Algorithm redundancy

Real-life representations are often prone to corruption. Biological codes, like RNA, may mutate naturally¹ and during measurement; cosmic radiation and other ambient noise can flip bits in computer storage². One way to recover from corrupted data is to introduce or exploit redundancy.

Consider the following algorithm to introduce redundancy in a string of 0s and 1s.

Create redundancy by repeating each bit three times

```
1 procedure redun3(a_{k-1}\cdots a_0): a nonempty bitstring)
2 for i:=0 to k-1
3 c_{3i}:=a_i
4 c_{3i+1}:=a_i
5 c_{3i+2}:=a_i
6 return c_{3k-1}\cdots c_0
```

Decode sequence of bits using majority rule on consecutive three bit sequences

```
procedure decode3(c_{3k-1}\cdots c_0): a nonempty bitstring whose length is an integer multiple of 3)

for i:=0 to k-1

if exactly two or three of c_{3i}, c_{3i+1}, c_{3i+2} are set to 1

a_i:=1

else

a_i:=0

return a_{k-1}\cdots a_0
```

Give a recursive definition of the set of outputs of the redun3 procedure, Out,

```
Consider the message m = 0001 so that the sender calculates redun3(m) = redun3(0001) = 000000000111.
```

Introduce ____ errors into the message so that the signal received by the receiver is _____ but the receiver is still able to decode the original message.

Challenge: what is the biggest number of errors you can introduce?

Building a circuit for lines 3-6 in *decode* procedure: given three input bits, we need to determine whether the majority is a 0 or a 1.

c_{3i}	c_{3i+1}	c_{3i+2}	a_i
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

Circuit

¹Mutations of specific RNA codons have been linked to many disorders and cancers.

²This RadioLab podcast episode goes into more detail on bit flips: https://www.wnycstudios.org/story/bit-flip

Algorithm rna mutation insertion deletion

 $mutation: S \times \mathbb{Z}^+ \times B \to S$

return $c_1 \cdots c_m$ {The return value is a RNA strand made of the c_i values}

Recall that S is defined as the set of all RNA strands, nonempty strings made of the bases in $B = \{A, U, G, C\}$. We define the functions

 $insertion: S \times \mathbb{Z}^+ \times B \to S$

```
deletion: \{s \in S \mid rnalen(s) > 1\} \times \mathbb{Z}^+ \to S
                                                                                                         with rules
    procedure mutation(b_1 \cdots b_n): a RNA strand, k: a positive integer, b: an element of B)
    for i := 1 to n
 2
       if i = k
 3
          c_i := b
       else
          c_i := b_i
    return c_1 \cdots c_n {The return value is a RNA strand made of the c_i values}
    procedure insertion(b_1 \cdots b_n): a RNA strand, k: a positive integer, b: an element of B)
       for i := 1 to n
 3
          c_i := b_i
       c_{n+1} := b
 5
 6
    _{
m else}
       for i := 1 to k-1
         c_i := b_i
       c_k := b
       for i := k+1 to n+1
10
11
         c_i := b_{i-1}
    return c_1 \cdots c_{n+1} {The return value is a RNA strand made of the c_i values}
12
    procedure deletion(b_1 \cdots b_n): a RNA strand with n > 1, k: a positive integer)
    if k > n
       m := n
 3
       for i := 1 to n
 4
          c_i := b_i
       \mathbf{for} \ i \ := \ 1 \ \mathbf{to} \ k-1
 9
          c_i := b_i
       \mathbf{for} \ i \ := \ k \ \mathbf{to} \ n-1
10
         c_i := b_{i+1}
11
```

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

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

defined by

logb((b,n)) = greatest integer y so that b^y is less than or equal to n

Calculating integer part of base b logarithm

```
procedure logb(b,n): positive integers with b>1)

i:=0

while n>b-1

i:=i+1

n:=n div b

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?

Base expansion algorithms

Two algorithms for constructing base b expansion from decimal representation

Most significant first: Start with left-most coefficient of expansion (highest value)

Informally: Build up to the value we need to represent in "greedy" approach, using units determined by base.

Calculating base b expansion, from left

```
procedure baseb1(n,b): positive integers with b>1)

v:=n

k:=1+ output of logb algorithm with inputs b and b

for i:=1 to b

a_{k-i}:=0

while v\geq b^{k-i}

a_{k-i}:=a_{k-i}+1

v:=v-b^{k-i}

return (a_{k-1},\ldots,a_0)\{(a_{k-1}\ldots a_0)_b \text{ is the base } b \text{ expansion of } n\}
```

Least significant first: Start with right-most coefficient of expansion (lowest value)

Idea: (when k > 1)

$$n = a_{k-1}b^{k-1} + \dots + a_1b + a_0$$

= $b(a_{k-1}b^{k-2} + \dots + a_1) + a_0$

so $a_0 = n \mod b$ and $a_{k-1}b^{k-2} + \cdots + a_1 = n \operatorname{div} b$.

Calculating base b expansion, from right

```
procedure baseb2(n,b): positive integers with b>1)

q:=n

k:=0

while q\neq 0

a_k:=q \mod b

q:=q \operatorname{div} b

k:=k+1

return (a_{k-1},\ldots,a_0)\{(a_{k-1}\ldots a_0)_b \text{ is the base } b \text{ expansion of } n\}
```

Base conversion algorithm

Practice: write an algorithm for converting from base b_1 expansion to base b_2 expansion: