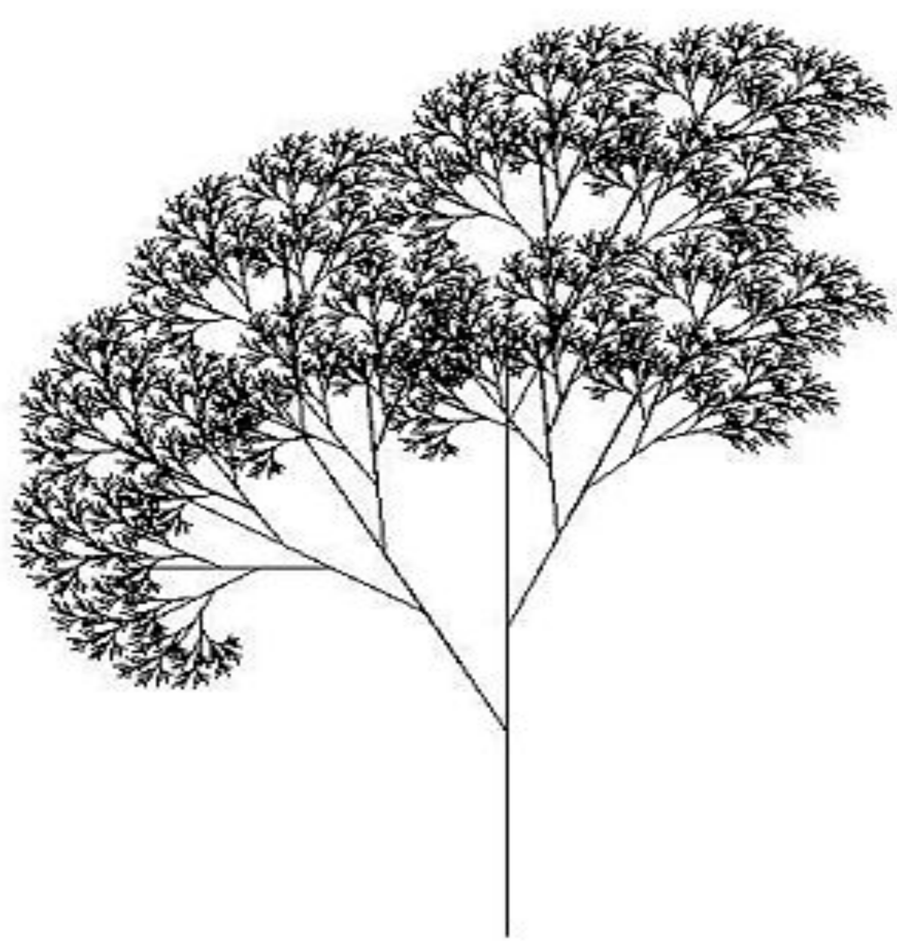
- (x - y + 1)
- ((z / 3) - y)
- ((z / 3) - (6 + 5))
- ((z / (2 * 4)) - (6 + 5))
- etc.

Recursive Algorithms

- Review: An algorithm is recursive if it is partly defined by simpler versions of itself
- AKA: An algorithm is recursive if it solves a mathematical problem in part by reducing the problem to itself at a smaller (simpler) input
  - There could be multiple inputs, AKA formal parameters
  - At least one of which is simpler in recursive use/call
- Example: Recursive Euclidean Algorithm
- procedure gcd(a, b) for: nonnegative integers a, b and with a < b
- if a = 0 then gcd(a, b) = b
- else gcd(a, b) = gcd(b mod a, a)

Recursive Tree

Each branch is a smaller version of this tree



Recursive Algorithms

- For every recursive algorithm, there is an equivalent iterative algorithm
- Recursive algorithms are often shorter, more elegant, and easier to understand than iterative counterparts
- However, iterative algorithms are usually more efficient in their use of execution space and time

Summary

- **Sequence**: ordered list of 0 or more elements that may repeat; elements are position dependent (as opposed to elements of a **set**)
- A **series** is the sum of elements of a sequence
- **Induction** is an efficient method for theorem proving
- **Recursion** is a mathematical principle that expresses an algorithm partly in terms of simpler versions of itself

References

1. Wiki sequence: https://en.wikipedia.org/wiki/Sequence
2. Arithmetic geometric sequence: https://en.wikipedia.org/wiki/Arithmetico–geometric_sequence
3. Recursion: https://en.wikipedia.org/wiki/Recursion_(computer_science)
4. Series vs. sequence: https://www.tutapoint.com/knowledge-center/view/difference-between-sequence-and-series
5. Math an Experimental Science: https://www2.math.upenn.edu/~wilf/website/Mathematics_AnExperimentalScience.pdf