You can watch it work here
http://www.csit.parkland.edu/~kurban/permanent/labs/recursivegame/runme.cgi

Thinking Recursively

on a simple game

Requirements

Create file in python with a **comment** containing the academic honesty pledge as shown below. Add another, separate comment to the file containing your name

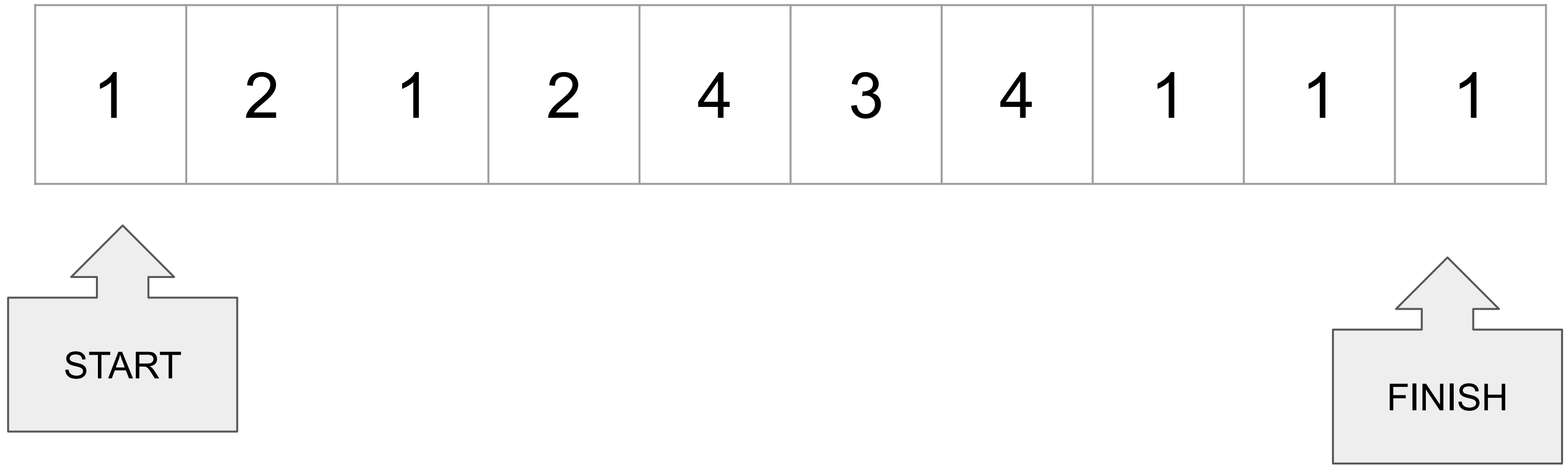
- Write a python program that prints the solution to the game board supplied in a textarea as described on a later slide.
- Attach your python source in using the dropbox link in cobra learning. (I don't need any html files)
- **Put a link to your running file in the message area**

```
# I honor Parkland's core values by affirming that I have
# followed all academic integrity guidelines for this work.

# your name
```

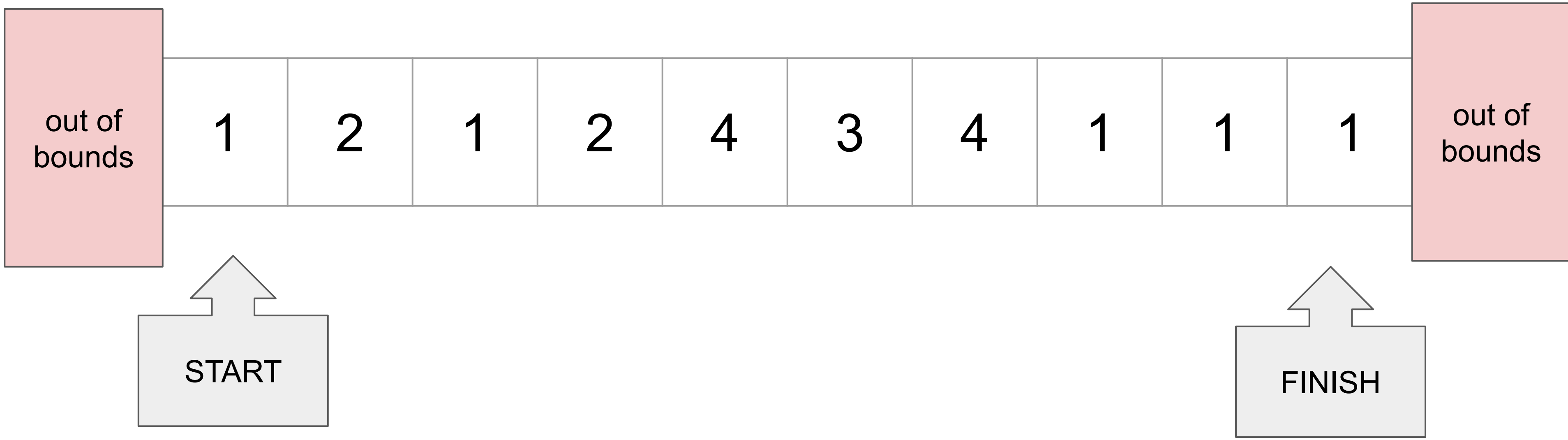
The game.

You will be given a series of integers in the text area. Think of them as a game board with a start and a finish



The game.

To win the game, you need to start at the beginning and end exactly on the finish. You can only move the number of spaces listed in the cell, but you can choose to move left or right as long as you stay on the board. It doesn't matter what's in the last cell, you'll win if you get there.



The game.

So for this game, the end is at position == [9]. You always start with a position == 0 and end with a position == len()-1

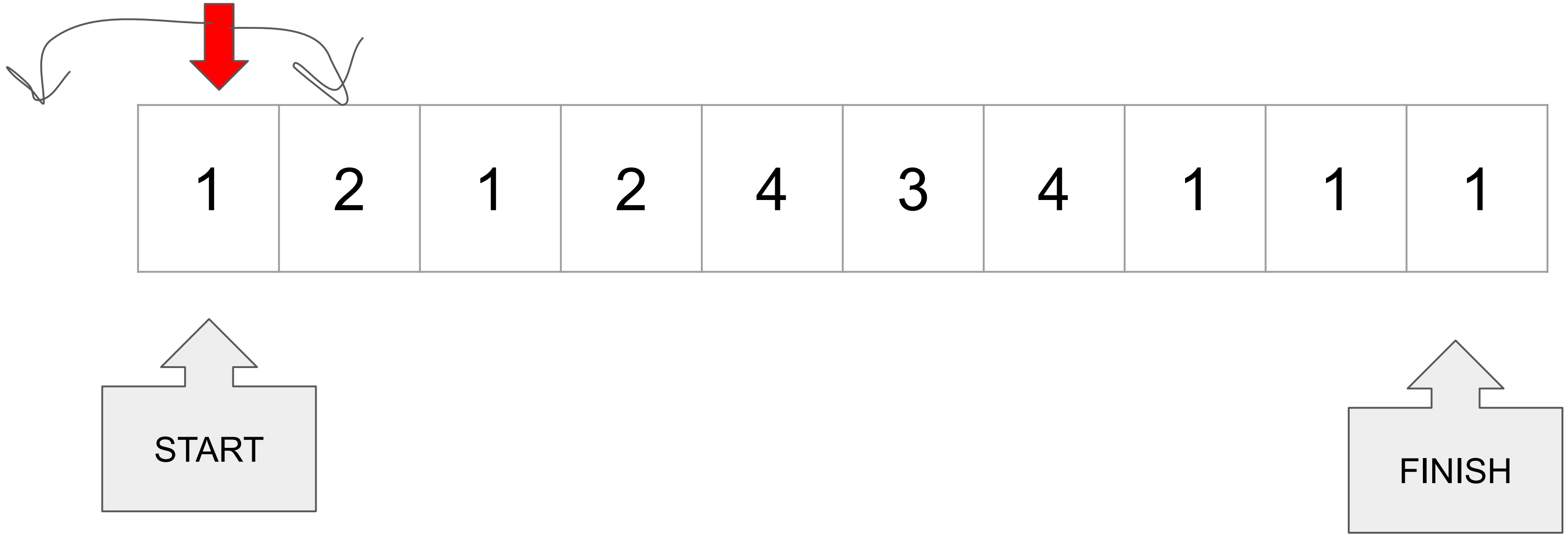
Positions									
0	1	2	3	4	5	6	7	8	9
1	2	1	2	4	3	4	1	1	1

START

FINISH

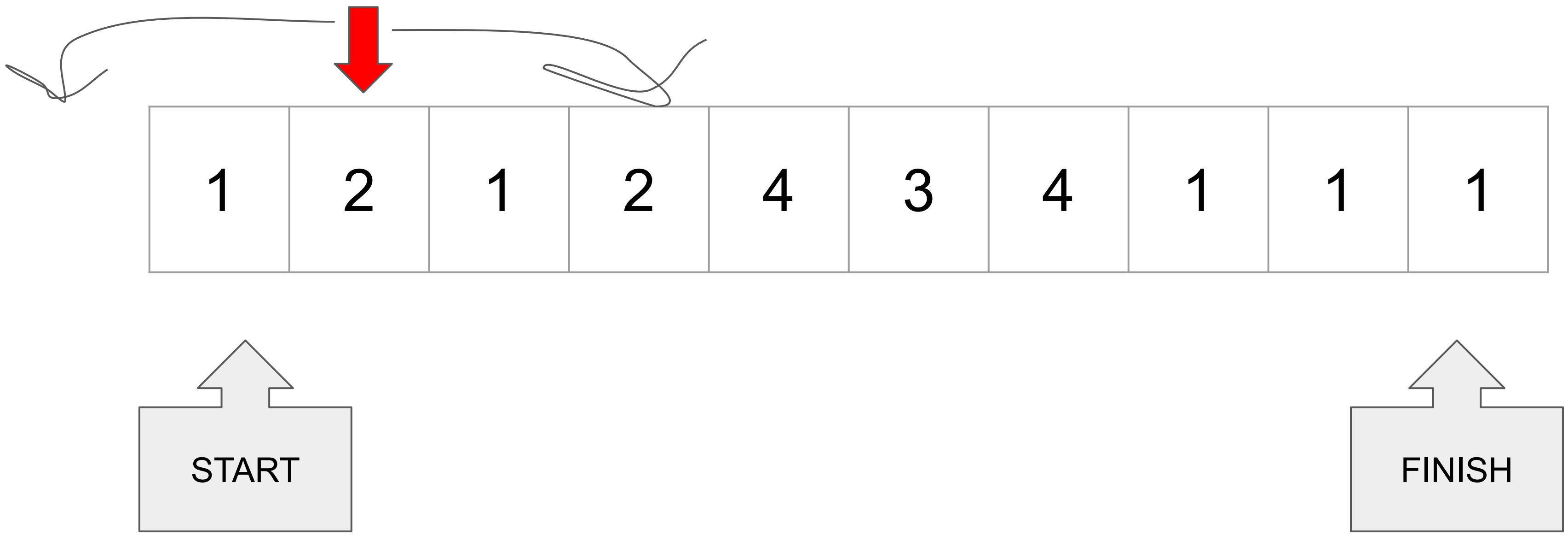
Move 1

We start at position == 0. At this spot, we can only move to the right if we want to 'win' since 1 to the left will 'lose'



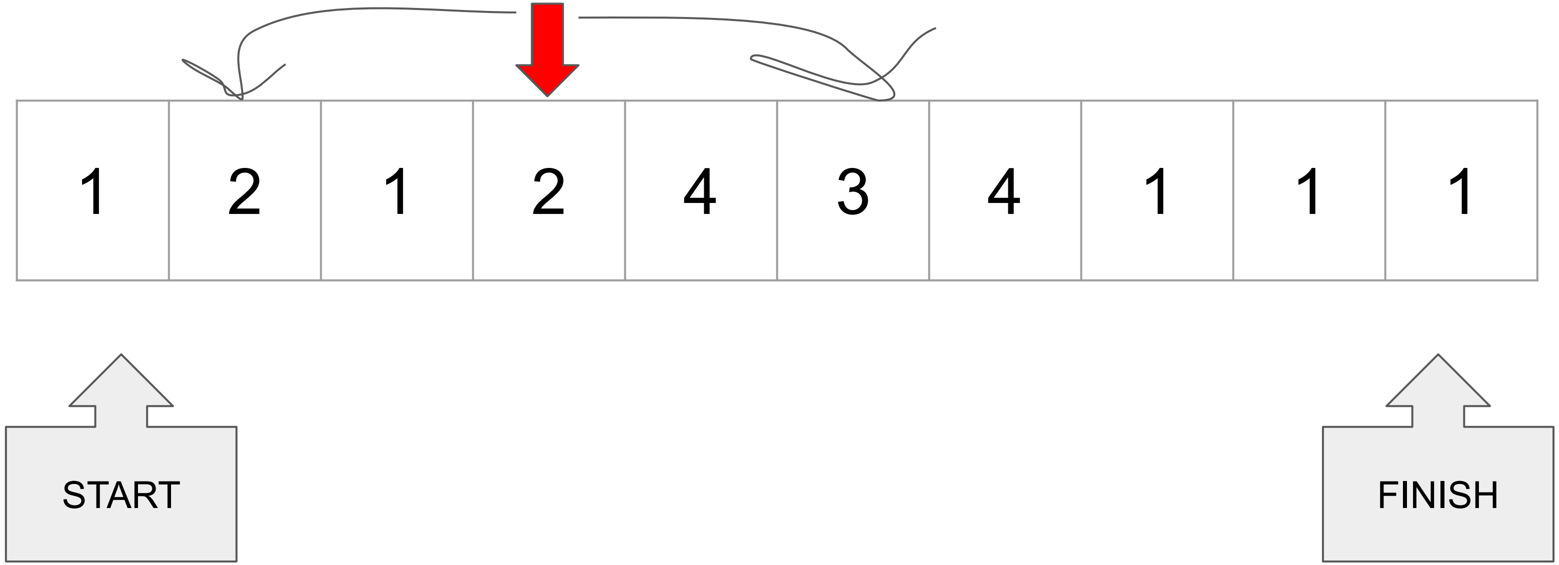
Move 2

At position == 1. At this spot, we can also only move to the right if we want to 'win' since 2 to the left will 'lose'



Move 3

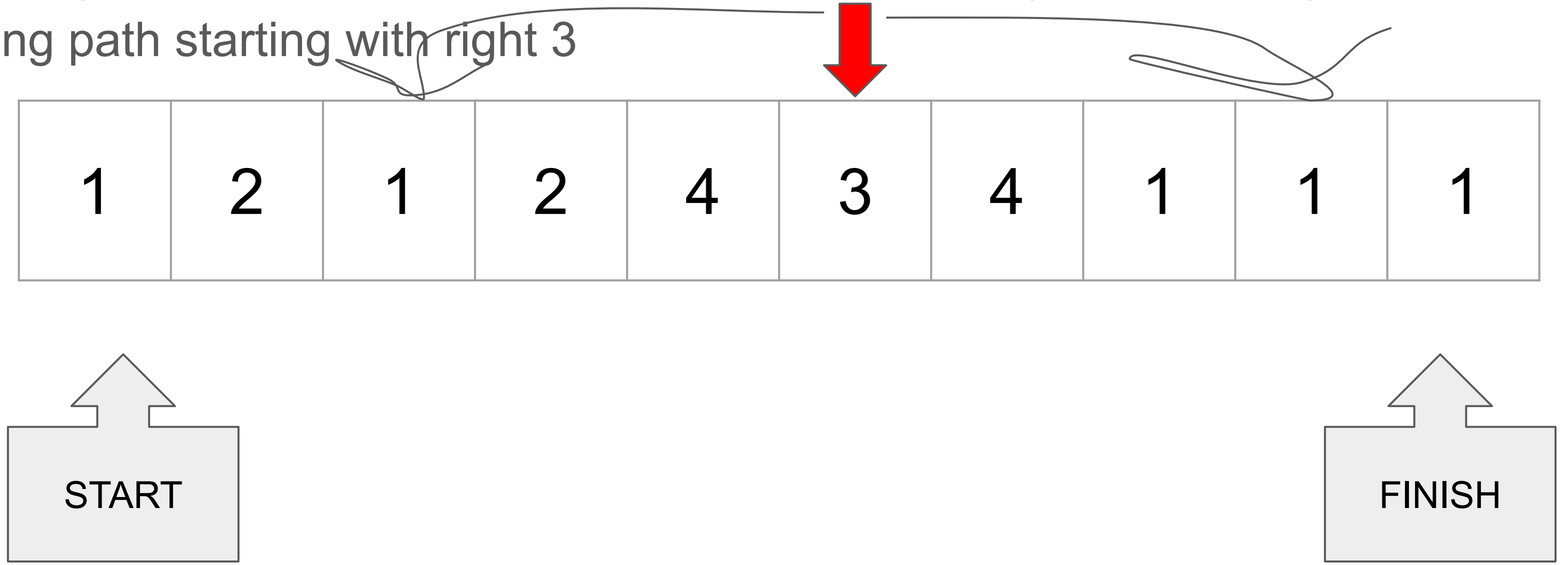
At this spot, we can move to the left or right. Moving to the left will bring us to a spot we've already been (HINT: keep track of where you've been!!!) and won't help us win.



Move 4

At this spot, we can move to the left or right. Both bring up to spot that may or may not win … we don't know. This is where 'recursion' is more obvious.

A winning path from here to the end is either a winning path starting with left 3 or a winning path starting with right 3



Other things to know

- You may have a 0 in a cell. This will mean you can't win this way if you need to move from there. (If it's the last cell, you've won)
- There are numerous base cases that stop the recursion
    - 'winning' (reaching the last cell) should also stop the recursion.
    - Too high (off the end of the list)
    - Too low (off the beginning of the list)
    - A value of zero in the cell
    - The cell has been visited before
- When you have 'won' you still need to print the 'path' that wins in the correct order. If you just print it after the recursive call, it will come out backwards, so put it in a list and print that list out backwards after all the recursive calls are done. This is when your return True.
- Any correct solution is fine, it doesn't need to be the shortest.

There might be no solution.

The recursion may return with no solution, like if the first cell is a zero. You'll need to report if there is no solution. HINT: If this is the case, there shouldn't be anything in your output list.

Input (not the example above)

I'll put in a space separated series of integers like:

1 2 1 2 3 4 3 1 1 1

And you'll print out the positions to the winning spot:

0 1 3 5 9

Input (another example)

I'll put in a space separated series of integers like:

1 4 3 5 2 1 5 2 7 0 4 0

Results

0 → -1, 1

0 → 1 → -3, 5

0 → 1 → 5 → 4, 6

0 → 1 → 5 → 6 → 1, 11

0 → 1 → 5 → 6 → 11, WINNING PATH

How to think of it recursively

Define the problem in terms of itself … but with a smaller problem and a trivial solution(s).

Use path length as the smaller solution.

there is a path to win the game from the current position IF there is a path to win from the right OR there is a path to win to the left.

```
def pathCanWin( ...current... ):
    # code to return True or False if you can win or can't win directly goes here
    if pathCanWin(...left…) or pathCanWin(...right…):
        return True
    else
        return False
```

Some to try

`10 8 6 4 2 6 1 3 5 7 9 5 1 1 2 1 1` `0 --> 10 --> 1 --> 9 --> 16 --> Win or 0, 10, 1, 9, 2, 8, 3, 7, 4, 6, 5, 11, 16`

`10 8 6 4 2 6 1 3 5 7 9 0` `0 --> 10 --> 1 --> 9 --> 2 --> 8 --> 3 --> 7 --> 4 --> 6 --> 5 --> 11 --> Win`