Algorithm Design & Problem Solving: Tower of Hanoi

Contents

The Tower of Hanoi

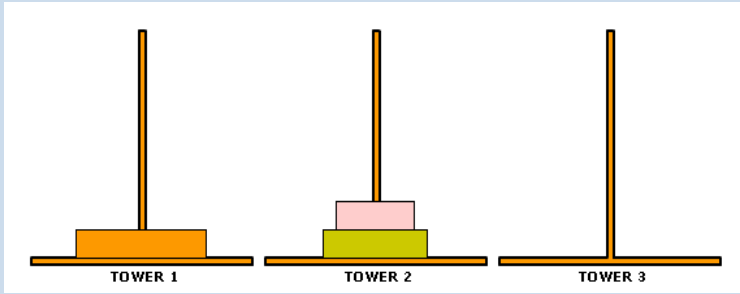
- **Rules**
  - Move all disks to Tower 3
  - Only one disk can be moved at a time
  - A disk can never be put on a smaller disk

The Tower of Hanoi

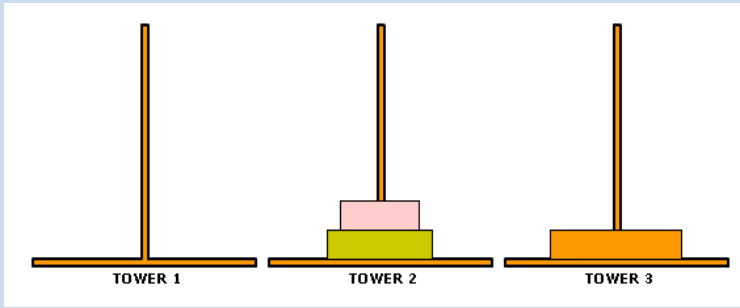
- What is the problem?
  - Move the largest disk, disk2, to Tower 3
  - Move the middle disk, disk1, to Tower 3
  - Move smallest disk, disk0, to Tower 3

The Tower of Hanoi

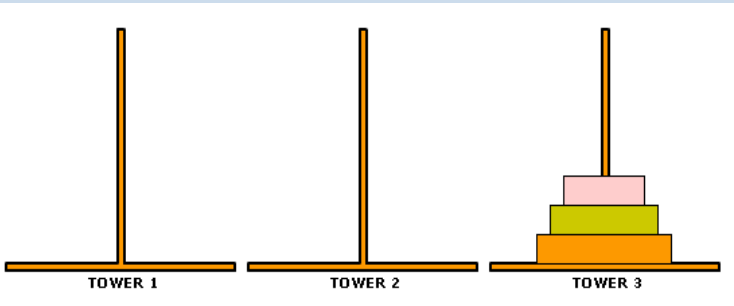
- To move disk0, disk1 and disk2 from 1 to 3:
  1. Move disk0 and disk1 from 1 to 2,
     i. Move disk0 from 1 to 3, and
     ii. Move disk1 from 1 to 2, and
     iii. Move disk0 from 3 to 2



  2. Move disk2 from 1 to 3



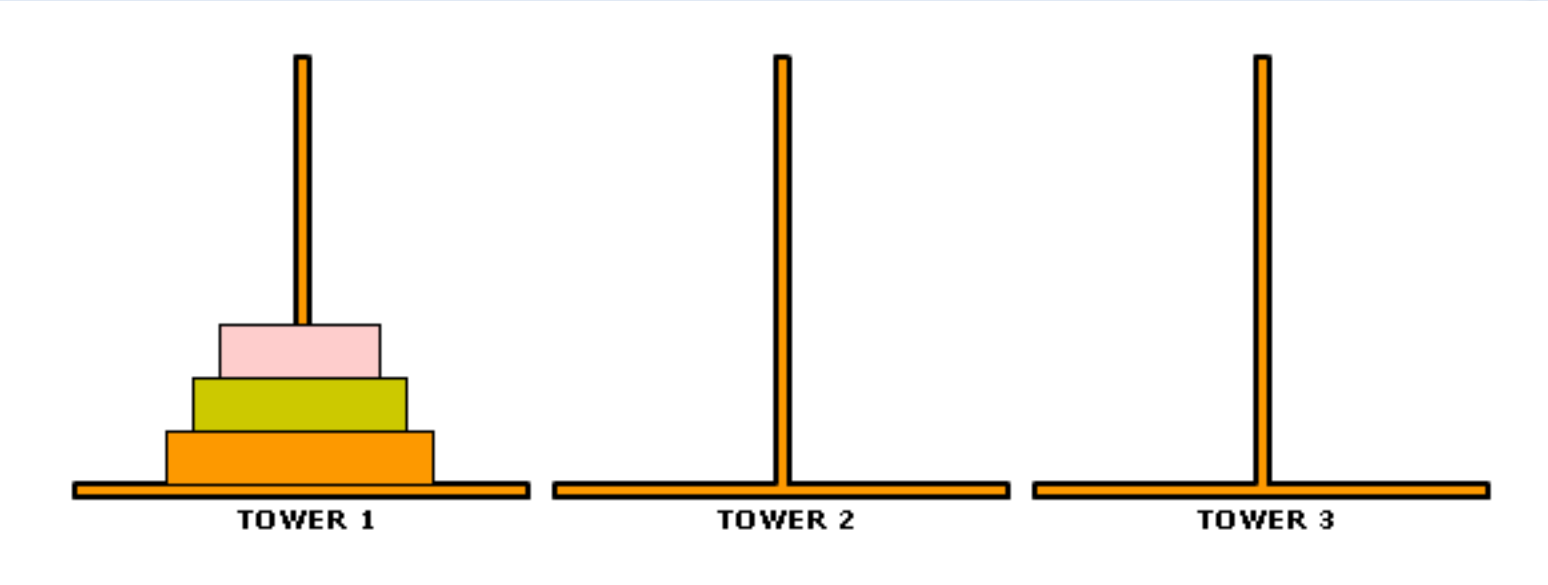
  3. Move disk0 and disk1 from 2 to 3
     i. Move disk0 from 2 to 1, and
     ii. Move disk1 from 2 to 3, and
     iii. Move disk0 from 1 to 3



Towers of Hanoi

- Why is this suitable for **recursion**?
  - Because there is a base case
  - The problem is iteratively getting smaller

The Tower of Hanoi



Source

Spare

Dest

Towers of Hanoi: A recursive algorithm

Which disk am I moving?

moveTower (disks, source, dest, spare)

From which tower?

To which tower?

In your TOH handout from yesterday, fill up the column "Relevant function call"

Towers of Hanoi: A recursive algorithm….. Start of

moveTower (disks, source, dest, spare)

```
If disk = 0
        Move disk from source to dest
```

Base case: if it is the smallest disk then move it

Towers of Hanoi: A recursive algorithm

moveTower (disks, source, dest, spare)

If disk = 0

Move disk from source to dest

else

moveTower (disk-1, source, spare, dest)

move disk from source to dest

moveTower (disk-1, spare, dest, source)

Base case: only small disk can be moved

Towers of Hanoi: A recursive algorithm

```
moveTower (disks, source, dest, spare)
        If disk = 0
                Move disk from source to dest
        else
                moveTower (disk-1, source, spare, dest)
                move disk from source to dest
                moveTower (disk-1, spare, dest, source)
```

This moves the big & medium disks

Towers of Hanoi: A recursive algorithm

moveTower (disks, source, dest, spare)

If disk = 0

Move disk from source to dest

else

moveTower (disk-1, source, spare, dest)

Recursive calls

move disk from source to dest

moveTower (disk-1, spare, dest, source)

Towers of Hanoi: A recursive algorithm

```
moveTower (disks, source, dest, spare)
        If disk = 0
                Move disk from source to dest
        else
                moveTower (disk-1, source, spare, dest)
                move disk from source to dest
                moveTower (disk-1, spare, dest, source)
```

Note how the towers change position in the call

Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

In your TOH handout, fill up the column "Line of code that move takes place"

Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

moveTower(2,Tower1,Tower3,Tower2)

Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

moveTower(2,Tower1,Tower3,Tower2)

TOP OF STACK

Towers of Hanoi: A recursive algorithm

The call was made at line 5

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

5

Call Stack

moveTower(2,Tower1,Tower3,Tower2)
moveTower(1,Tower1,Tower2,Tower3)

Towers of Hanoi: A recursive algorithm

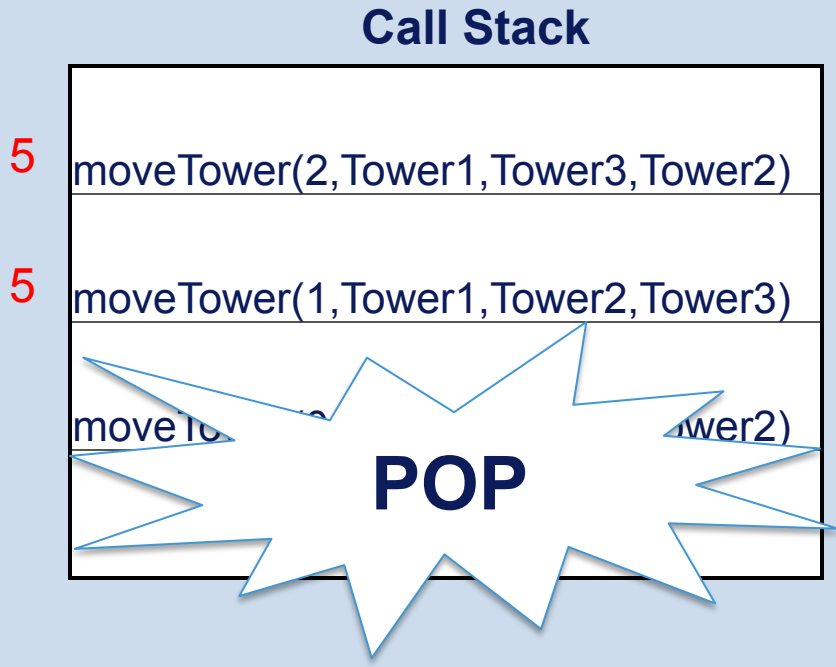
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

5	moveTower(2,Tower1,Tower3,Tower2)
5	moveTower(1,Tower1,Tower2,Tower3)
	moveTower(0,Tower1,Tower3,Tower2)

Towers of Hanoi: A recursive algorithm

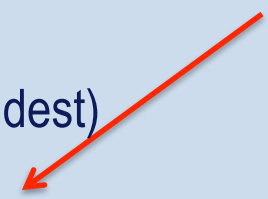
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



1(i) Move disk0 from 1 to 3

Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



Call Stack

5	moveTower(2,Tower1,Tower3,Tower2)
5	moveTower(1,Tower1,Tower2,Tower3

1(ii) Move disk1 from 1 to 2

Towers of Hanoi: A recursive algorithm

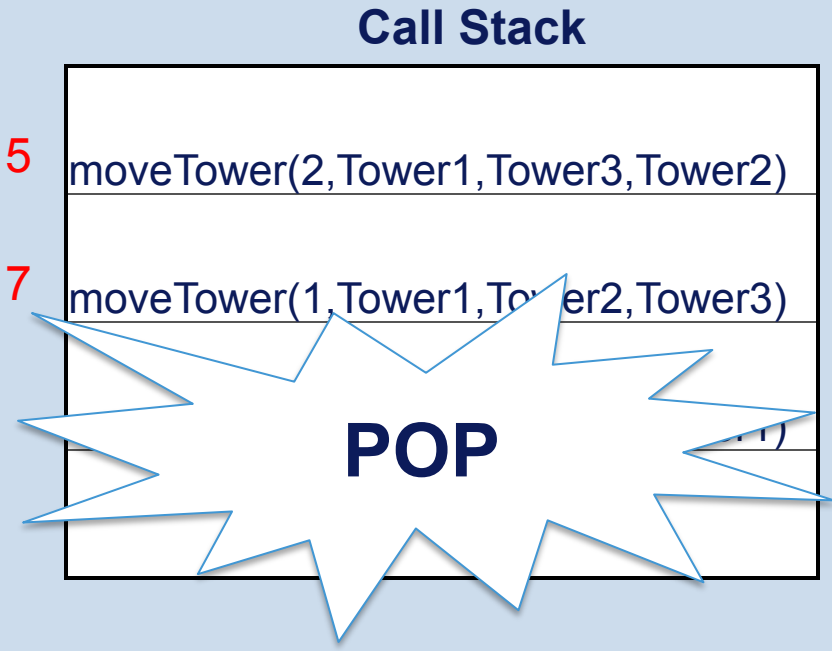
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

5	moveTower(2,Tower1,Tower3,Tower2)
7	moveTower(1,Tower1,Tower2,Tower3)
	moveTower(0,Tower3,Tower2,Tower1)

Towers of Hanoi: A recursive algorithm

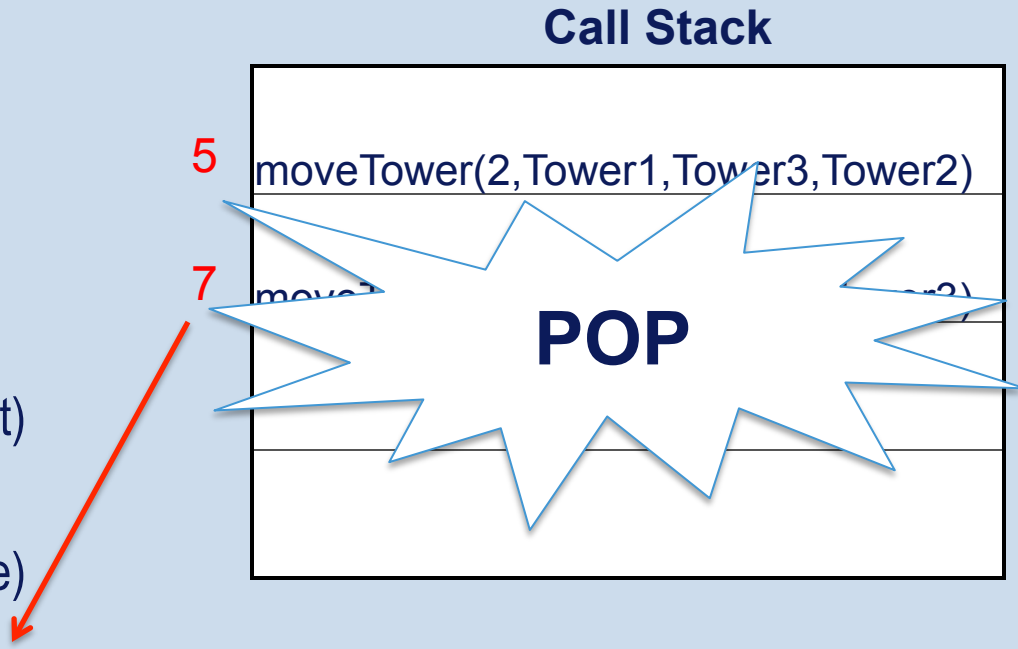
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



1(iii) Move disk0 from 3 to 2

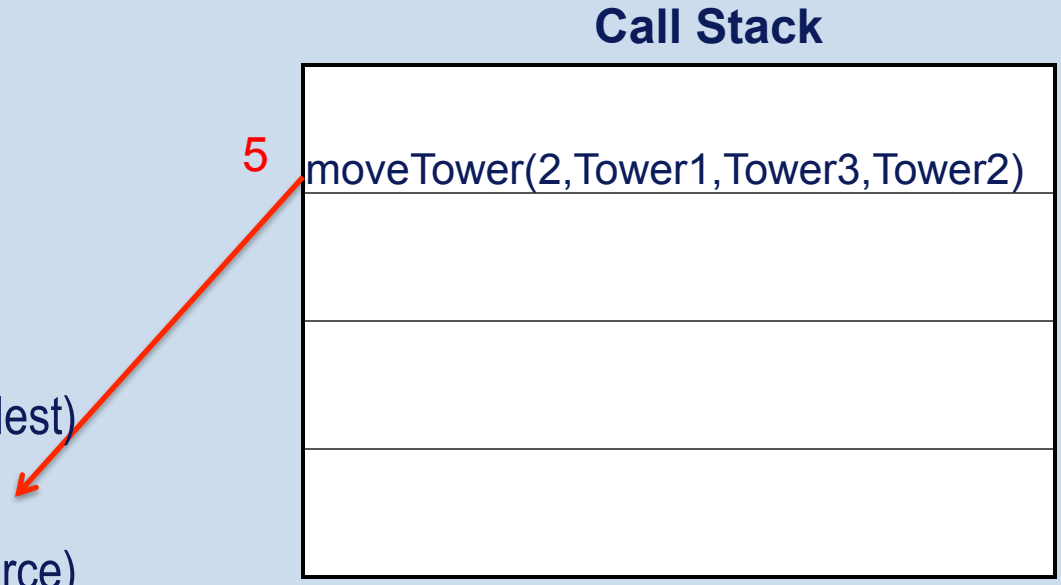
Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



2. Move disk2 from 1 to 3

Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

7	moveTower(2,Tower1,Tower3,Tower2)
	moveTower(1,Tower2,Tower3,Tower1)

Towers of Hanoi: A recursive algorithm

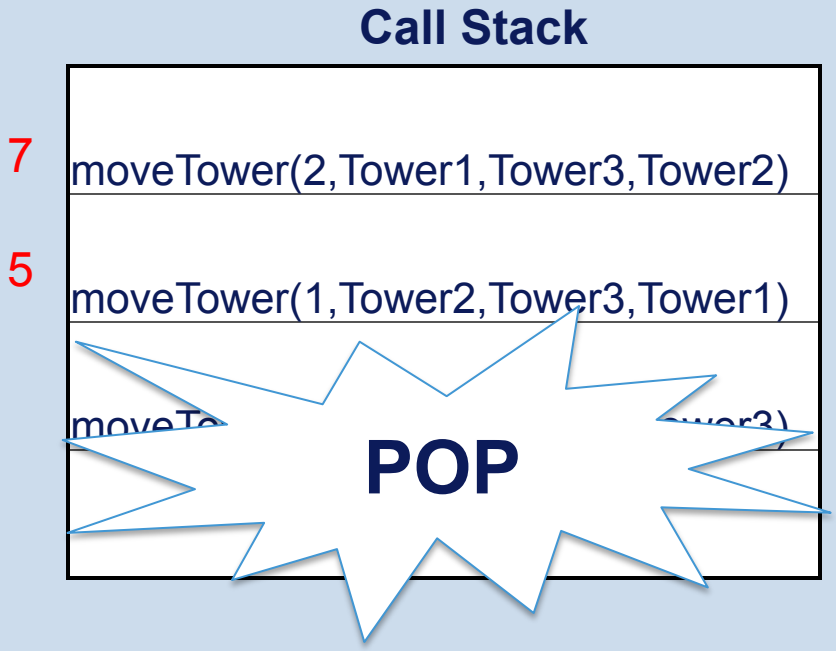
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

7	moveTower(2,Tower1,Tower3,Tower2)
5	moveTower(1,Tower2,Tower3,Tower1)
	moveTower(0,Tower2,Tower1,Tower3)

Towers of Hanoi: A recursive algorithm

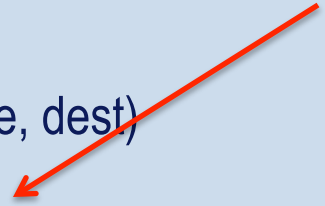
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



3(i) Move disk0 from 2 to 1

Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



Call Stack

7 moveTower(2,Tower1,Tower3,Tower2)
5 moveTower(1,Tower2,Tower3,Tower1)

3(ii) Move disk1 from 2 to 3

Towers of Hanoi: A recursive algorithm

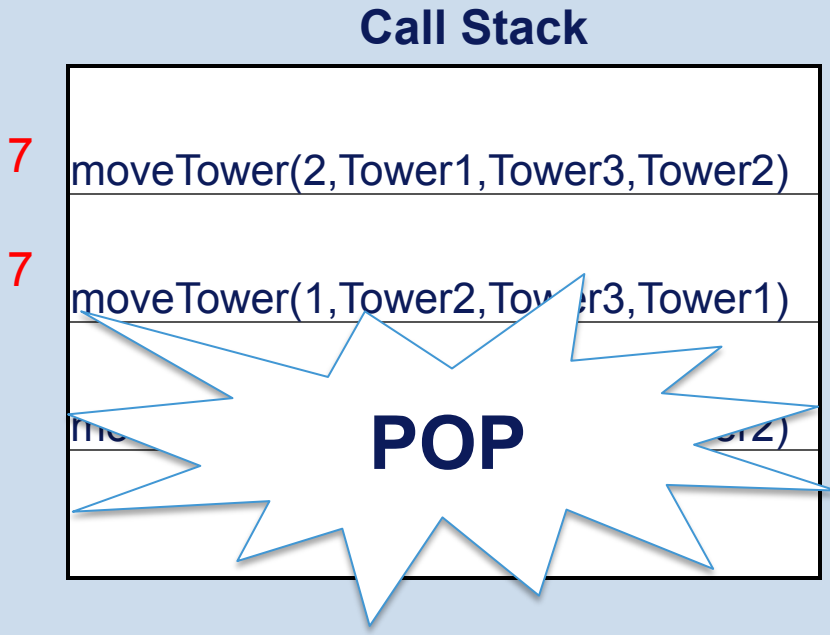
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

7	moveTower(2,Tower1,Tower3,Tower2)
7	moveTower(1,Tower2,Tower3,Tower1)
	moveTower(0,Tower1,Tower3,Tower2)

Towers of Hanoi: A recursive algorithm

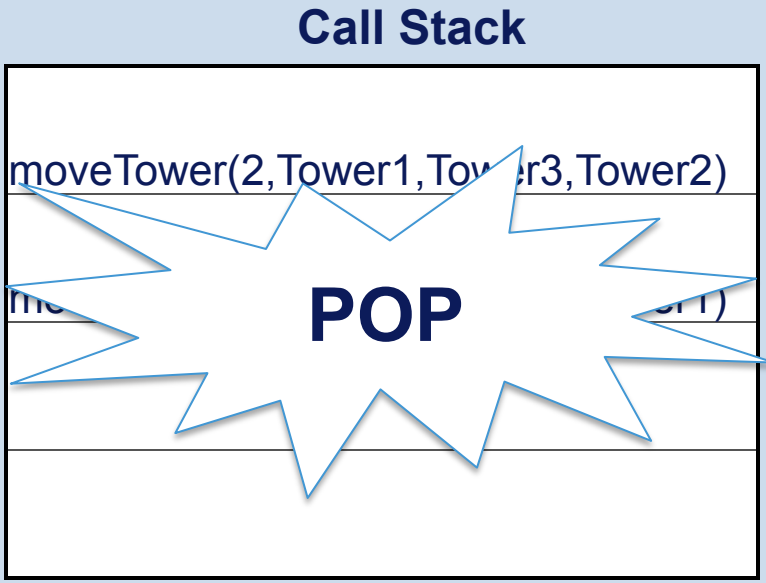
```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



3(iii) Move disk0 from 1 to 3

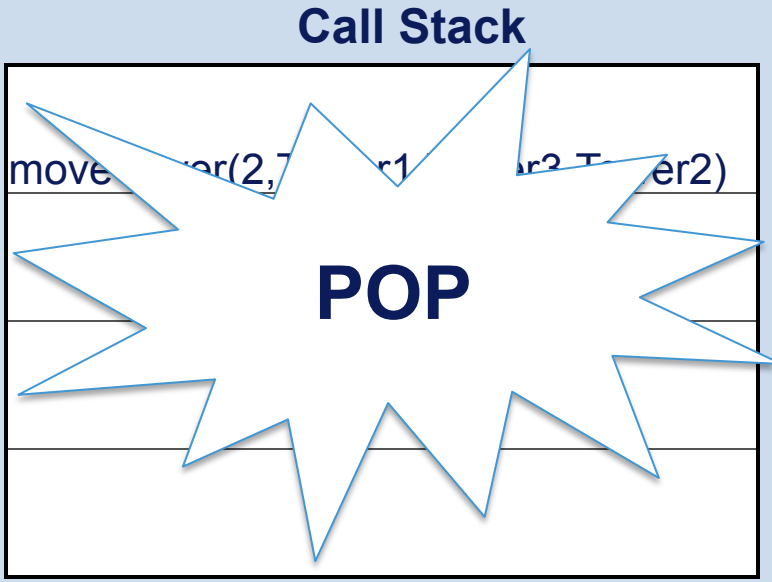
Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```



Towers of Hanoi: A recursive algorithm

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

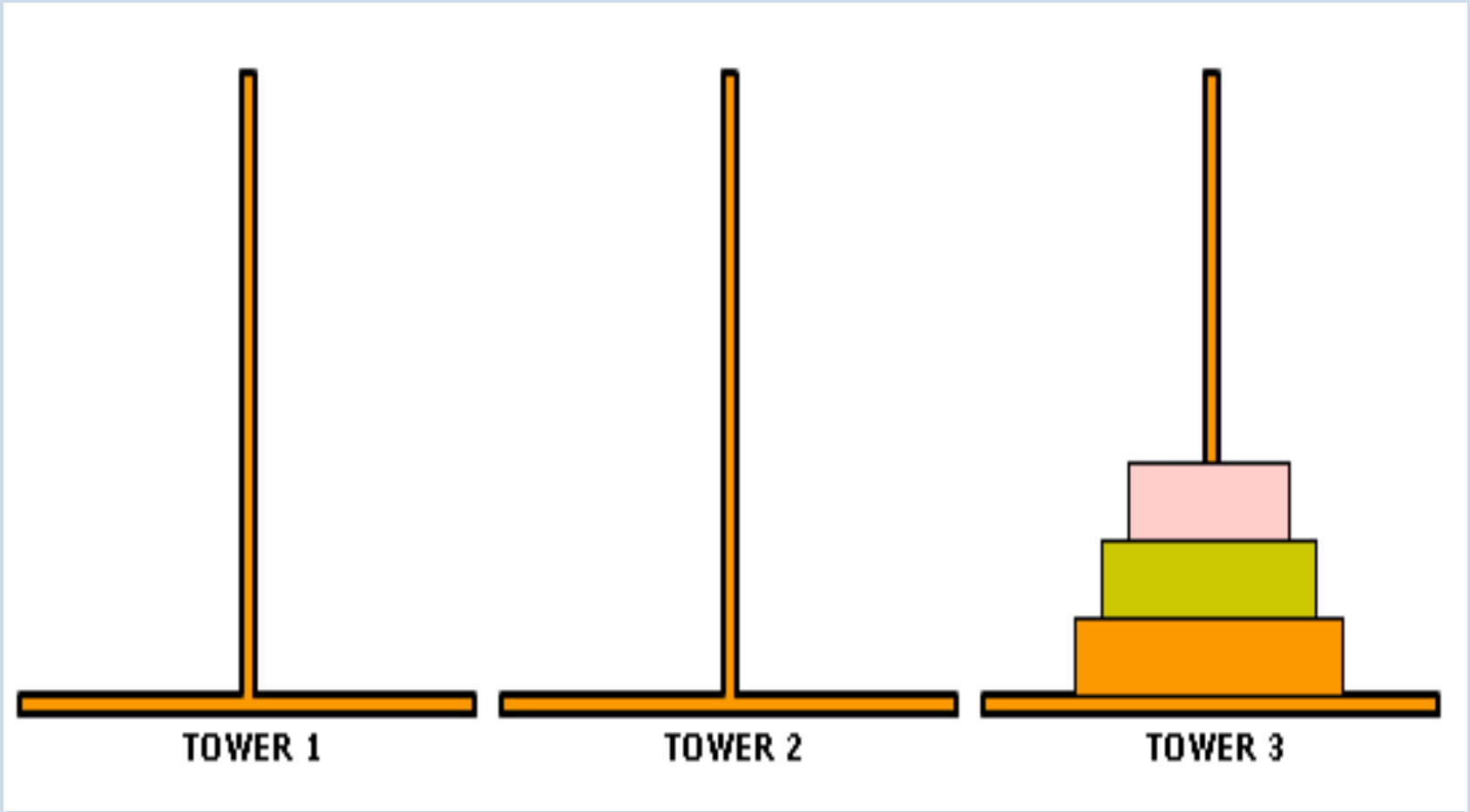


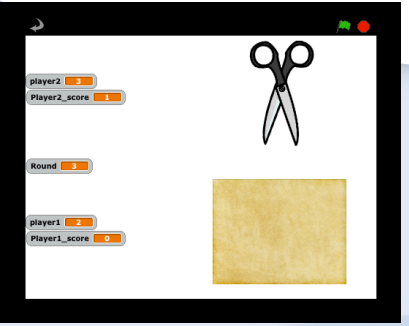
Towers of Hanoi: Bingo!!!

```
1  moveTower (disks, source, dest, spare)
2  If disk = 0
3       Move disk from source to dest
4  Else
5       moveTower (disk-1, source, spare, dest)
6       move disk from source to dest
7       moveTower (disk-1, spare, dest, source)
```

Call Stack

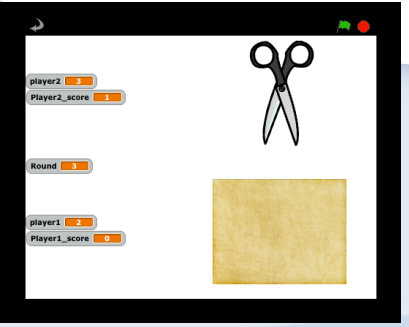
Towers of Hanoi: Bingo!!!





Rock Paper Scissors

- Rules:
  - There are 2 players (you and the computer)
  - If a player wins, their score is incremented by 1
  - If it is a draw, neither player receives any points
  - You choose the number of rounds
  - *Scissors* beats *paper*
  - *Paper* beats *rock*
  - *Rock* beats *scissors*



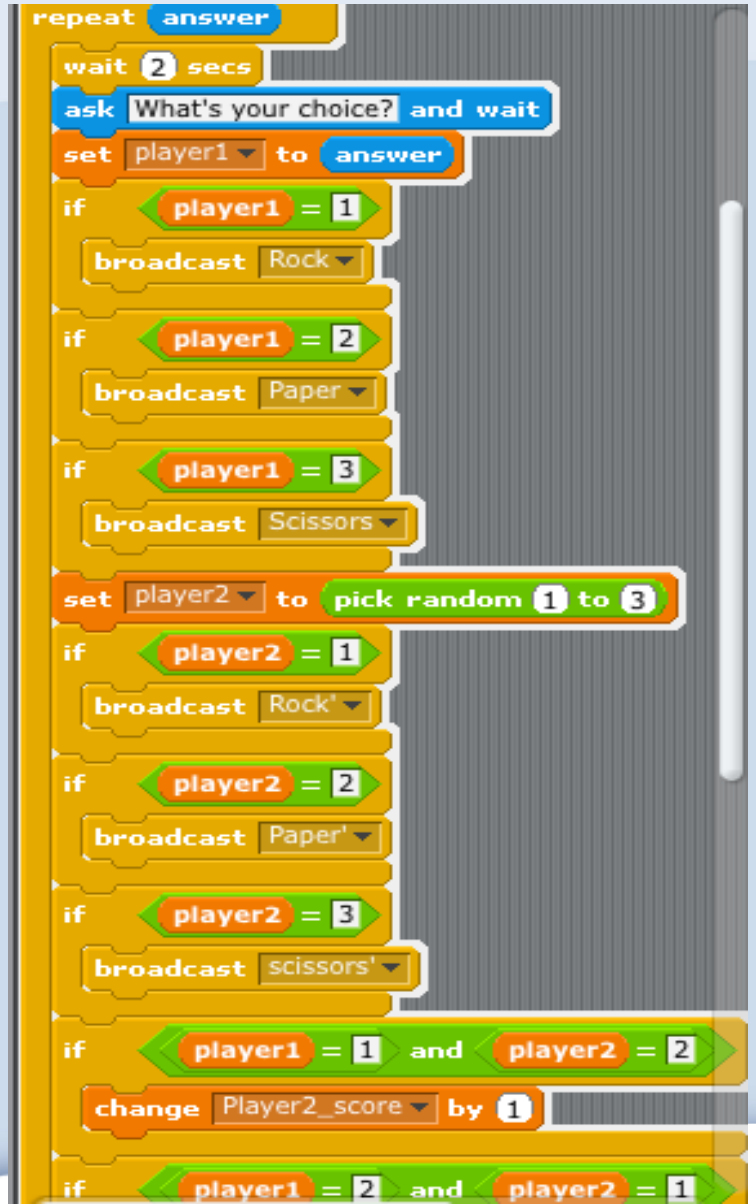
A solution ….

scissors= 1, paper=2, rock = 3
Enter number of rounds
While round!=0
    Enter player1ans
    player2ans = random choice

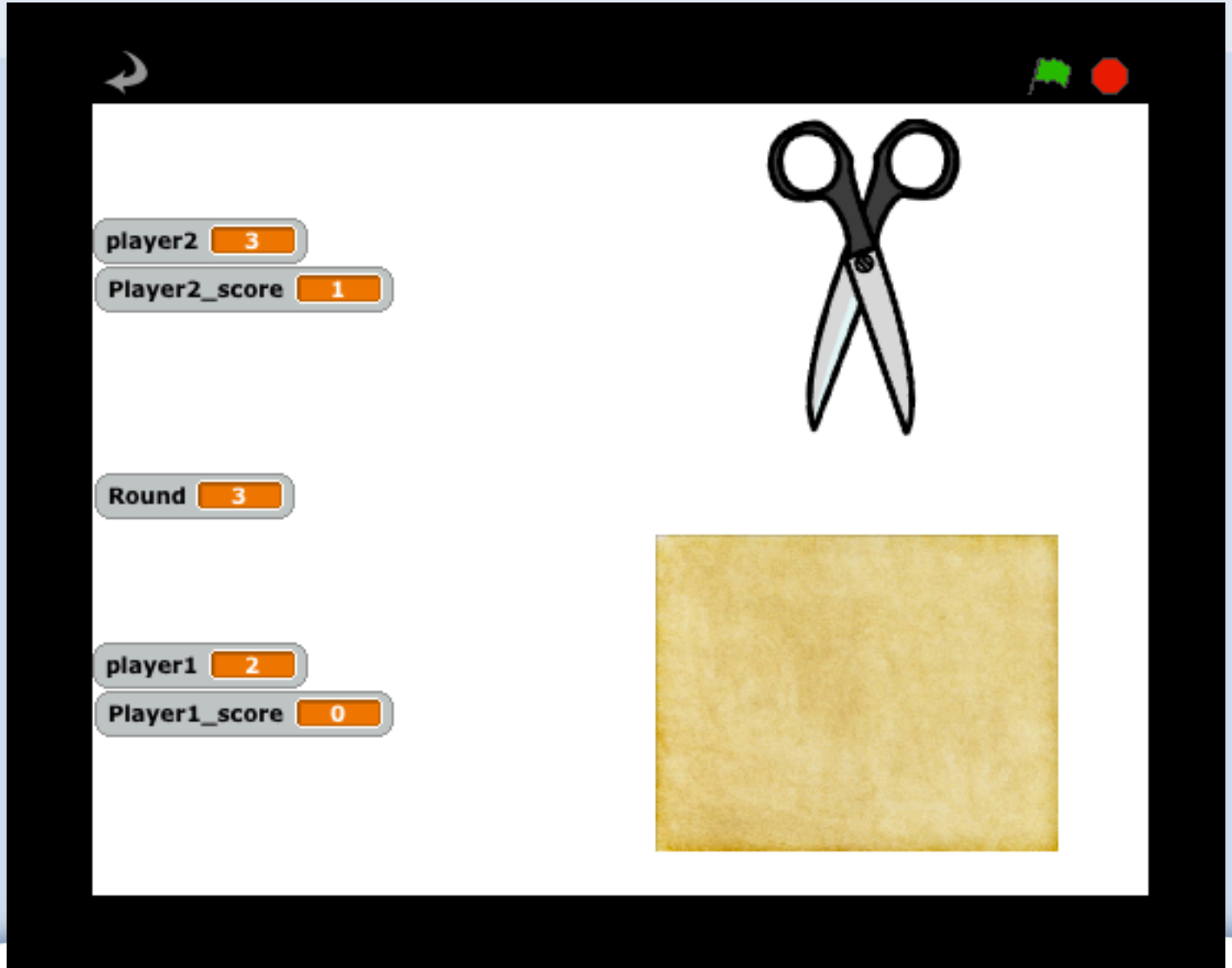
    if player1ans = scissors and player2ans = paper
        player1score = player1score + 1
    else if player1ans = paper and player2ans = scissors
        player2score = player2score + 1

……..

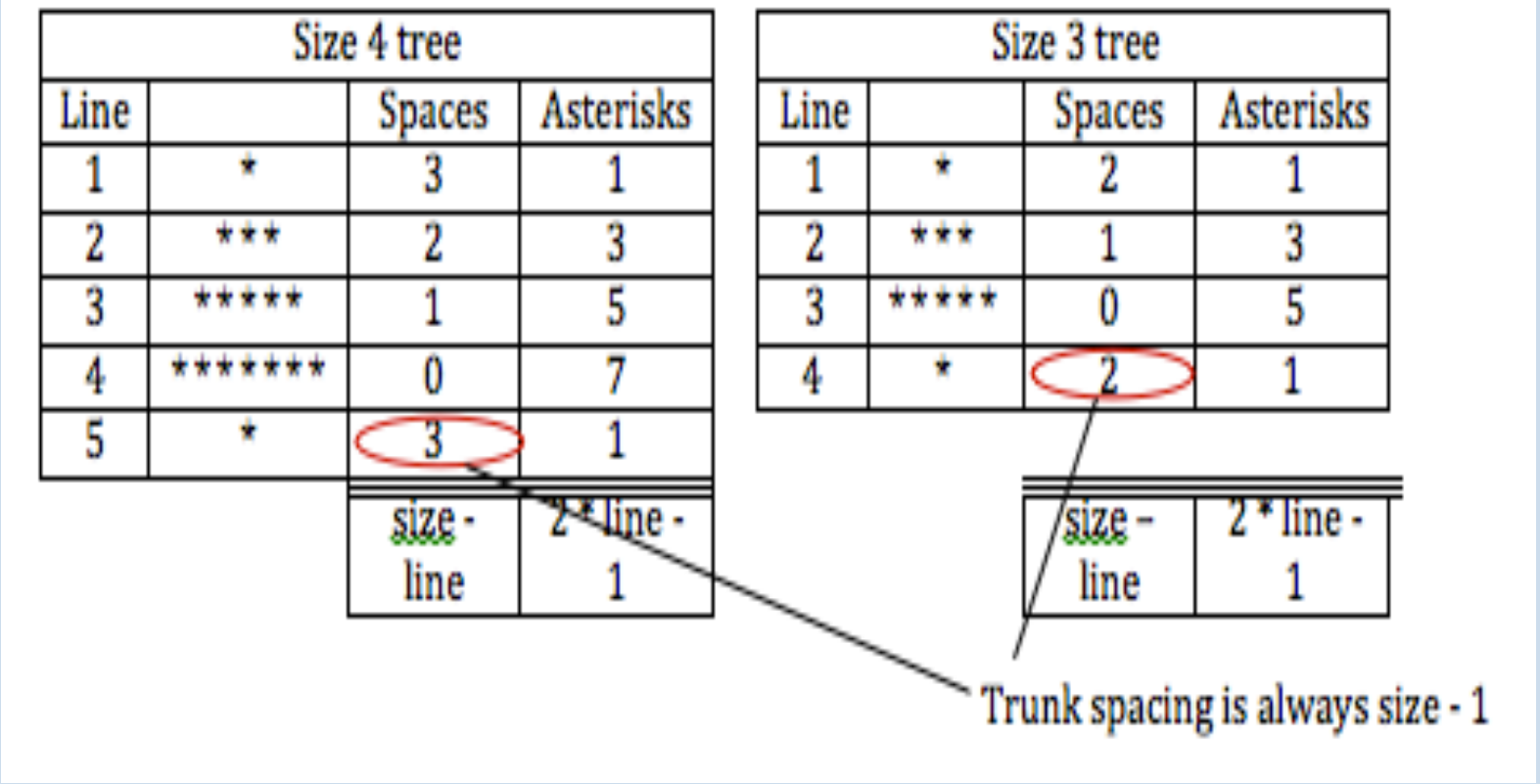
Rock Paper Scissors

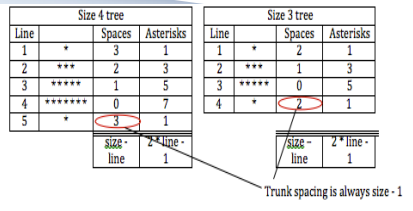


Rock Paper Scissors



Tree-Drawing Algorithm





Size 4 tree

Line		Spaces	Asterisks
1	*	3	1
2	***	2	3
3	*****	1	5
4	*******	0	7
5	*	3	1
		size - line	2 * line - 1

Size 3 tree

Line		Spaces	Asterisks
1	*	2	1
2	***	1	3
3	*****	0	5
4	*	2	1
		size - line	2 * line - 1

Trunk spacing is always size - 1

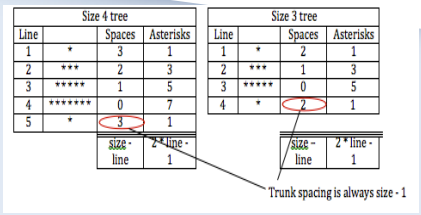
Tree-Drawing Algorithm

Read in size (the size of the tree)

Loop over the lines in the tree (that's what the size represents)

Write out some spaces (tree size minus the line we're on)
Write out some asterisks (line times 2 minus 1)

Draw a trunk (that's tree size – 1 spaces followed by a '*')



Tree-Drawing Algorithm

```
read size                                   (the size of the tree)

for i=1, i<=size                            (that's what the size represents)
        for j=0, j<size-i                   (tree size minus the line we're on)
                print ' '
        for k=0, k<((i*2)-1)                (line times 2 minus 1)
                print '*'
for j=0, j< size-1                          (that's tree size – 1 spaces followed by a '*')
        print ' '
print '*'
```

Test it! Does it work??

Thank You !


D·I·T
Dublin Institute of Technology
Institiúid Teicneolaíochta Átha Cliath