Algorithm definition

New! An algorithm is a finite sequence of precise instructions for solving a problem.

Algorithms can be expressed in English or in more formalized descriptions like pseudocode or fully executable programs.

Sometimes, we can define algorithms whose output matches the rule for a function we already care about. Consider the (integer) logarithm function

$$logb : \{b \in \mathbb{Z} \mid b > 1\} \times \mathbb{Z}^+ \quad \rightarrow \quad \mathbb{N}$$

defined by

$$logb(\ (b, n)\ ) = \text{greatest integer } y \text{ so that } b^y \text{ is less than or equal to } n$$

Calculating integer part of base b logarithm

```
1    procedure logb(b,n: positive integers with b>1)
2    i := 0
3    while n > b-1
4      i := i+1
5      n := n div b
6    return i {i holds the integer part of the base b logarithm of n}
```

Trace this algorithm with inputs $b = 3$ and $n = 17$

	b	n	i	$n > b - 1$?
Initial value	3	17		
After 1 iteration				
After 2 iterations				
After 3 iterations				

Compare: does the output match the rule for the (integer) logarithm function?

Fixed width definition

Definition For b an integer greater than 1, w a positive integer, and n a nonnegative integer _____, the **base** b **fixed-width** w **expansion of** n is

$$(a_{w-1} \cdots a_1 a_0)_{b,w}$$

where $a_0, a_1, \ldots, a_{w-1}$ are nonnegative integers less than b and

$$n = \sum_{i=0}^{w-1} a_i b^i$$

Fixed width example

Decimal $b = 10$	Binary $b = 2$	Binary fixed-width 10 $b = 2$, $w = 10$	Binary fixed-width 7 $b = 2$, $w = 7$	Binary fixed-width 4 $b = 2$, $w = 4$
$(20)_{10}$				

Fixed width fractional definition

Definition For b an integer greater than 1, w a positive integer, w' a positive integer, and x a real number the **base b fixed-width expansion of x with integer part width w and fractional part width w'** is $(a_{w-1} \cdots a_1 a_0 . c_1 \cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and

$$x \geq \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} \qquad \text{and} \qquad x < \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} + b^{-w'}$$

3.75 in fixed-width binary, integer part width 2, fractional part width 8	
0.1 in fixed-width binary, integer part width 2, fractional part width 8	

```
welcome $jshell
|  Welcome to JShell -- Version 10.0.1
|  For an introduction type: /help intro

jshell> 0.1
$1 ==>

jshell> 0.2
$2 ==>

jshell> 0.1 + 0.2
$3 ==>

jshell> Math.sqrt(2)
$4 ==>

jshell> Math.sqrt(2)*Math.sqrt(2)
$5 ==>

jshell>
```

Note: Java uses floating point, not fixed width representation, but similar rounding errors appear in both.

Negative int expansions

Representing negative integers in binary: Fix a positive integer width for the representation w, $w > 1$.

	To represent a positive integer n	To represent a negative integer $-n$
Sign-magnitude	$[0a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1a_{w-2} \cdots a_0]_{s,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$:
2s complement	$[0a_{w-2} \cdots a_0]_{2c,w}$, where $n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $n = 17$, $w = 7$:	$[1a_{w-2} \cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2} \cdots a_0)_{2,w-1}$ Example $-n = -17$, $w = 7$:

Calculating 2s complement

For positive integer n, to represent $-n$ in 2s complement with width w,

- Calculate $2^{w-1} - n$, convert result to binary fixed-width $w - 1$, pad with leading 1, or
- Express $-n$ as a sum of powers of 2, where the leftmost 2^{w-1} is negative weight, or
- Convert n to binary fixed-width w, flip bits, add 1 (ignore overflow)

Challenge: use definitions to explain why each of these approaches works.

Representing zero

Representing 0:

So far, we have representations for positive and negative integers. What about 0?

	To represent a **non-negative** integer n	To represent a **non-positive** integer $-n$
Sign-magnitude	$[0a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$:	$[1a_{w-2}\cdots a_0]_{s,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$:
2s complement	$[0a_{w-2}\cdots a_0]_{2c,w}$, where $n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $n = 0$, $w = 7$:	$[1a_{w-2}\cdots a_0]_{2c,w}$, where $2^{w-1} - n = (a_{w-2}\cdots a_0)_{2,w-1}$ Example $-n = 0$, $w = 7$:

Netflix intro

What data should we encode about each Netflix account holder to help us make effective recommendations?

In machine learning, clustering can be used to group similar data for prediction and recommendation. For example, each Netflix user's viewing history can be represented as a n-tuple indicating their preferences about movies in the database, where n is the number of movies in the database. People with similar tastes in movies can then be clustered to provide recommendations of movies for one another. Mathematically, clustering is based on a notion of distance between pairs of n-tuples.

Data types

Term	**Examples**: (add additional examples from class)
set unordered collection of elements *repetition doesn't matter* *Equal sets agree on membership of all elements*	$7 \in \{43, 7, 9\}$ $2 \notin \{43, 7, 9\}$
n-tuple ordered sequence of elements with n "slots" $(n > 0)$ *repetition matters, fixed length* *Equal n-tuples have corresponding components equal*	
string ordered finite sequence of elements each from specified set (called the alphabet over which the string is defined) *repetition matters, arbitrary finite length* *Equal strings have same length and corresponding characters equal*	

Special cases:

When $n = 2$, the 2-tuple is called an **ordered pair**.

A string of length 0 is called the **empty string** and is denoted λ.

A set with no elements is called the **empty set** and is denoted $\{\}$ or $\emptyset$.

Set operations

To define a set we can use the roster method, set builder notation, a recursive definition, and also we can apply a set operation to other sets.

New! Cartesian product of sets and **set-wise concatenation of sets of strings**

Definition: Let X and Y be sets. The **Cartesian product** of X and Y, denoted $X \times Y$, is the set of all ordered pairs (x, y) where $x \in X$ and $y \in Y$

$$X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$$

Conventions: (1) Cartesian products can be chained together to result in sets of n-tuples and (2) When we form the Cartesian product of a set with itself $X \times X$ we can denote that set as X^2, or X^n for the Cartesian product of a set with itself n times for a positive integer n.

Definition: Let X and Y be sets of strings over the same alphabet. The **set-wise concatenation** of X and Y, denoted $X \circ Y$, is the set of all results of string concatenation xy where $x \in X$ and $y \in Y$

$$X \circ Y = \{xy \mid x \in X \text{ and } y \in Y\}$$

Pro-tip: the meaning of writing one element next to another like xy depends on the data-types of x and y. When x and y are strings, the convention is that xy is the result of string concatenation. When x and y are numbers, the convention is that xy is the result of multiplication. This is (one of the many reasons) why is it very important to declare the data-type of variables before we use them.

Fill in the missing entries in the table:

Set	Example elements in this set and their data type:
B	A C G U
	(A, C) (U, U)
$B \times \{-1, 0, 1\}$	
$\{-1, 0, 1\} \times B$	
	$(0, 0, 0)$
$\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\} \circ \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$	
	GGGG

Defining functions

New! Defining functions A function is defined by its (1) domain, (2) codomain, and (3) rule assigning each element in the domain exactly one element in the codomain.

The domain and codomain are nonempty sets.
The rule can be depicted as a table, formula, piecewise definition, or English description.
The notation is

"Let the function FUNCTION-NAME: DOMAIN $\to$ CODOMAIN be given by
FUNCTION-NAME(x) = ... for every $x \in DOMAIN$".

or

"Consider the function FUNCTION-NAME: DOMAIN $\to$ CODOMAIN defined as
FUNCTION-NAME(x) = ... for every $x \in DOMAIN$".

Example: The absolute value function

Domain

Codomain

Rule

Defining functions recursively

When the domain of a function is a *recursively defined set*, the rule assigning images to domain elements (outputs) can also be defined recursively.

Recall: The set of RNA strands S is defined (recursively) by:

Basis Step:	$\mathtt{A} \in S, \mathtt{C} \in S, \mathtt{U} \in S, \mathtt{G} \in S$
Recursive Step:	If $s \in S$ and $b \in B$, then $sb \in S$

where sb is string concatenation.

Definition (Of a function, recursively) A function *rnalen* that computes the length of RNA strands in S is defined by:

		$rnalen : S$	$\to \mathbb{Z}^+$
Basis Step:	If $b \in B$ then	$rnalen(b)$	$= 1$
Recursive Step:	If $s \in S$ and $b \in B$, then	$rnalen(sb)$	$= 1 + rnalen(s)$

The domain of *rnalen* is

The codomain of *rnalen* is

Example function application:

$$rnalen(\mathtt{ACU}) =$$

Example: A function *basecount* that computes the number of a given base b appearing in a RNA strand s is defined recursively:

		$basecount : S \times B$	$\to \mathbb{N}$	
Basis Step:	If $b_1 \in B, b_2 \in B$	$basecount(\ (b_1, b_2)\ )$	$= \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases}$	
Recursive Step:	If $s \in S, b_1 \in B, b_2 \in B$	$basecount(\ (sb_1, b_2)\ )$	$= \begin{cases} 1 + basecount(\ (s, b_2)\ ) & \text{when } b_1 = b_2 \\ basecount(\ (s, b_2)\ ) & \text{when } b_1 \neq b_2 \end{cases}$	