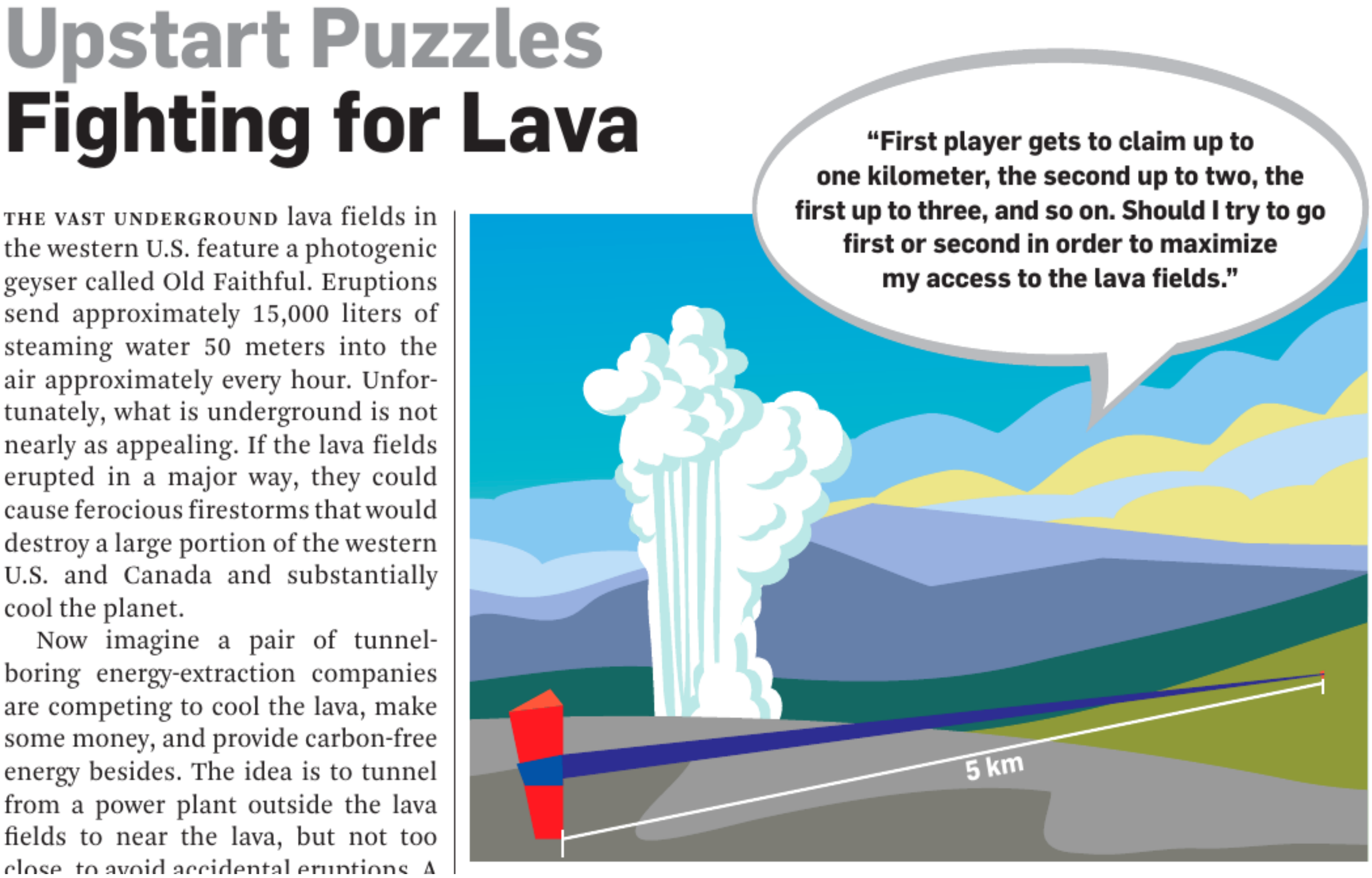
# SOLUTIONS

ACH
ENB
ACH



## Upstart Puzzles

# Fighting for Lava

THE VAST UNDERGROUND lava fields in the western U.S. feature a photogenic geyser called Old Faithful. Eruptions send approximately 15,000 liters of steaming water 50 meters into the air approximately every hour. Unfortunately, what is underground is not nearly as appealing. If the lava fields erupted in a major way, they could cause ferocious firestorms that would destroy a large portion of the western U.S. and Canada and substantially cool the planet.

Now imagine a pair of tunnel-boring energy-extraction companies are competing to cool the lava, make some money, and provide carbon-free energy besides. The idea is to tunnel from a power plant outside the lava fields to near the lava, but not too close to avoid accidental eruptions. A

"First player gets to claim up to one kilometer, the second up to two, the first up to three, and so on. Should I try to go first or second in order to maximize my access to the lava fields."
5 km

Source of this puzzle:

DOI:10.1145/3314071

## Legend

Player 1

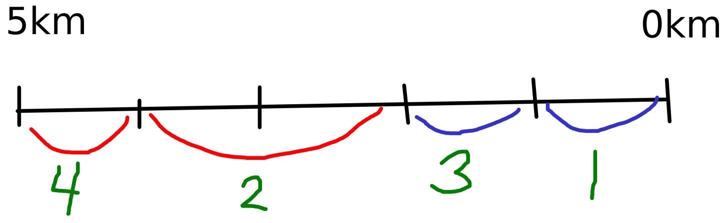
Player 2

move#

↑

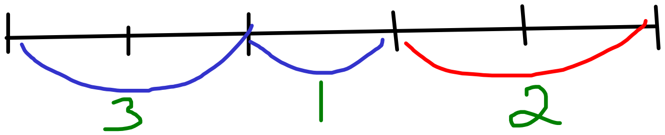
to show the order of the moves as the game is played

**Warm-up.** Suppose the line segment is five kilometers long from a stake at kilometer 0 to a stake at kilometer 5. Suppose the first player takes between 0 and 1 kilometer. Which player would get more of the line segment, assuming each plays optimally?



← bad 1st move by Player 1

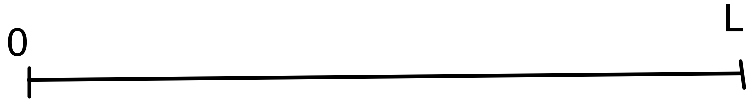
**Question 1.** By playing differently, beginning with the first move, could the first player acquire the rights to more of the line segment than the second player?

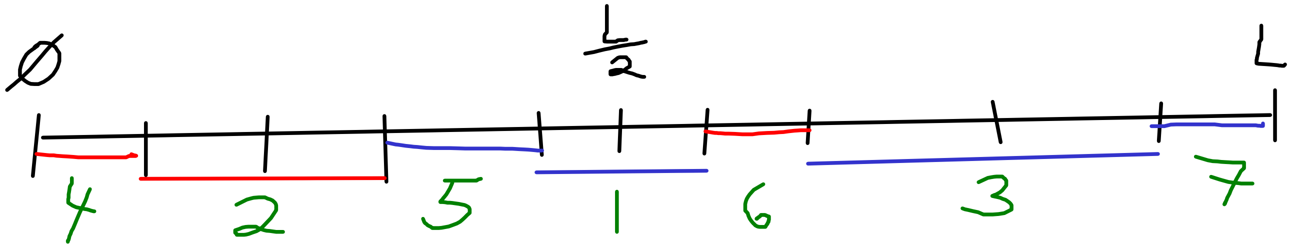


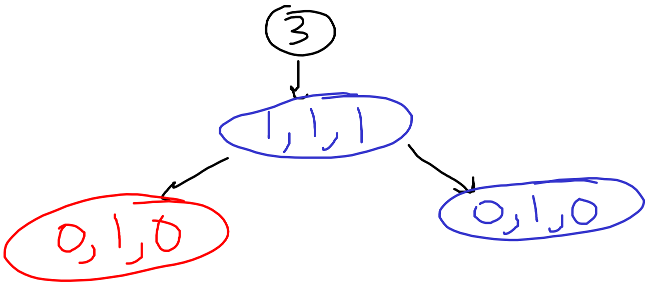
← good 1st move by Player 1

*Solution to question 1.* Yes. If the first player takes kilometers 2 to 3, then the second player could take kilometers 0 to 2, but then the first player would take kilometers 3 to 5. The first player would thus get three of the five kilometers.

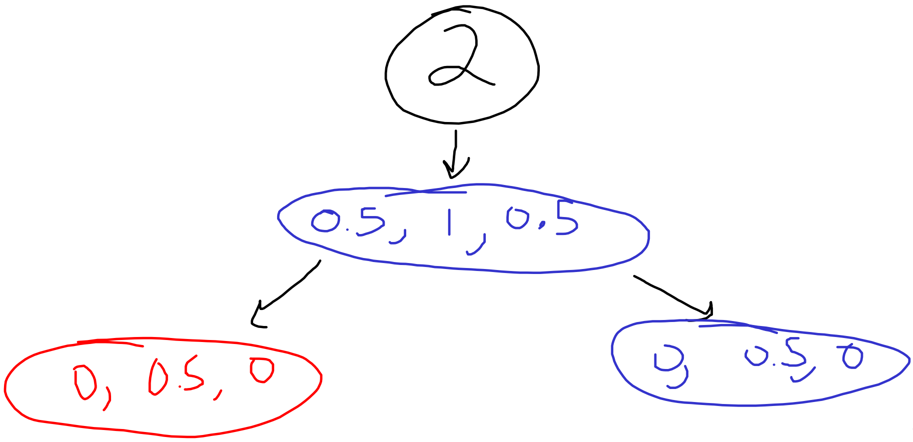
**Question 2.** Is there a minimal amount by which one player can win regardless of the length $L$ of the segment?







Player1 wins by 1

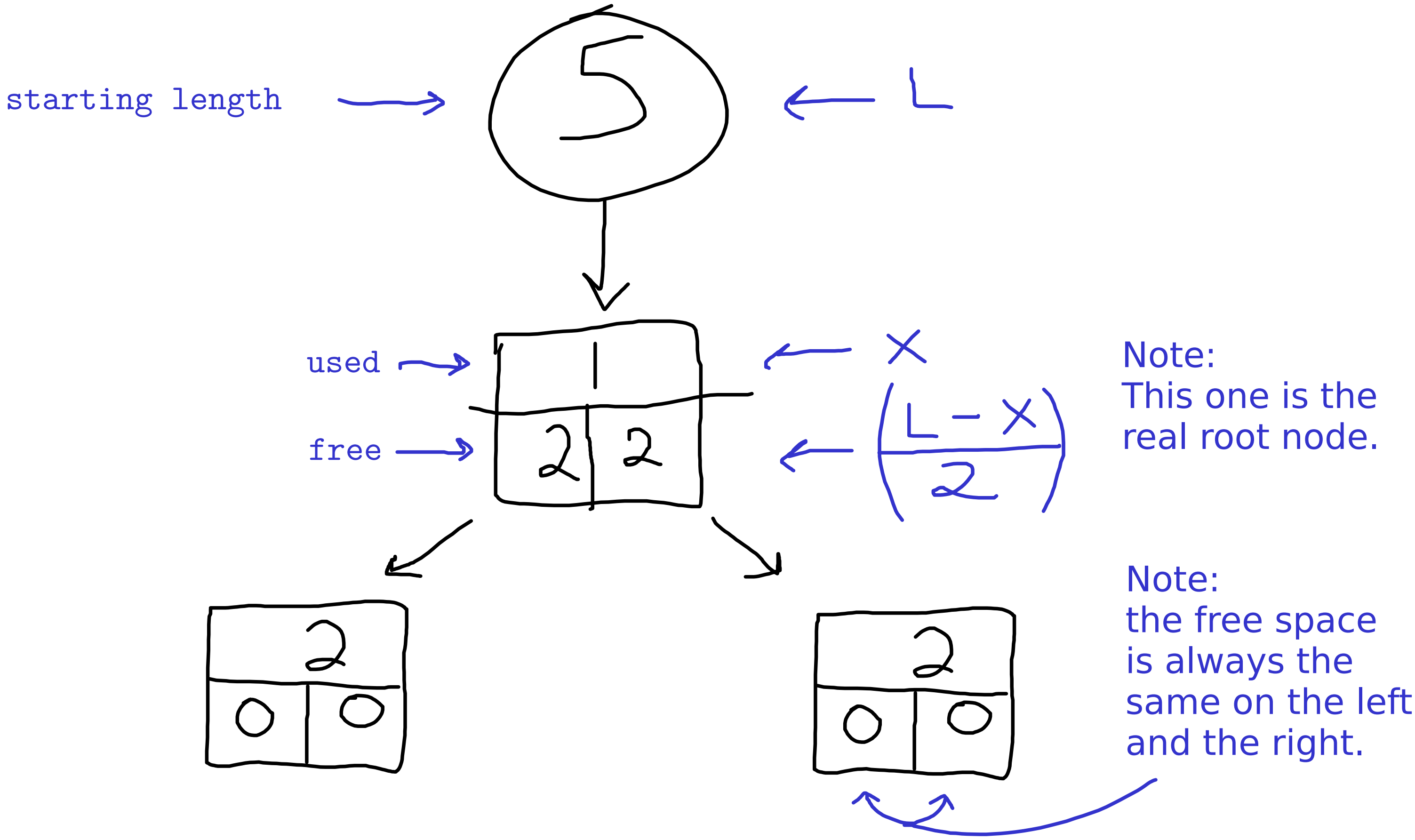


Player1 wins by 1

As you use start with higher numbers, the difference between the two players' scores increases.
Assuming L>1, then player1 will always win by at least 1 point.

*Answer.* Yes. The first player can win by at least one kilometer every time by going in the middle, meaning the halfway point of the first player's kilometer is at position $L/2$. After that, the first player would mirror the second player's moves. So if the second player takes $x$ to $x$+2 to the left of the middle kilometer, then the first player takes $(x + L/2)$ to $(x + 2 + L/2)$ on the right of the middle. The net effect is the first player can always guarantee to capture at least as much territory as the second player on the two sides of that middle kilometer. The first player wins by at least the kilometer of the first move.

# Using a Tree to Visualize the Problem



Note:
This one is the real root node.

Note:
the free space is always the same on the left and the right.

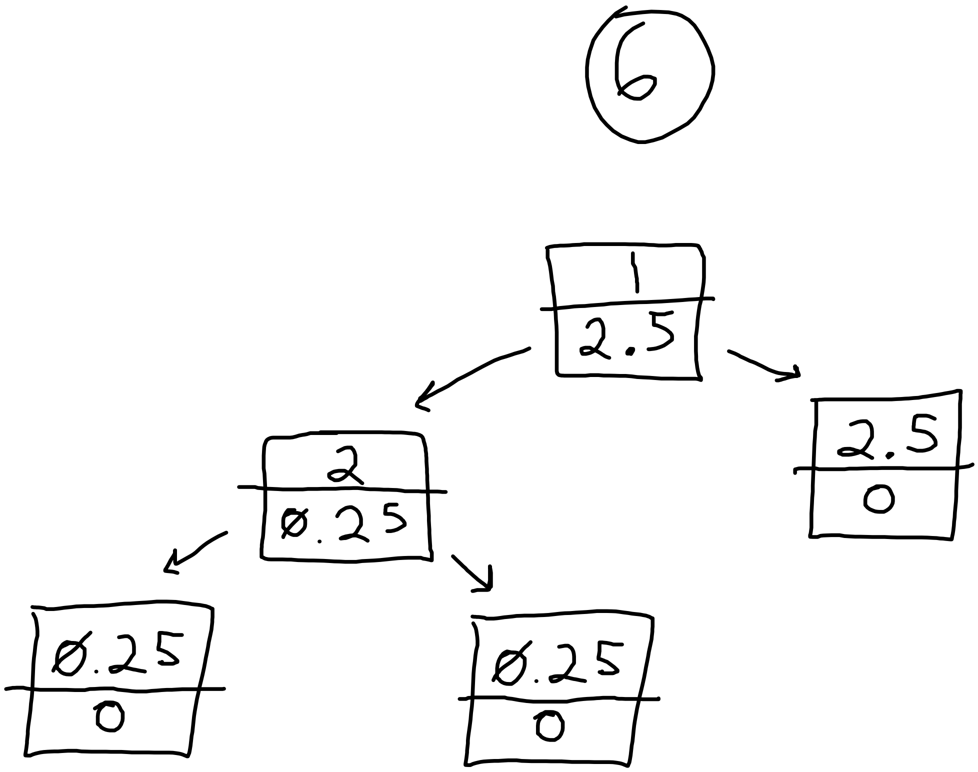
Here's a program (written in Python3) that prints information about a specific turn. It prints the space Used, and the remaining space Free.

L: length of the free space availiable in the segment
x: max amount of space you are allowed to take.

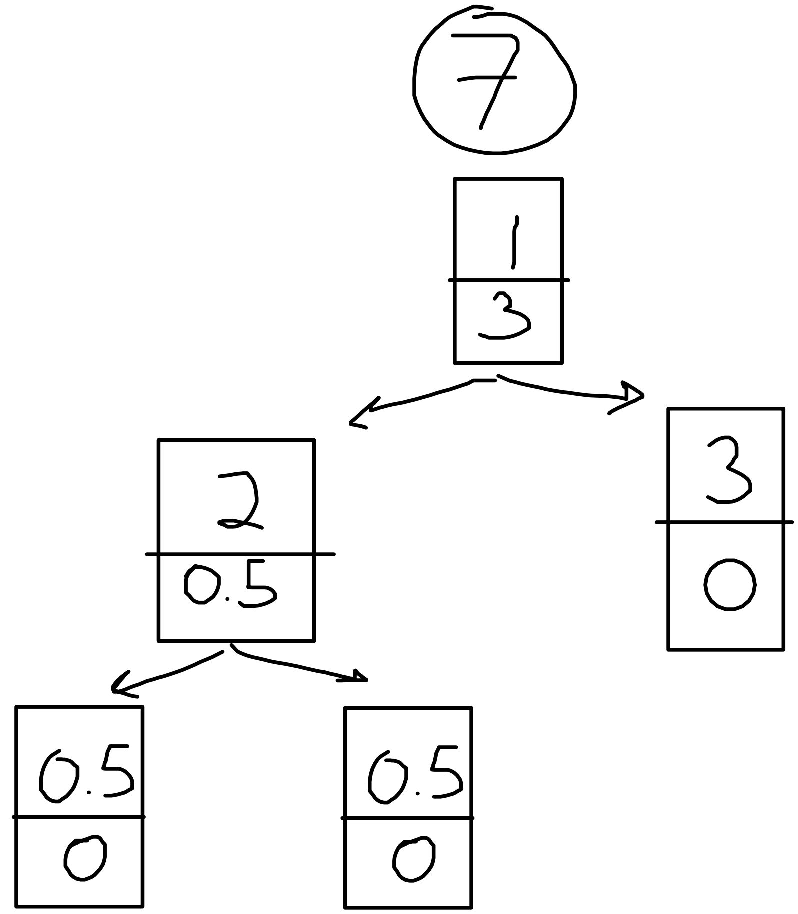
```
def f(L, x):
        if x > L:
                used = L
                free = 0
        else:
                used = x
                free = (L-x)/2
        print("used: {}, free: {}".format(used, free))
```

The information can be used again. where,
L is set to the new value for FREE, and x is incremented by 1.

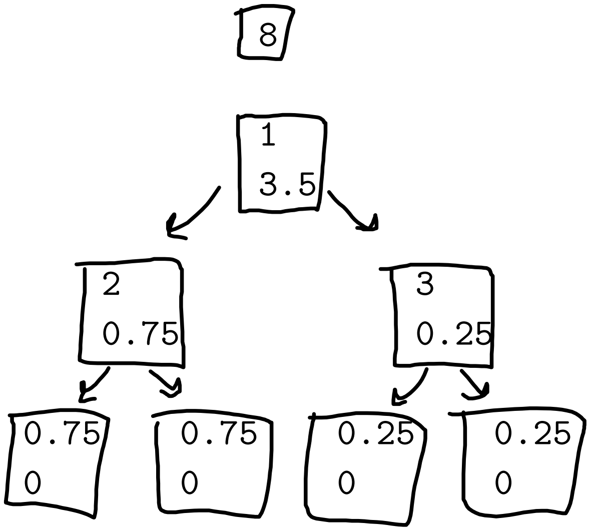
# Example Games visualized with the Tree.



```
>>> f(6, 1)
used: 1, free: 2.5

>>> f(2.5, 2)
used: 2, free: 0.25

>>> f(2.5, 3)
used: 2.5, free: 0

>>> f(0.25, 4)
used: 0.25, free: 0

>>> f(0.25, 5)
used: 0.25, free: 0
```



```
>>> f(7, 1)
used: 1, free: 3.0

>>> f(3, 2)
used: 2, free: 0.5

>>> f(3, 3)
used: 3, free: 0.0

>>> f(0.5, 4)
used: 0.5, free: 0

>>> f(0.5, 5)
used: 0.5, free: 0
```



```
>>> f(8, 1)
used: 1, free: 3.5
>>> f(3.5, 2)
used: 2, free: 0.75
>>> f(3.5, 3)
used: 3, free: 0.25
>>> f(0.75, 4)
used: 0.75, free: 0
>>> f(0.75, 5)
used: 0.75, free: 0
>>> f(0.25, 6)
used: 0.25, free: 0
>>> f(0.25, 7)
used: 0.25, free: 0
```

There are a few reasons why this tree-form is useful and/or insightful

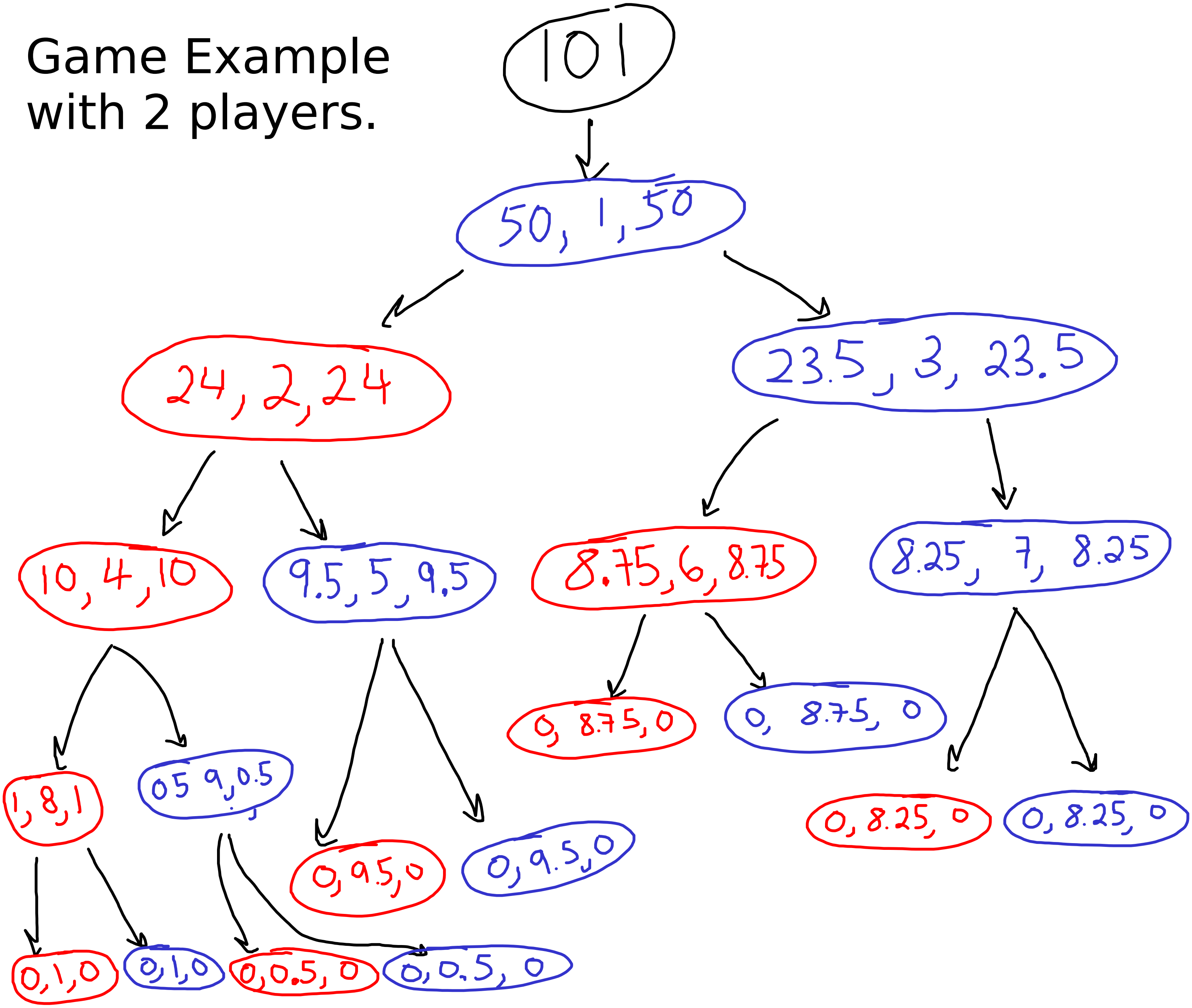
* Each node has either exactly 2 children, or exactly 0 children

* As L increases, new nodes appear in a way that always tries to fill up the tree. There are no nodes at depth N+1 unless depth N is full.

* The number of nodes = the number of moves to complete the game.

* Since each move is the "optimal move a given player can make", it is actually generalized away from the original 2 players. You can use exactly the same tree to analyze the game with k players.

# Game Example with 2 players.



Starting with L=101, player1 (blue) splits the segment directly into 2 halves. In this example, player2 (red) will always choose the leftmost portion if given the choice between 2 equal portions.

This picture shows the endgame result after the full game has been played out. Visit each of the nodes, and add up the sum of the middle numbers. That sum will be 101.

Scores

Player1 : $1 + 3 + 5 + 7 + 9 + 9.5 + 8.75 + 8.25 + 1 + 0.5 = \boxed{53}$

Player2 : $2 + 4 + 6 + 8 + 9.5 + 8.75 + 8.25 + 1 + 0.5 = \boxed{48}$

# Program to Build a Full Game Tree

```
class Node:
    def __init__(self, used, free):
        self.used = used
        self.free = free
```

Each "node" could be thought of as a single "move" made by some player. We don't care who.

```
def makeNode(L, x):
    if x>L:
        used = L
        free = 0
    else:
        used = x
        free = float(L - x) / 2
    return Node(used, free)
```

We saw this function in the previous sections, when we were calculating a single move.

```
def currentParentNode(tree):
    currentIndex = len(tree) - 1
    parentIndex = currentIndex >> 1
    return tree[parentIndex]
```

Finding the index of the parent uses the exact same technique as a binary heap.

```
def buildTree(L):
    tree = []
    tree.append(makeNode(L, 1))
    n = currentParentNode(tree)
    x = 2
    while (n.free > 0):
        tree.append(makeNode(n.free, x))
        tree.append(makeNode(n.free, x + 1))
        n = currentParentNode(tree)
        x += 2
    return tree
```

Since the tree acts like a heap, we can represent it using an array.

Each move is appended onto the array. After the initial move is made, all of the other moves are done 2 at a time. We can do this because each node has either 2 or 0 children.

Once all the parent nodes are used up, we have completed the game tree.

```
def getScores(tree, nPlayers):
    scores = [0] * nPlayers
    for i, node in enumerate(tree):
        player = (i % nPlayers)
        points = node.used
        scores[player] += points
    return scores
```

After the game tree has already been built, we can use the tree array and tag them with a player.

This function just tallies up each of the scores, given some number of players.

# Program Example - Game where L=101

Now that the program has been written, let's confirm that it works with the handwritten example from L=101

( honestly, I can't believe I built the whole tree by hand the first time around, but whatever )

Inputs:

L = 101
nPlayers = 2

Program Output:

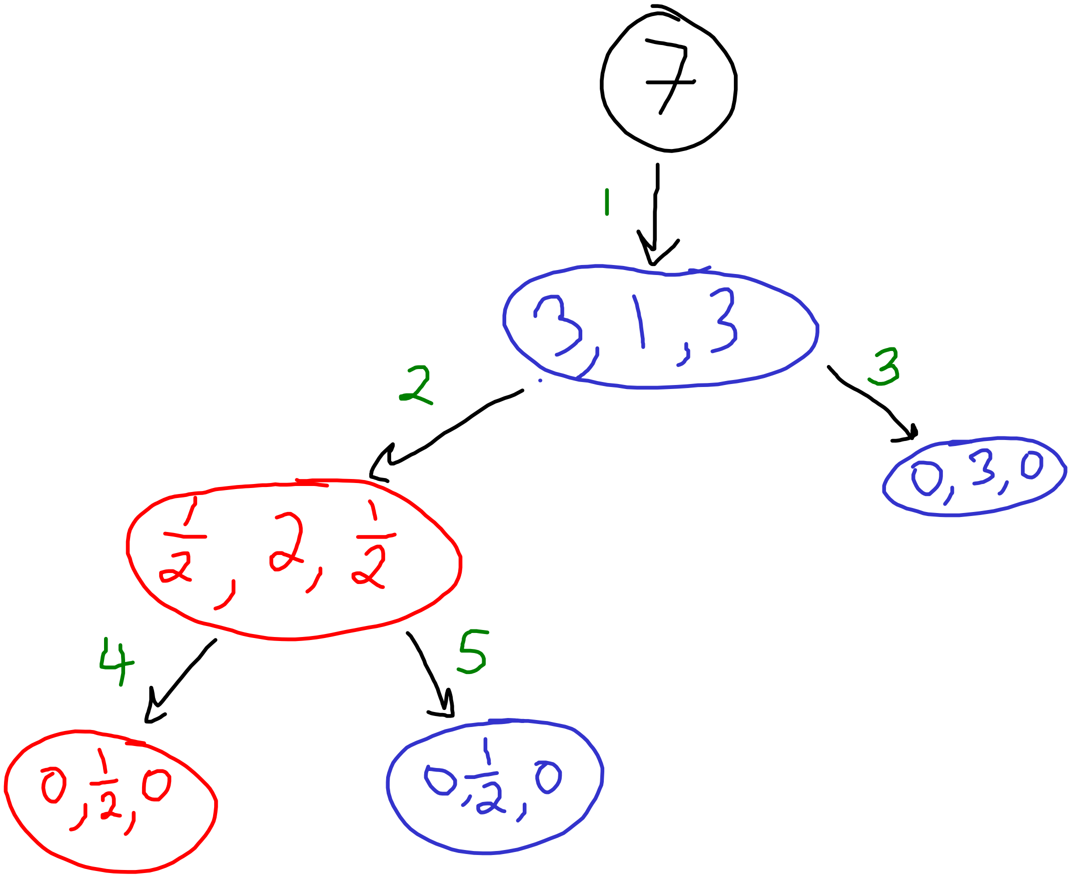
```
Turns:
[
    (used: 1, free: 50.0),
    (used: 2, free: 24.0),
    (used: 3, free: 23.5),
    (used: 4, free: 10.0),
    (used: 5, free: 9.5),
    (used: 6, free: 8.75),
    (used: 7, free: 8.25),
    (used: 8, free: 1.0),
    (used: 9, free: 0.5),
    (used: 9.5, free: 0),
    (used: 9.5, free: 0),
    (used: 8.75, free: 0),
    (used: 8.75, free: 0),
    (used: 8.25, free: 0),
    (used: 8.25, free: 0),
    (used: 1.0, free: 0),
    (used: 1.0, free: 0),
    (used: 0.5, free: 0),
    (used: 0.5, free: 0)
]

Scores:
[53.0, 48.0]
```

← omG!
yay!

It Worked =D

**Upstart 1.** Characterize situations in which the first player can guarantee to win by more than one kilometer or prove it cannot be done.

Player 1 will win by more than 1 point
for all games with L>5 and k=2



Scores

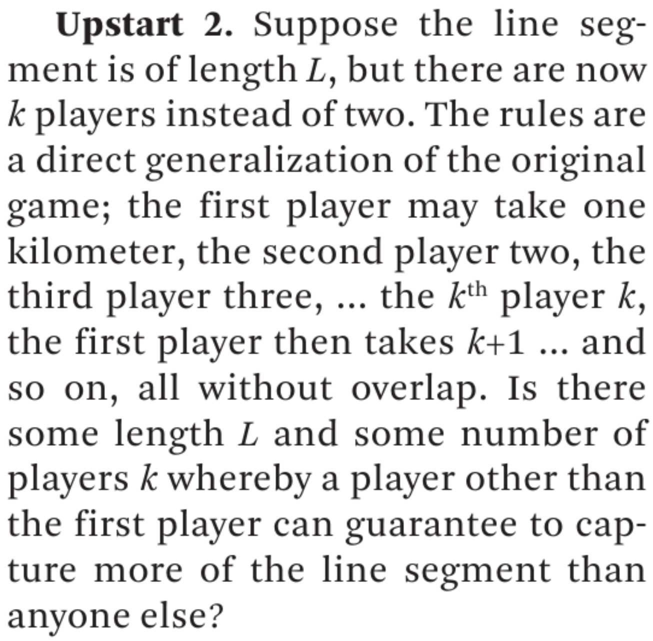
Player1: 1 + 3 + 0.5 = 4.5

Player2: 2 + 0.5 = 2.5

Player1
wins by
2 points

More evidence of this is presented in Summary of Scores,
which is part of the next question.

This is only the tip of iceberg.



**Upstart 2.** Suppose the line segment is of length $L$, but there are now $k$ players instead of two. The rules are a direct generalization of the original game; the first player may take one kilometer, the second player two, the third player three, ... the $k^{th}$ player $k$, the first player then takes $k$+1 ... and so on, all without overlap. Is there some length $L$ and some number of players $k$ whereby a player other than the first player can guarantee to capture more of the line segment than anyone else?

Short answer:

Yes, there are many cases.

If each player follows the same move pattern, then there are many cases where different players win the game.

Here are some of the summaries for small values of L and k. More on the next page. "Maxed out on players" means that any additional players would never play a turn. "Wins" means "The winners". Notice how there is a tie when L=5 and k=3.

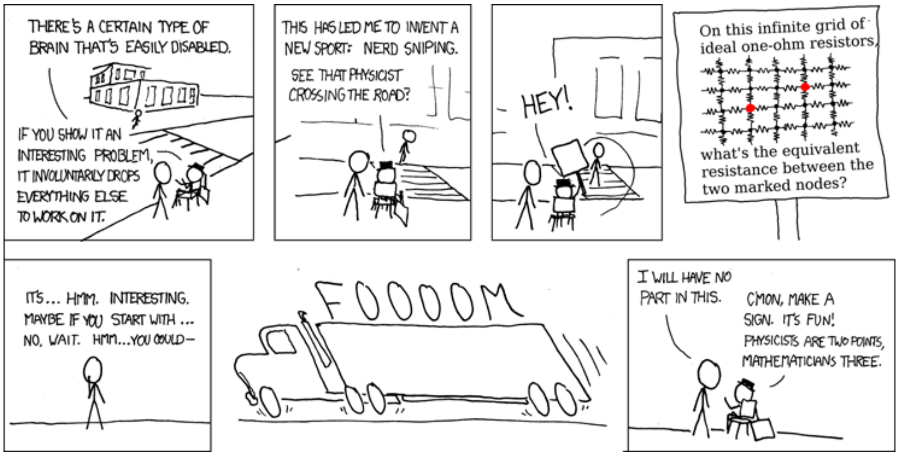
```
L: 5, k: 2      Wins: [0],      Scores: [3.0, 2]
L: 5, k: 3      Wins: [1, 2],   Scores: [1, 2, 2.0]
Maxed out on players.
L: 6, k: 2      Wins: [0],      Scores: [3.75, 2.25]
L: 6, k: 3      Wins: [2],      Scores: [1.25, 2.25, 2.5]
L: 6, k: 4      Wins: [2],      Scores: [1.25, 2, 2.5, 0.25]
L: 6, k: 5      Wins: [2],      Scores: [1, 2, 2.5, 0.25, 0.25]
Maxed out on players.
L: 7, k: 2      Wins: [0],      Scores: [4.5, 2.5]
L: 7, k: 3      Wins: [2],      Scores: [1.5, 2.5, 3]
L: 7, k: 4      Wins: [2],      Scores: [1.5, 2, 3, 0.5]
L: 7, k: 5      Wins: [2],      Scores: [1, 2, 3, 0.5, 0.5]
Maxed out on players.
L: 8, k: 2      Wins: [0],      Scores: [5.0, 3.0]
L: 8, k: 3      Wins: [2],      Scores: [2.0, 2.75, 3.25]
L: 8, k: 4      Wins: [2],      Scores: [1.75, 2.25, 3.25, 0.75]
L: 8, k: 5      Wins: [2],      Scores: [1.25, 2.25, 3, 0.75, 0.75]
L: 8, k: 6      Wins: [2],      Scores: [1.25, 2, 3, 0.75, 0.75, 0.25]
L: 8, k: 7      Wins: [2],      Scores: [1, 2, 3, 0.75, 0.75, 0.25, 0.25]
Maxed out on players.
L: 9, k: 2      Wins: [0],      Scores: [5.5, 3.5]
L: 9, k: 3      Wins: [2],      Scores: [2.5, 3.0, 3.5]
L: 9, k: 4      Wins: [2],      Scores: [2.0, 2.5, 3.5, 1.0]
L: 9, k: 5      Wins: [2],      Scores: [1.5, 2.5, 3, 1.0, 1.0]
L: 9, k: 6      Wins: [2],      Scores: [1.5, 2, 3, 1.0, 1.0, 0.5]
L: 9, k: 7      Wins: [2],      Scores: [1, 2, 3, 1.0, 1.0, 0.5, 0.5]
Maxed out on players.
```

# Summary of Scores for Different Cases

```
L: 10, k: 2    Wins: [0],     Scores: [6.0, 4.0]
L: 10, k: 3    Wins: [2],     Scores: [3.0, 3.25, 3.75]
L: 10, k: 4    Wins: [2],     Scores: [2.25, 2.75, 3.75, 1.25]
L: 10, k: 5    Wins: [2],     Scores: [1.75, 2.75, 3, 1.25, 1.25]
L: 10, k: 6    Wins: [2],     Scores: [1.75, 2, 3, 1.25, 1.25, 0.75]
L: 10, k: 7    Wins: [2],     Scores: [1, 2, 3, 1.25, 1.25, 0.75, 0.75]
Maxed out on players.
L: 11, k: 2    Wins: [0],     Scores: [6.5, 4.5]
L: 11, k: 3    Wins: [2],     Scores: [3.5, 3.5, 4.0]
L: 11, k: 4    Wins: [2],     Scores: [2.5, 3.0, 4.0, 1.5]
L: 11, k: 5    Wins: [1, 2],  Scores: [2.0, 3.0, 3, 1.5, 1.5]
L: 11, k: 6    Wins: [2],     Scores: [2.0, 2, 3, 1.5, 1.5, 1.0]
L: 11, k: 7    Wins: [2],     Scores: [1, 2, 3, 1.5, 1.5, 1.0, 1.0]
Maxed out on players.
L: 12, k: 2    Wins: [0],     Scores: [7.0, 5.0]
L: 12, k: 3    Wins: [2],     Scores: [4.0, 3.75, 4.25]
L: 12, k: 4    Wins: [2],     Scores: [2.75, 3.25, 4.25, 1.75]
L: 12, k: 5    Wins: [1],     Scores: [2.25, 3.25, 3, 1.75, 1.75]
L: 12, k: 6    Wins: [2],     Scores: [2.25, 2, 3, 1.75, 1.75, 1.25]
L: 12, k: 7    Wins: [2],     Scores: [1, 2, 3, 1.75, 1.75, 1.25, 1.25]
Maxed out on players.
L: 13, k: 2    Wins: [0],     Scores: [7.5, 5.5]
L: 13, k: 3    Wins: [0, 2],  Scores: [4.5, 4.0, 4.5]
L: 13, k: 4    Wins: [2],     Scores: [3.0, 3.5, 4.5, 2.0]
L: 13, k: 5    Wins: [1],     Scores: [2.5, 3.5, 3, 2.0, 2.0]
L: 13, k: 6    Wins: [2],     Scores: [2.5, 2, 3, 2.0, 2.0, 1.5]
L: 13, k: 7    Wins: [2],     Scores: [1, 2, 3, 2.0, 2.0, 1.5, 1.5]
Maxed out on players.
L: 14, k: 2    Wins: [0],     Scores: [8.0, 6.0]
L: 14, k: 3    Wins: [0],     Scores: [5.0, 4.25, 4.75]
L: 14, k: 4    Wins: [2],     Scores: [3.25, 3.75, 4.75, 2.25]
L: 14, k: 5    Wins: [1],     Scores: [2.75, 3.75, 3, 2.25, 2.25]
L: 14, k: 6    Wins: [2],     Scores: [2.75, 2, 3, 2.25, 2.25, 1.75]
L: 14, k: 7    Wins: [2],     Scores: [1, 2, 3, 2.25, 2.25, 1.75, 1.75]
Maxed out on players.
L: 15, k: 2    Wins: [0],     Scores: [8.5, 6.5]
L: 15, k: 3    Wins: [0],     Scores: [5.5, 4.5, 5.0]
L: 15, k: 4    Wins: [2],     Scores: [3.5, 4.0, 5.0, 2.5]
L: 15, k: 5    Wins: [1],     Scores: [3.0, 4.0, 3, 2.5, 2.5]
L: 15, k: 6    Wins: [0, 2],  Scores: [3.0, 2, 3, 2.5, 2.5, 2.0]
L: 15, k: 7    Wins: [2],     Scores: [1, 2, 3, 2.5, 2.5, 2.0, 2.0]
Maxed out on players.
```

# Congratulations!

You've just won 3 xkcd points!


THERE'S A CERTAIN TYPE OF BRAIN THAT'S EASILY DISABLED.
IF YOU SHOW IT AN INTERESTING PROBLEM, IT INVOLUNTARILY DROPS EVERYTHING ELSE TO WORK ON IT.
THIS HAS LED ME TO INVENT A NEW SPORT: NERD SNIPING.
SEE THAT PHYSICIST CROSSING THE ROAD?
HEY!
On this infinite grid of ideal one-ohm resistors, what's the equivalent resistance between the two marked nodes?
IT'S... HMM. INTERESTING. MAYBE IF YOU START WITH ... NO, WAIT. HMM...YOU COULD—
FOOOOM
I WILL HAVE NO PART IN THIS.
C'MON, MAKE A SIGN. IT'S FUN! PHYSICISTS ARE TWO POINTS, MATHEMATICIANS THREE.

https://xkcd.com/356/