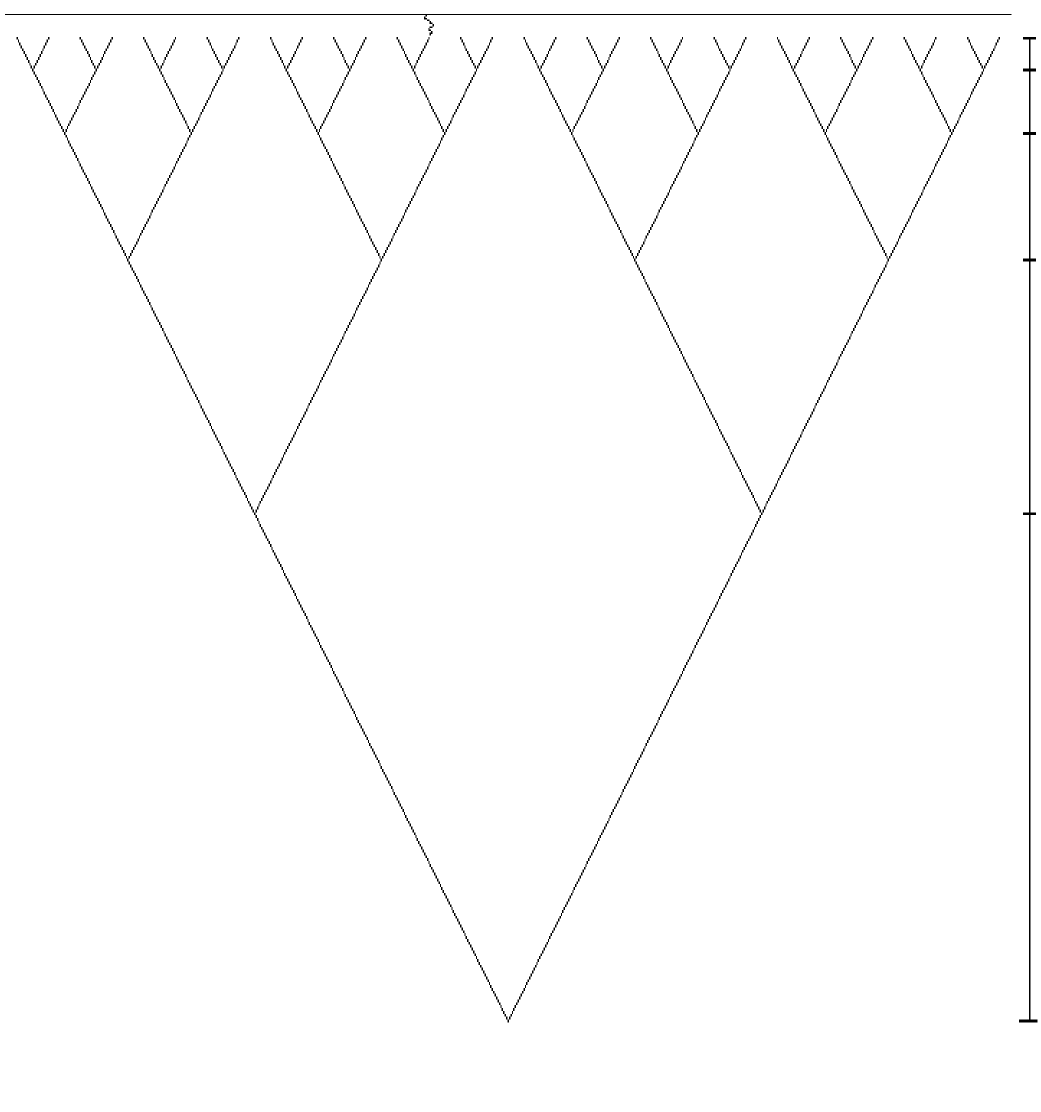
CSE 380	Discrete Mathematics II
	Even More About Infinity
14 January 2021	*Learn*

How Many Nodes and How Many Paths?

Answer 2 is correct. The Infinite Binary Tree (IBT) has

- A *countable* infinity of nodes.
- An *uncountable* infinity of paths.

```
curl -O https://firstthreeodds.org/img/infinitebinarytree.png
```



Remember this:

$$\mathbf{N} \leftrightarrow \bigcup_{n=2}^{\infty} \left(\overbrace{\mathbf{N} \times \mathbf{N} \times \cdots \times \mathbf{N}}^{n} \right)$$

Meaning

the Union of a Countable Number of Countable Sets is countable!

How to Biject?

How do you map a number n to its address in the IBT?

Definition and Implementation

Define address to be a pair: (level, position) where both level and position start at zero.

```
from math import floor, log2

def to_IBT_address(n):
  level = floor(log2(n))
  position = n - 2 ** level
  return level, position
```

```
print(to_IBT_address(1029))
```

How do you do the reverse-mapping of an IBT address back to n?

```
def from_IBT_address(IBT_address):
  level, position = IBT_address
  return 2 ** level + position
```

```
print(from_IBT_address([10, 5]))
```

```
print(from_IBT_address(to_IBT_address(1029)))
```

Recall the Powerset of the Positive Integers

	1	2	3	4	...
1	0	0	0	0	
2	1	0	0	0	
3	0	1	0	0	
4	1	1	0	0	
5	0	0	1	0	
6	1	0	1	0	
7	0	1	1	0	
8	1	1	1	0	
9	0	0	0	1	
10	1	0	0	1	

Speaking of Cantor

Here is a pretty good explanation of his Diagonal Argument (and a lot more!):

`http://www.coopertoons.com/education/diagonal/diagonalargument.html`

Amazing Power

On page 5 of Brother Bessey's "To Infinity And Beyond" paper, he introduces the concept that a one-dimensional space, (a line), has the exact same number of points as a two-dimensional space, (a plane).

How could this possibly be true?

Answer

It was Cantor's discovery of this amazing Power of the Continuum that led him to exclaim:

> Je le vois, mais je ne le crois pas! (I see it, but I don't believe it!)

```
(org-sbe shunffle)
(shuffle [0 2 4 6 8] [1 3 5 7 9])
```

```
(defun shuffle (&rest vectors)
  (apply 'vconcat (apply 'mapcar* 'vector vectors)))

(defun unshuffle (vector &optional num-parts)
  (loop with answer = nil
        with l = (length vector)
        with n = (or num-parts 2)
        for i from 0 below n
        do (push (loop for j from i below l by n
                       vconcat (subseq vector j (+ j 1)))
                 answer)
        finally return (nreverse answer)))
```

```
(unshuffle [1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 6 2 6] 5)
```

```
(apply 'shuffle '([1 7] [2 8] [3 9] [4 7] [5 3] [6 4]))
```

Thanks to Kyle for this Python version of `shuffle` that can take an indefinite number of arguments. (You do the `unshuffle`!)

```
shuffle = lambda *a:sum([*map(list,zip(*a))],[])
print(shuffle([0,2,4,6,8],[1,3,5,7,9]))
```

Meaning

A 1-dimensional line

$\leftrightarrow$ 2-dimensional plane

$\leftrightarrow$ 3-dimensional space

$\leftrightarrow \ldots$.

$\leftrightarrow$ n-dimensional space

$\leftrightarrow \ldots$.

$0.314159265\ldots \leftrightarrow (0.34525.., 0.1196\ldots)$

$\aleph_0$ = countable infinity

$\aleph_1$ = uncountable infinity

$\aleph_1 = 2^{\aleph_0}$

$\aleph_2 = 2^{\aleph_1}$

$\aleph_n = 2^{\aleph_{n-1}}$

Pros and Cons

Raw LaTeX

scratch-paper.tex $\rightarrow$ scratch-paper.pdf

```
curl -O https://firstthreeodds.org/scratch-paper.tex
curl -O https://firstthreeodds.org/scratch-paper.pdf
```

```
scratch-paper.tex
```

TEST CENTER PERSONNEL TAKE NOTE: The instructor of CS 238, Rick Neff, hereby authorizes the holder of this piece of scratch paper to **keep it** after taking one of his online tests, for the purpose of learning where he or she may have erred.

Org-mode LaTeX

Like you've (perhaps) been using.

Experimenting with Pairing Function in CDL

```
from math import ceil, floor, sqrt
```

```
def f(m, n):
    x = m + n
    return (((x - 2) * (x - 1)) // 2) + m

def f_inverse_1(y):
    x = ceil((3 + sqrt(8 * y + 1)) / 2) - 1
    m = y - ((x - 2) * (x - 1)) // 2
    n = x - m
    return m, n

def f_inverse_2(y):
    x = int((3 + sqrt(8 * y)) / 2)
    m = y - ((x - 2) * (x - 1)) // 2
    n = x - m
    return m, n

def f_inverse_3(y):
    x = int((3 + sqrt(8 * y + 1)) / 2)
    m = y - ((x - 2) * (x - 1)) // 2
    n = x - m
    return m, n

assert(all([f_inverse_1(y) == f_inverse_2(y) for y in range(1,300)]))

for y in range(1,30):
  if (f_inverse_2(y) != f_inverse_3(y)):
    print(y, f_inverse_2(y), f_inverse_3(y))

for y in range(1,30000):
  m, n = f_inverse_3(y)
  assert(y == f(m, n))

def tri(x):
  return x * (x - 1) // 2

for n in range(1, 10):
  y = tri(n)
  print(n, y, (sqrt(8 * y + 1) - 1) / 2)
```