Introduction to Programming

Chapter 7

Recursion

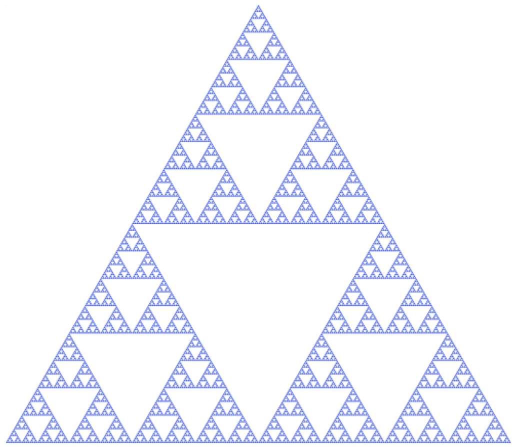
Overview

Q. What is recursion

A. When something is specified in terms of *itself*.

Why learn recursion?
- Represents a new mode of thinking.
- Provides a powerful programming paradigm.
- Enables reasoning about correctness.
- Gives insight into the nature of computation.

Many computational artifacts are naturally self-referential.
- File system with folders containing folders.
- Fractal graphical patterns.
- Divide-and-conquer algorithms (stay tuned).



Example: Convert an integer to binary

Recursive program

To compute a function of a positive integer N

- Base case. Return a value for small N .
- Reduction step. Assuming that it works for smaller values of its argument, use the function to compute a return value for N.

```
string convert(int N) {
    if(N==1) return "1";
    return convert(N/2) + to_string(N%2);
}

int main() {
    int N; cin >> N;
    cout << convert(N);
}
```

Q. How can we be convinced that this method is correct?

A. Use *mathematical induction.*

Mathematical induction (quick review)

To prove a statement involving a positive integer N

- Base case. Prove it for some specific values of N.
- Induction step. Assuming that the statement is true for all positive integers less than N, use that fact to prove it for N.

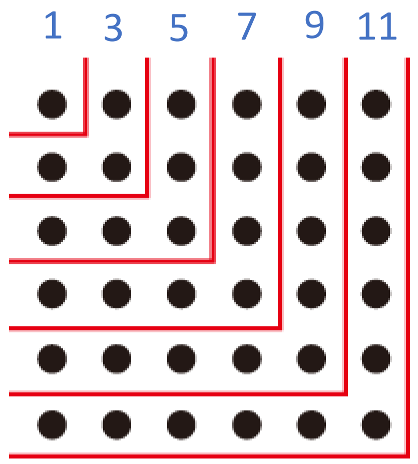
Example The sum of the first N odd integers is N^2.

Base case. True for $N = 1$.

Induction step. The N-th odd integer is $2N - 1$.

Let $T_N = 1 + 3 + 5 + \cdots + (2N - 1)$ be the sum of the first N odd integers.

- Assume that $T_{N-1} = (N-1)^2$.
- Then $\mathrm{T_N} = (N-1)^2 + (2N-1) = N^2$.



an alternate proof

Proving a recursive program correct

Recursion

To compute a function of N

- Base case. Return a value for small N .
- Reduction step. Assuming that it works for smaller values of its argument, use the function to compute a return value for N.

Mathematical induction

To prove a statement involving N

- Base case. Prove it for small N.
- Induction step. Assuming that the statement is true for all positive integers less than N, use that fact to prove it for N.

Recursive program

```
string convert(int N) {
    if(N==1) return "1";
    return convert(N/2) + to_string(N%2);
}
```

Correctness proof, by induction

convert() computes the binary representation of N

- Base case. Returns `"1"` for $N = 1$.
- Induction step. Assume that `convert()` works for $\frac{N}{2}$

1. Correct to append `"0"` if N is even, since $N = 2(N/2)$.
2. Correct to append `"1"` if N is odd since $N = 2(N/2) + 1$.

Mechanics of a function call

System actions when *any* function is called

- *Save environment* (values of all variables and call location).
- *Initialize* argument variables.
- *Transfer control* to the function.
- *Restore environment* (and assign return value)
- *Transfer control* back to the calling code.

```
string convert(int N) {
    if(N==1) return "1";
    return convert(N/2) + to_string(N%2);
}

int main() {
    int N; cin >> N;
    cout << convert(N);
}
```

```
convert(26)
    if(N==1) return "1";
    return "1101" + "0";
```

```
convert(13)
    if(N==1) return "1";
    return "110" + "1";
```

```
convert(6)
    if(N==1) return "1";
    return "11" + "0";
```

```
convert(3)
    if(N==1) return "1";
    return "1" + "1";
```

```
convert(1)
    if(N==1) return "1";
    return convert(0) + "0";
```

Programming with recursion: typical bugs

Missing base case

```
double bad(int N) {
    return bad(N-1) + 1.0/N;
}
```

No convergence guarantee

```
double bad(int N) {
    if (N == 1) return 1.0;
    return bad(1 + N/2) + 1.0/N;
}
```

try $N = 2$

Both lead to *infinite recursive loops* (bad news).

Collatz Sequence

Collatz function of N.

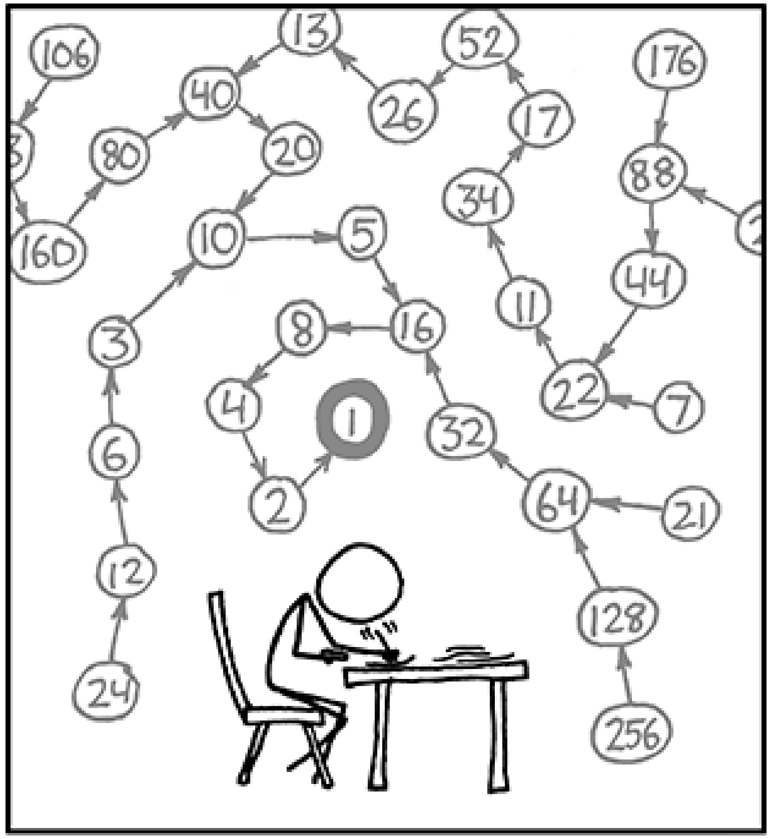
- If N is 1, stop.
- If N is even, divide by 2.
- If N is odd, multiply by 3 and add 1.

```
void collatz(int N) {
    cout << N << " ";
    if (N == 1) return;
    if (N % 2 == 0)
        collatz(N / 2);
    collatz(3*N + 1);
}
```

```
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Amazing fact. No one knows whether or not this function terminates for all N (!)

Note. We usually ensure termination by only making recursive calls for smaller N.



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

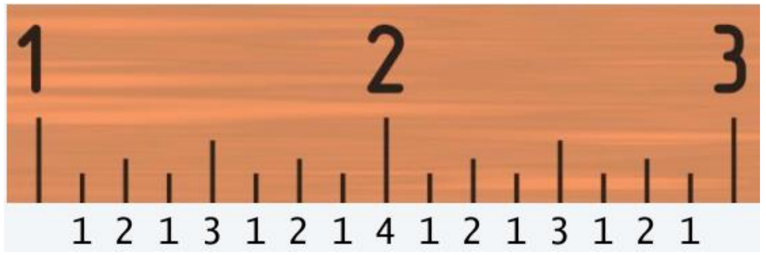
A classic example

Tower of Hanoi puzzle

Warmup: subdivisions of a ruler (revisited)

ruler(n): create subdivisions of a ruler to $1/2^n$ inches.

- Return one space for $n = 0$.
- Otherwise, sandwich n between two copies of `ruler(n-1)`.



```
string ruler(int n) {
    if (n == 0) return " ";
    return ruler(n-1) + to_string(n) + ruler(n-1);
}
int main() {
    int N; cin >> N;
    cout << ruler(N);
}
```

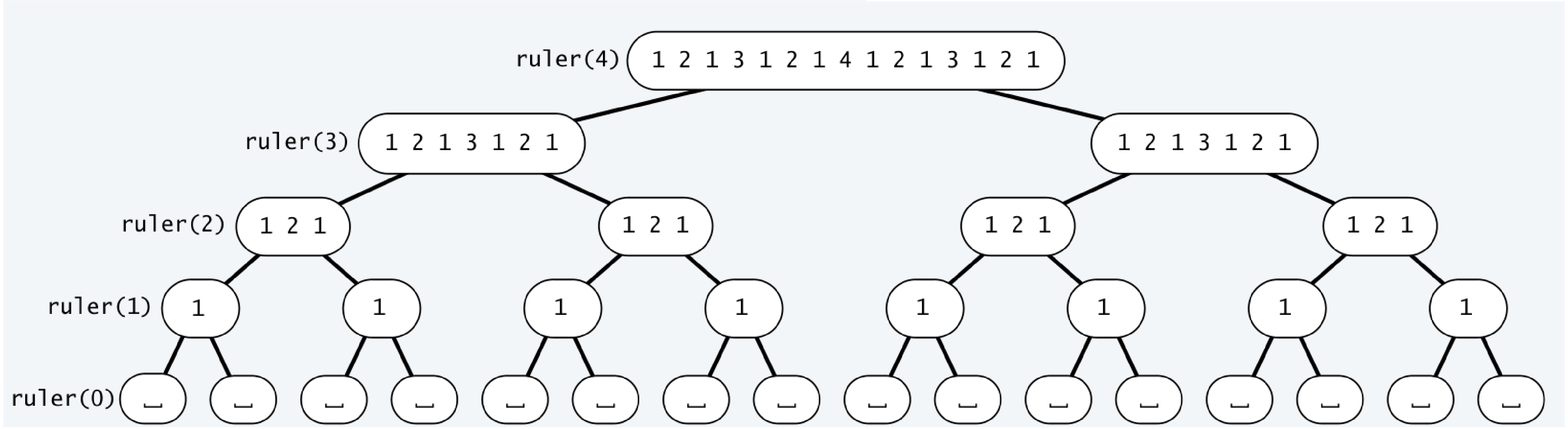
```
➢ a.exe
4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

➢ a.exe
50
terminate called after throwing an
instance of 'std::bad_alloc'
```

$2^{50} - 1$ integers in output!

Tracing a recursive program

Use a *recursive call tree*

- One node for each recursive call.
- Label node with return value after children are labeled.



Tower of Hanoi puzzle

A legend of uncertain origin

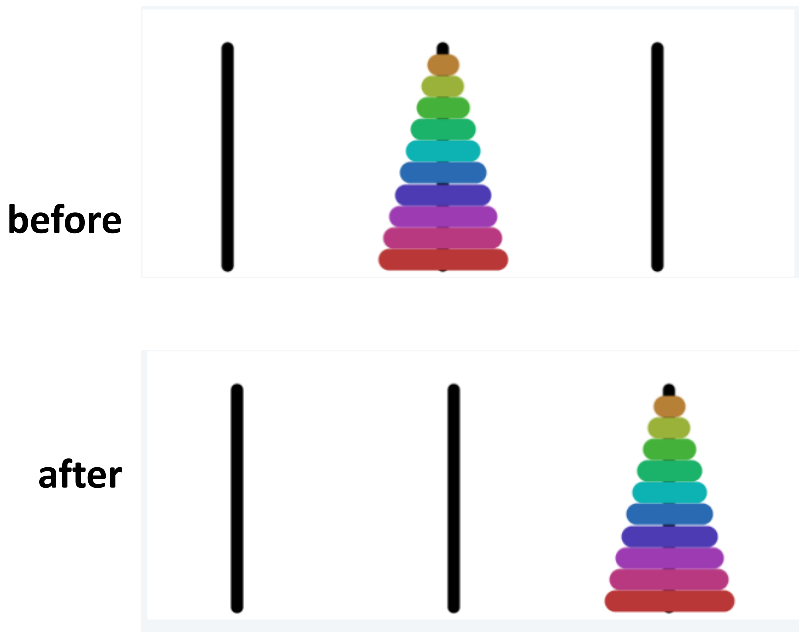
- n = 64 discs of differing size; 3 posts; discs on one of the posts from largest to smallest.
- An ancient prophecy has commanded monks to move the discs to another post.
- When the task is completed, *the world will end*.

Rules

- Move discs one at a time.
- Never put a larger disc on a smaller disc.

Q. Generate list of instruction for monks ?

Q. When might the world end ?

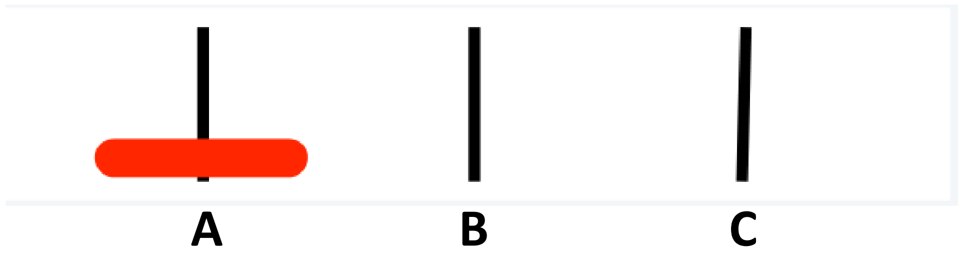


n=10 disks

Tower of Hanoi

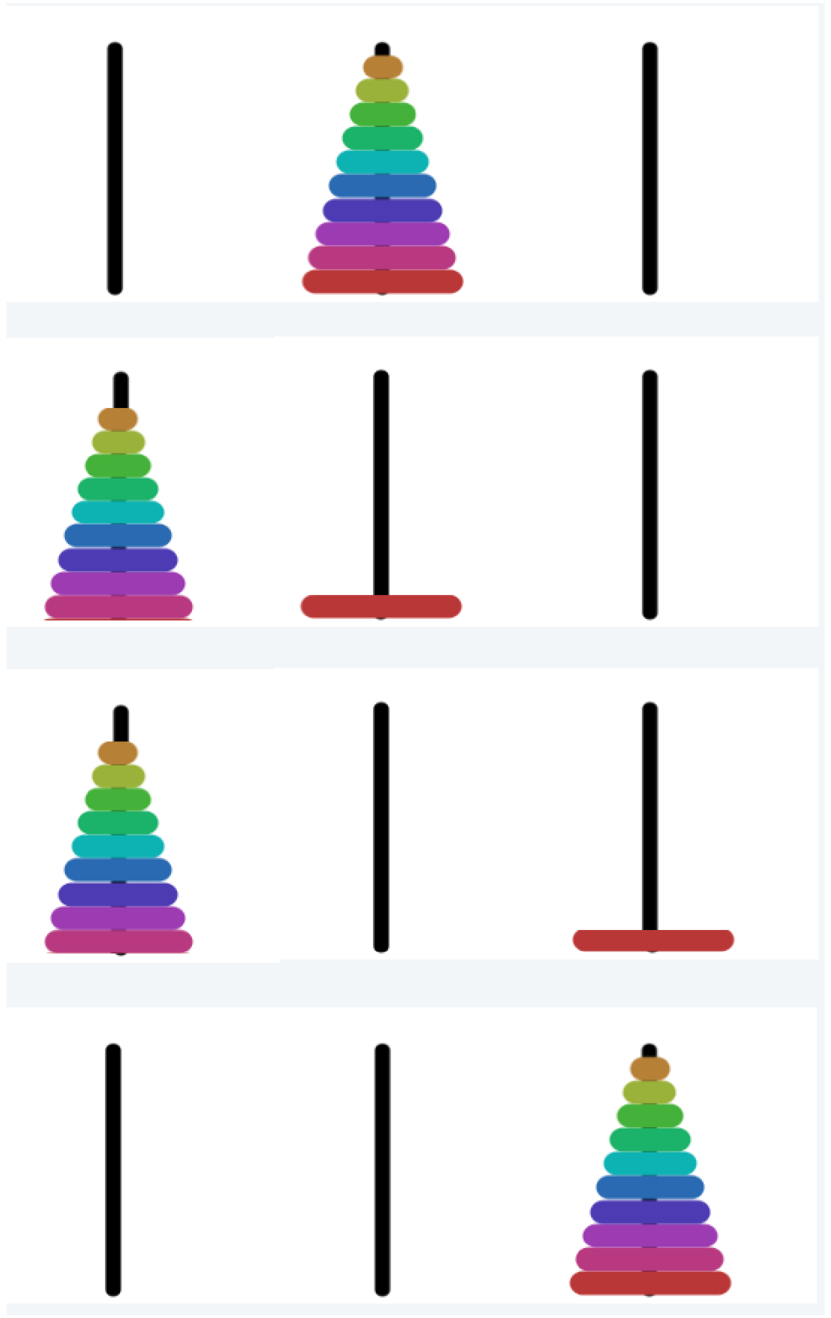
For simple instructions, use cyclic wraparound

- Move right means: A to B, B to C,   C to A
- Move left means: A to C, B to A,   B to A,

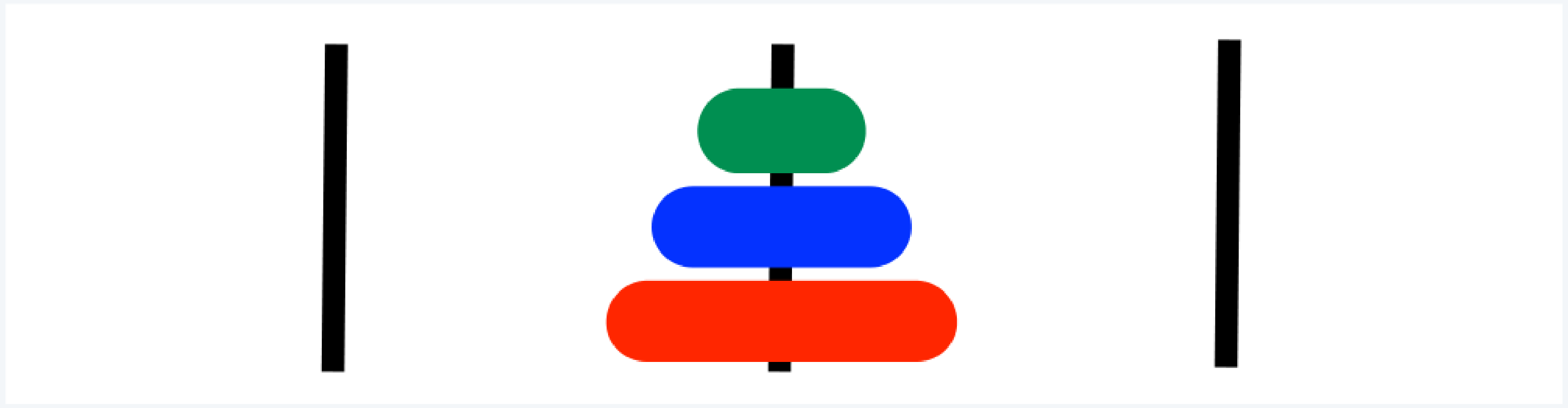


A recursive solution

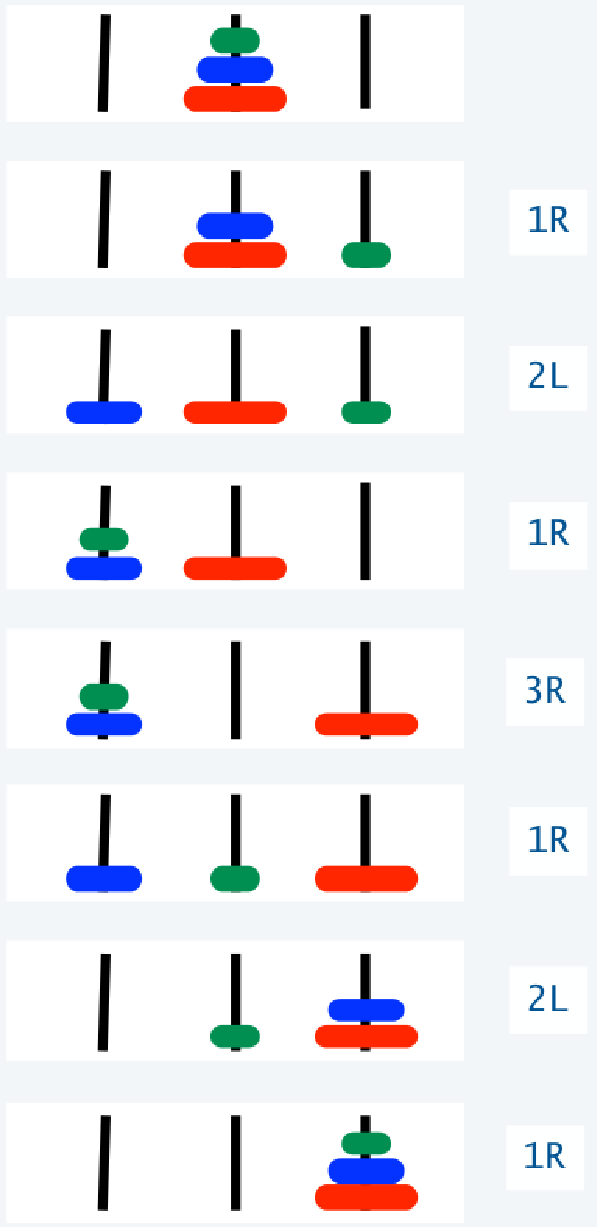
- Move n – 1 discs to the left (recursively).
- Move largest disc to the right.
- Move n – 1 discs to the left (recursively).



Tower of Hanoi solution (n=3)



1R	2L	1R	3R	1R	2L	1R



Towers of Hanoi: recursive solution

`hanoi(n)`: Print moves for n discs.

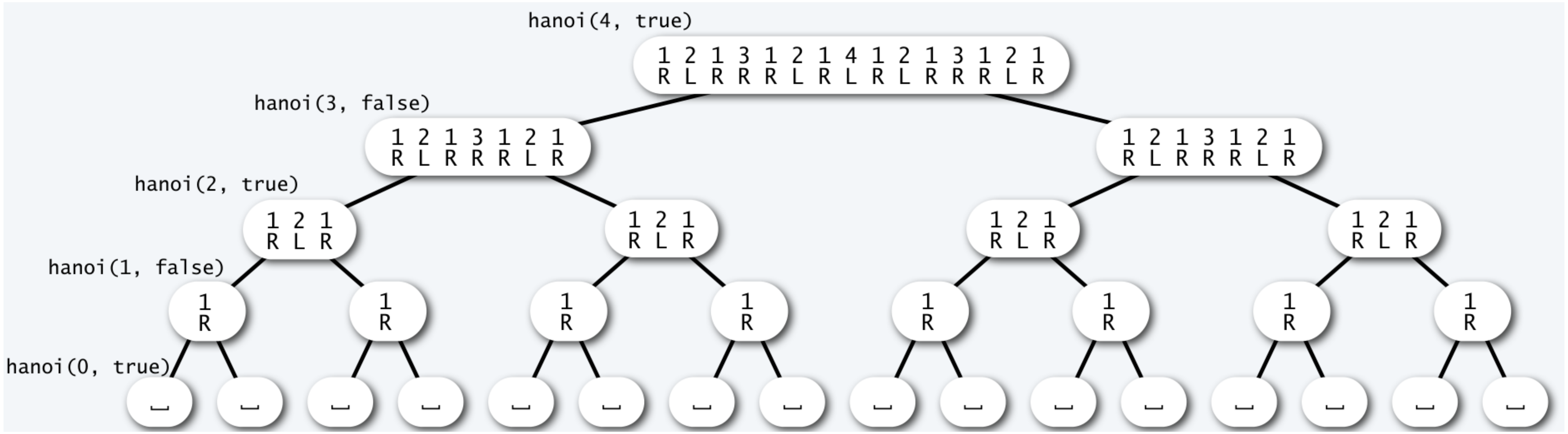
- Return one space for `n=0`.
- Otherwise, set move to the specified move for disc n.
- Then sandwich move between two copies of `hanoi(n-1)`.

```
string hanoi(int N, bool left) {
    if(N==0) return " ";
    string inst;
    if(left)
        inst =  to_string(N) + “L";
    else
        inst =  to_string(N) + “R";
    return hanoi(N-1, !left) + inst + hanoi(N-1, !left);
}
```

Recursive call tree for towers of Hanoi

Structure is the same as for the ruler function and suggests 3 useful and easy-to-prove facts.

- Each disc always moves in the same direction.
- Moving smaller disc always alternates with a unique legal move.
- Moving n discs requires 2^{n-1} moves.



Answers for towers of Hanoi

Q. Generate list of instructions for monks ?

A. (Long form). `1L 2R 1L 3L 1L 2R 1L 4R 1L 2R 1L 3L 1L 2R 1L 5L 1L 2R 1L 3L 1L 2R 1L 4R ...`

A. (Short form). Alternate "1L" with the only legal move not involving the disc 1.

"L" or "R" depends on whether n is odd or even

Q. When might the world end ?

A. Not soon: need $2^{64} - 1$ moves.

Note: Recursive solution has been proven optimal.

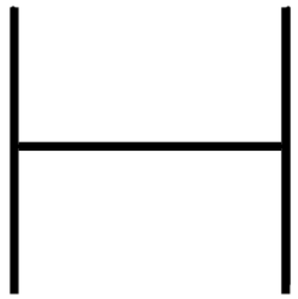
moves per second	end of world
1	5.84 billion centuries
1 billion	5.84 centuries

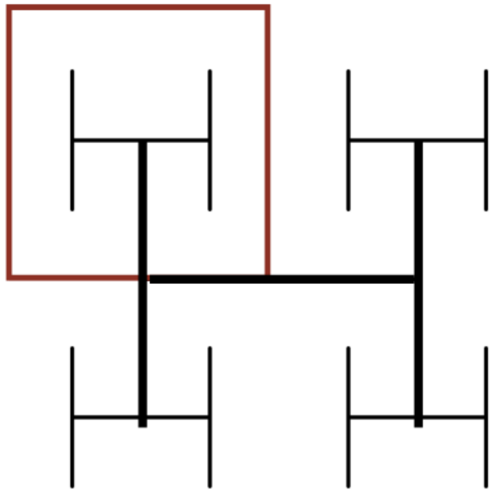
Recursive graphics

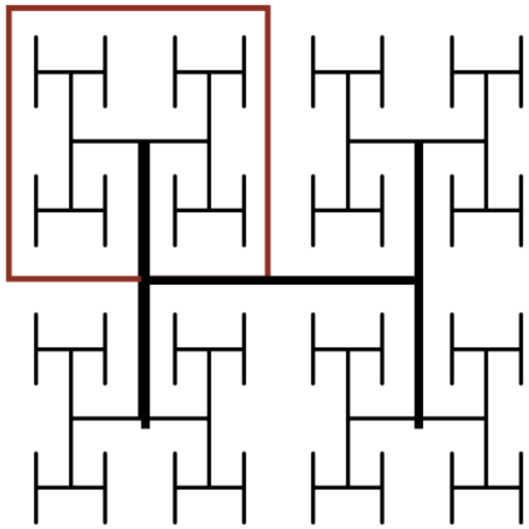
"Hello, World" of recursive graphics: H-trees

H-tree of order n

- If n is 0, do nothing.
- Draw an H, centered.
- Draw four H-trees of order n −1 and half the size, centered at the tips of the H.

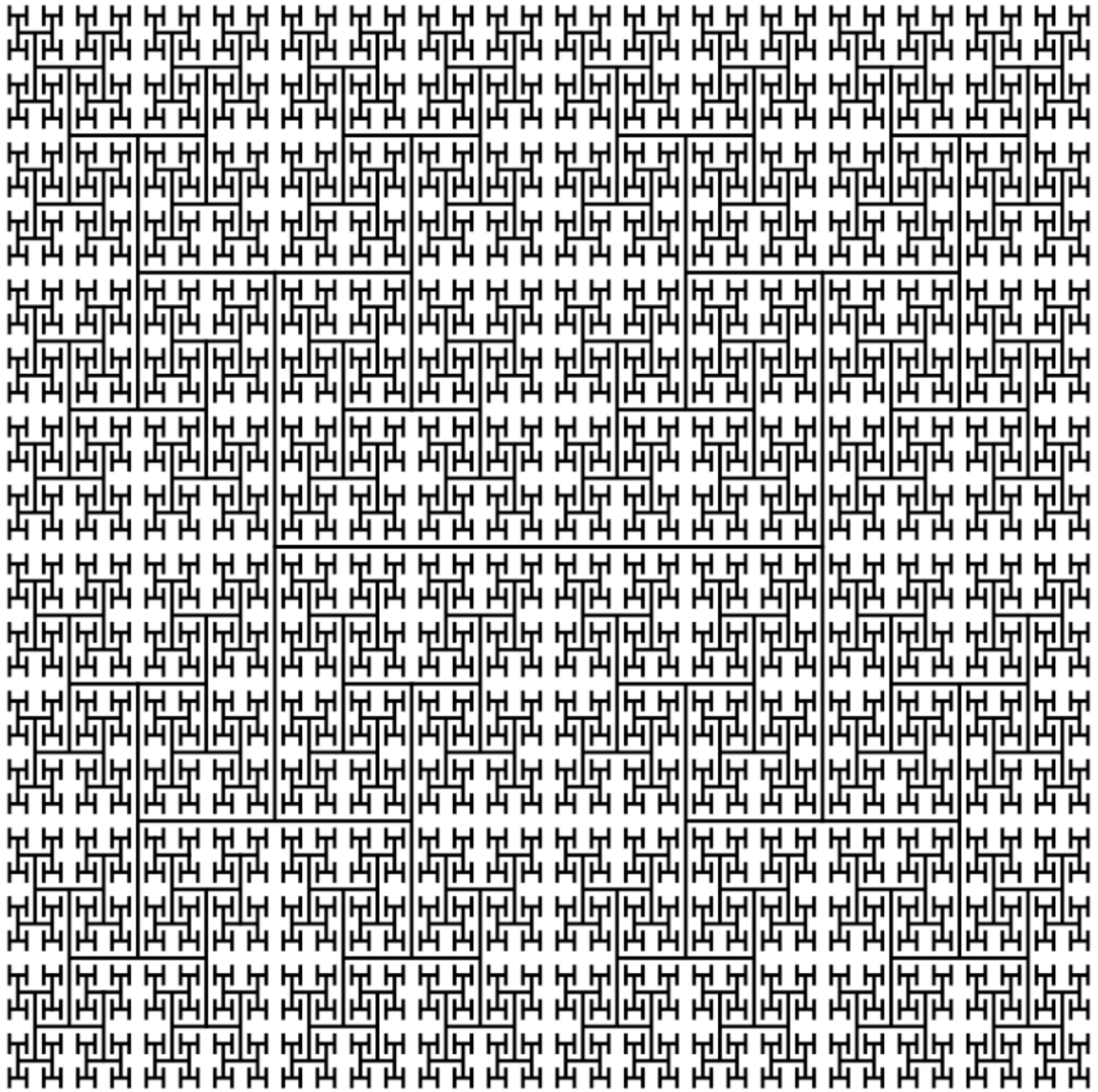

order 1


order 2


order 3

H Tree

Application. Connect a large set of regularly spaced sites to a single source.



order 6

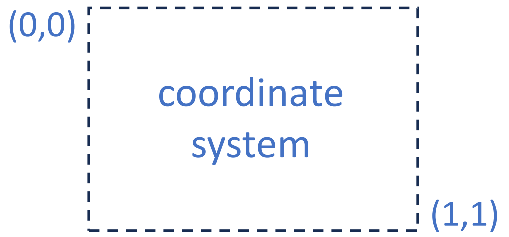
Recursive H-tree implementation

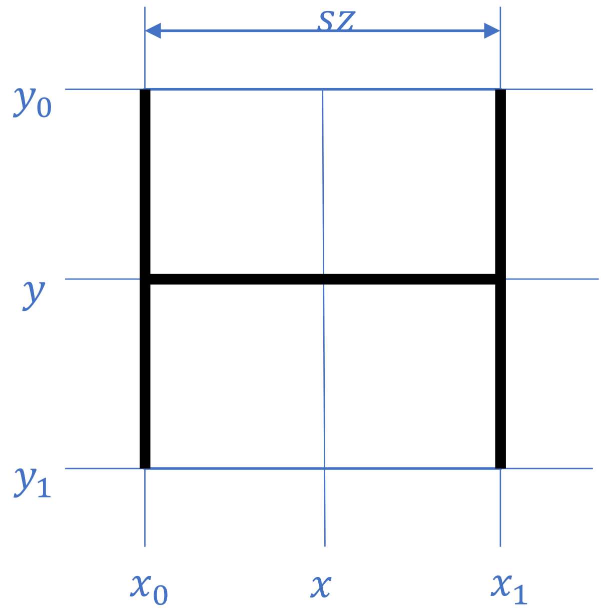
```
void draw(int n, double sz, double x, double y) {
    if(n==0) return;
    double x0 = x - sz/2, x1 = x + sz/2;
    double y0 = y - sz/2, y1 = y + sz/2;

    DrawLine(x0*W, y*H,  x1*W, y*H,  WHITE);
    DrawLine(x0*W, y0*H, x0*W, y1*H, WHITE);
    DrawLine(x1*W, y0*H, x1*W, y1*H, WHITE);

    draw(n-1, sz/2, x0, y0);
    draw(n-1, sz/2, x0, y1);
    draw(n-1, sz/2, x1, y0);
    draw(n-1, sz/2, x1, y1);
}
```

draw the H, centered on (x, y)

draw four half-size H-trees

```
BeginDrawing();
    draw(3, .5, .5, .5);
EndDrawing();
```


(0,0)
coordinate system
(1,1)


sz
y_0
y
y_1
x_0
x
x_1