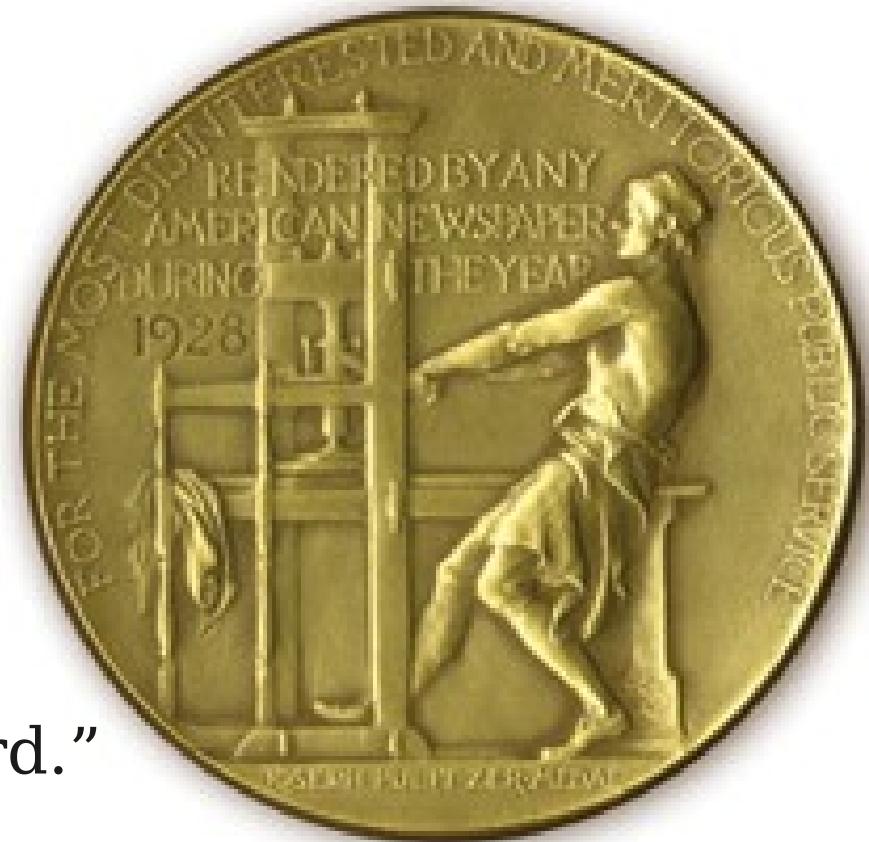


# Thinking Recursively

# The 2013 Pulitzer Prizes

“Awarded to the Sun Sentinel, Fort Lauderdale, FL, for its well documented investigation of off-duty police officers who recklessly speed and endanger the lives of citizens, leading to disciplinary action and other steps to curtail a deadly hazard.”



<http://www.pulitzer.org/citation/2013-Public-Service>

# Announcements

- YEAH hours (**Y**our **E**arly **A**ssignment **H**elp) hours for Assignment 2 this Thursday, April 18, in Gates B12 from 5:30PM – 6:30PM.
- Optional, but recommended!

# Thinking Recursively

# Recursive Problem-Solving

**if** (*problem is sufficiently simple*) {

*Directly solve the problem.*

*Return the solution.*

} **else** {

*Split the problem up into one or more smaller problems with the same structure as the original.*

*Solve each of those smaller problems.*

*Combine the results to get the overall solution.*

*Return the overall solution.*

}

```
int digitalRoot(int value);
int sumOfDigits(int value);

int sumOfDigits(int value) {
    if (value == 0) {
        return 0;
    } else {
        return sumOfDigits(value / 10) + (value % 10);
    }
}

int digitalRoot(int value) {
    if (value < 10) {
        return value;
    } else {
        return digitalRoot(sumOfDigits(value));
    }
}
```

```
int maximumCoverageFor(Vector<int>& populations) {
    if (populations.size() == 0) {
        return 0;
    } else if (populations.size() == 1) {
        return populations[0];
    } else {
        Vector<int> allButFirst      = tailOf(populations);
        Vector<int> allButFirstTwo = tailOf(allButFirst);

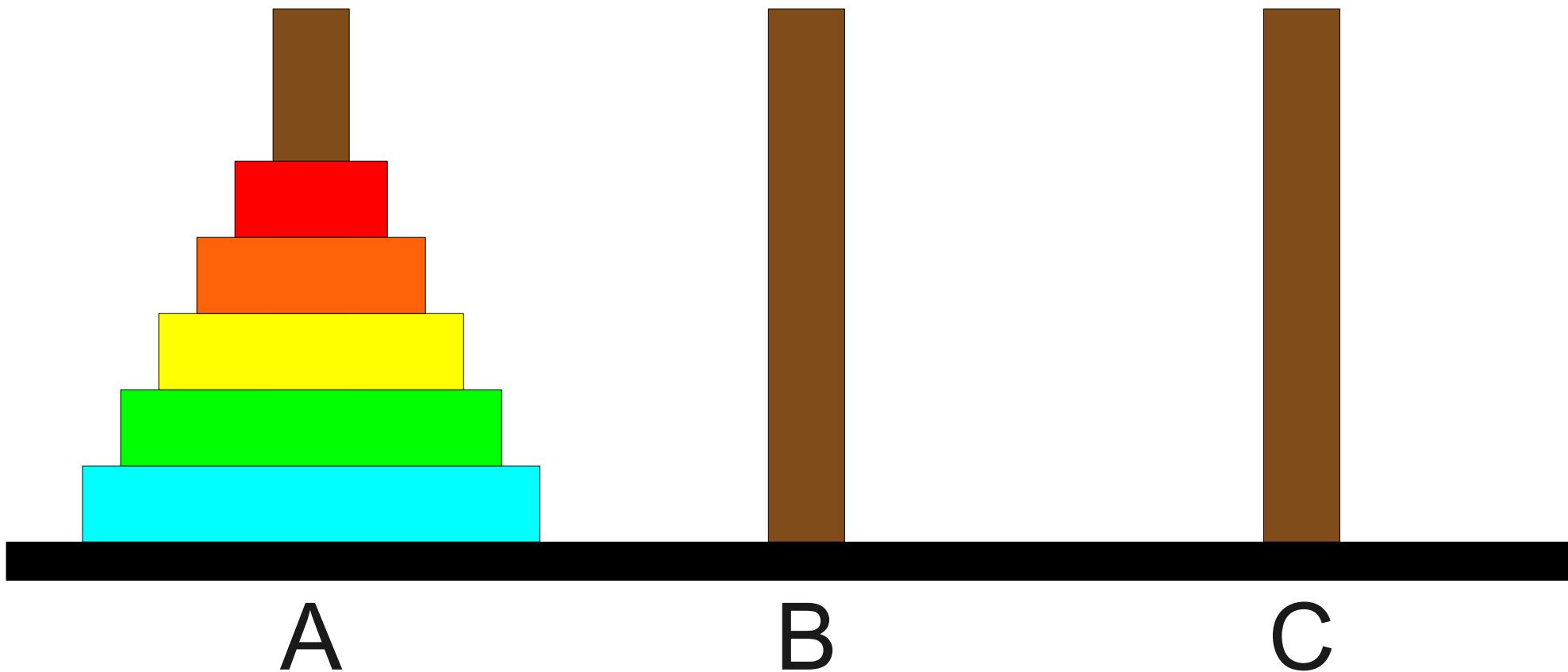
        return max(maximumCoverageFor(allButFirst),
                   populations[0] +
                   maximumCoverageFor(allButFirstTwo));
    }
}
```

# What's Going On?

- Recursion solves a problem by continuously simplifying the problem until it becomes simple enough to be solved directly.
- The **recursive decomposition** makes the problem slightly simpler at each step.
- The **base case** is what ultimately makes the problem solvable – it guarantees that when the problem is sufficiently simple, we can just solve it directly.

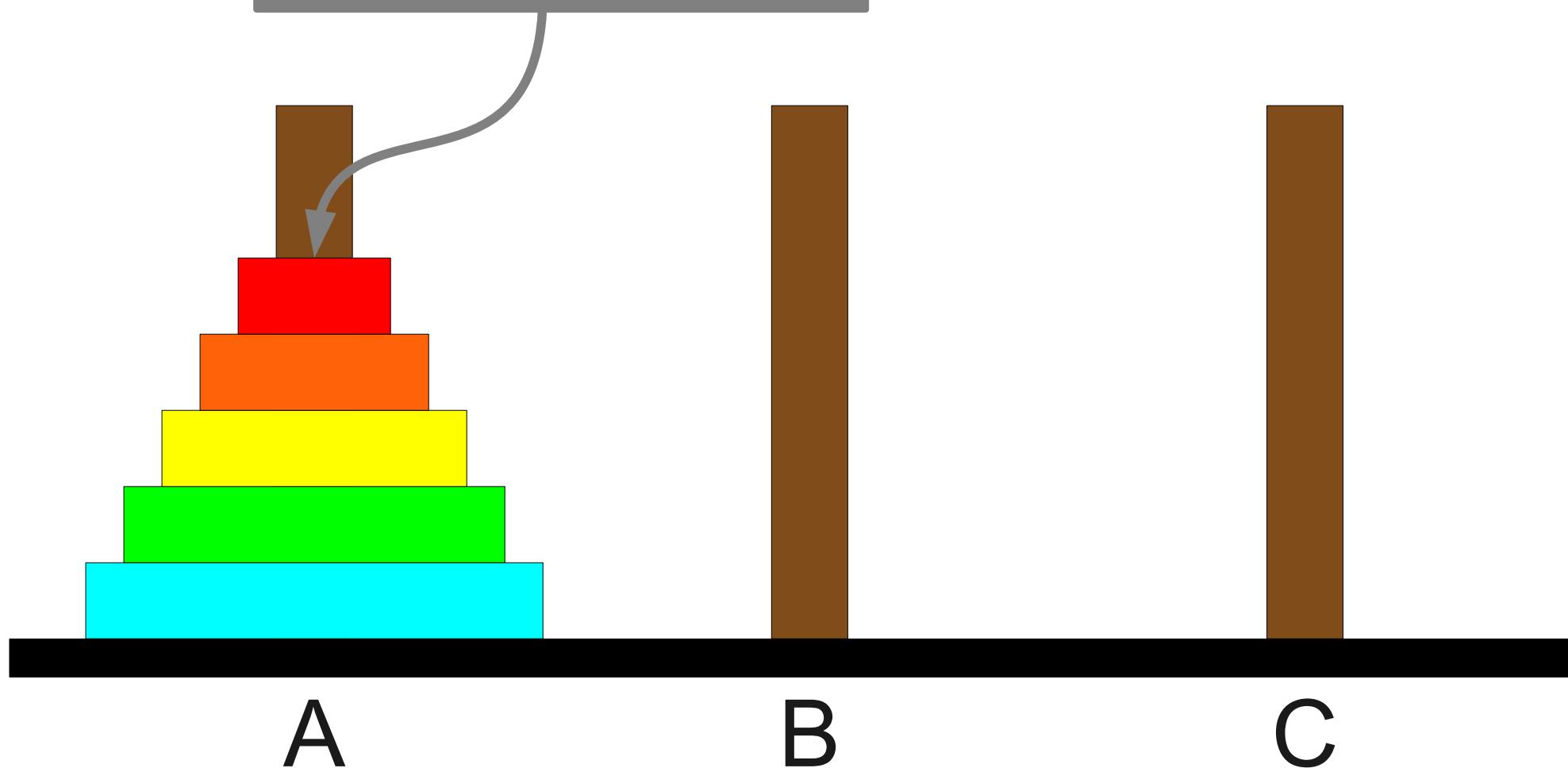
# The Towers of Hanoi Problem

# Towers of Hanoi



# Towers of Hanoi

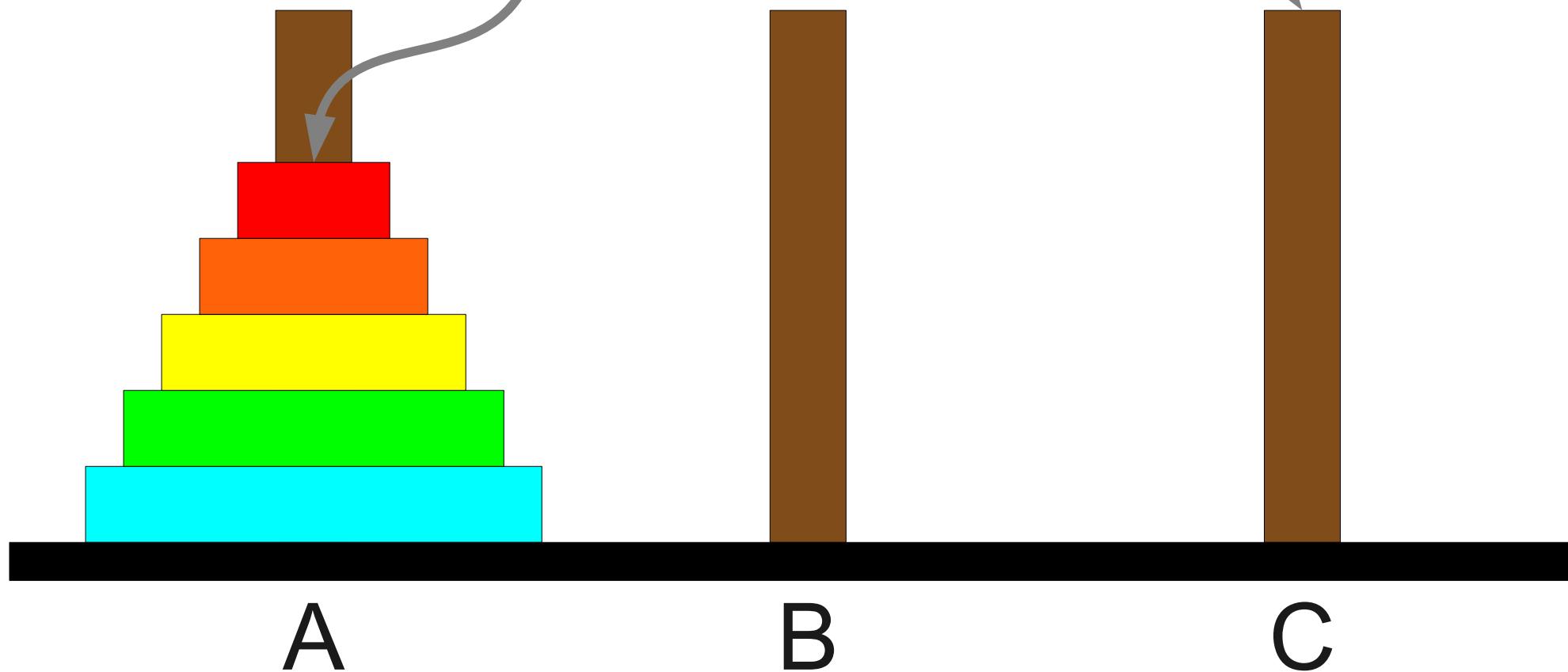
Move this tower...



# Towers of Hanoi

Move this tower...

...to this spindle.



# Towers of Hanoi

Move this tower...

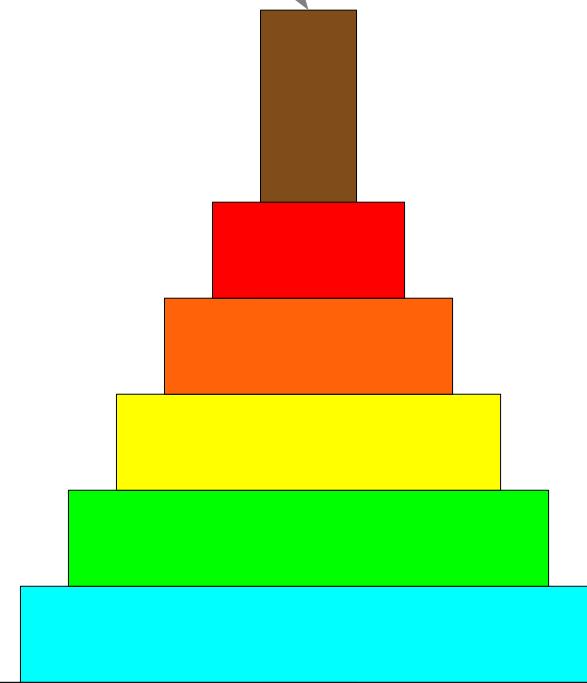
...to this spindle.



A

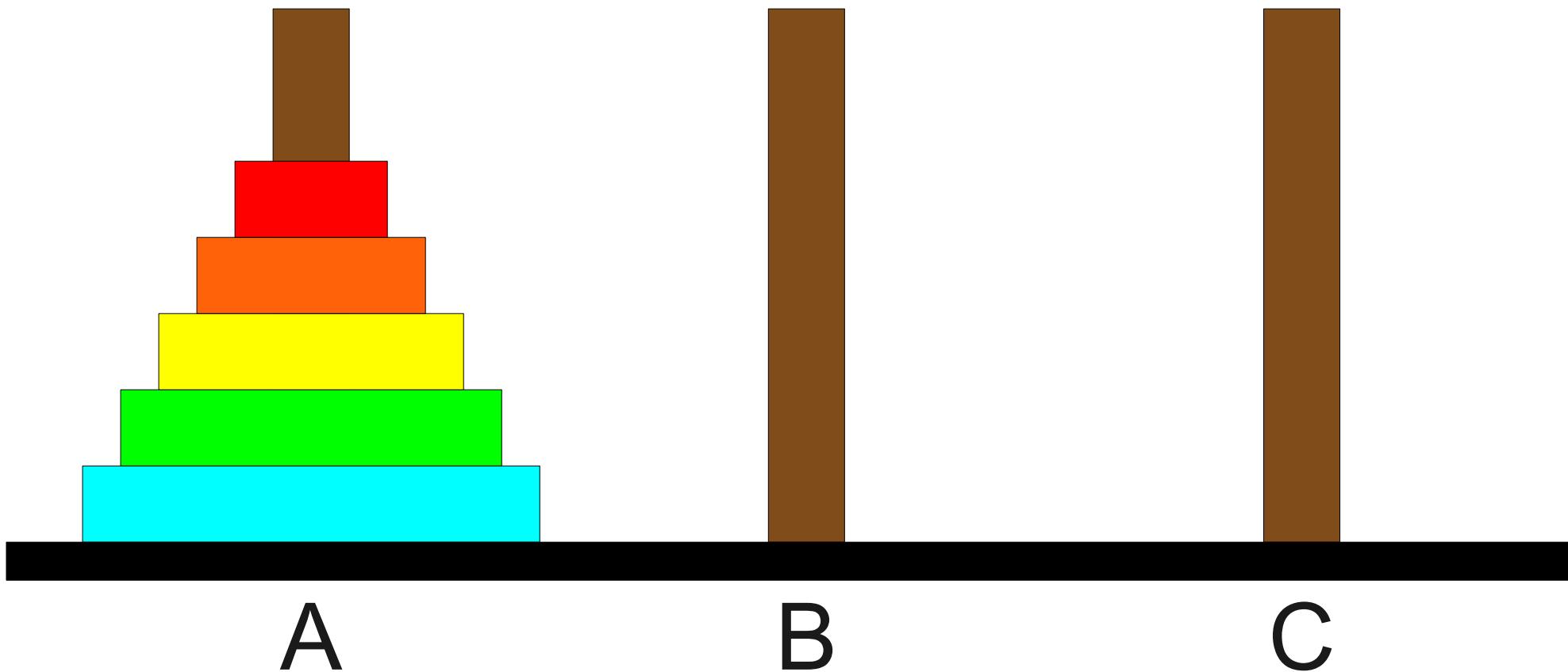


B

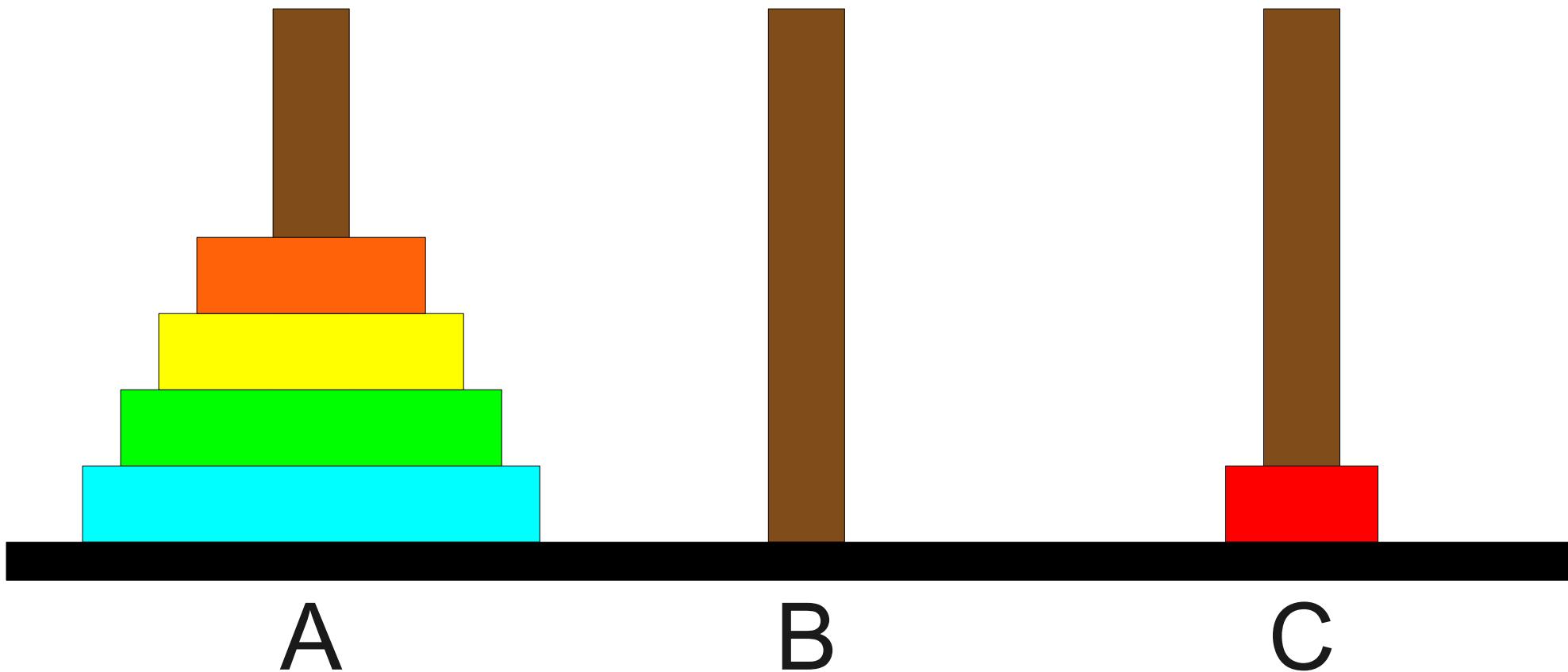


C

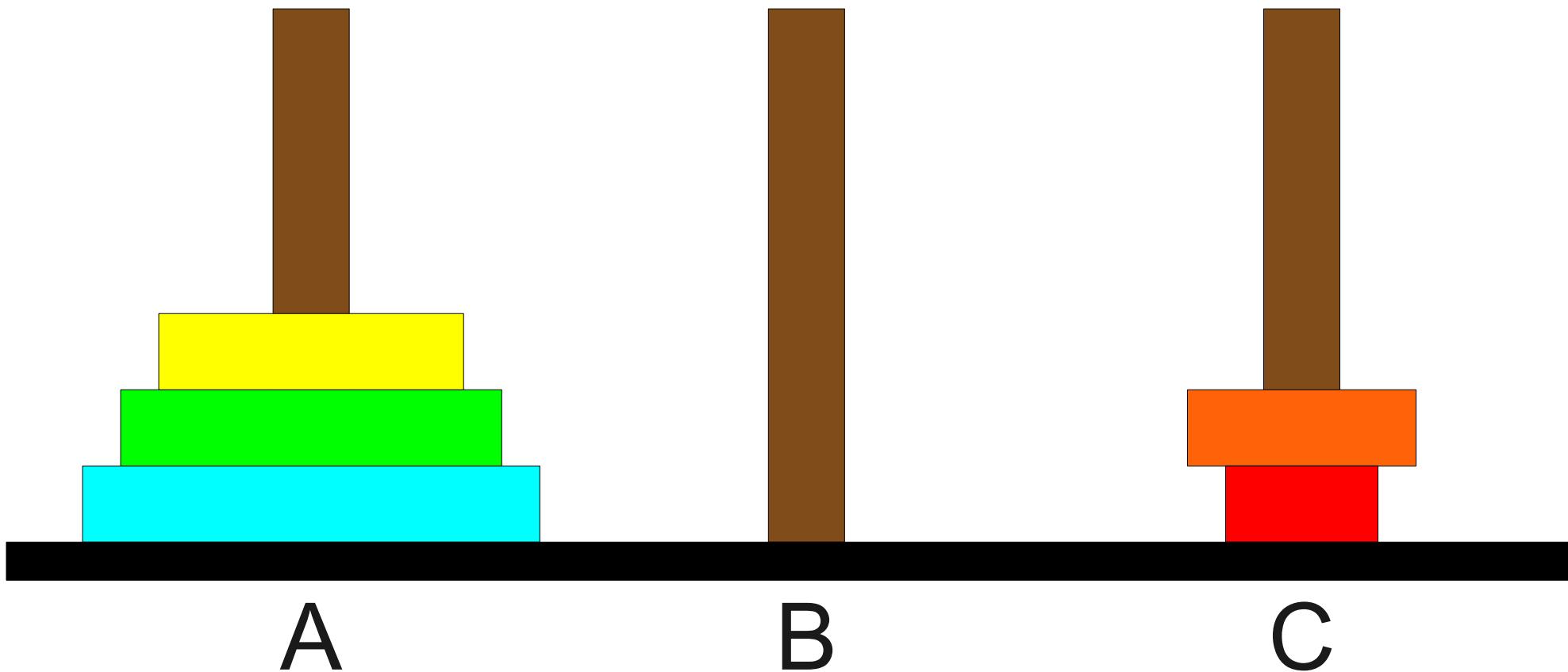
# Towers of Hanoi



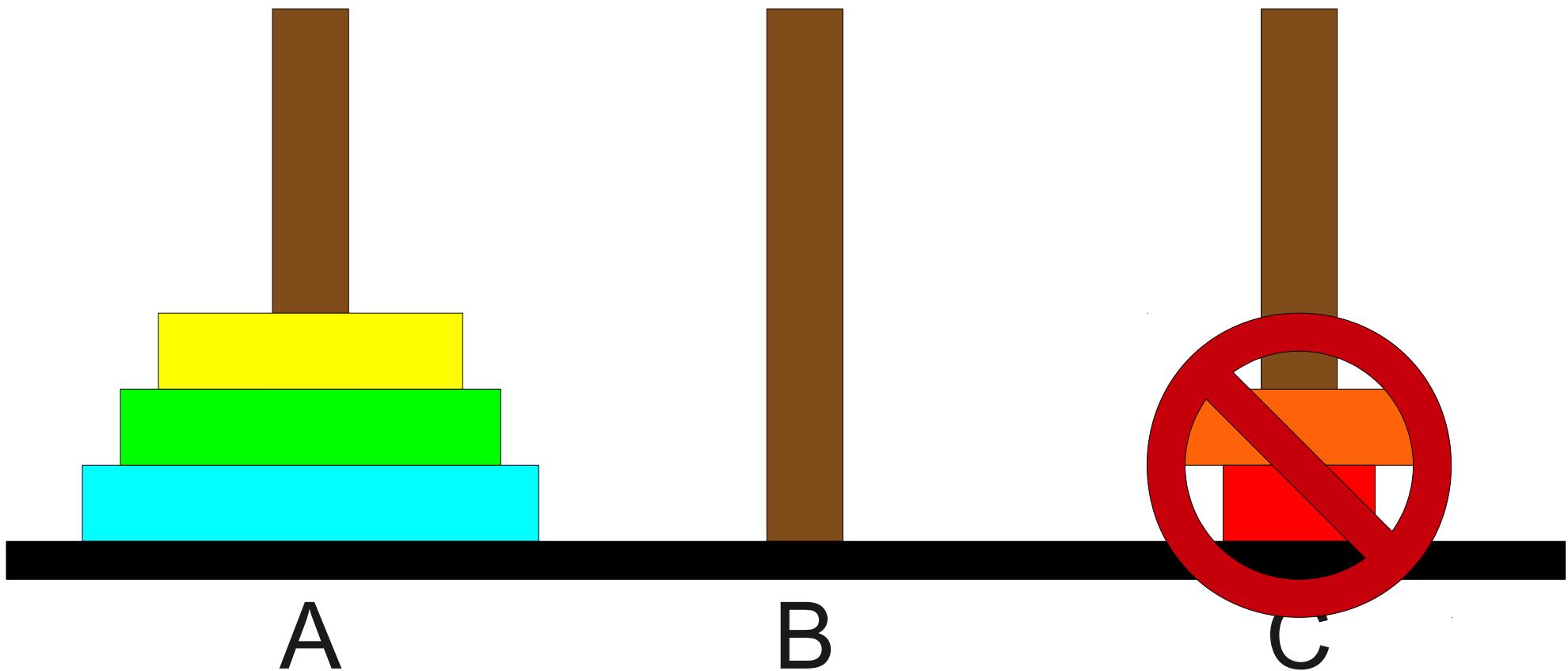
# Towers of Hanoi



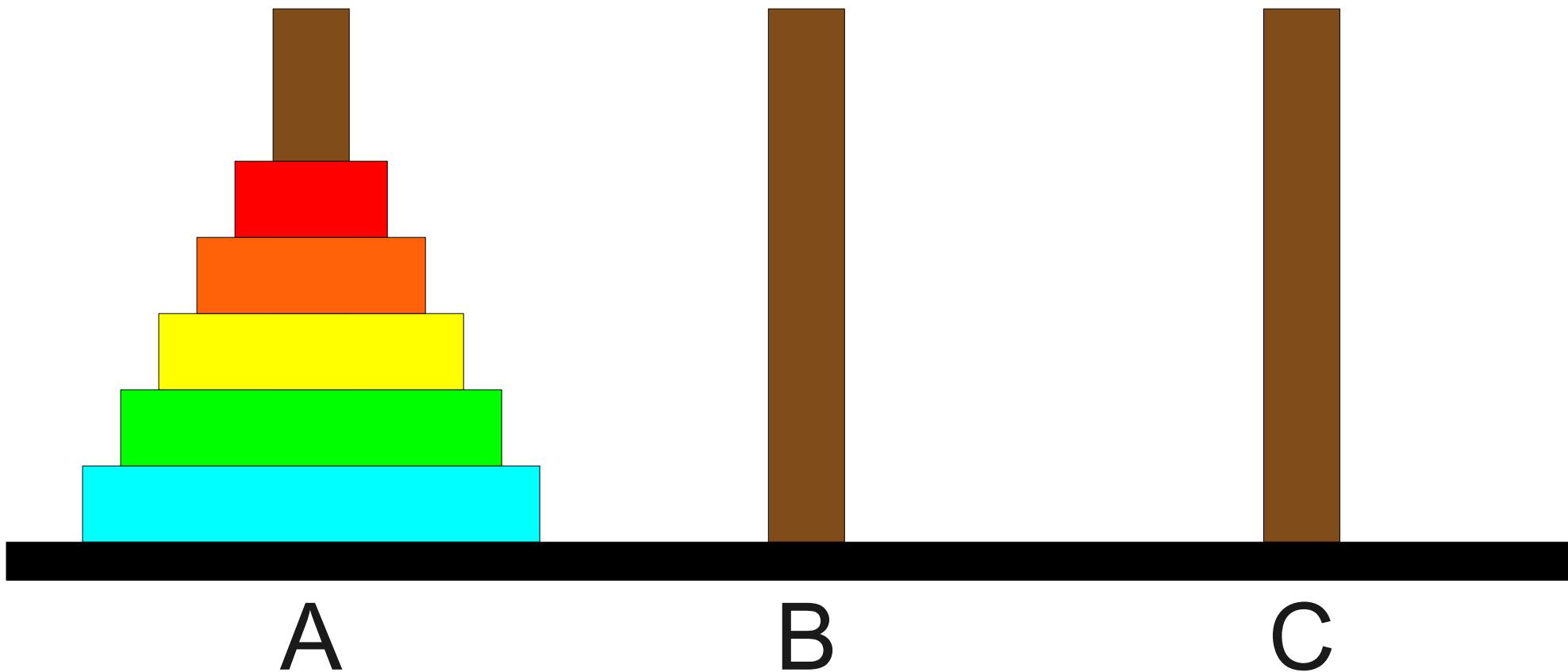
# Towers of Hanoi



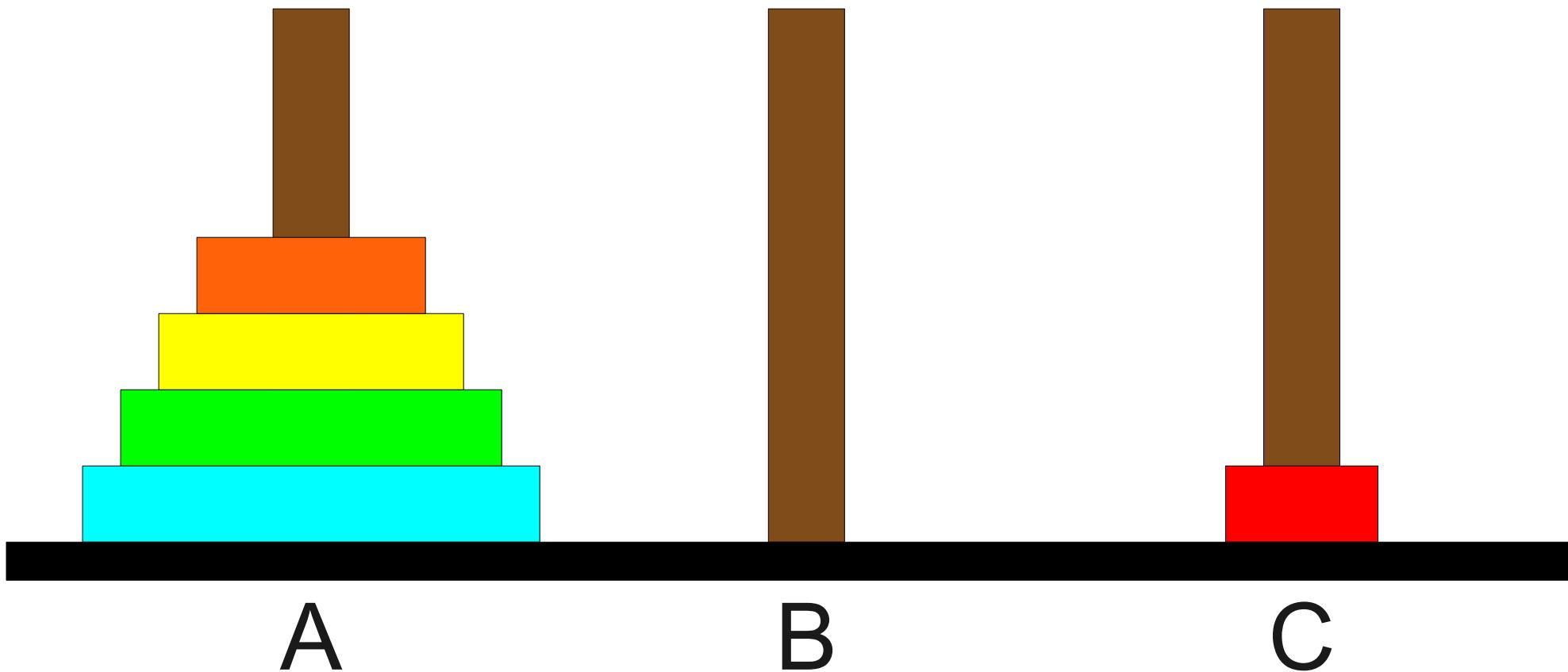
# Towers of Hanoi



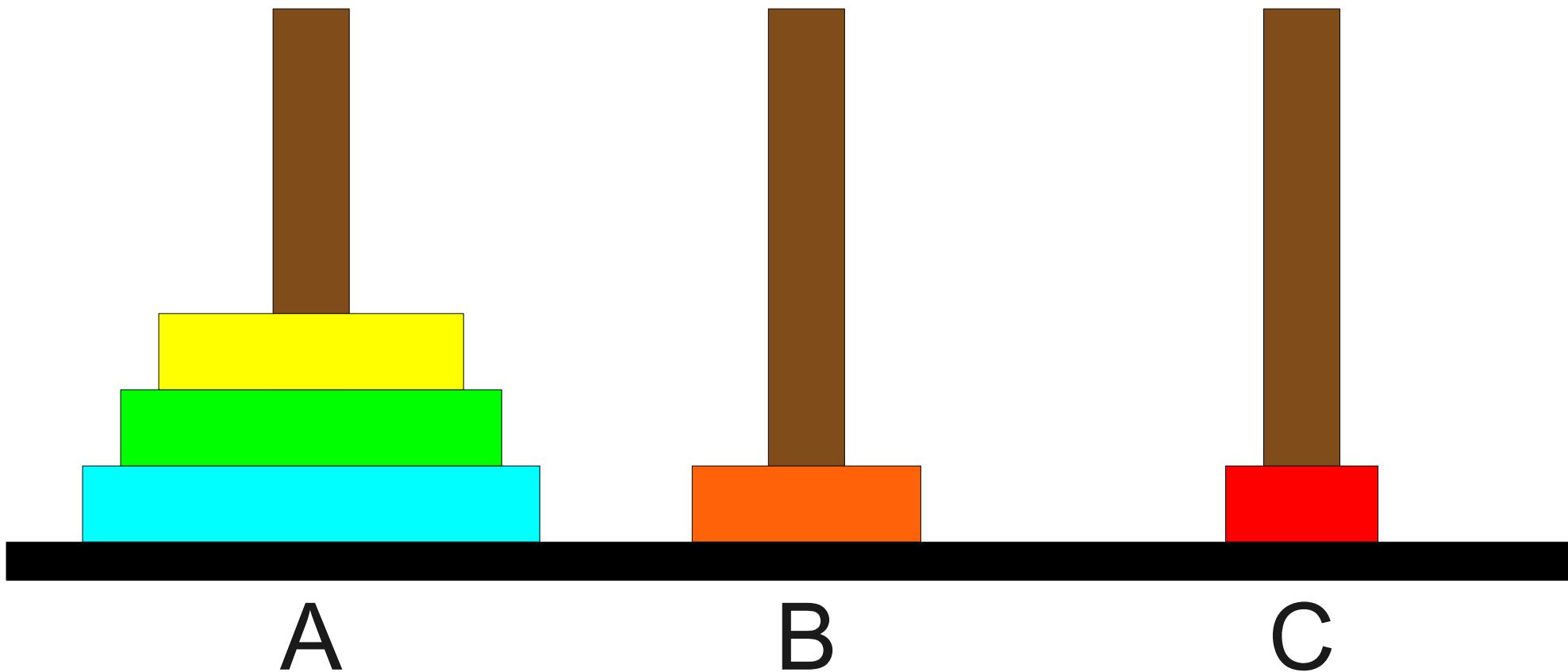
# Towers of Hanoi



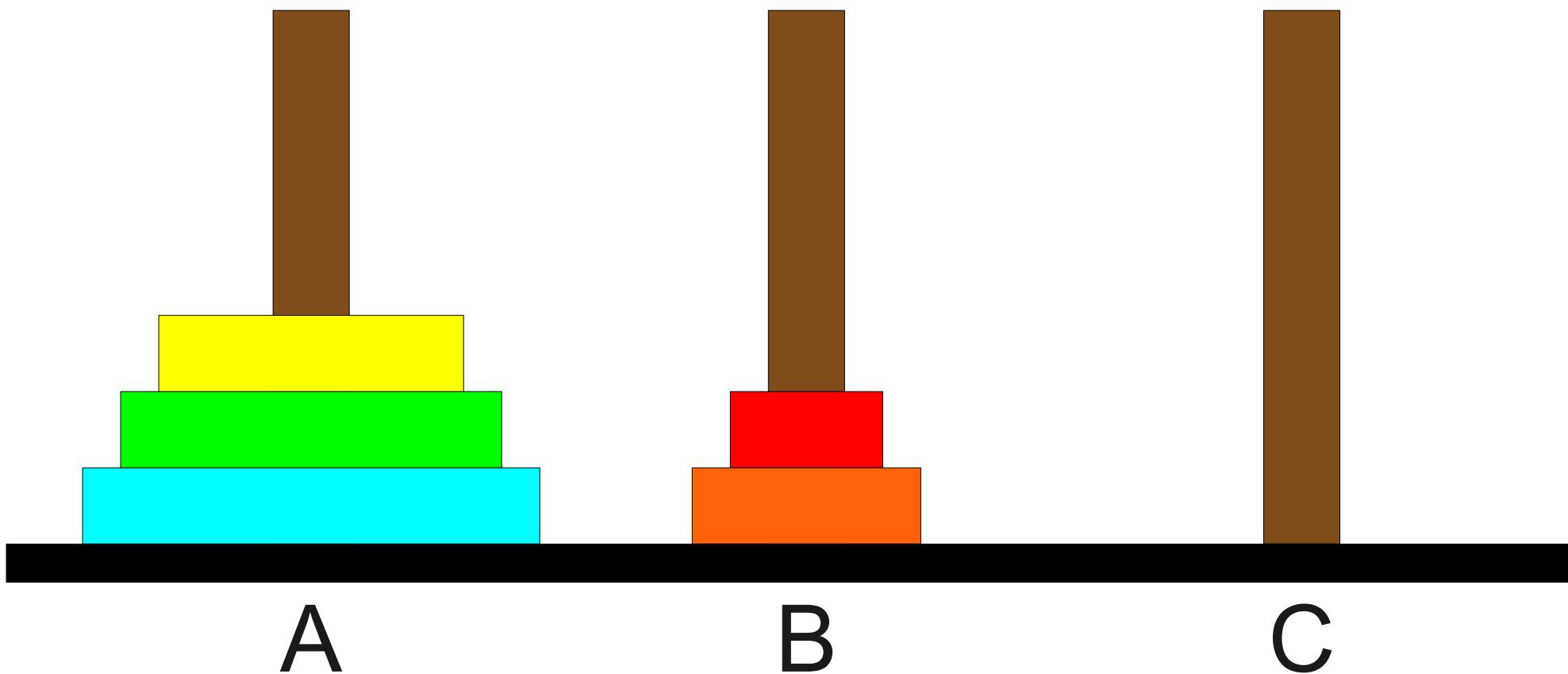
# Towers of Hanoi



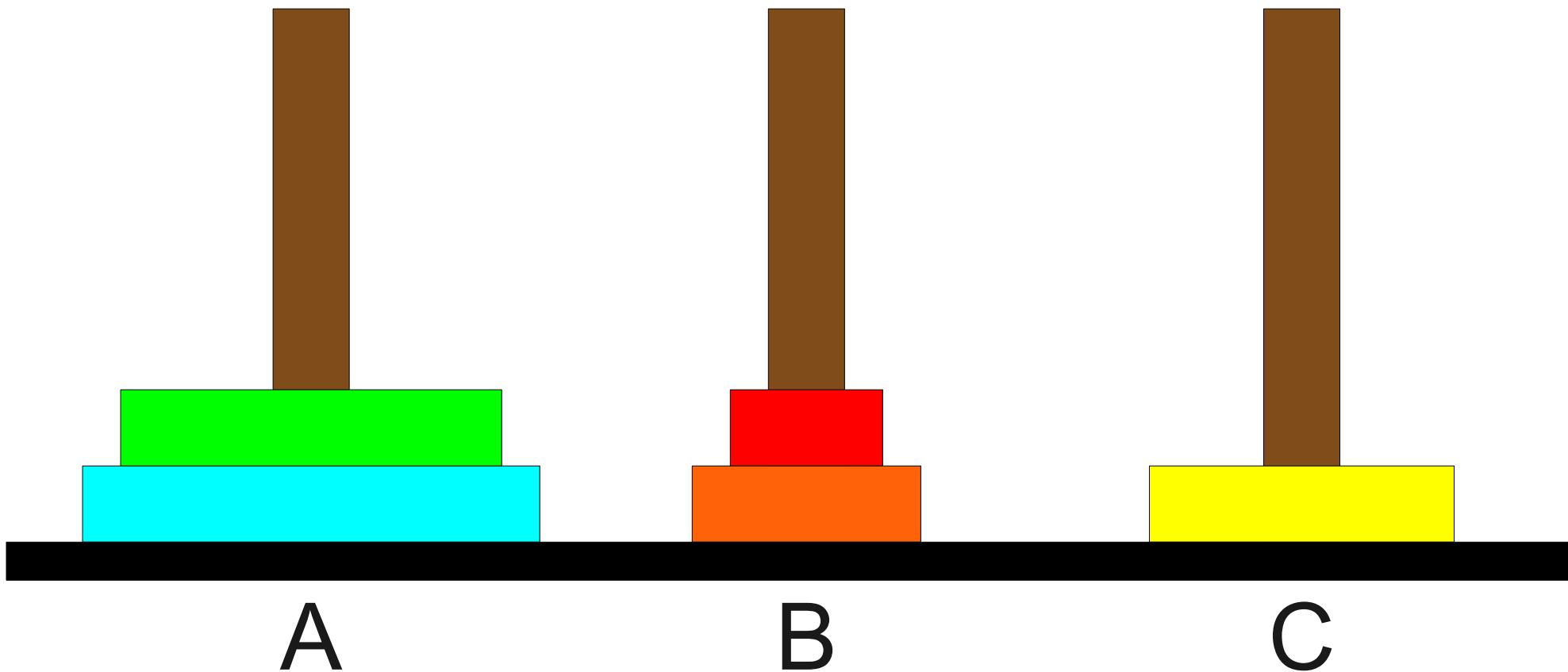
# Towers of Hanoi



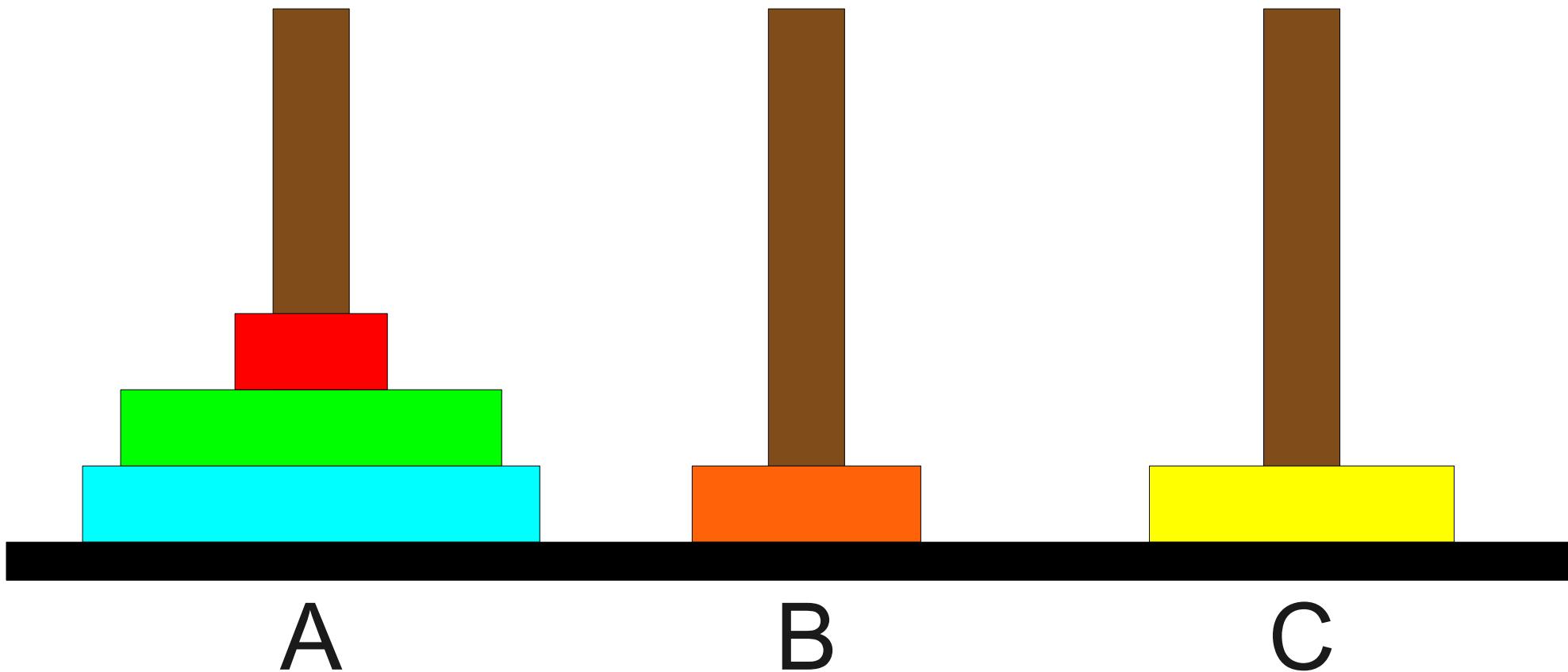
# Towers of Hanoi



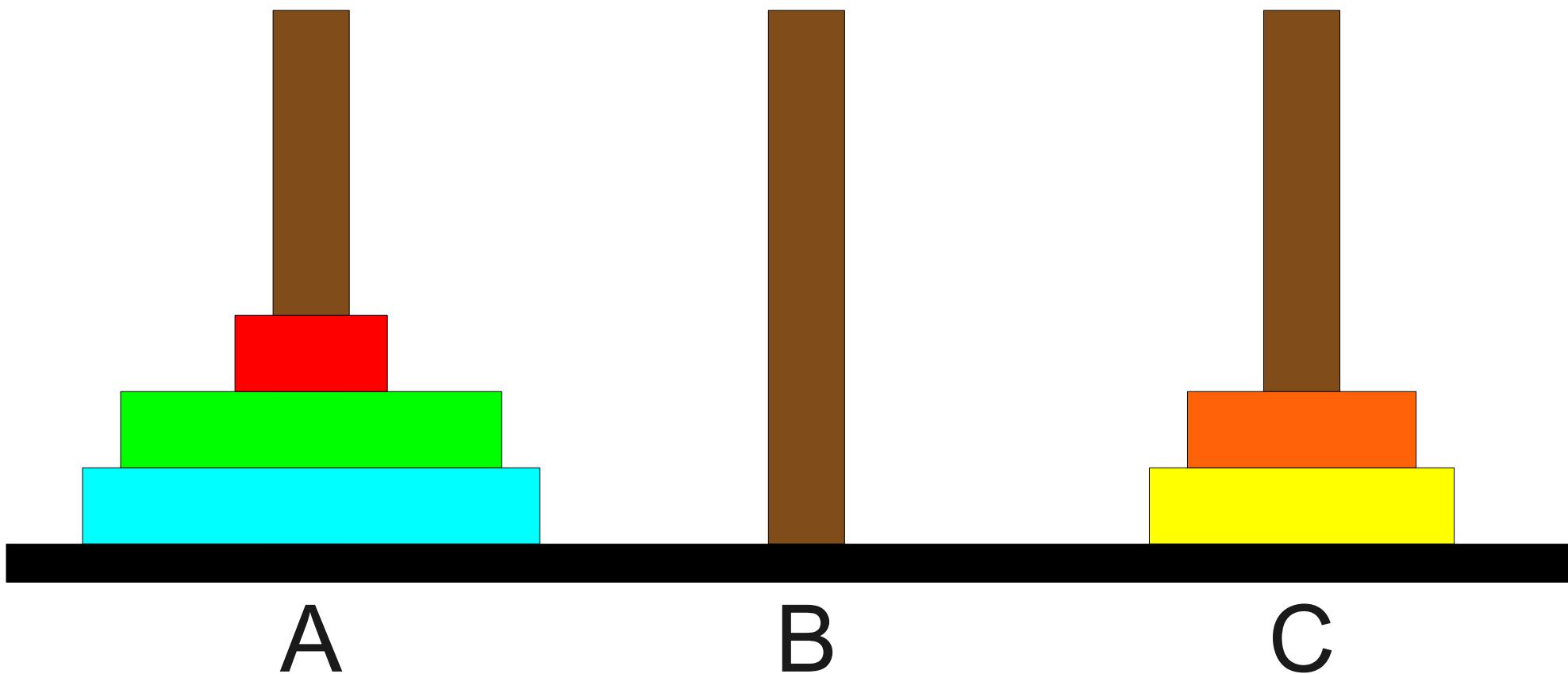
# Towers of Hanoi



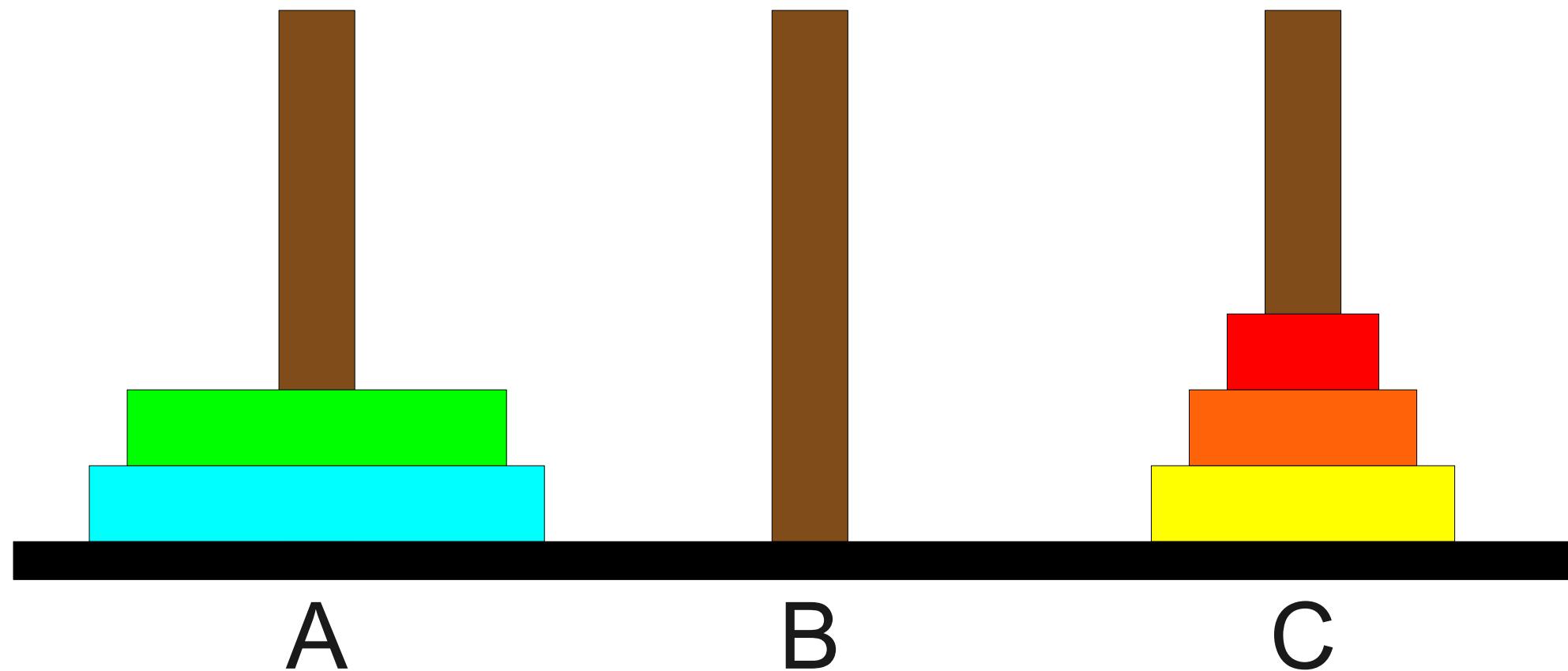
# Towers of Hanoi



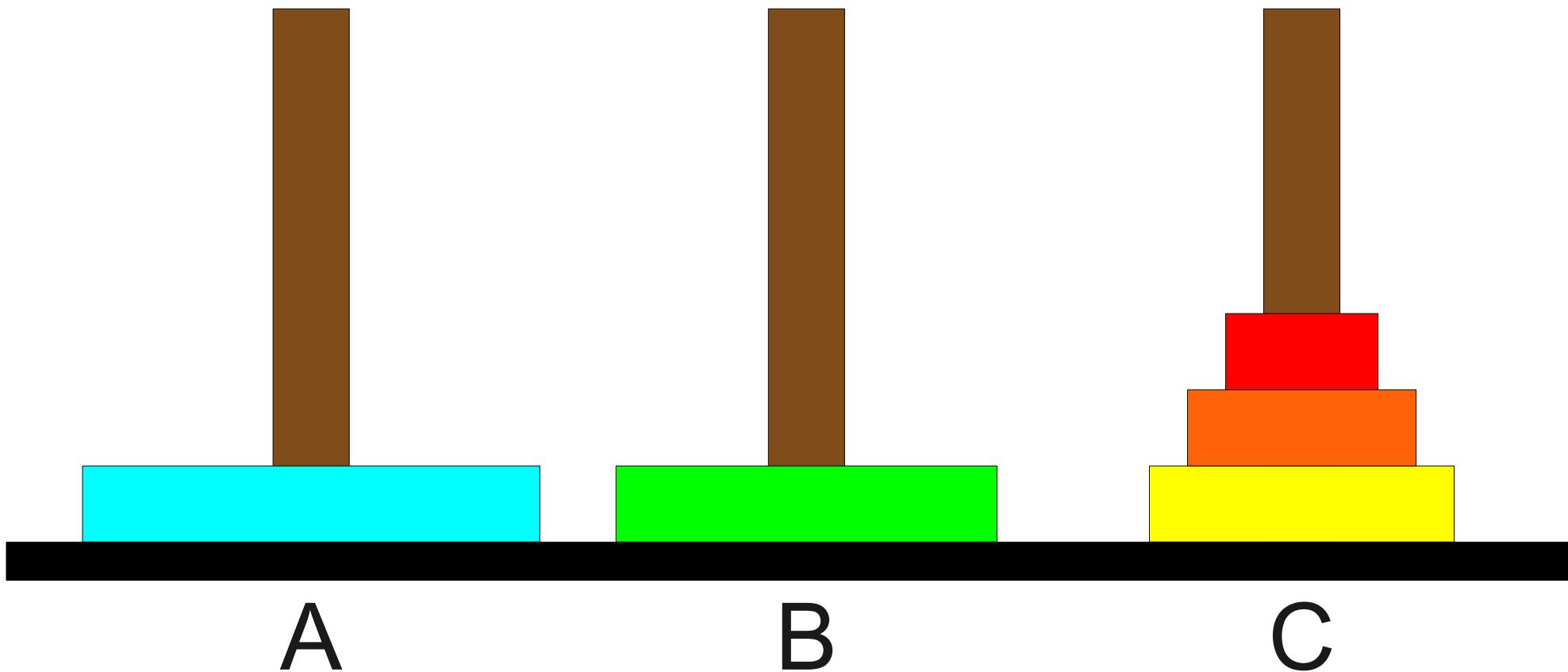
# Towers of Hanoi



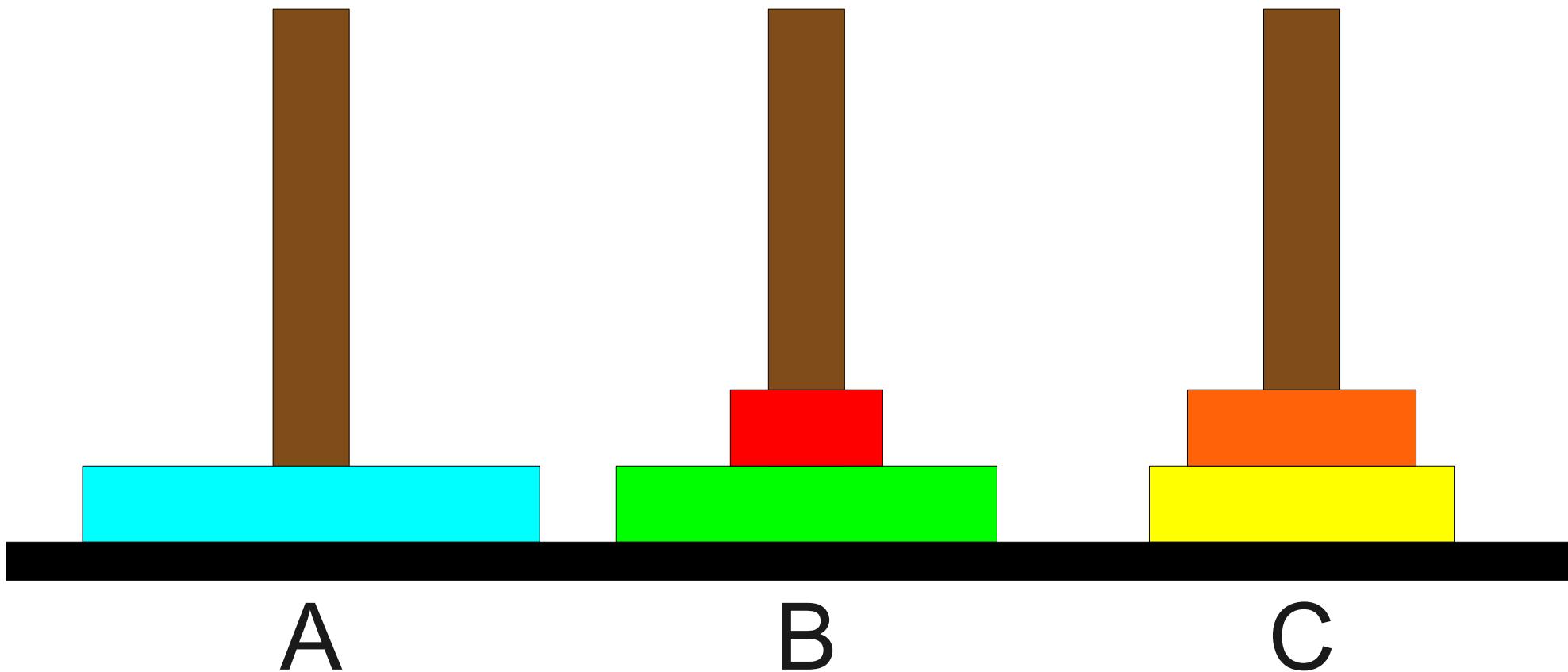
# Towers of Hanoi



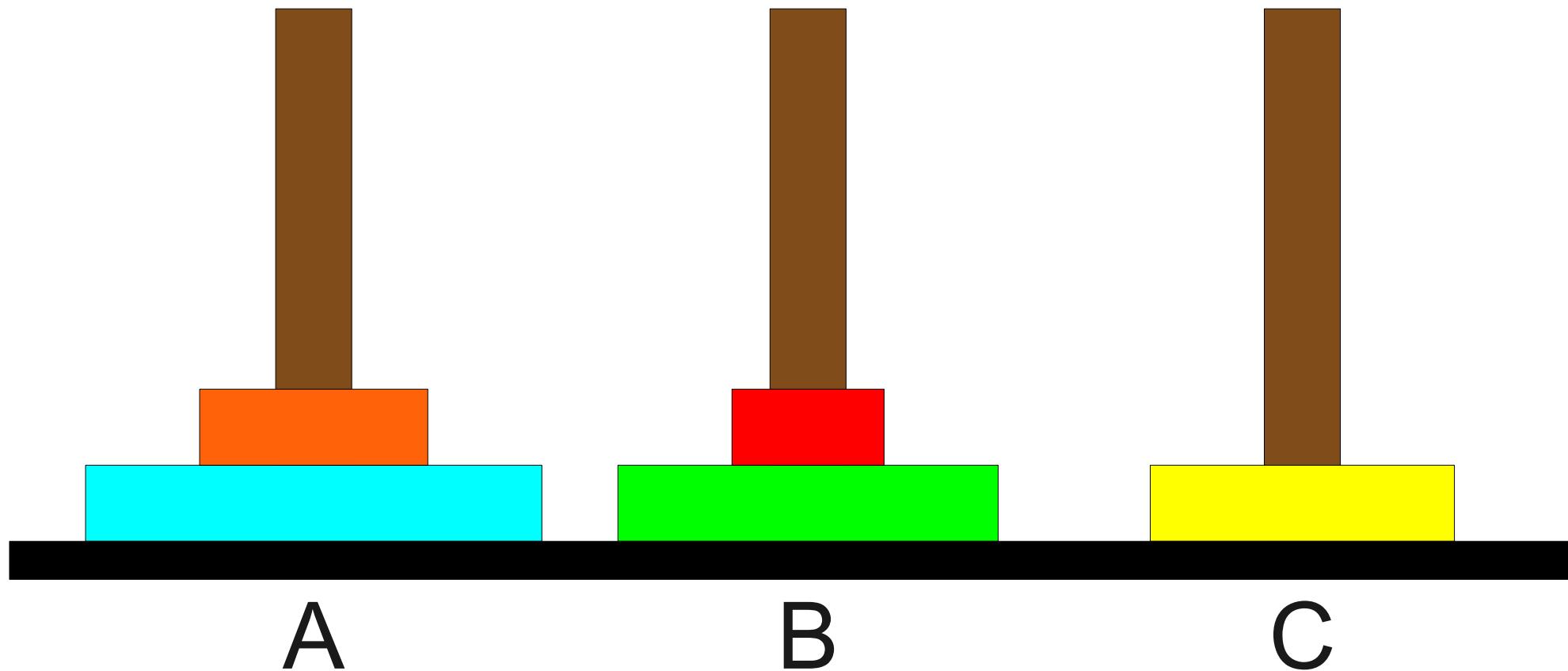
# Towers of Hanoi



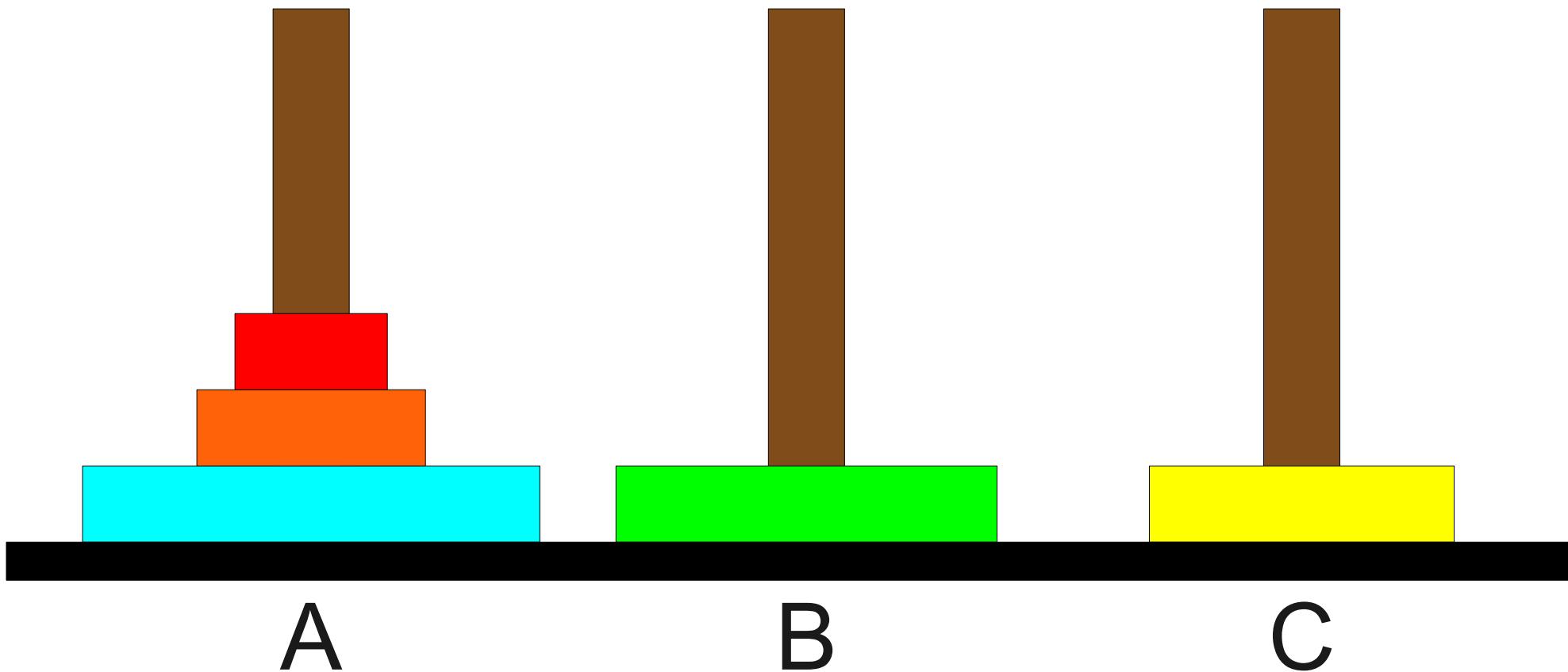
# Towers of Hanoi



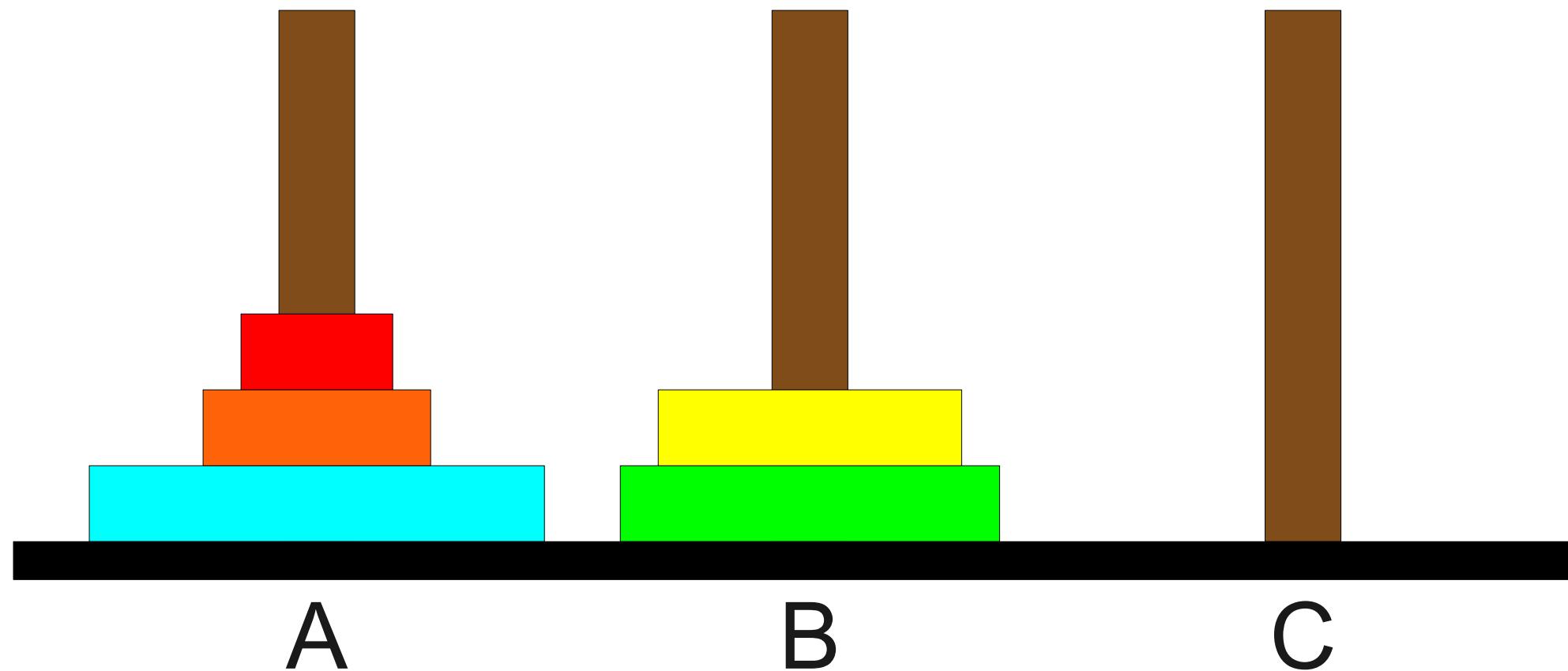
# Towers of Hanoi



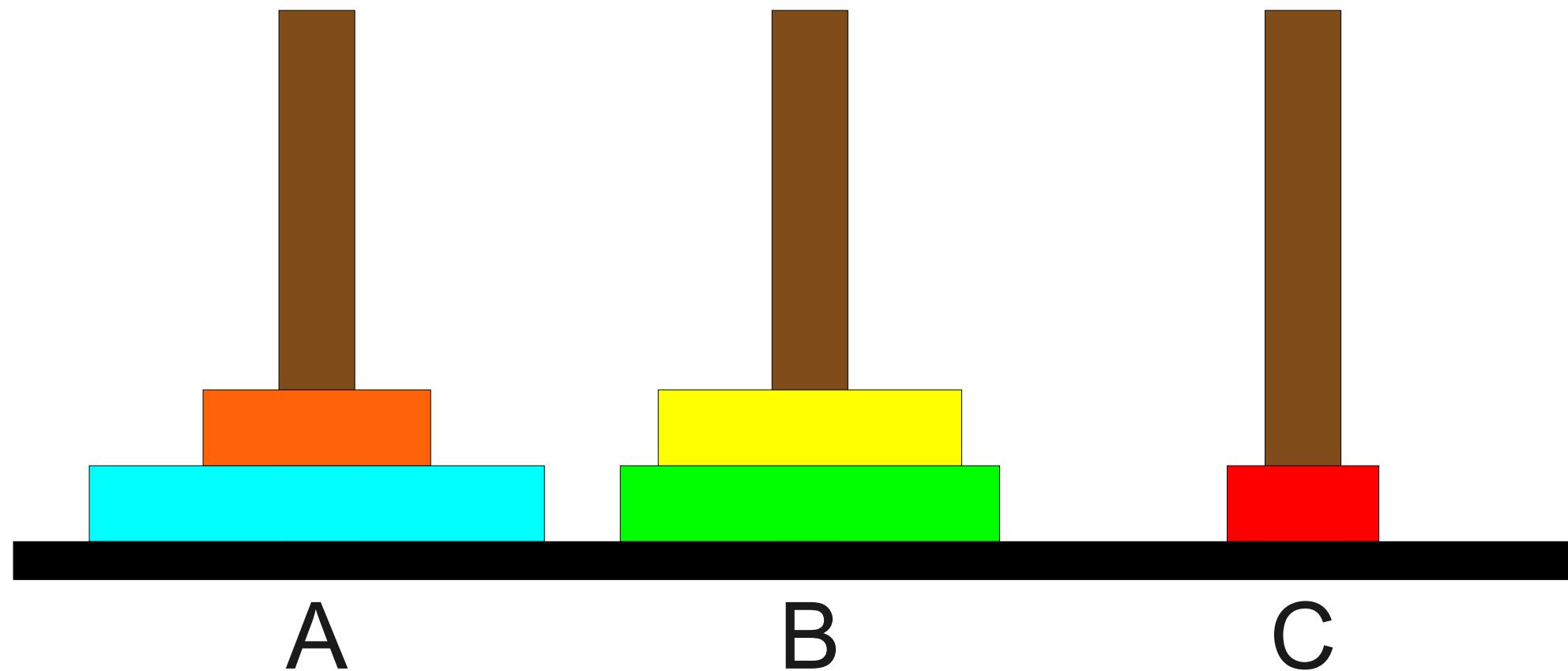
# Towers of Hanoi



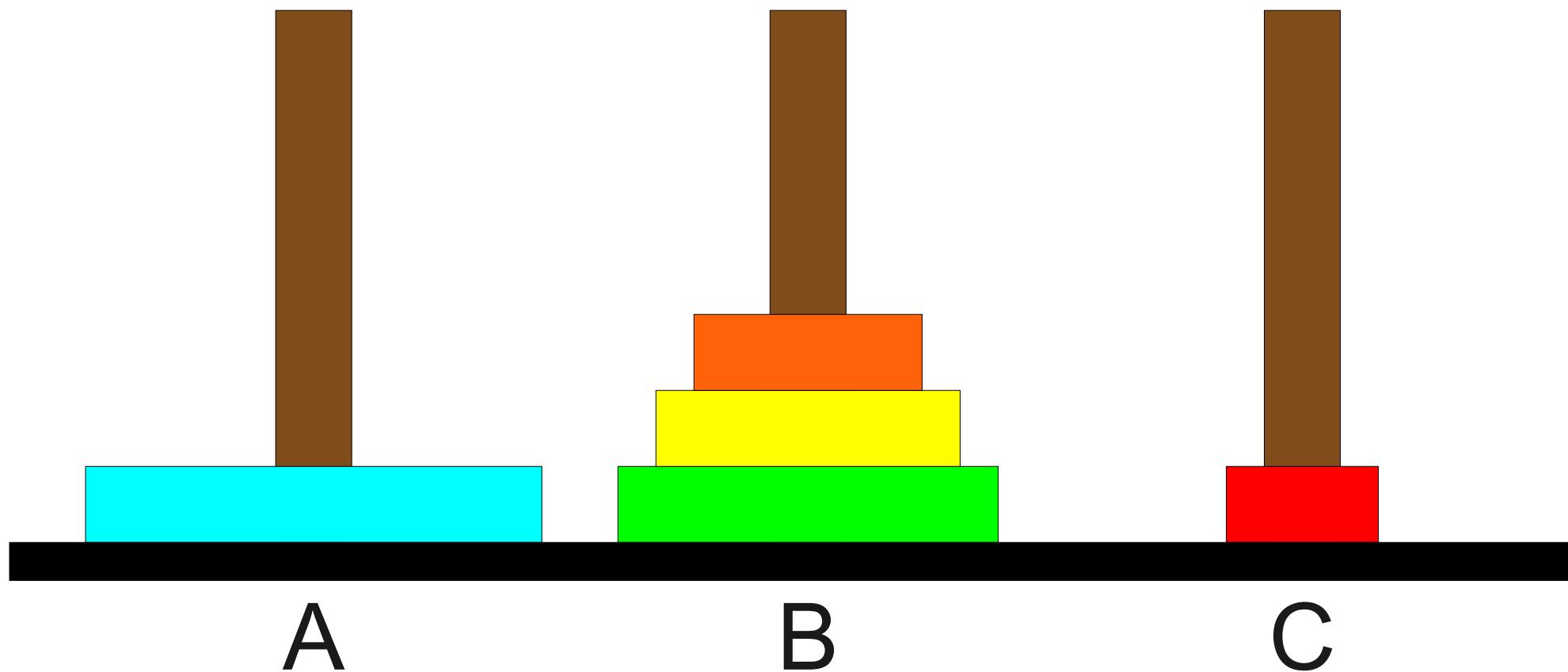
# Towers of Hanoi



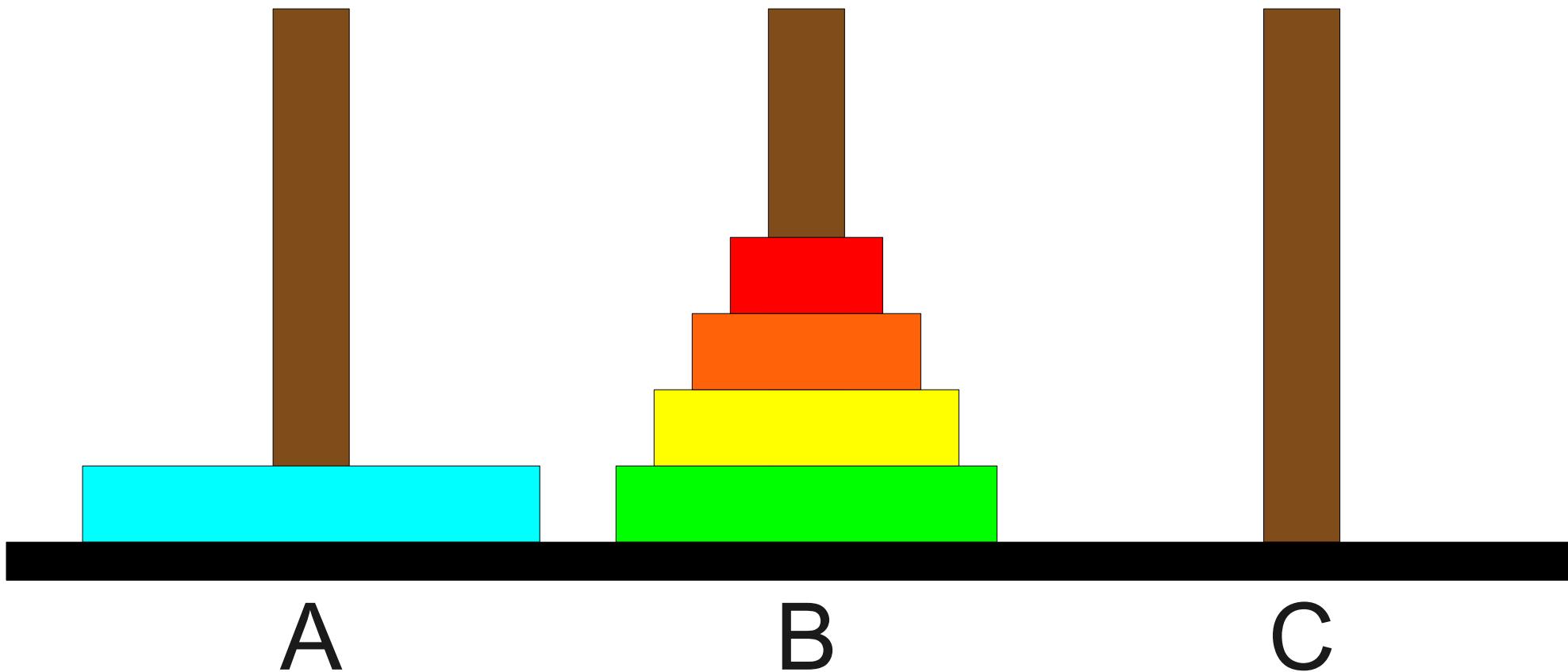
# Towers of Hanoi



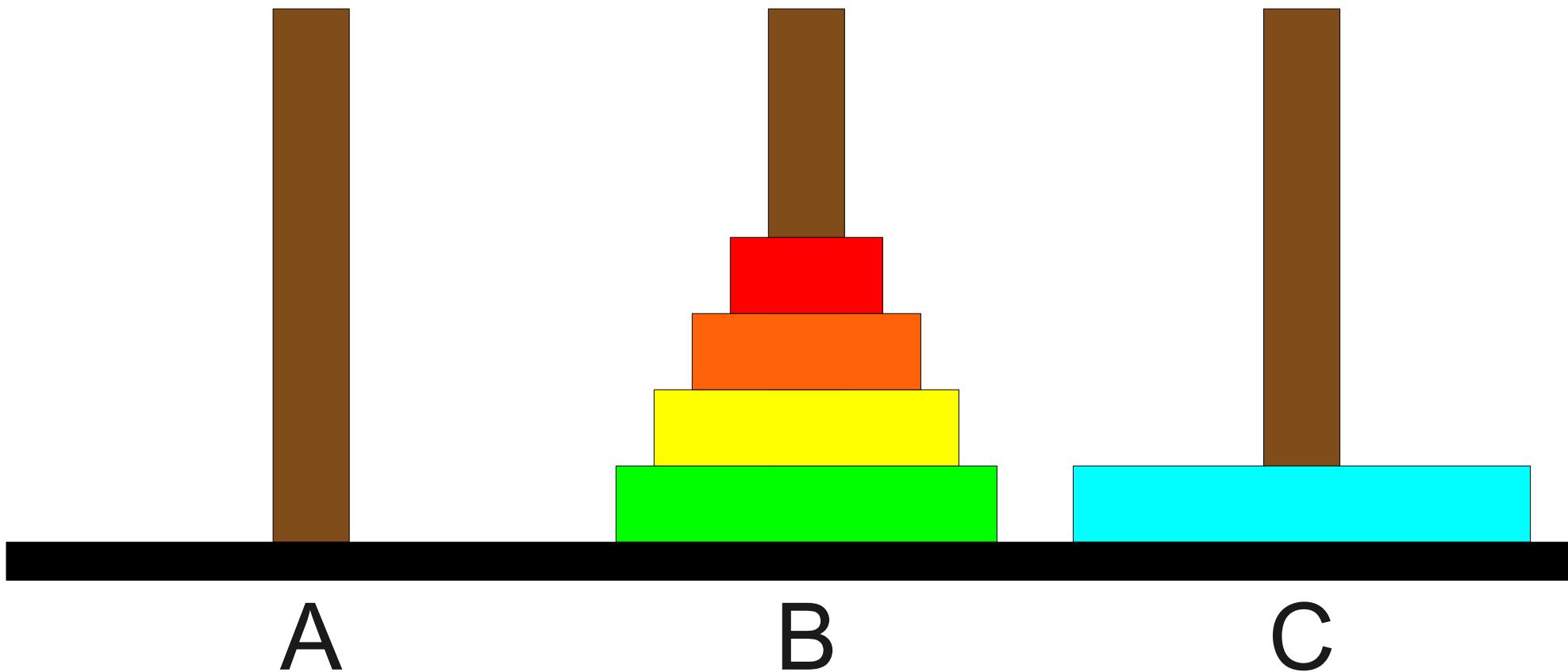
# Towers of Hanoi



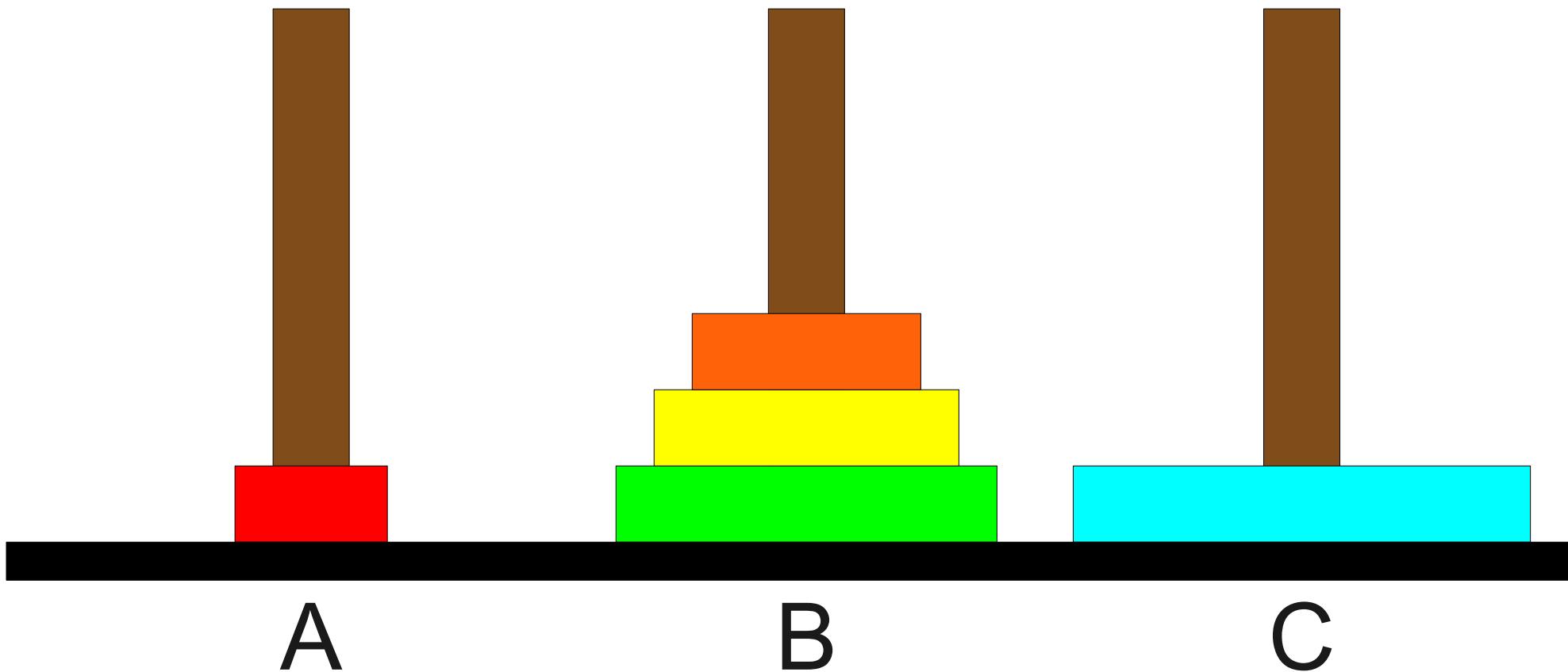
# Towers of Hanoi



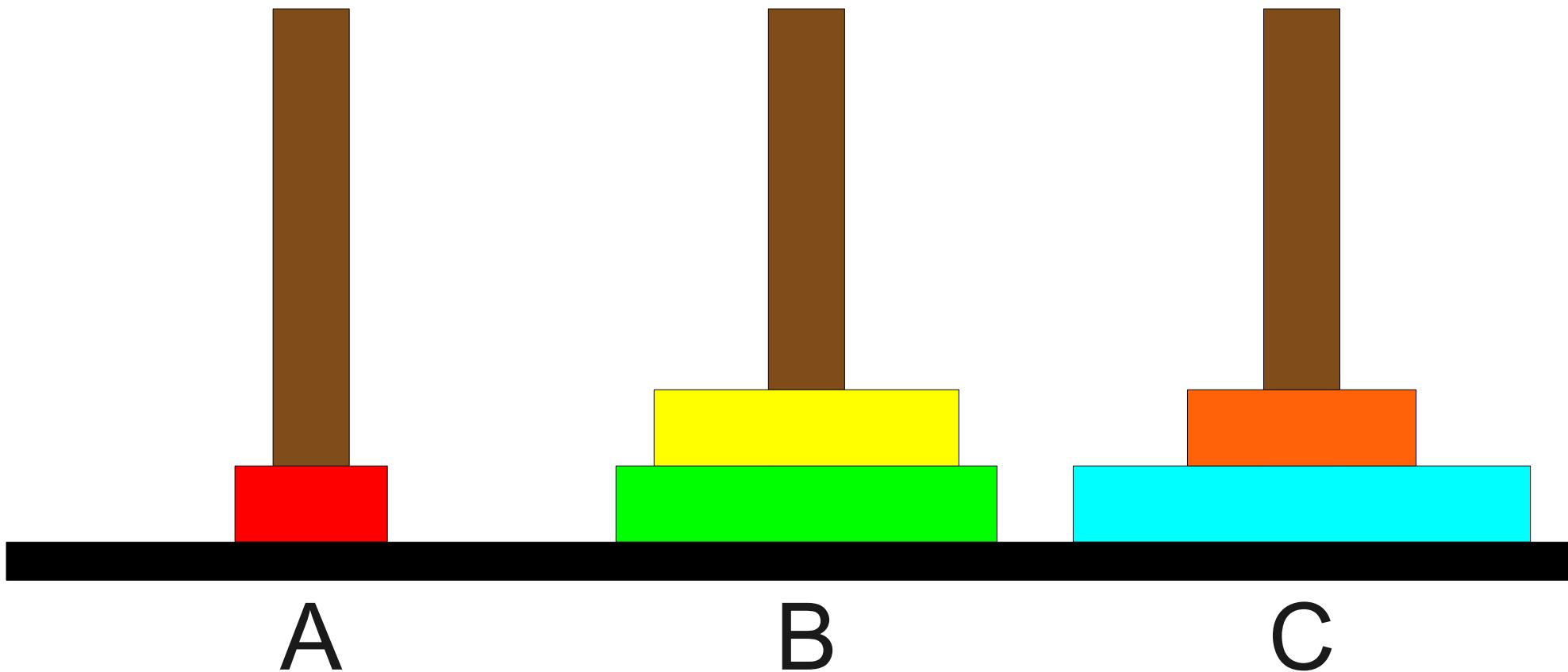
# Towers of Hanoi



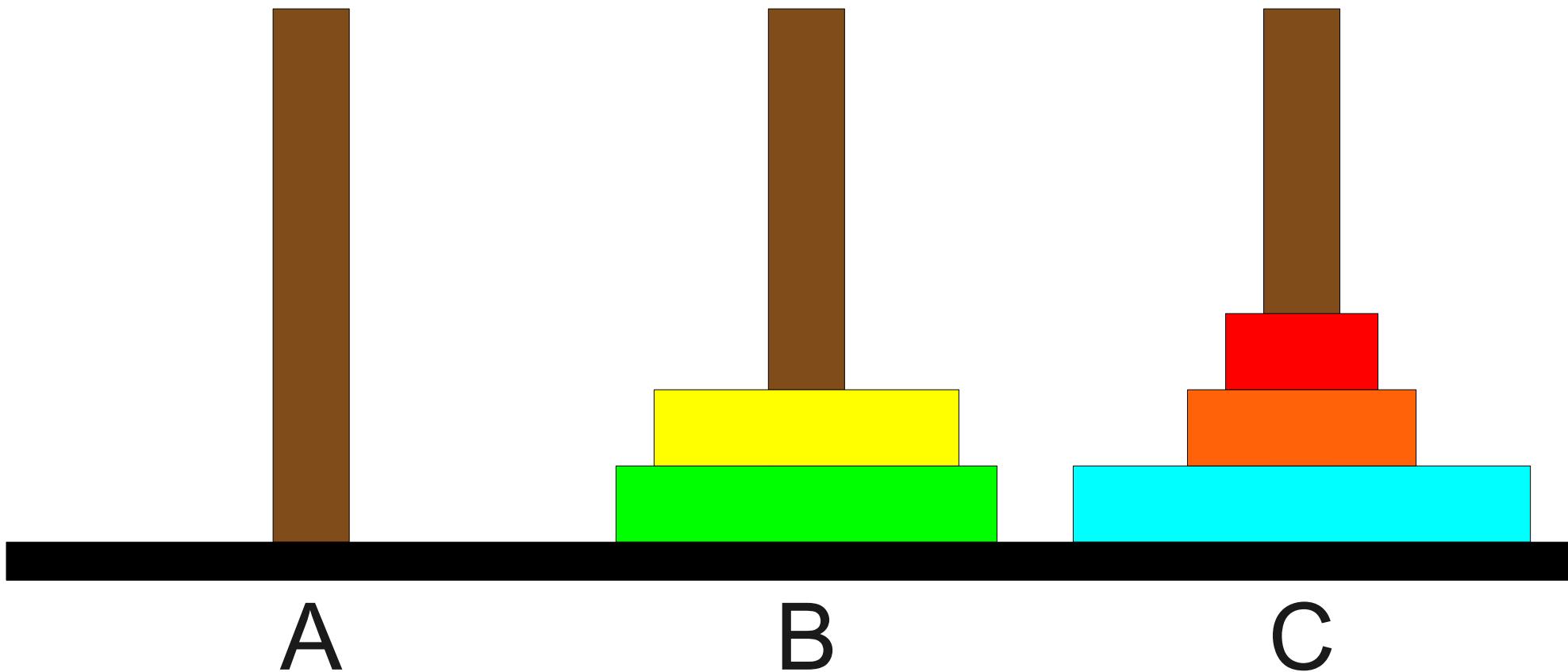
# Towers of Hanoi



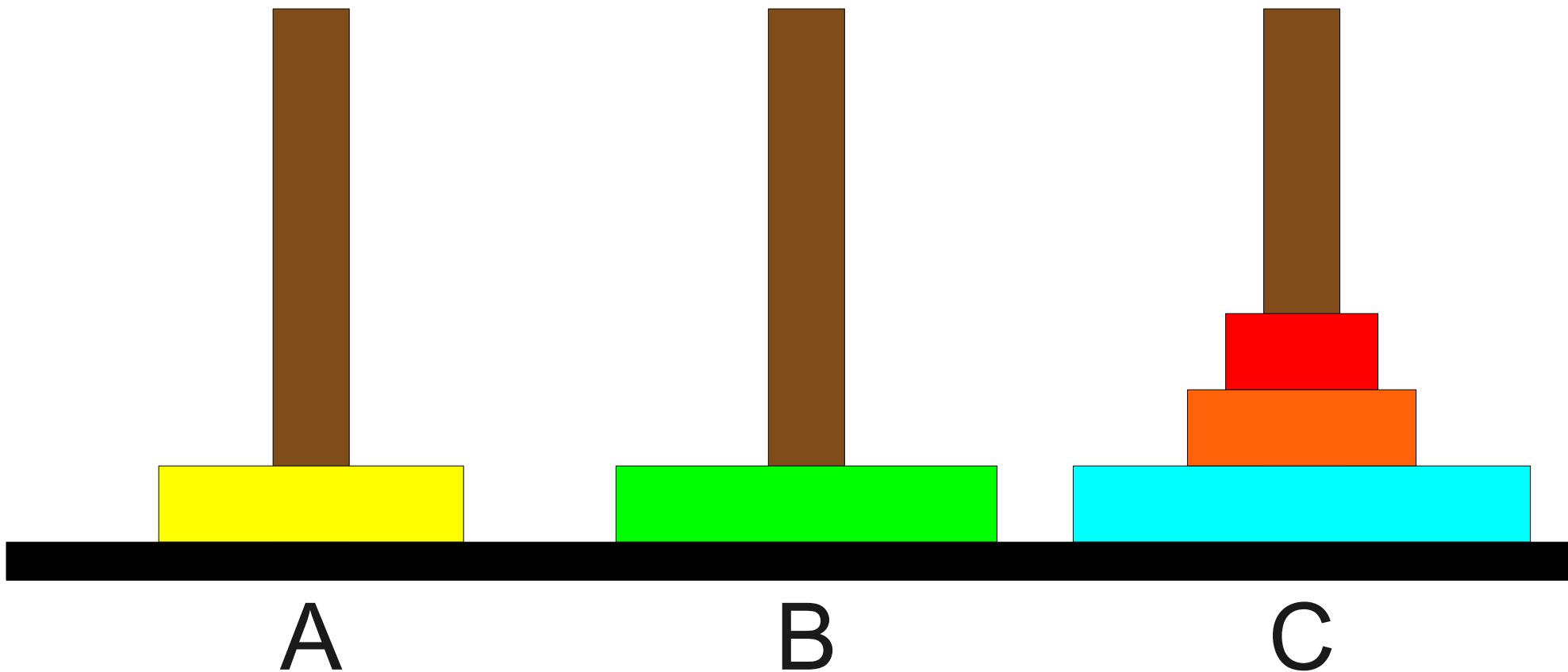
# Towers of Hanoi



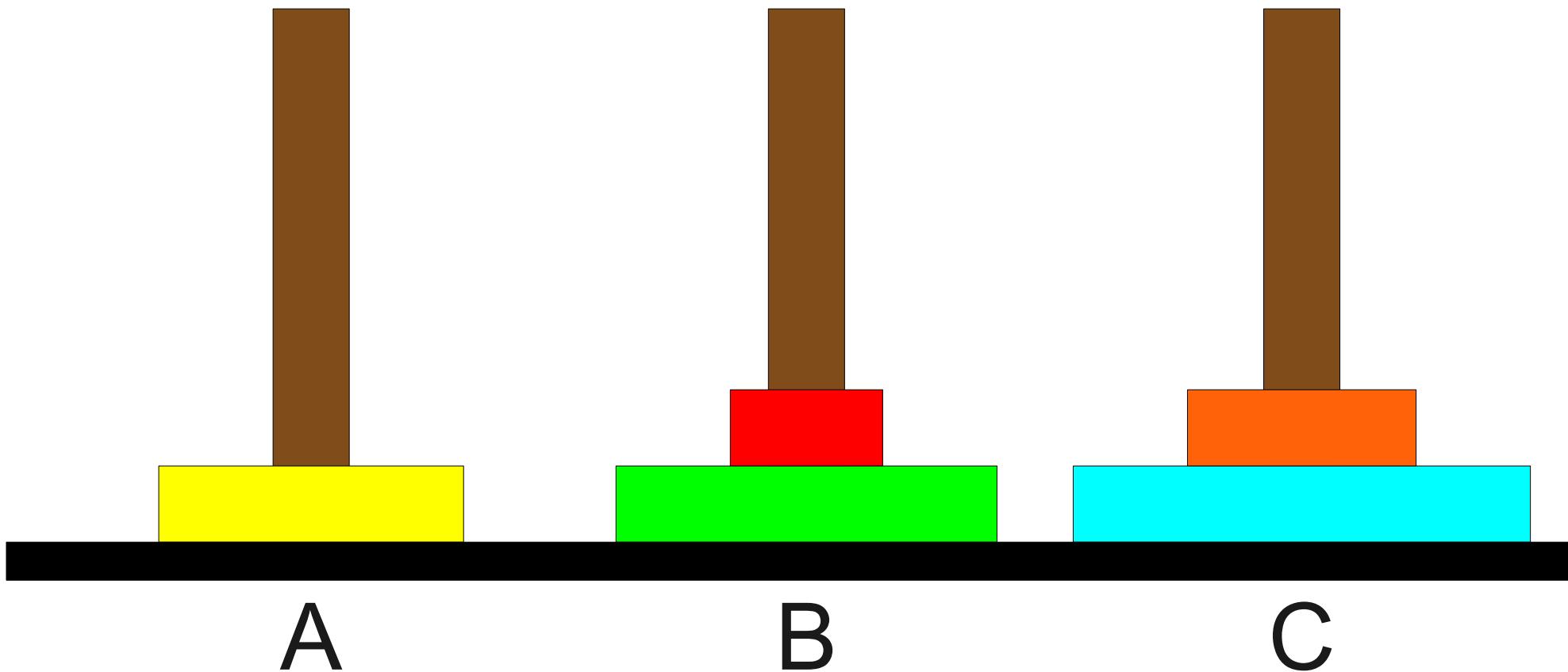
# Towers of Hanoi



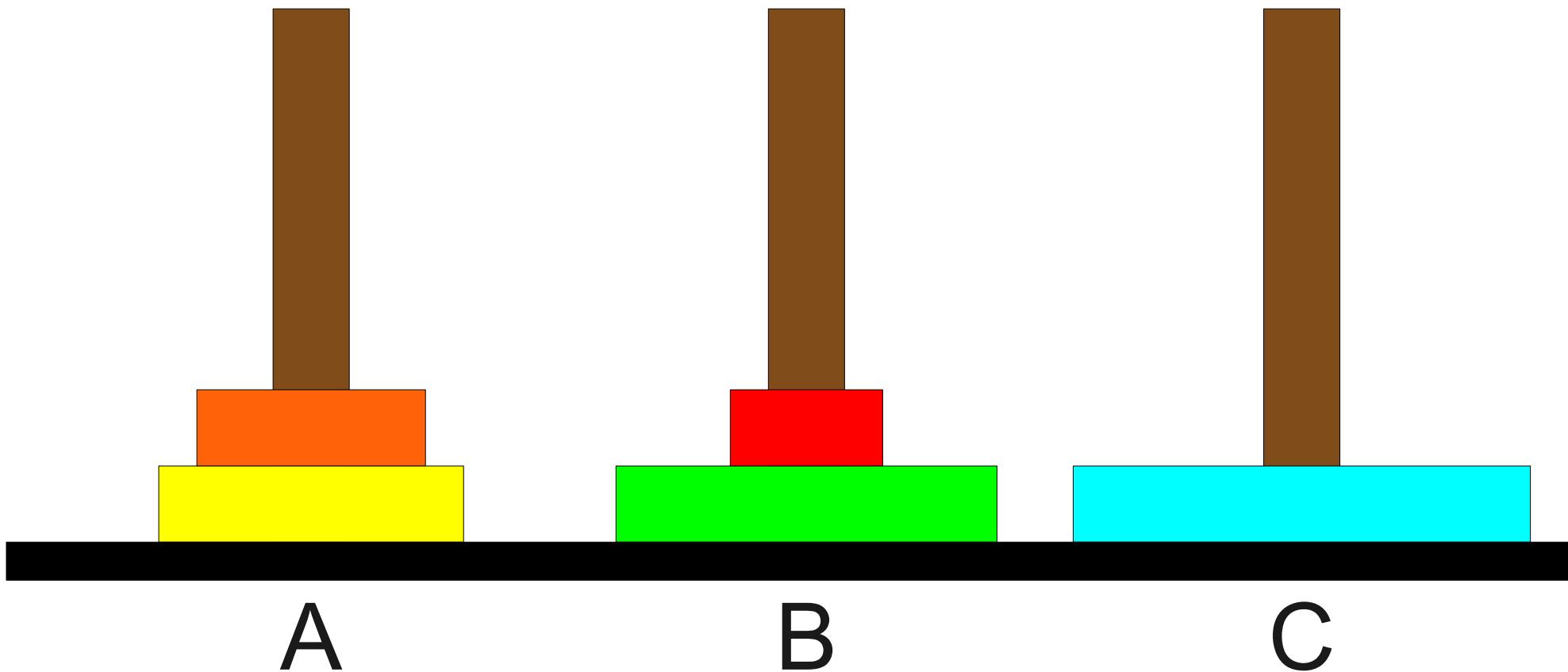
# Towers of Hanoi



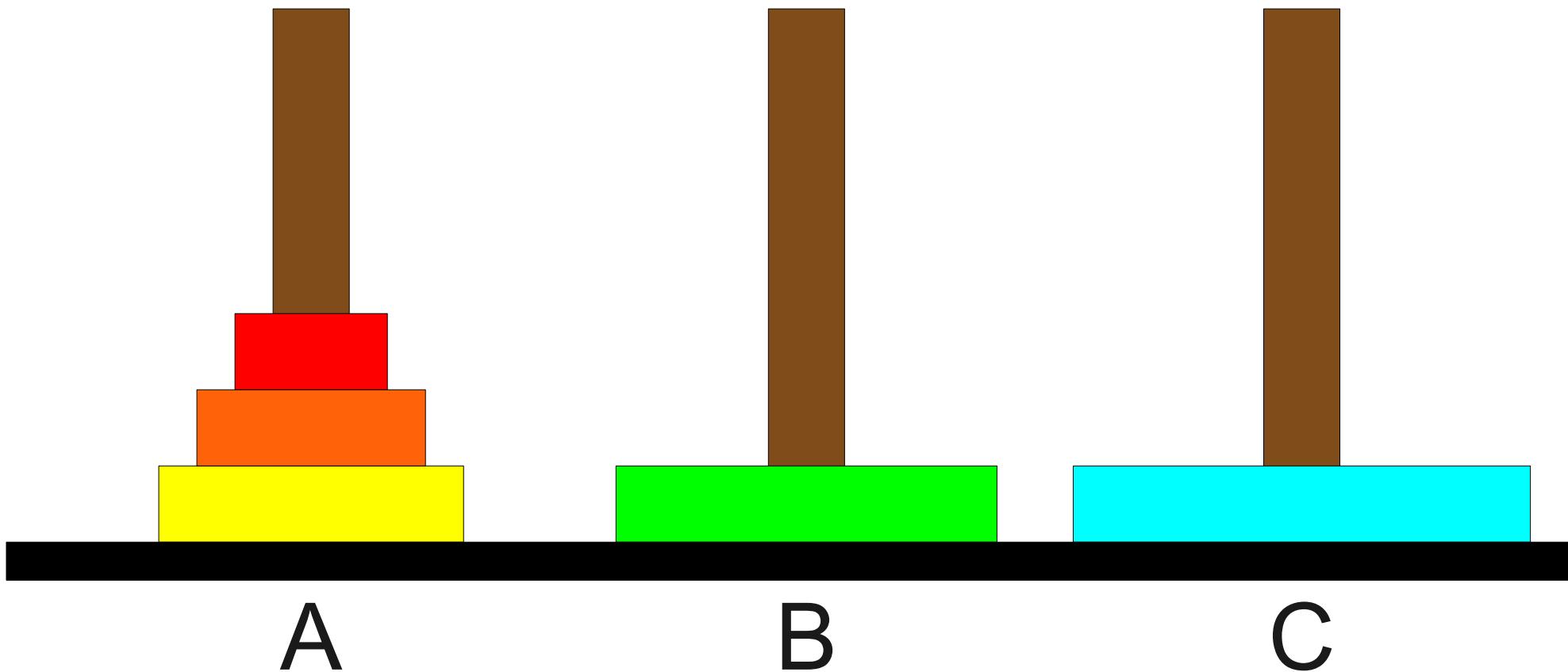
# Towers of Hanoi



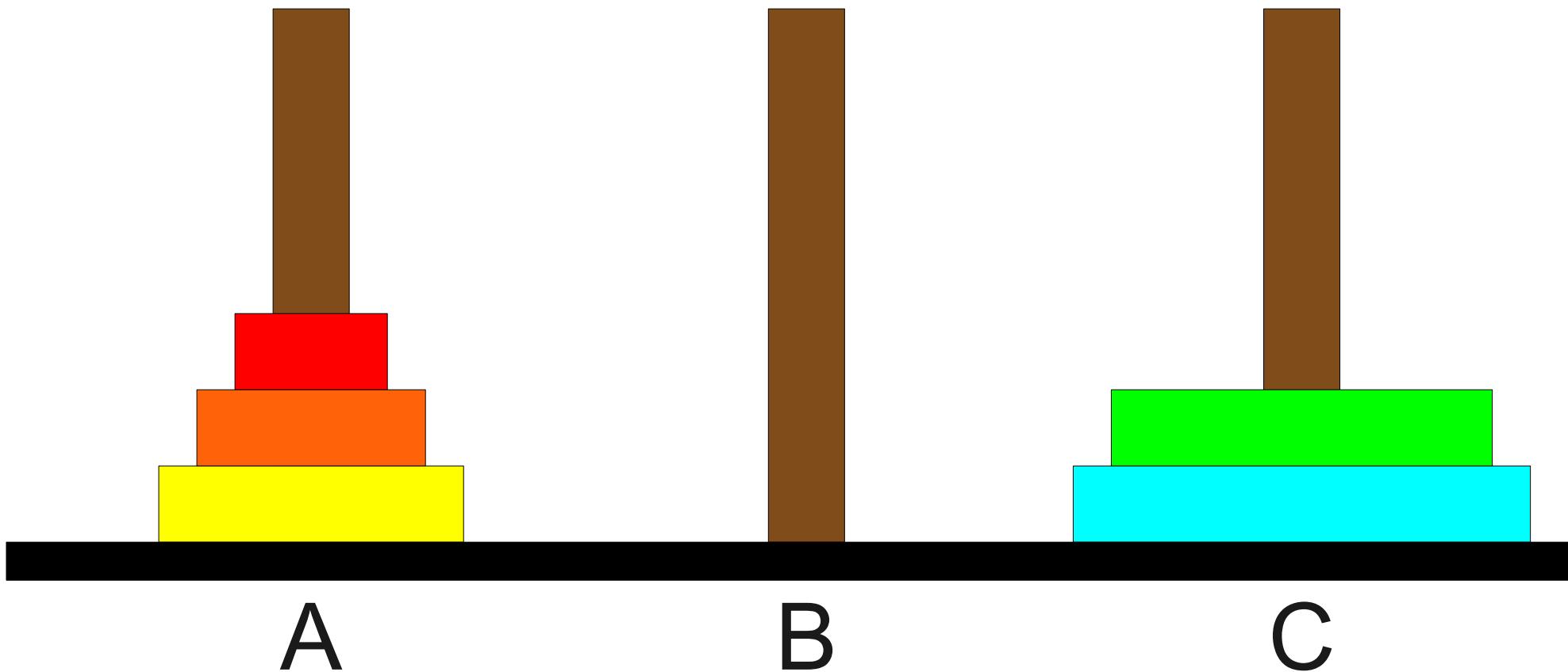
# Towers of Hanoi



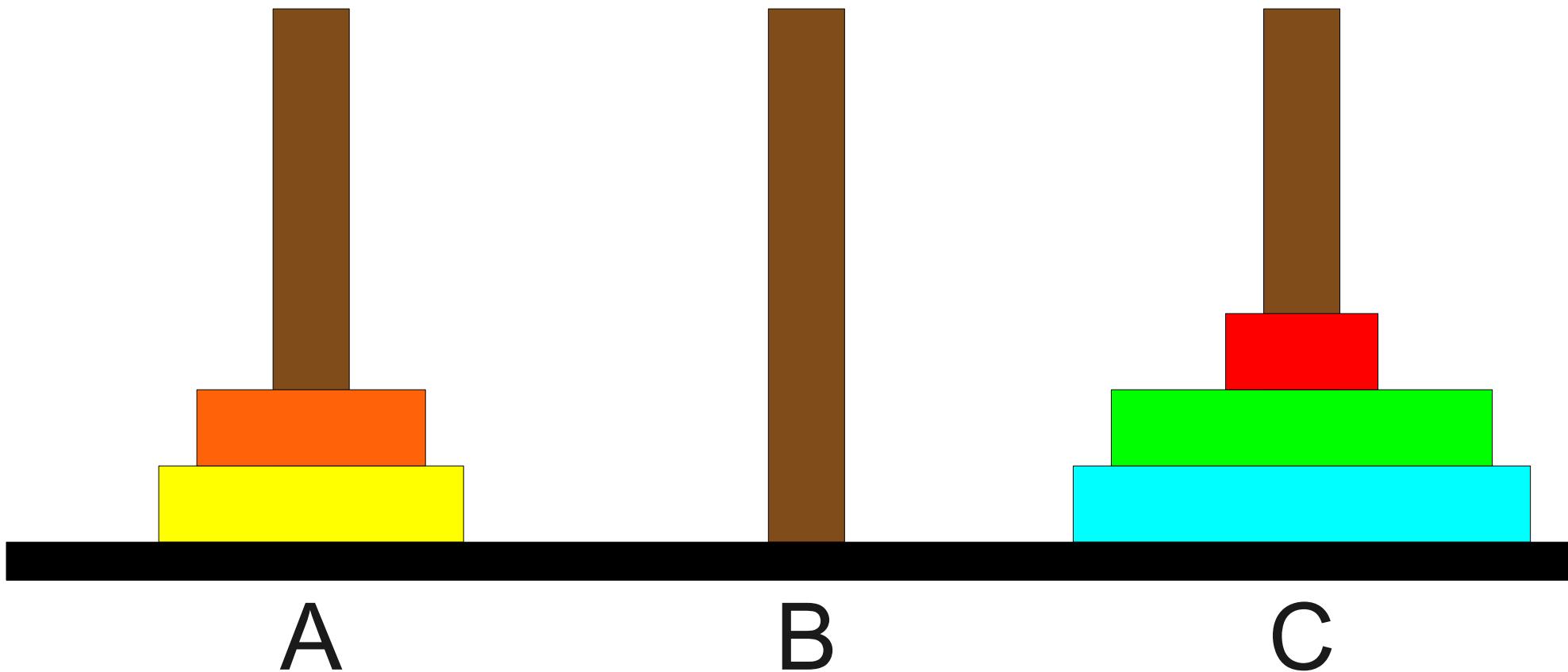
# Towers of Hanoi



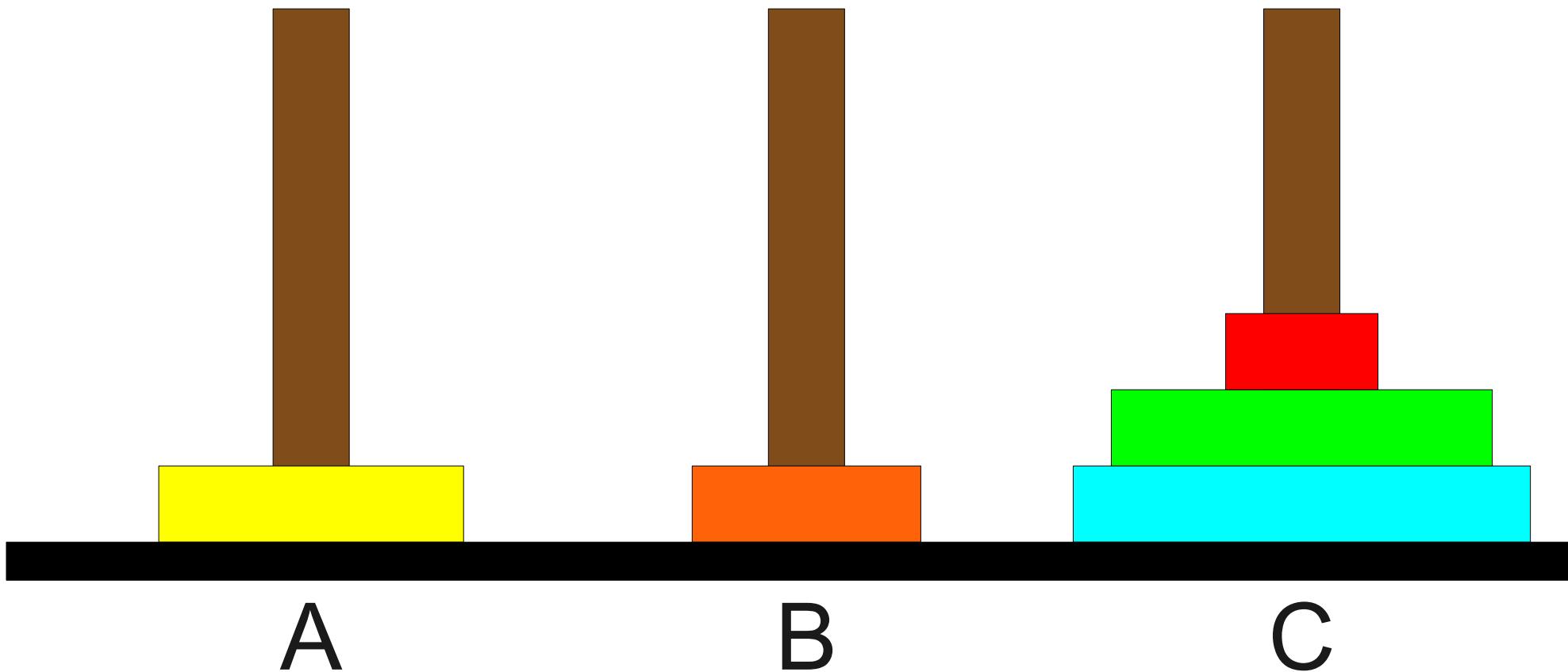
# Towers of Hanoi



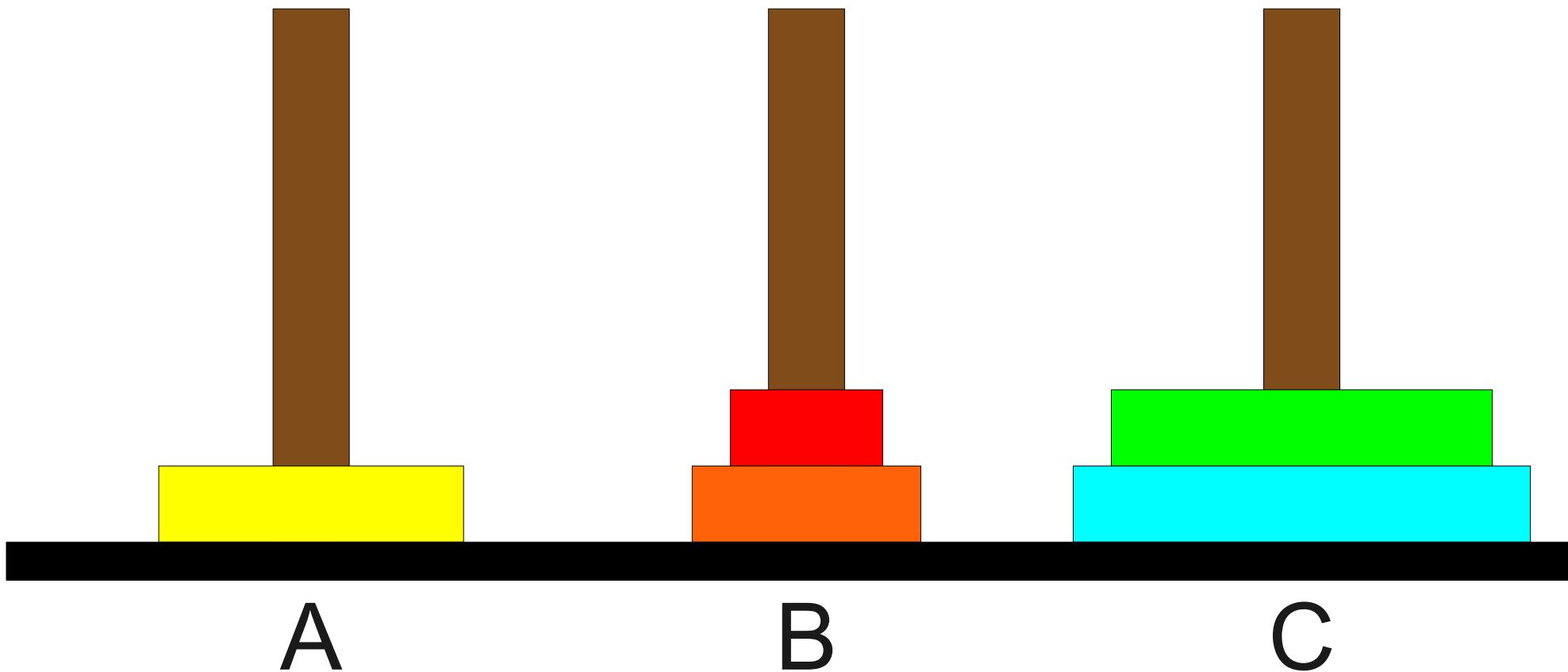
# Towers of Hanoi



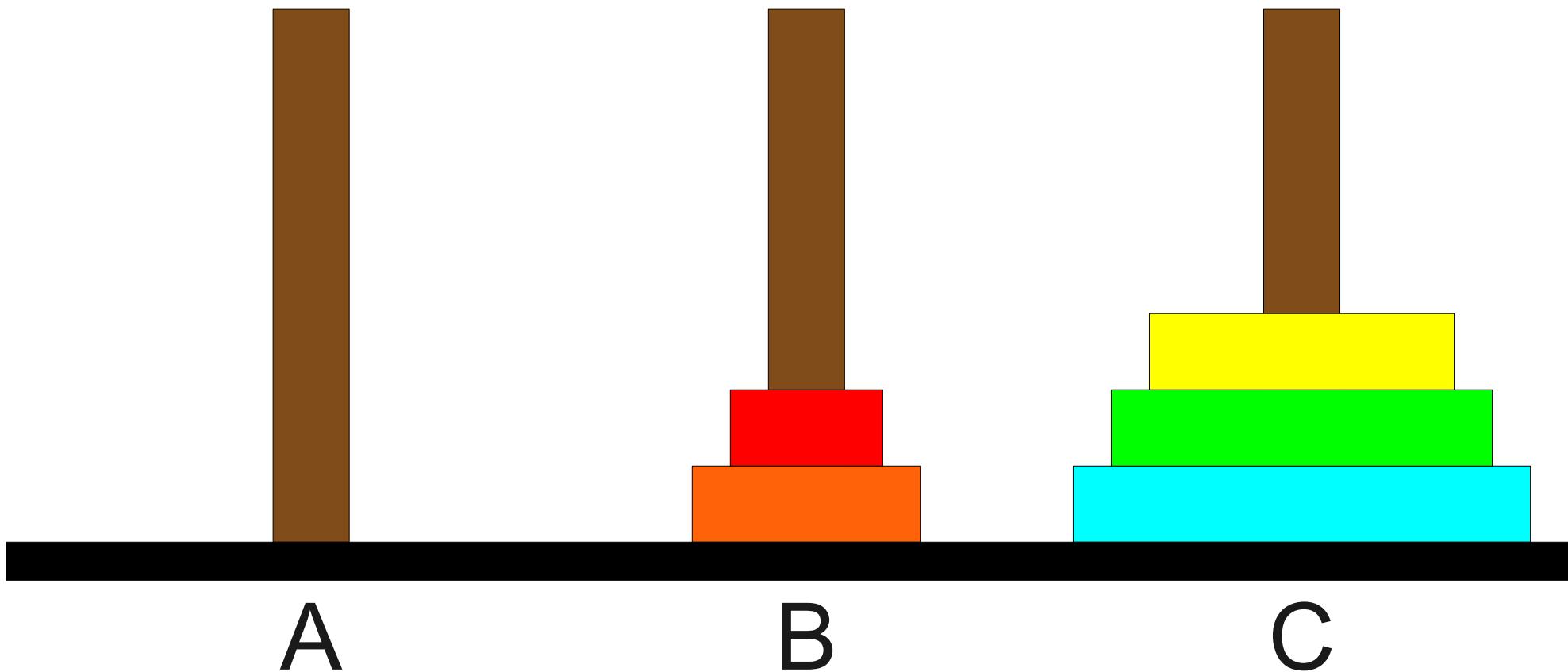
# Towers of Hanoi



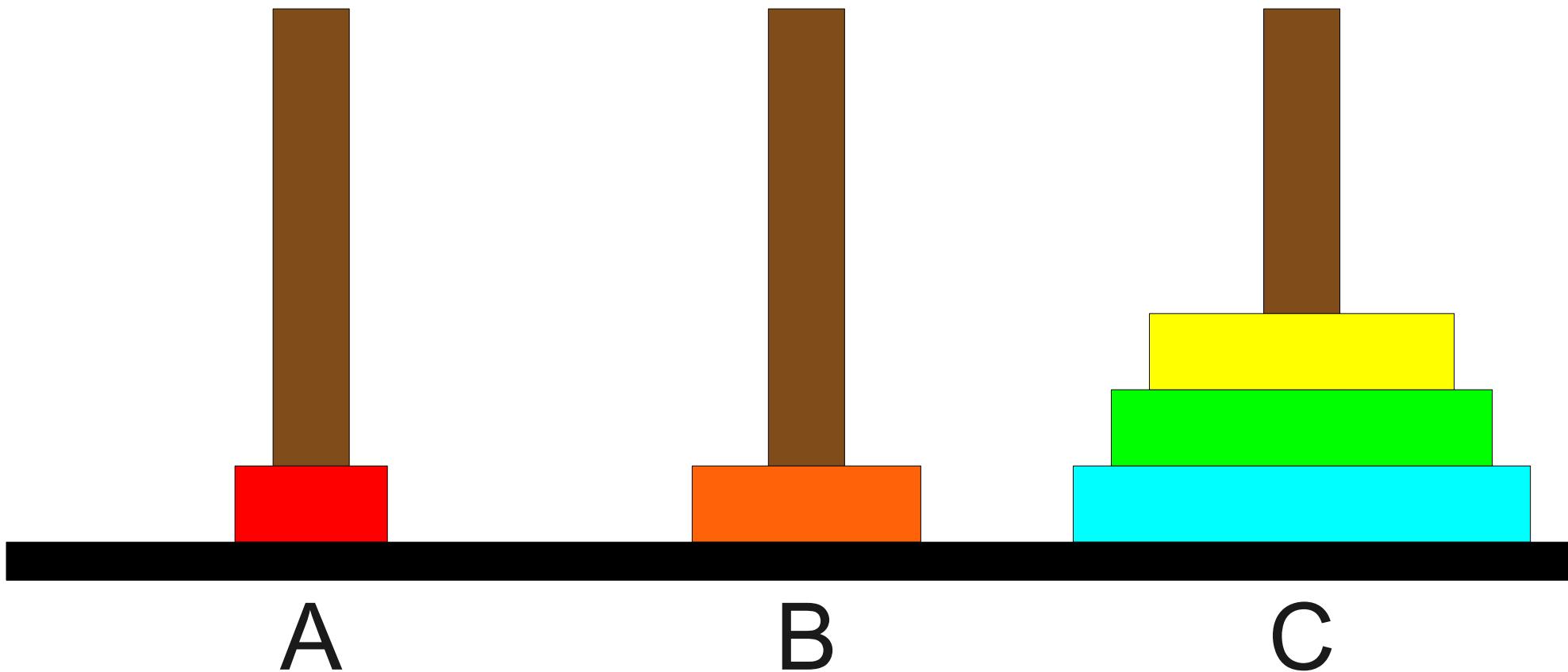
# Towers of Hanoi



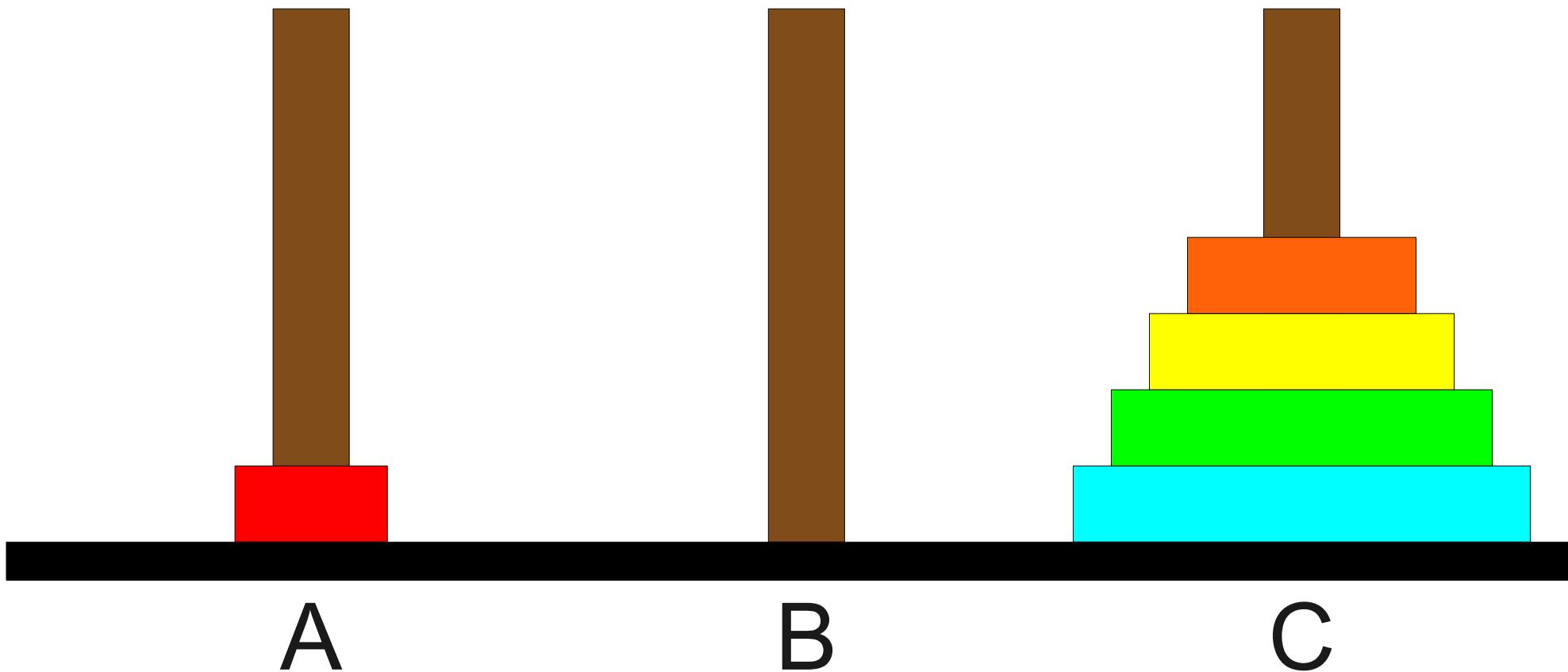
# Towers of Hanoi



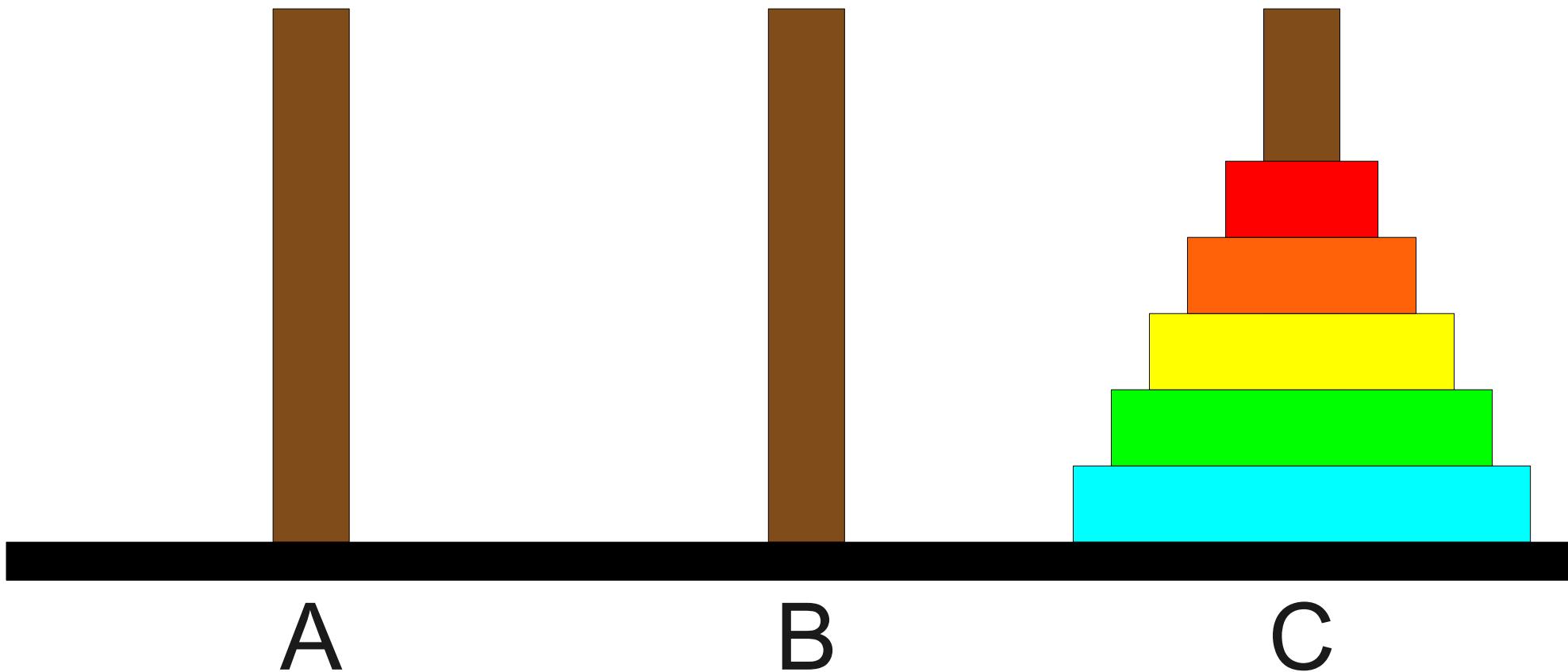
# Towers of Hanoi



# Towers of Hanoi



# Towers of Hanoi



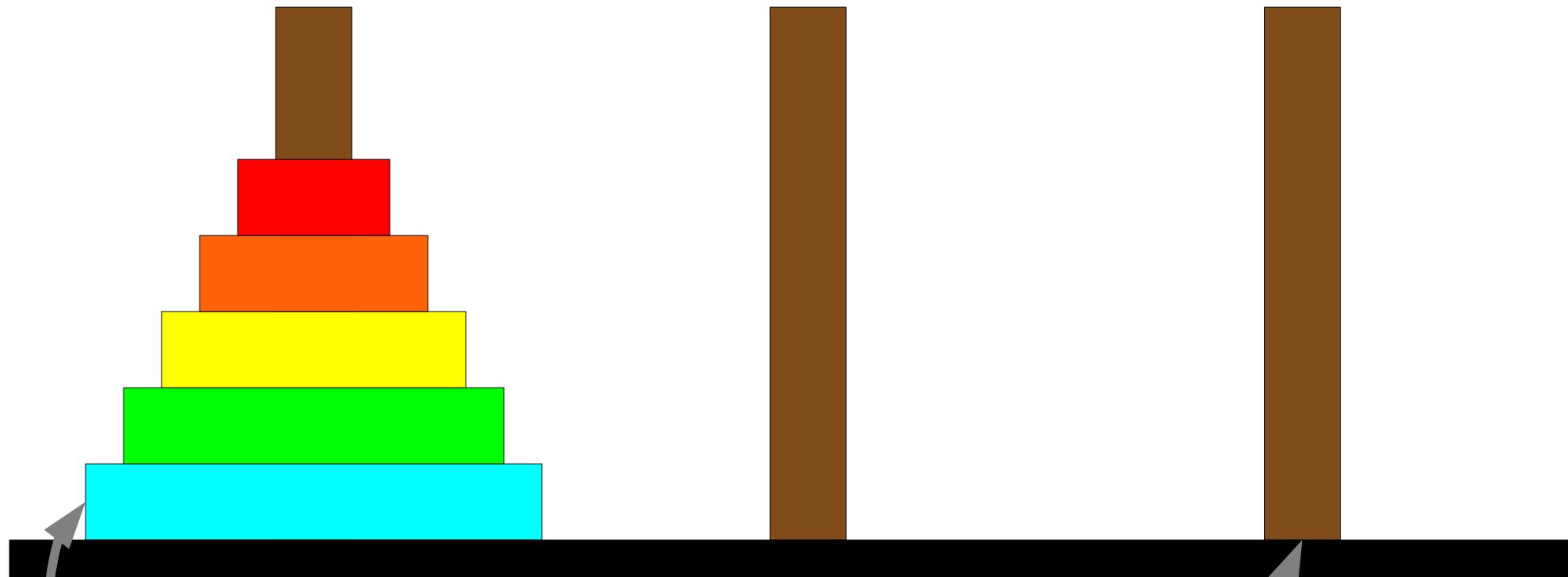
# Solving the Towers of Hanoi

# Solving the Towers of Hanoi

A

B

C



This disk...

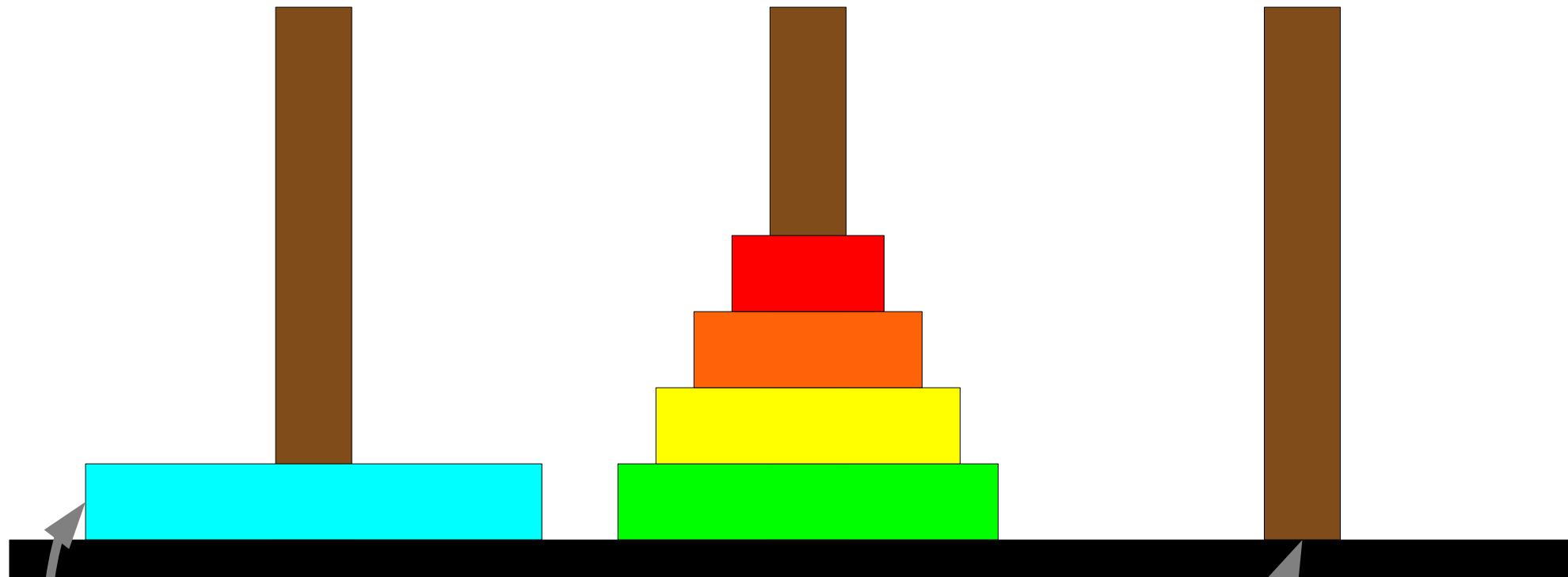
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C



This disk...

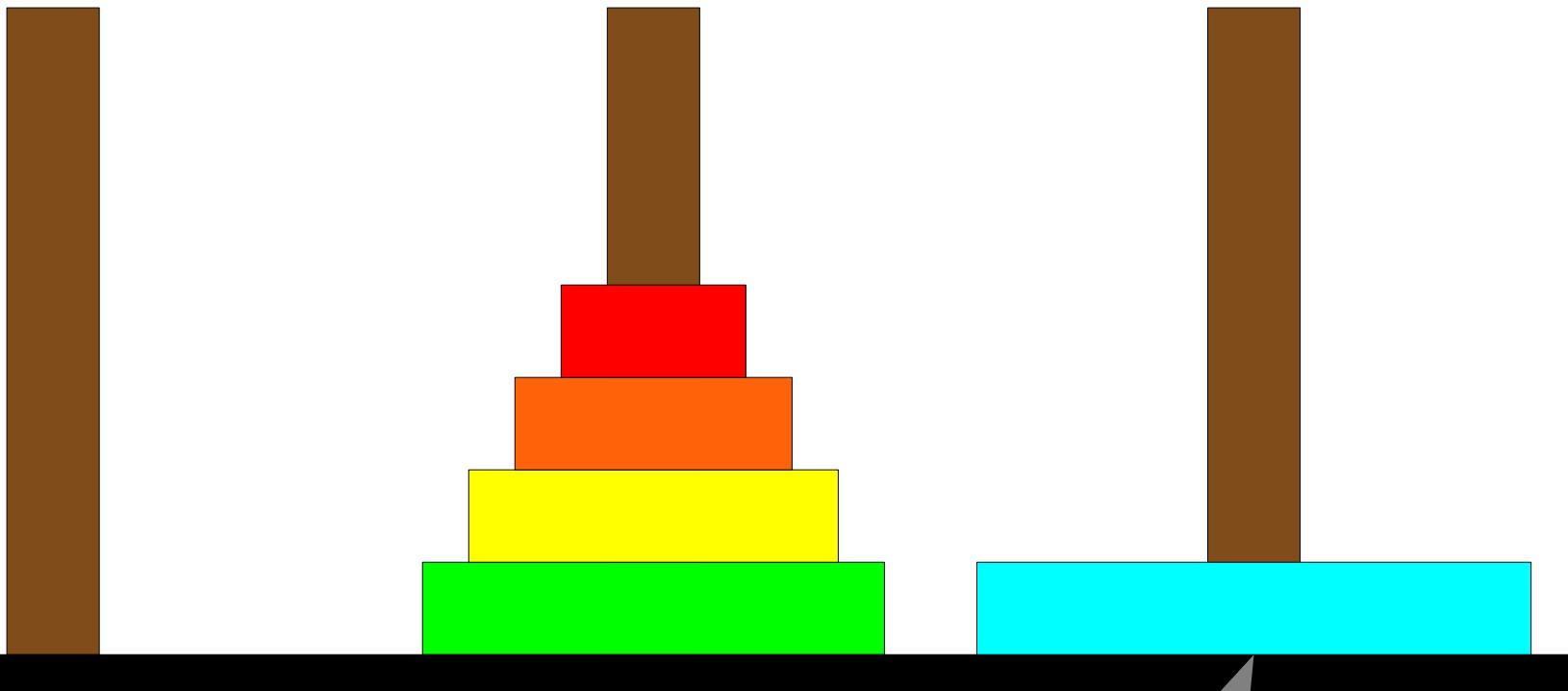
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C



This disk...

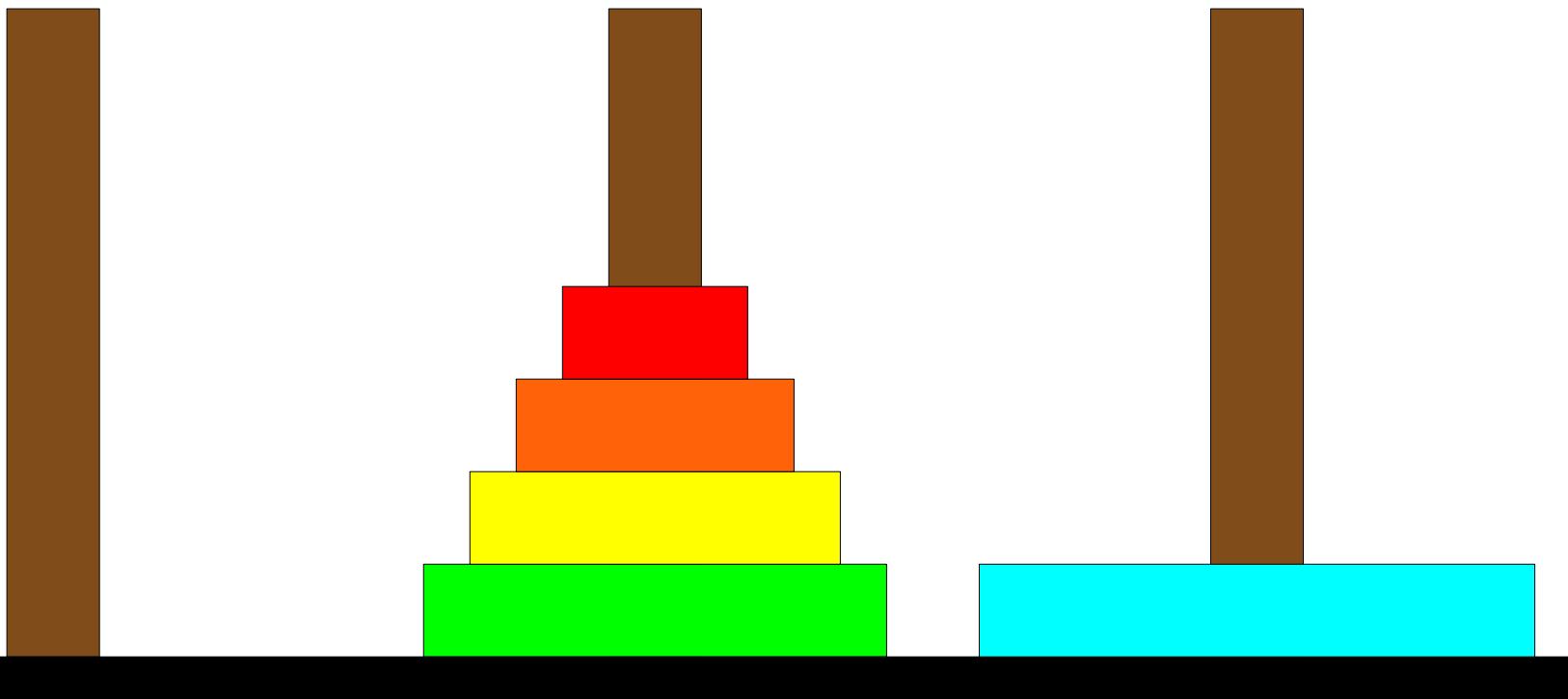
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C

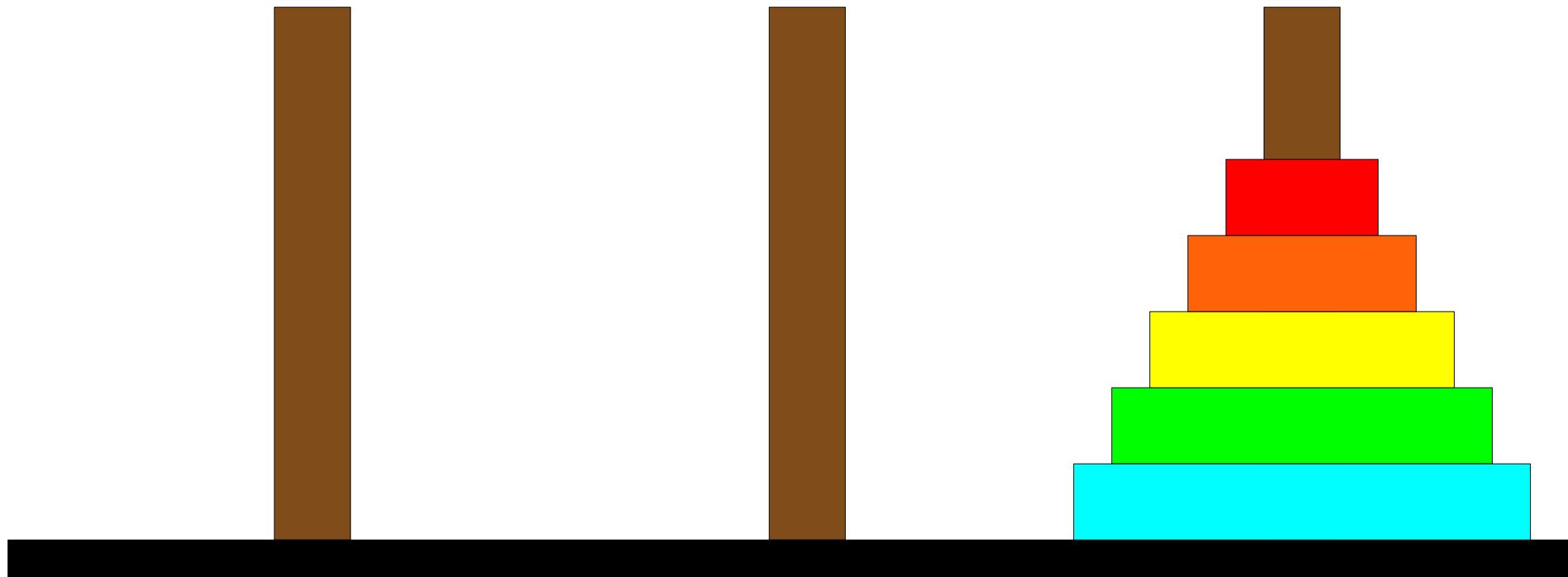


# Solving the Towers of Hanoi

A

B

C

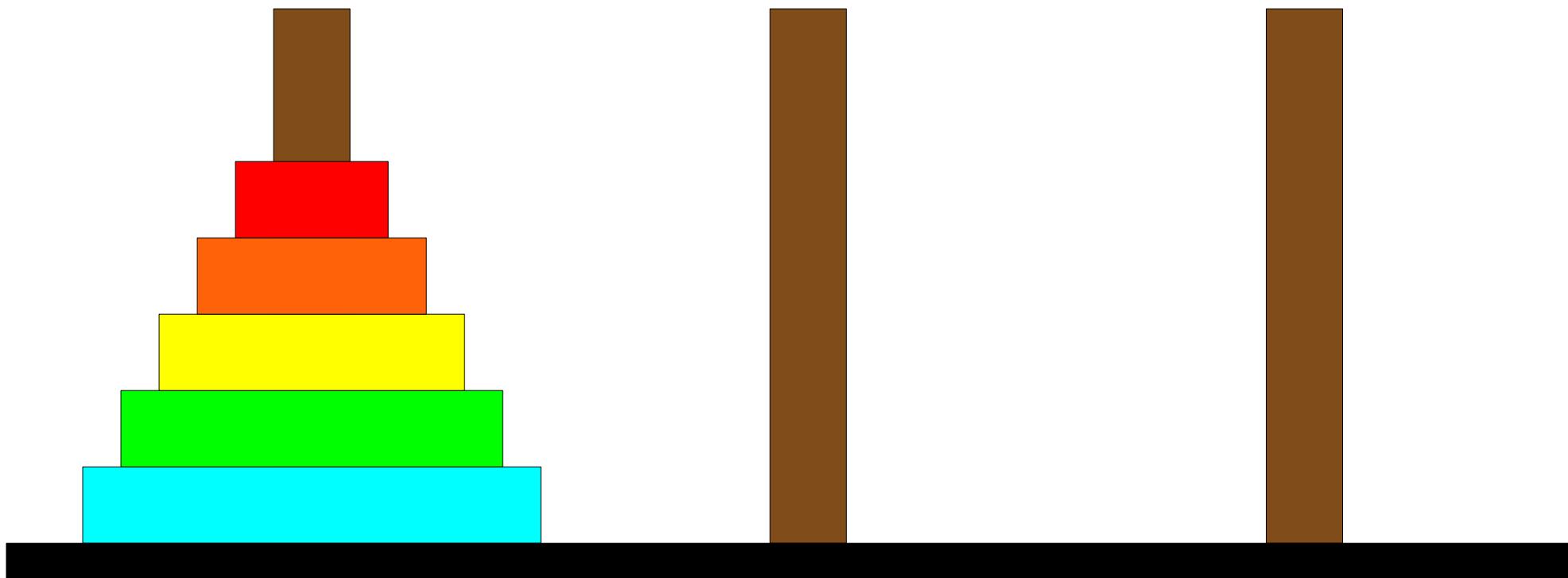


# Solving the Towers of Hanoi

A

B

C

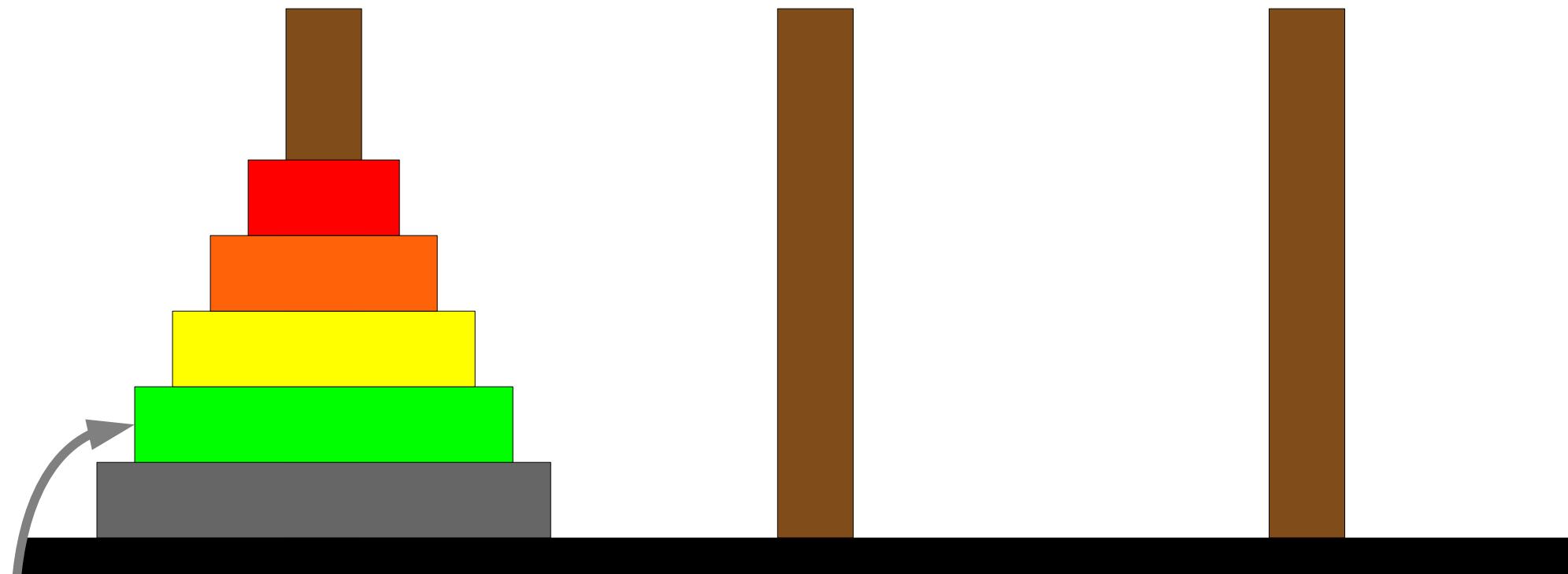


# Solving the Towers of Hanoi

A

B

C



This disk...

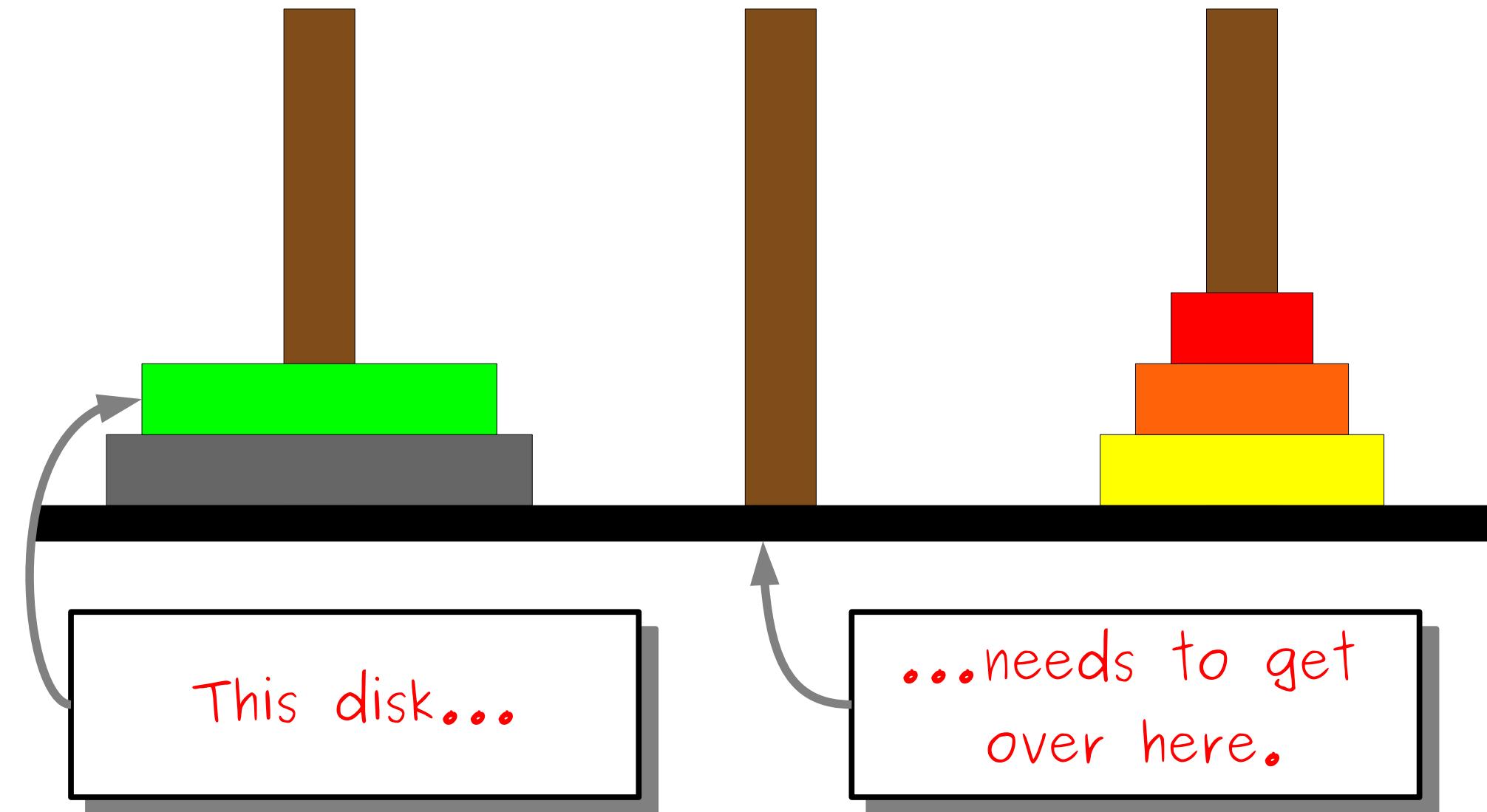
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C

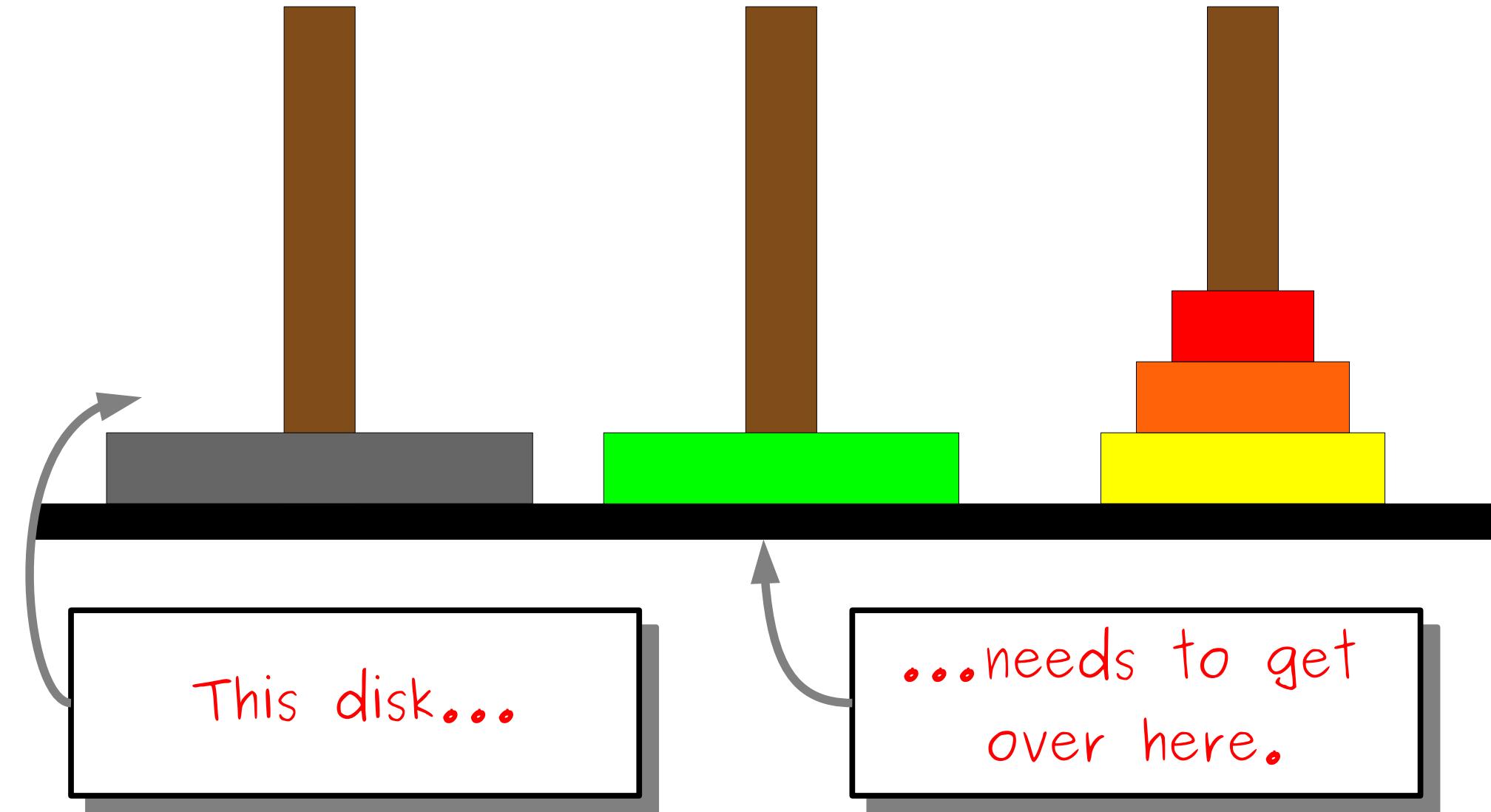


# Solving the Towers of Hanoi

A

B

C



# Solving the Towers of Hanoi

A

B

C



This disk...

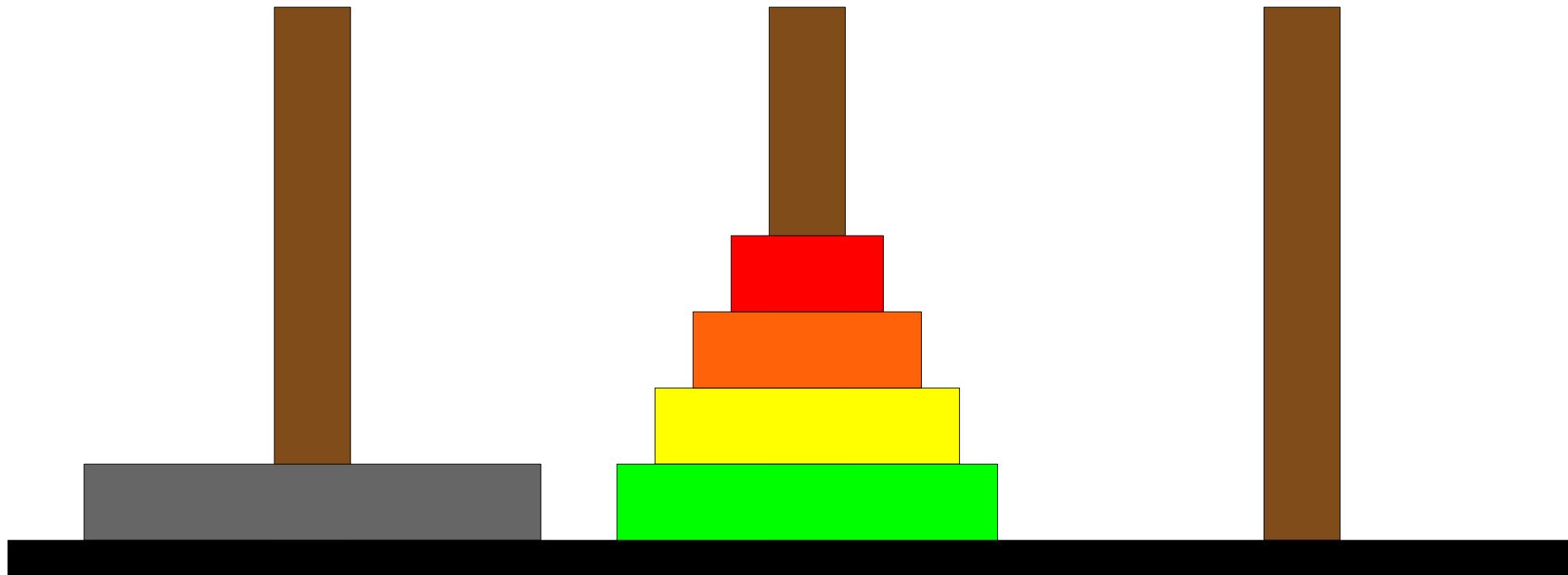
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C

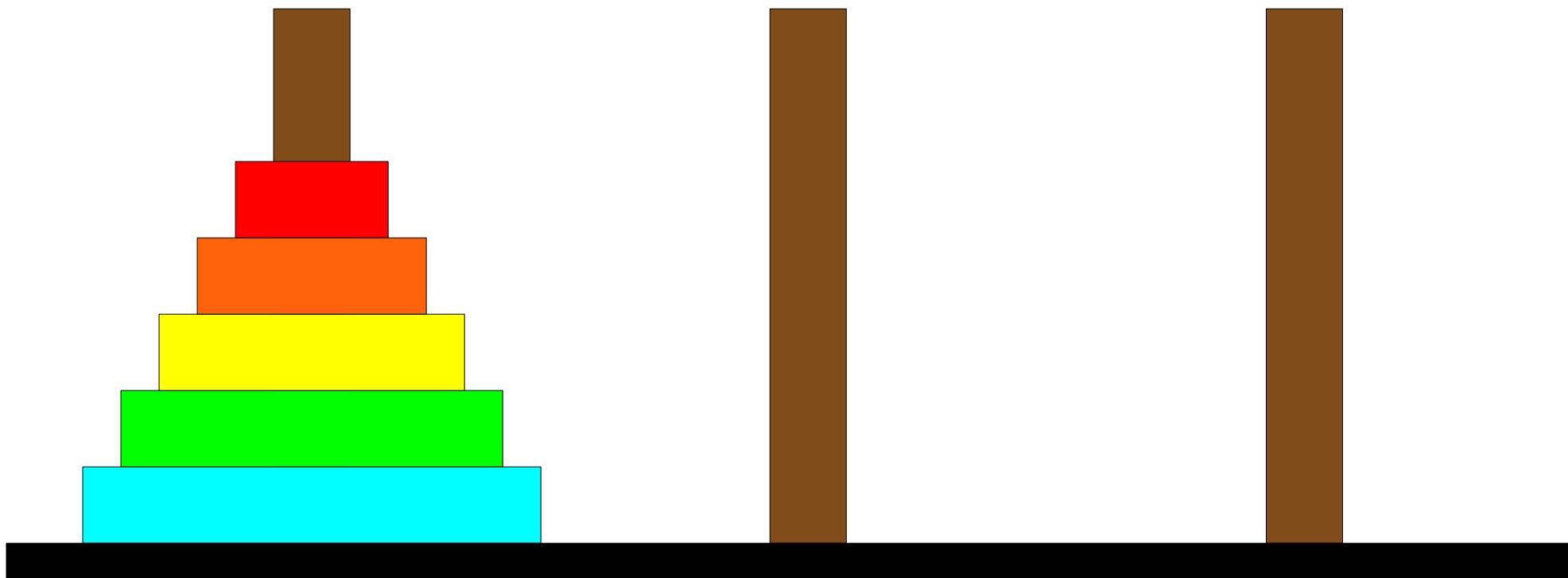


# Solving the Towers of Hanoi

A

B

C

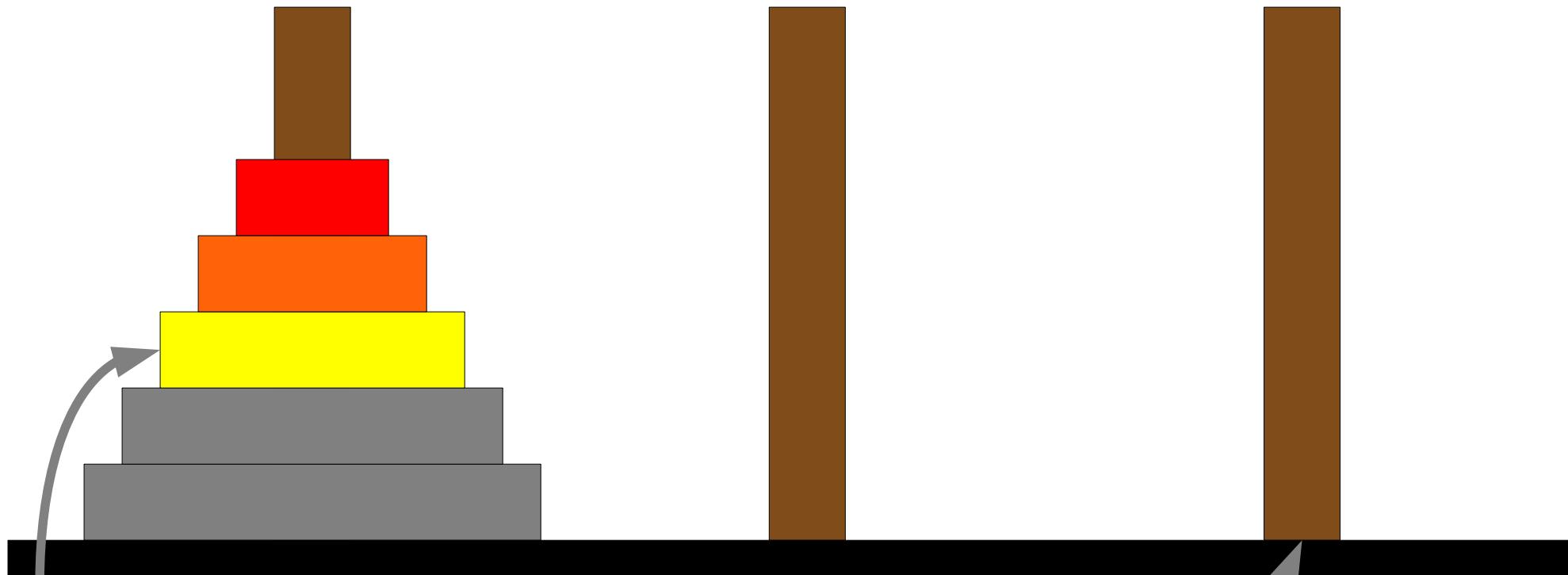


# Solving the Towers of Hanoi

A

B

C



This disk...

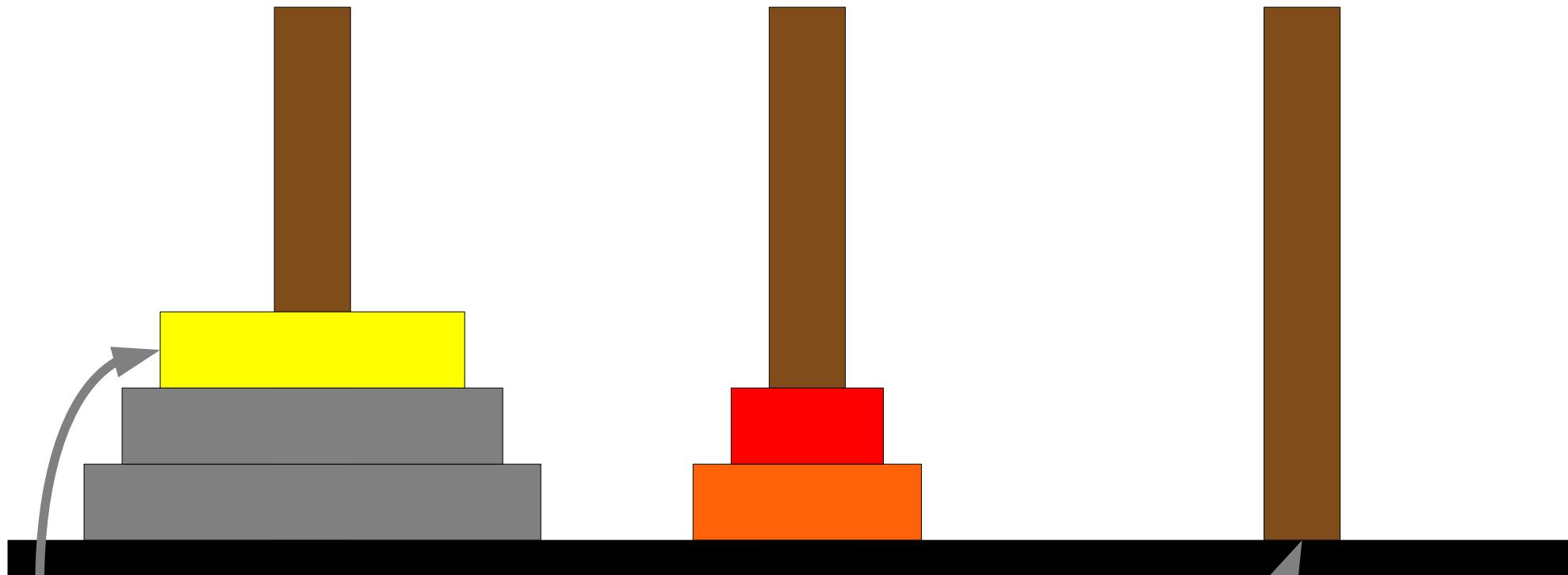
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C



This disk...

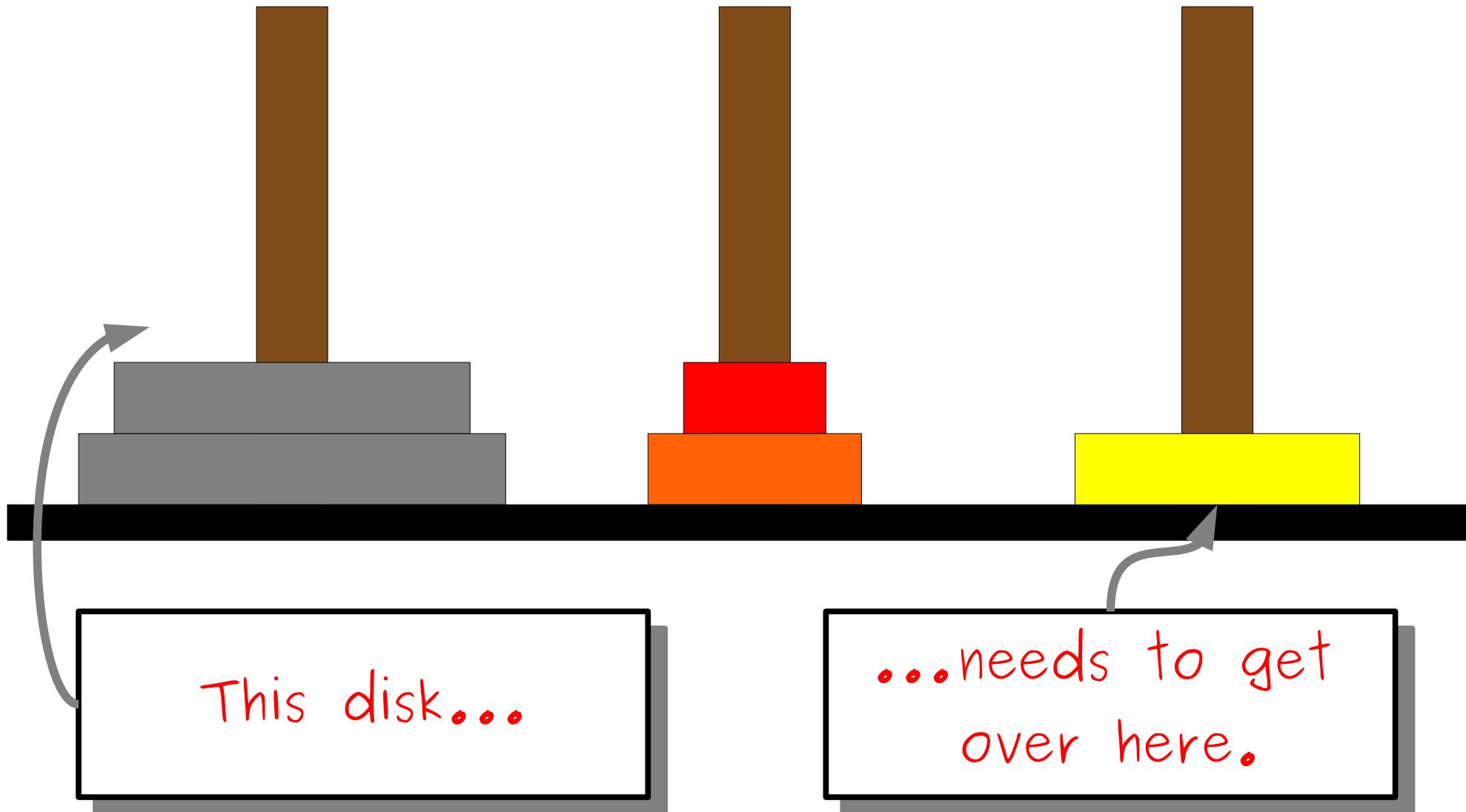
...needs to get  
over here.

# Solving the Towers of Hanoi

A

B

C

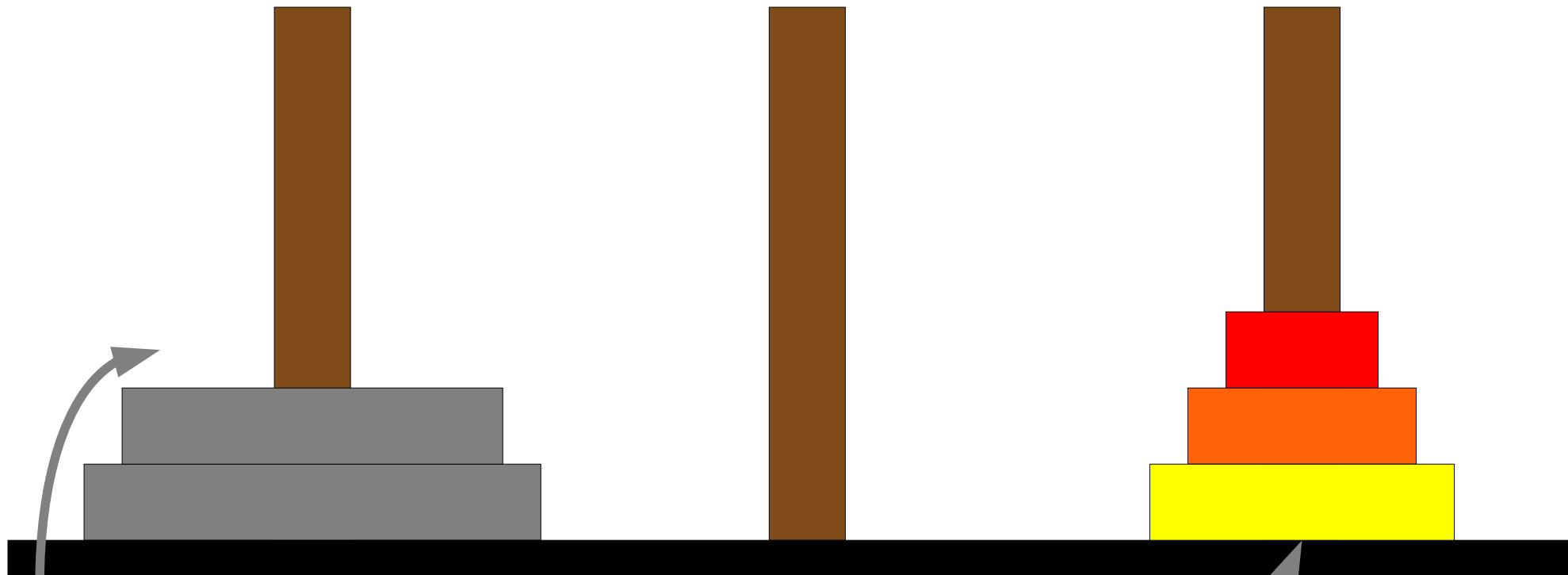


# Solving the Towers of Hanoi

A

B

C



This disk...

...needs to get  
over here.

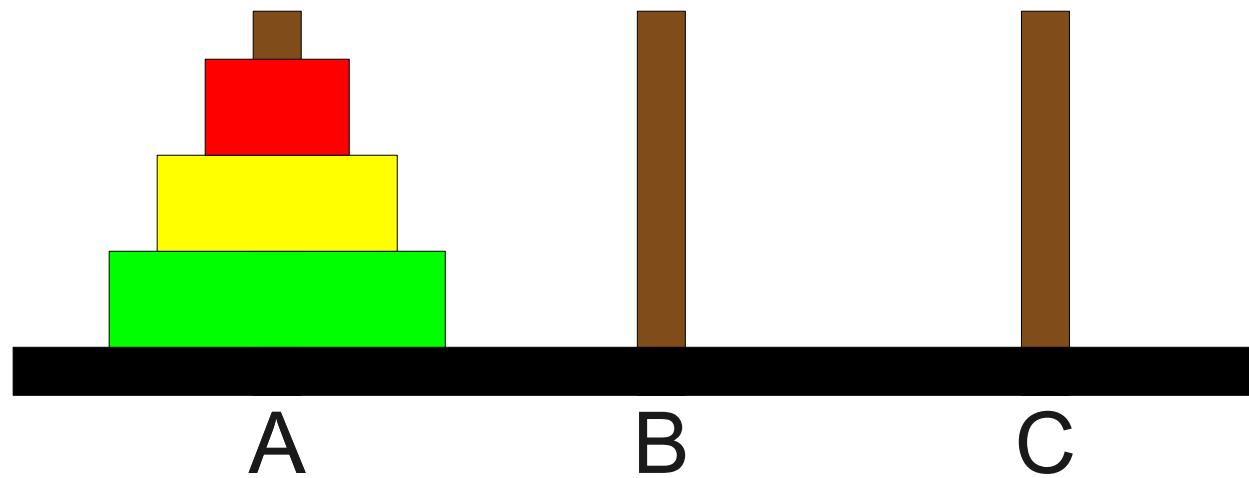
# The Base Case

- We need to find a very simple case that we can solve directly in order for the recursion to work.
- If the tower has size one, we can just move that single disk from the source to the destination.

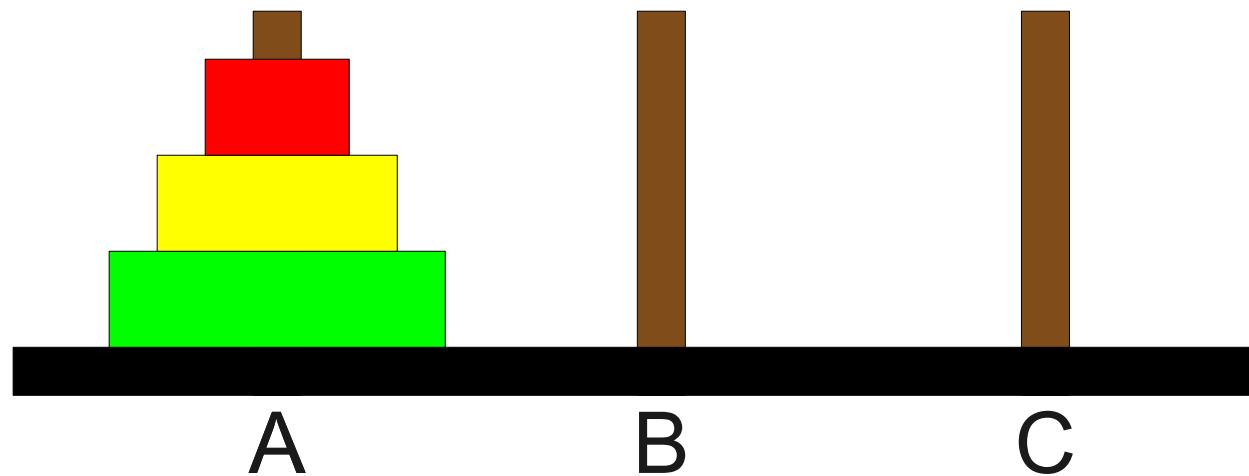
And now, the solution...

```
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

```
int main() {
    moveTower(3, 'a', 'c', 'b');
}
```

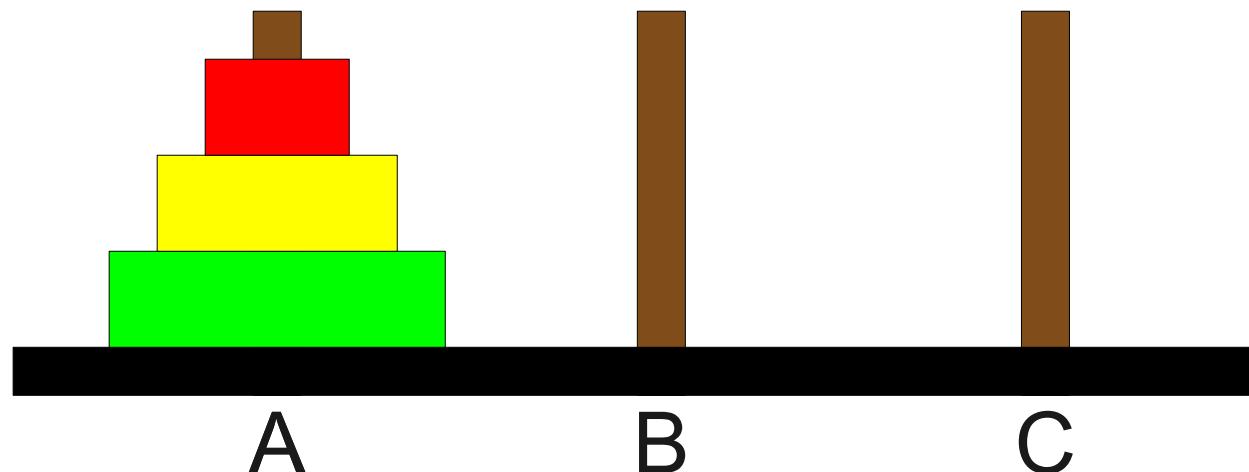


```
int main() {  
    moveTower(3, 'a', 'c', 'b');  
}
```



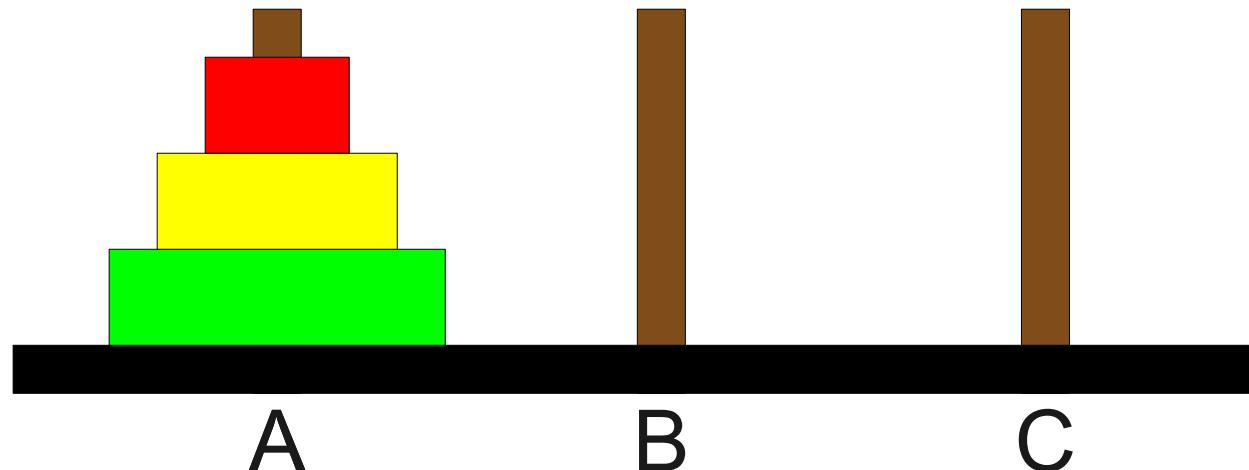
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



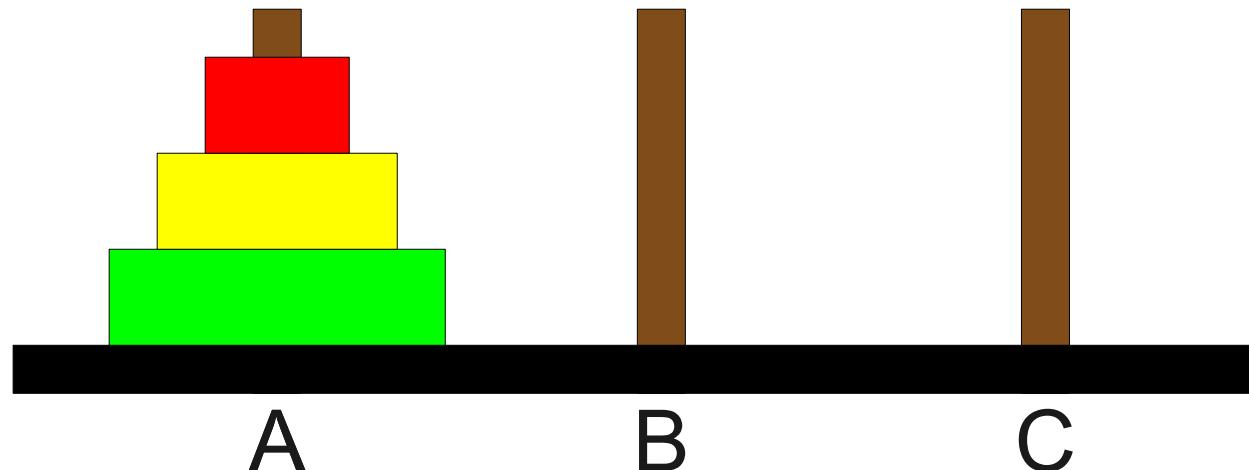
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



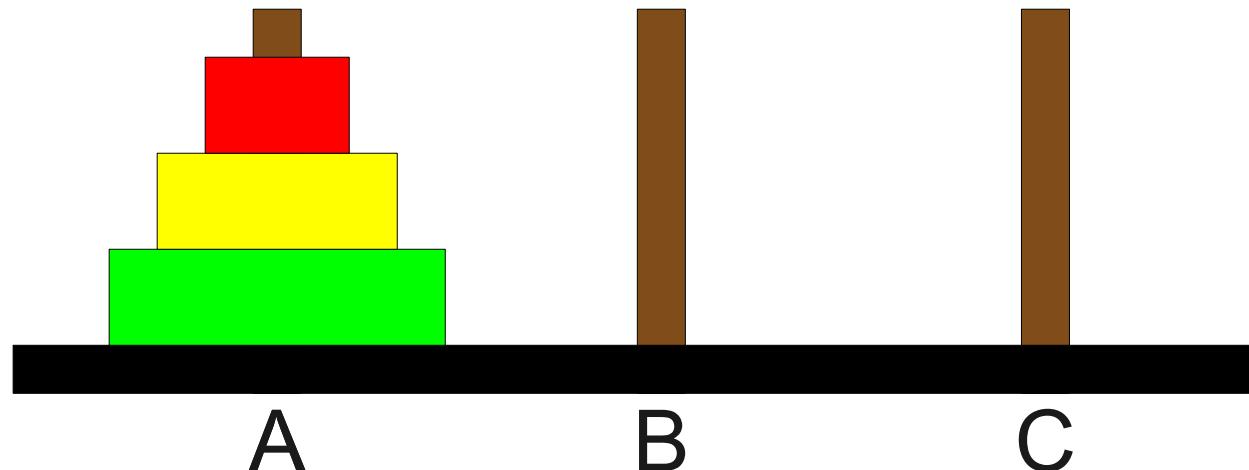
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



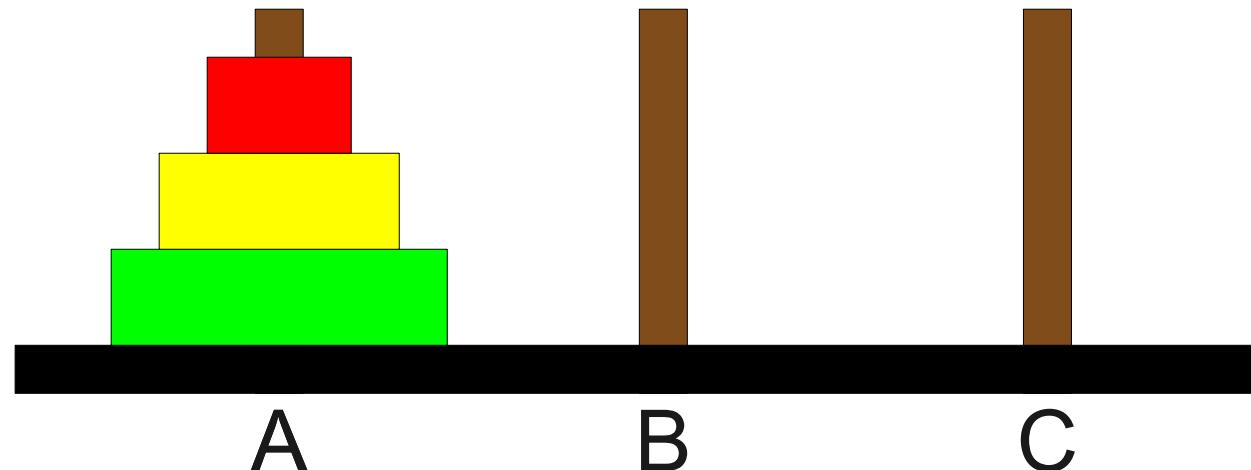
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



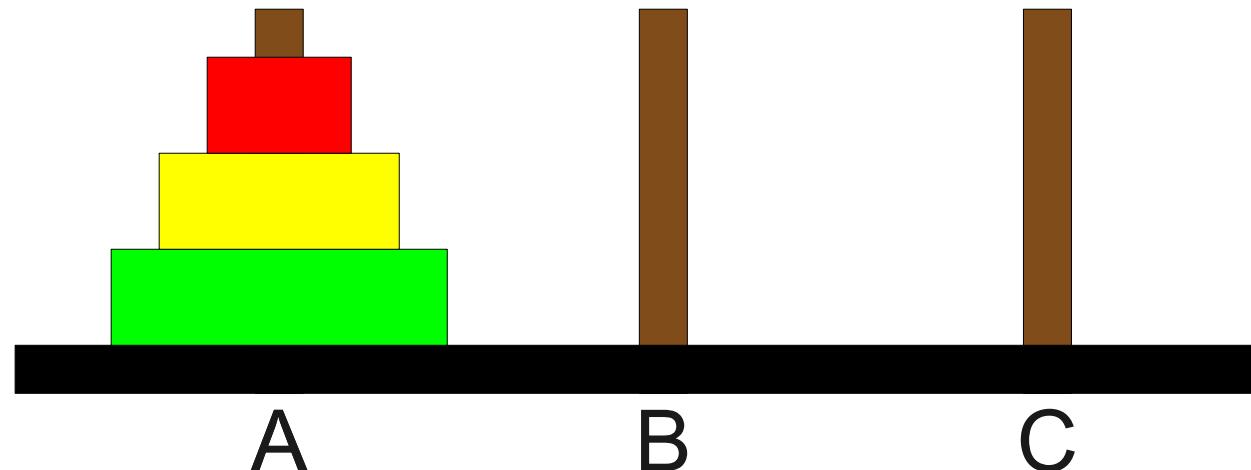
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from a    to b    temp c



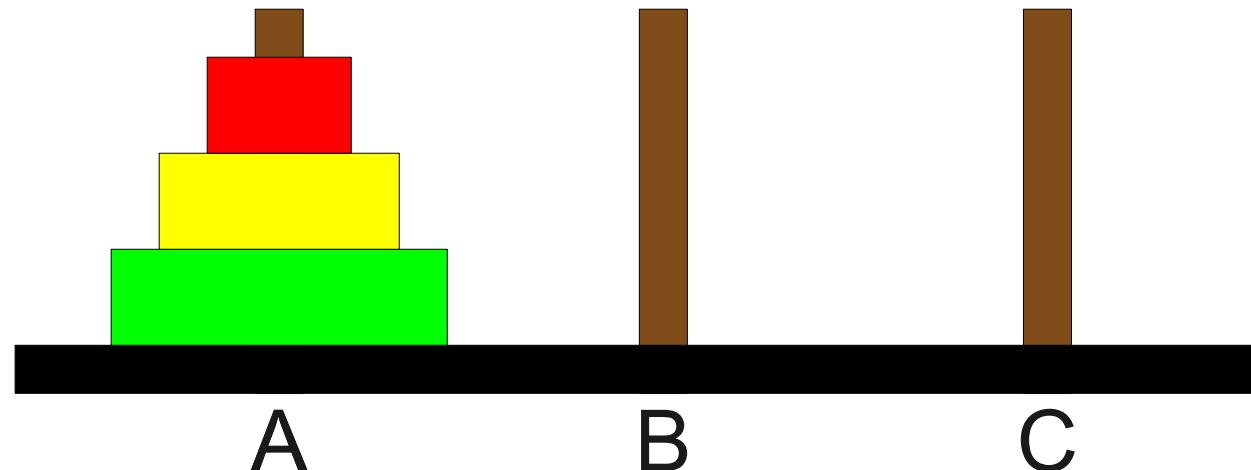
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from a    to b    temp c



```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from a    to b    temp c



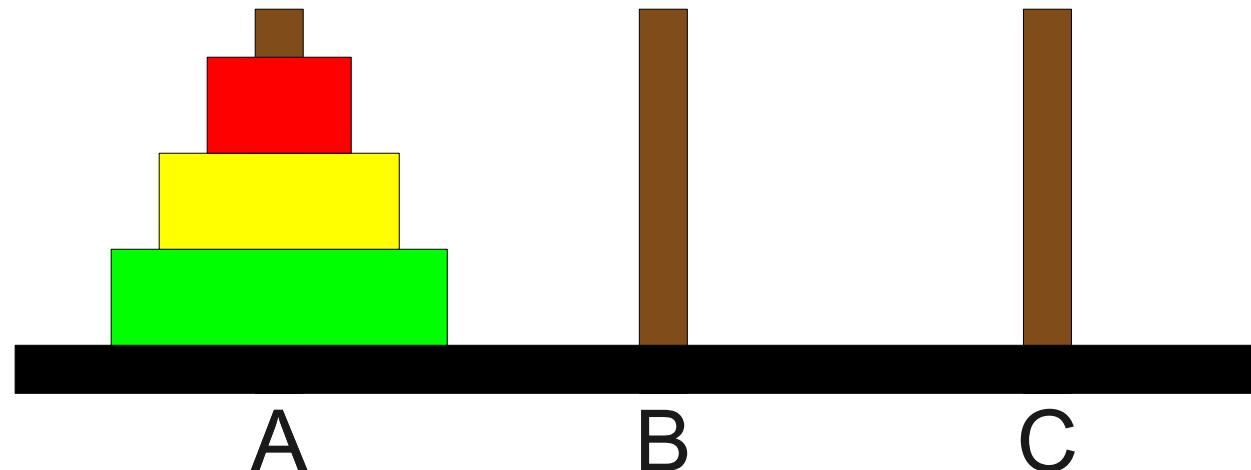
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        if (n == 1) {
            moveSingleDisk(from, to);
        } else {
            moveTower(n - 1, from, temp, to);
            moveSingleDisk(from, to);
            moveTower(n - 1, temp, to, from);
        }
    }
}

```

n 2    from a    to b    temp c



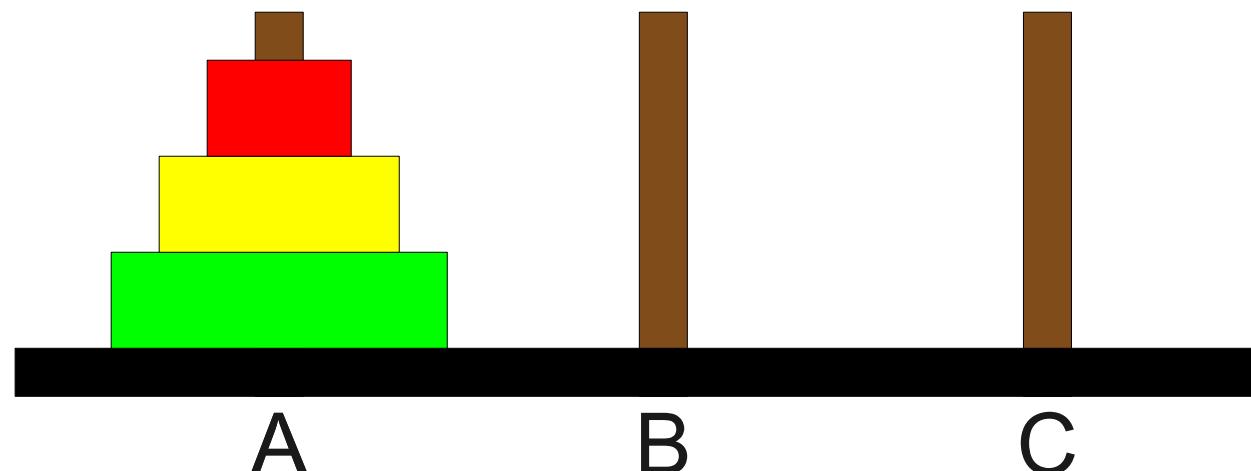
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

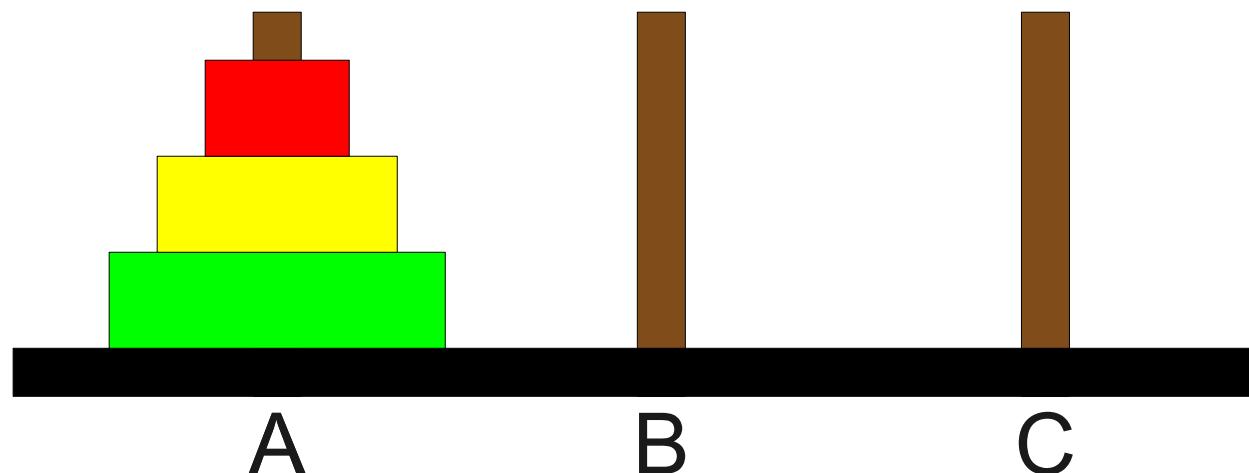
```

n      1      from      a      to      c      temp      b



```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n    1        from    a        to    c        temp    b



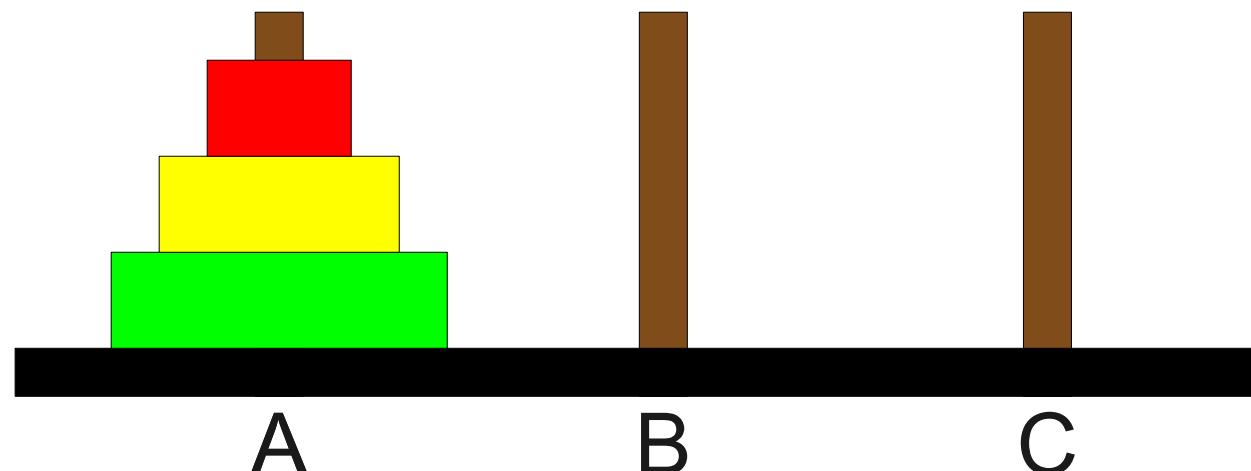
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

```

n      1      from      a      to      c      temp      b



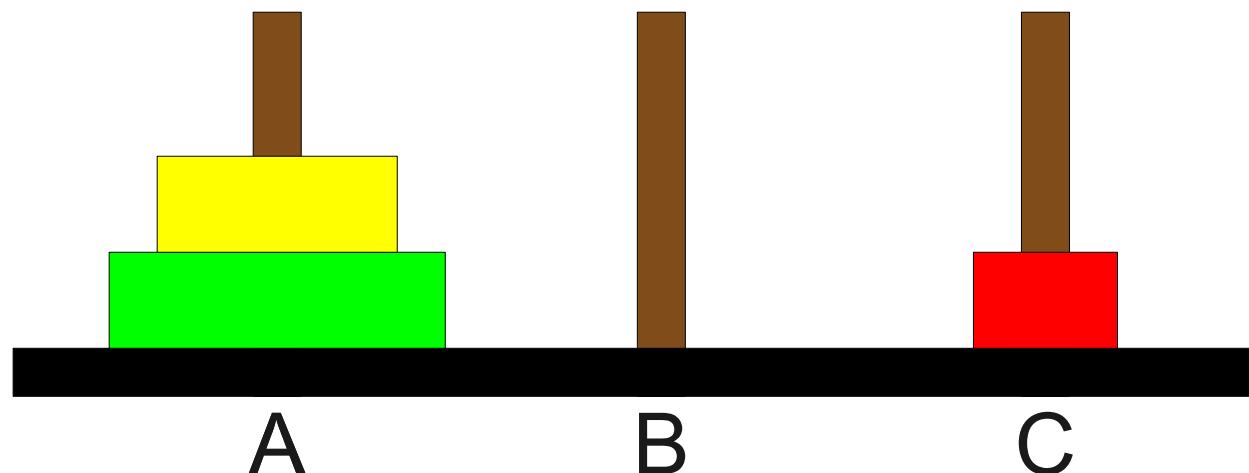
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

```

n    1      from    a      to    c      temp    b



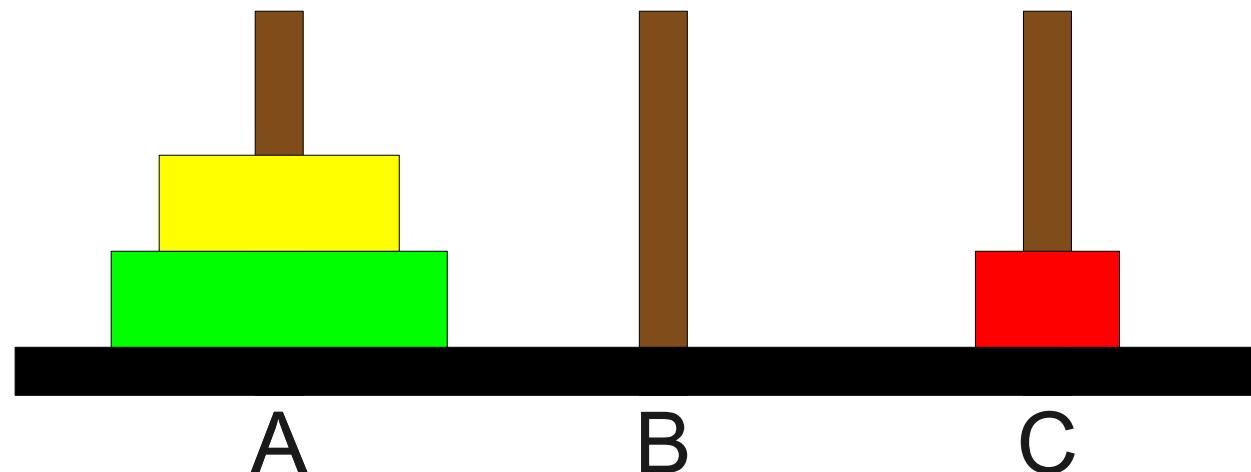
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        if (n == 1) {
            moveSingleDisk(from, to);
        } else {
            moveTower(n - 1, from, temp, to);
            moveSingleDisk(from, to);
            moveTower(n - 1, temp, to, from);
        }
    }
}

```

n 2    from a    to b    temp c



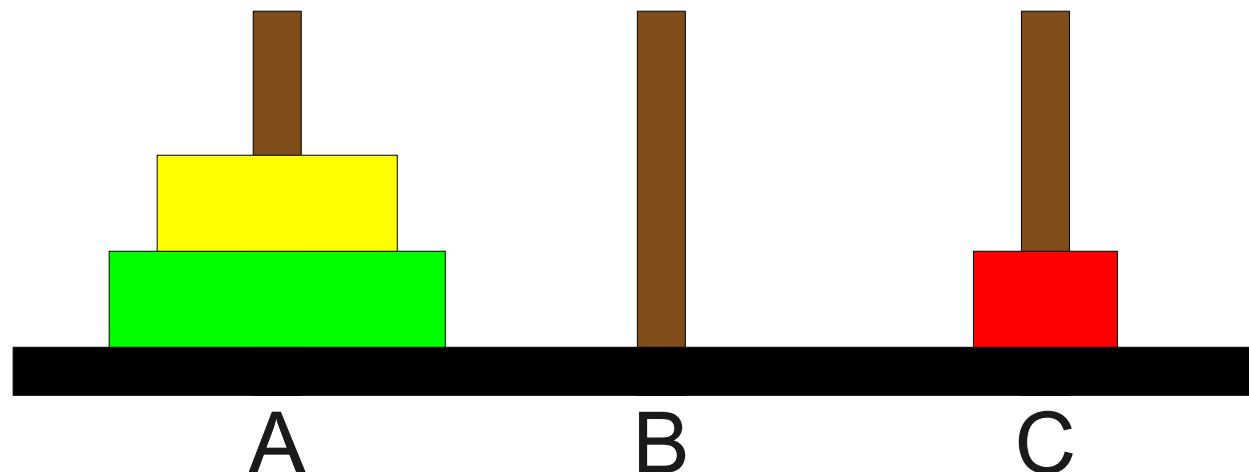
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        if (n == 1) {
            moveSingleDisk(from, to);
        } else {
            moveTower(n - 1, from, temp, to);
moveSingleDisk(from, to);
            moveTower(n - 1, temp, to, from);
        }
    }
}

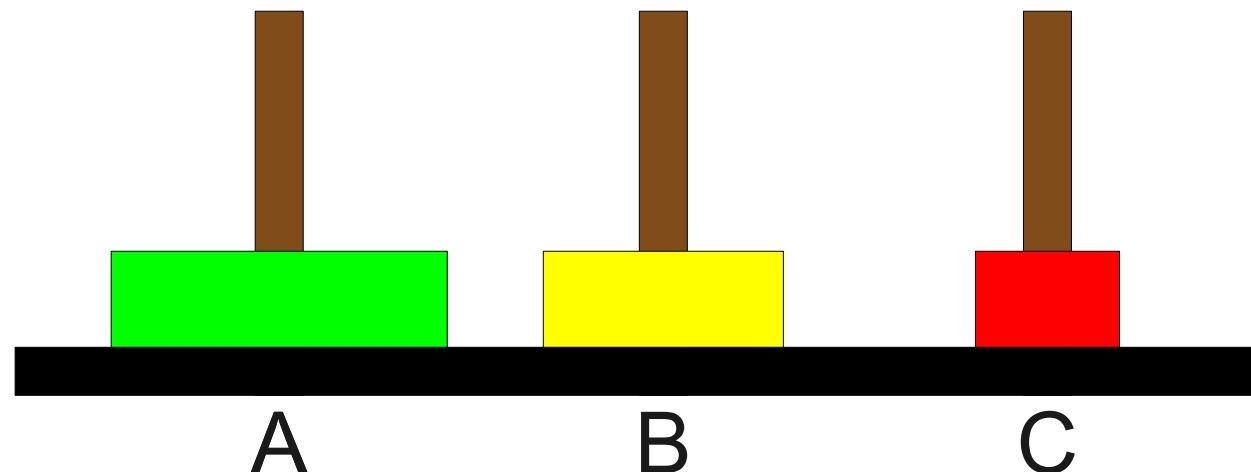
```

n 2    from a    to b    temp c



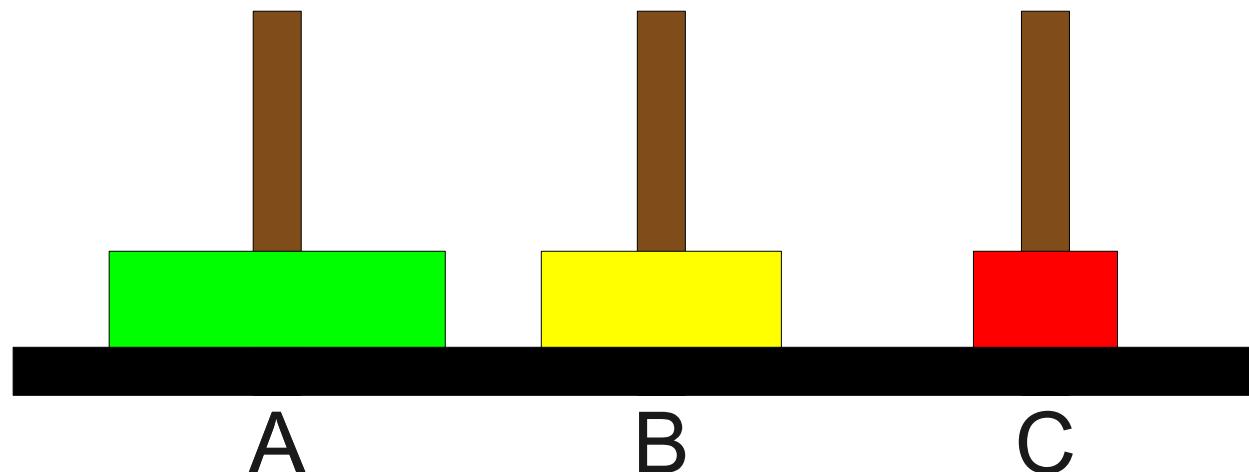
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from a    to b    temp c



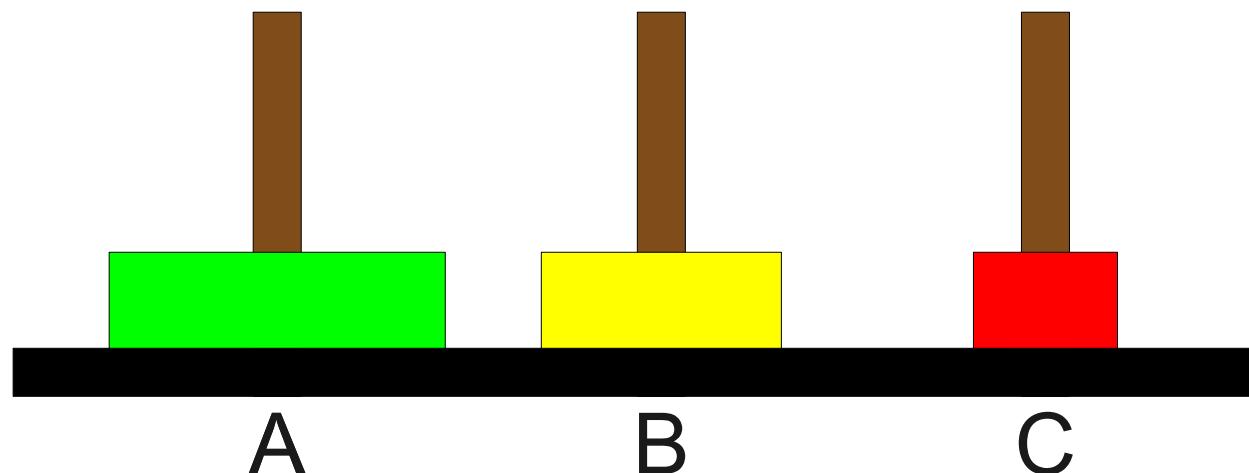
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from a    to b    temp c



```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n      1      from      C      to      b      temp      a



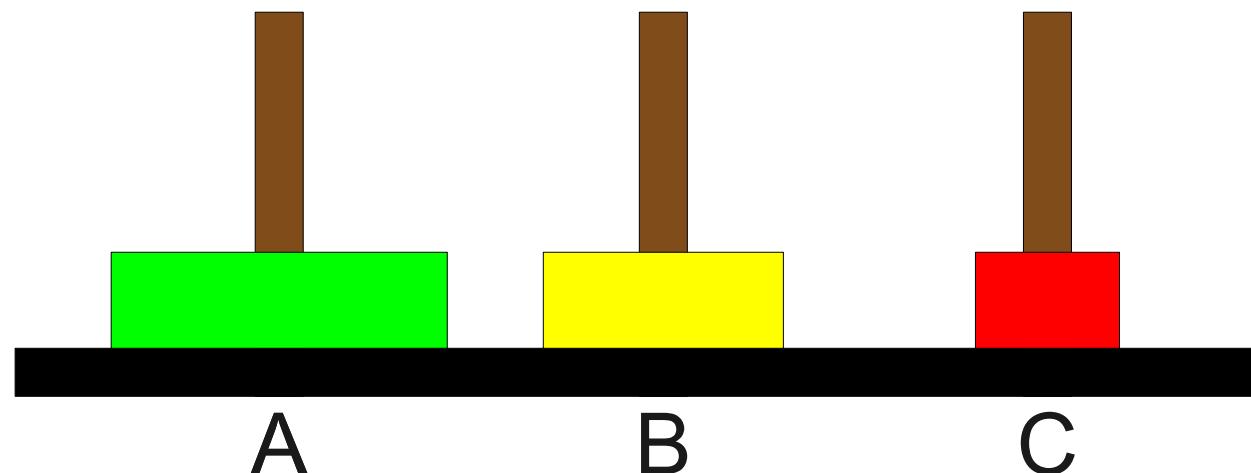
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

```

n      1      from      C      to      b      temp      a



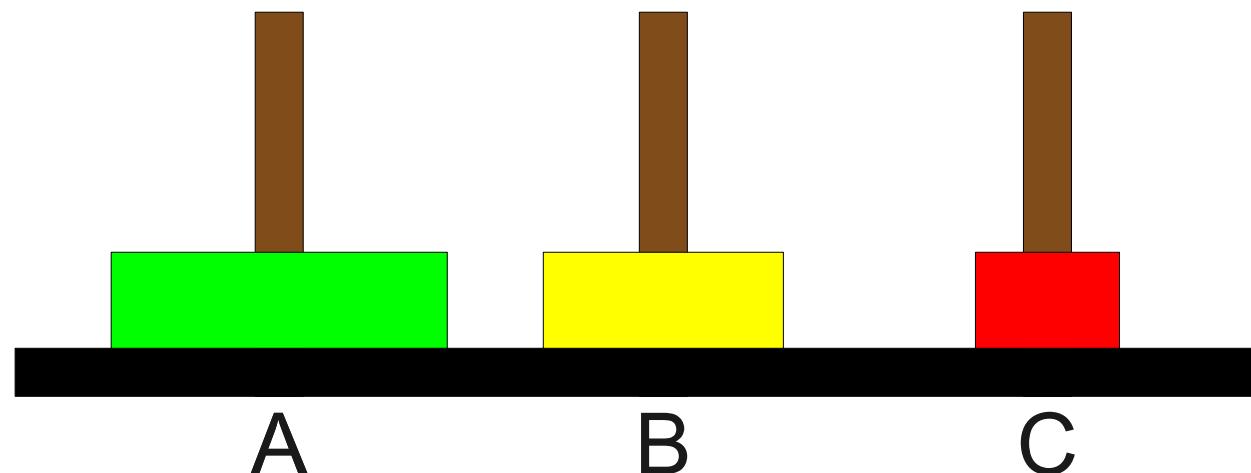
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

```

n 1      from C      to b      temp a



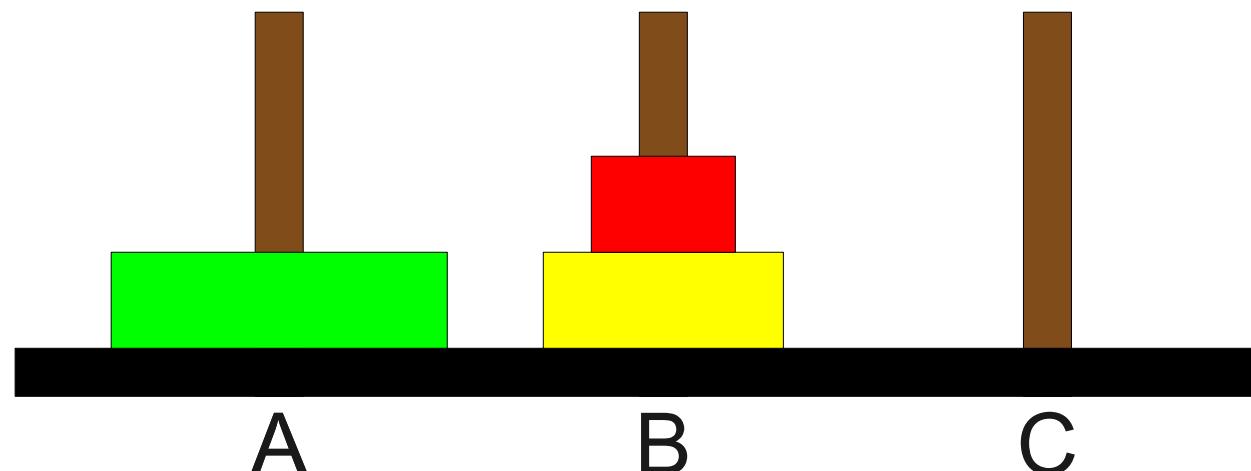
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

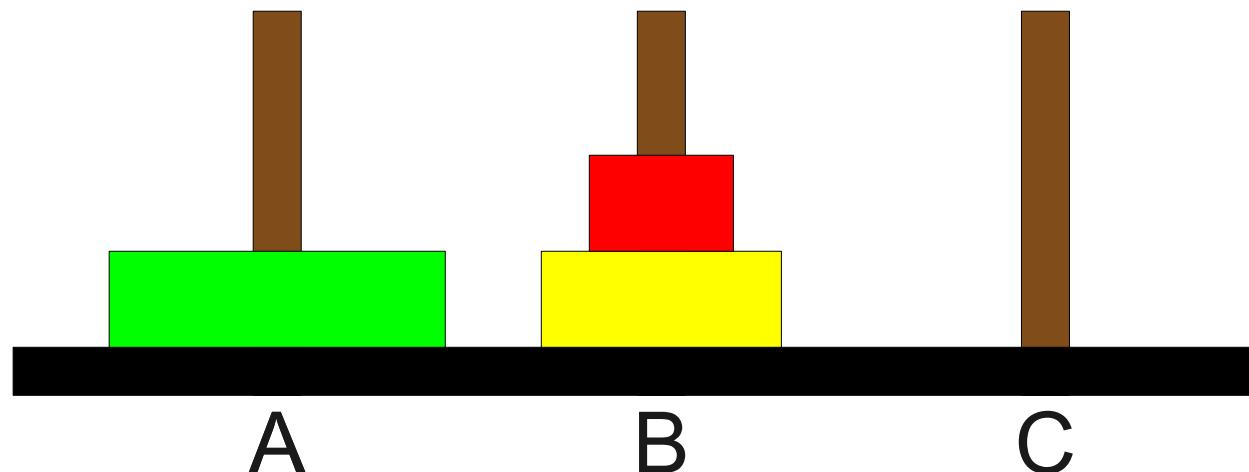
```

n      1      from      C      to      b      temp      a



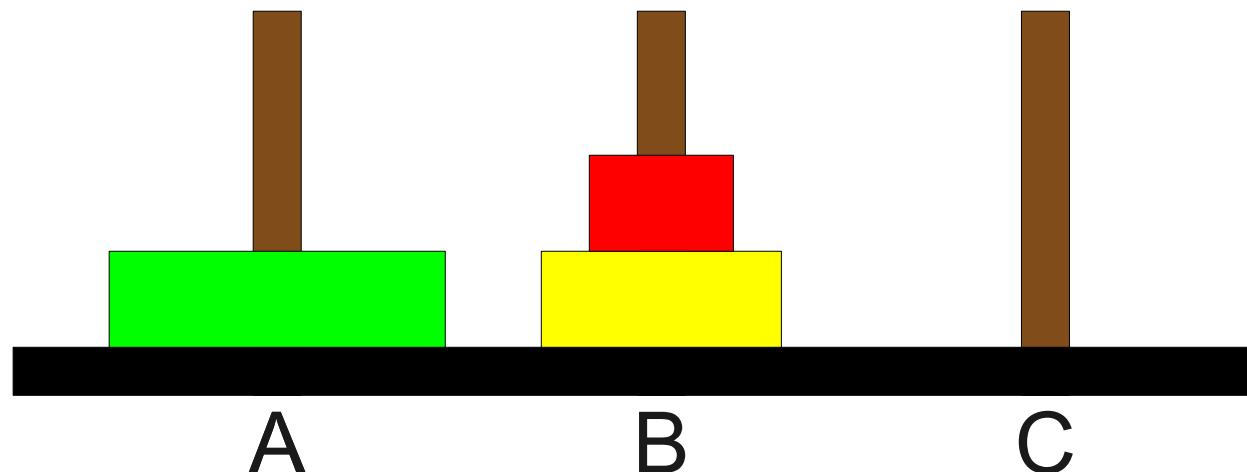
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from a    to b    temp c



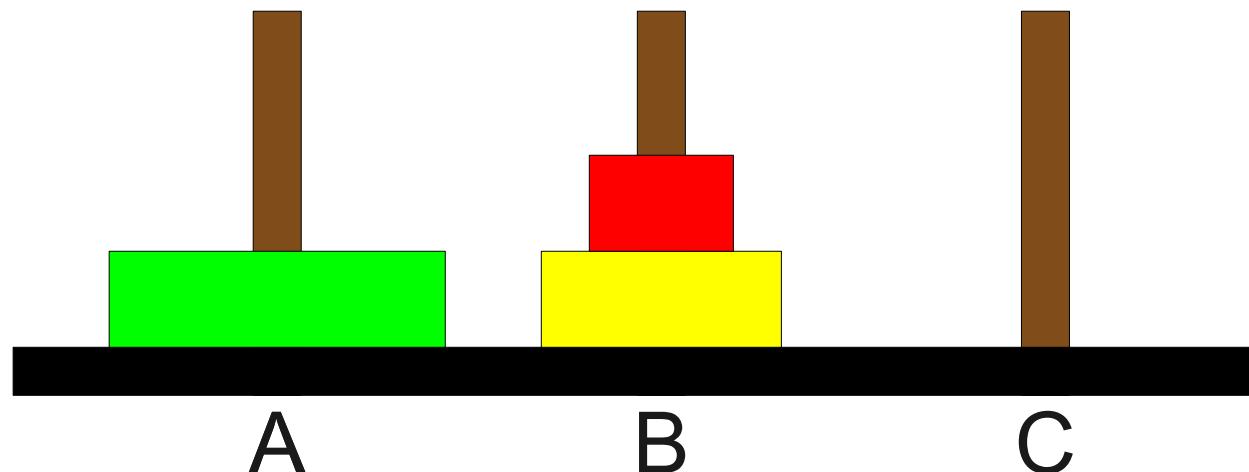
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



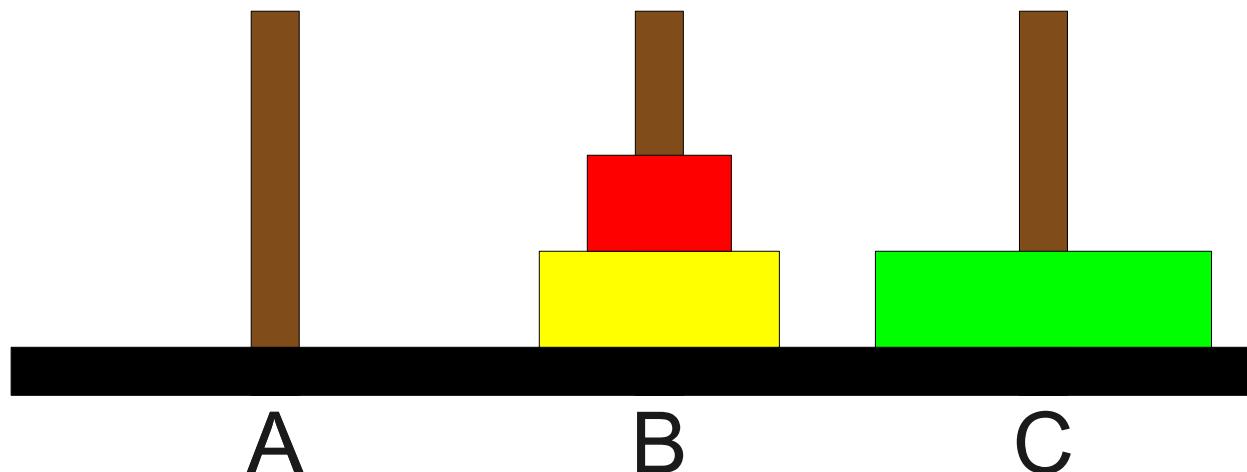
```
int main() {  
}  
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



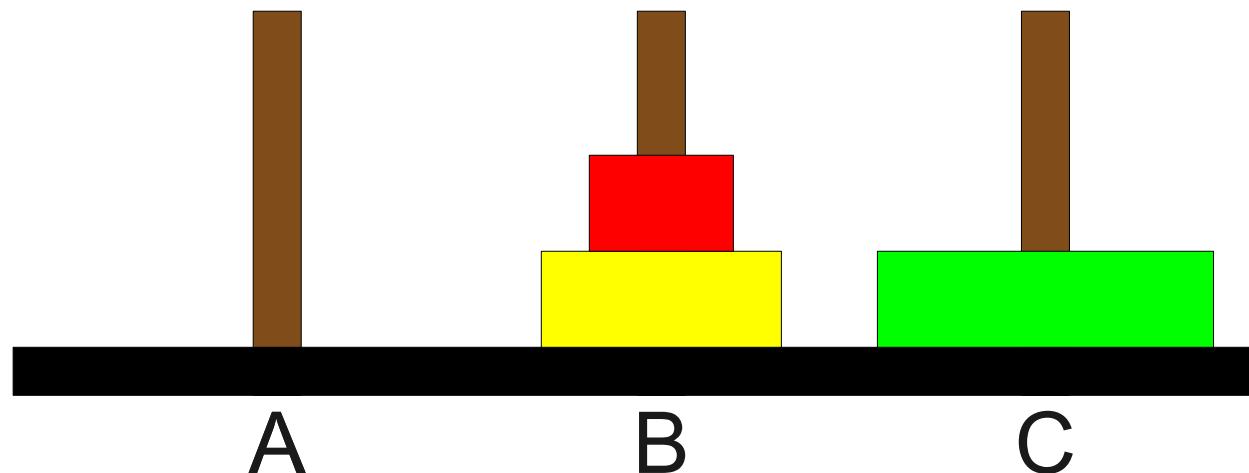
```
int main() {  
}  
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



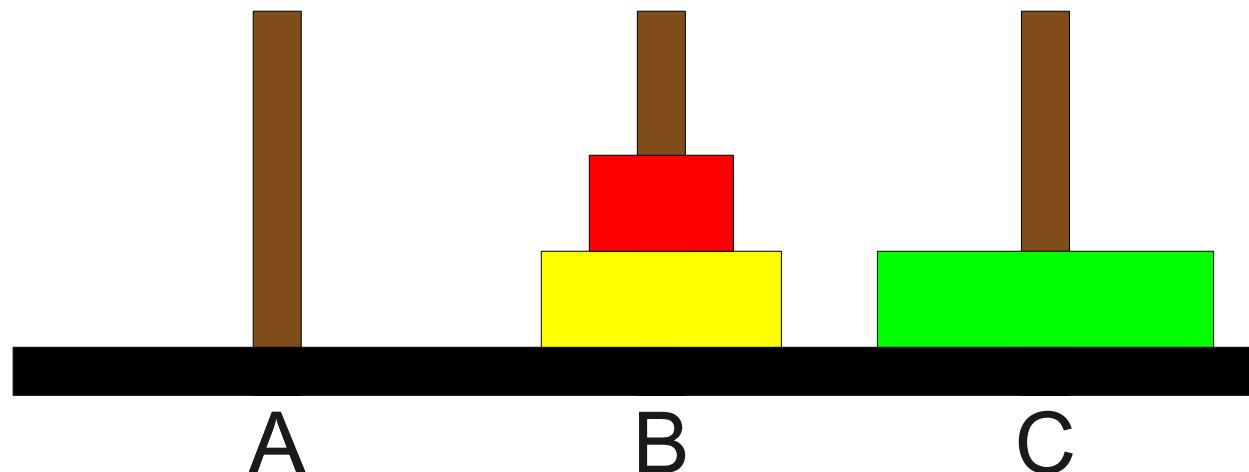
```
int main() {  
}  
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to c    temp b



```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a



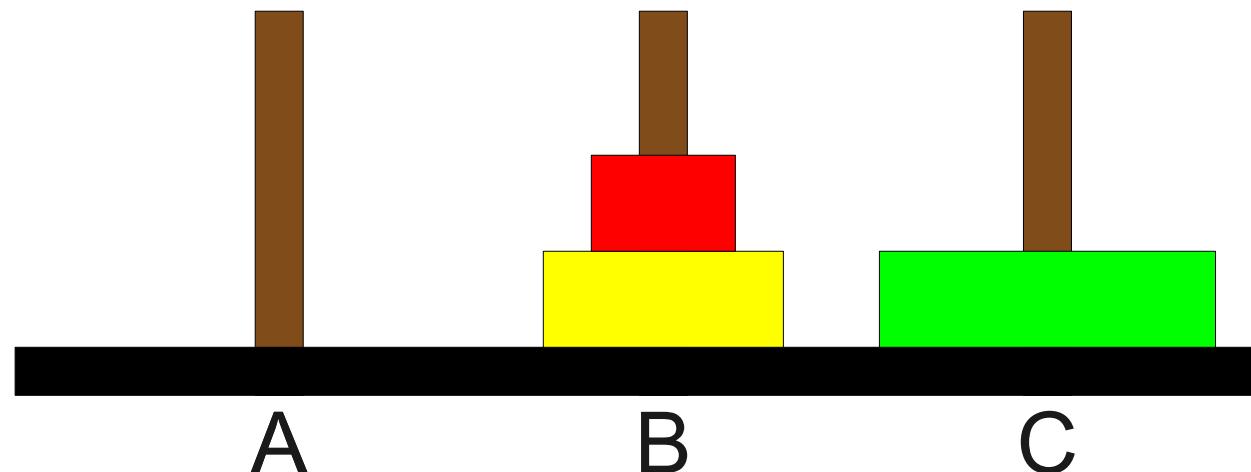
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        if (n == 1) {
            moveSingleDisk(from, to);
        } else {
            moveTower(n - 1, from, temp, to);
            moveSingleDisk(from, to);
            moveTower(n - 1, temp, to, from);
        }
    }
}

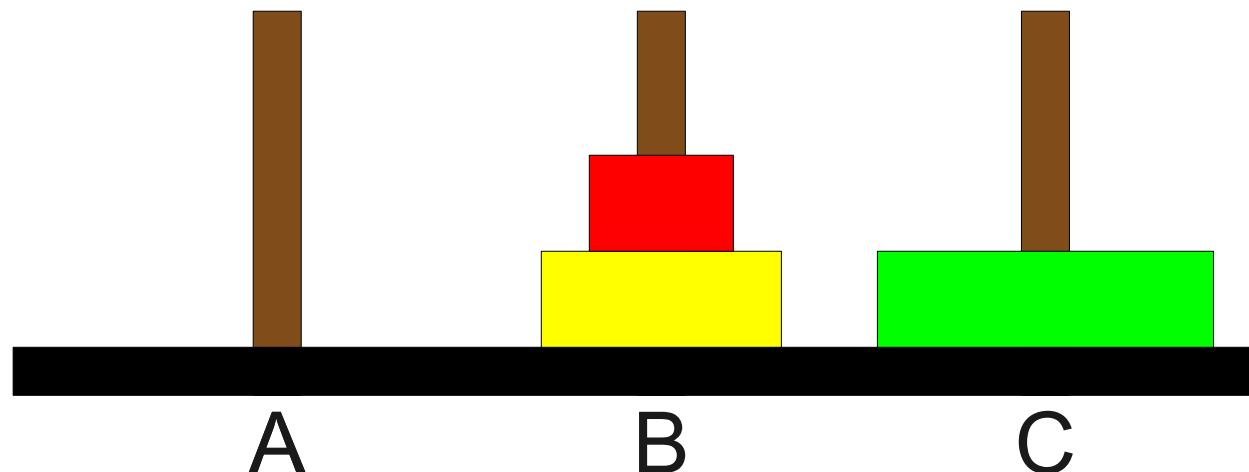
```

n 2    from b    to c    temp a



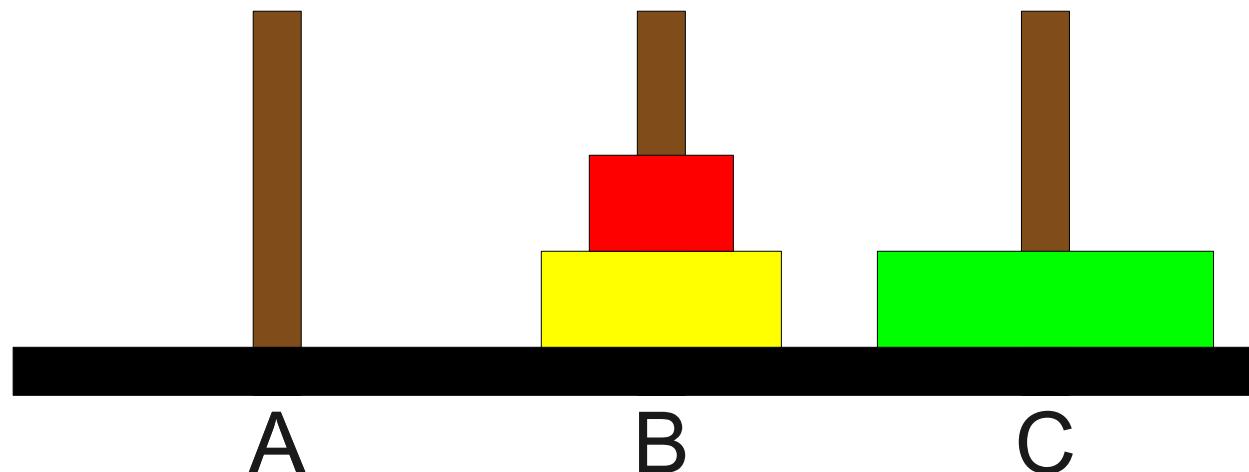
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a



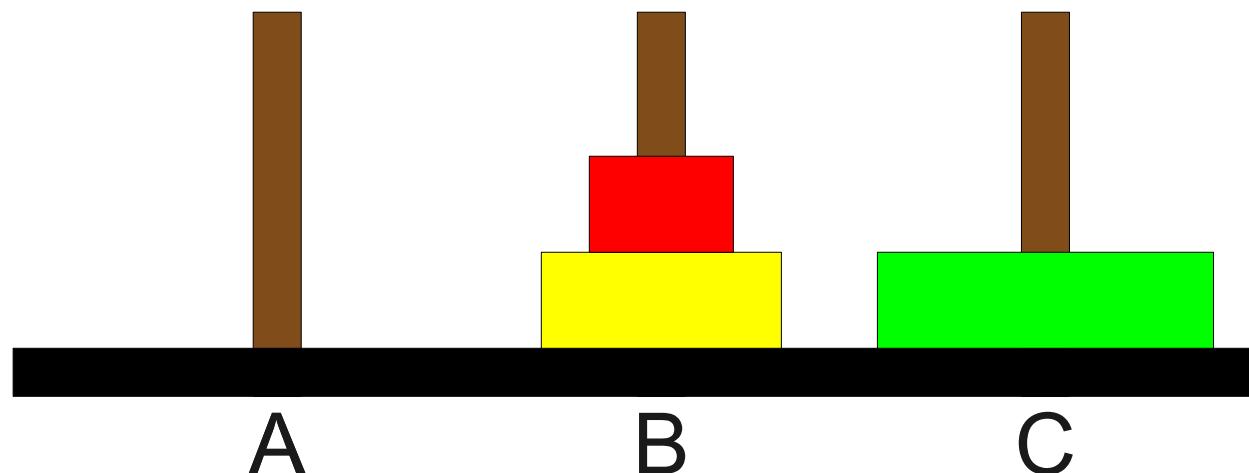
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a



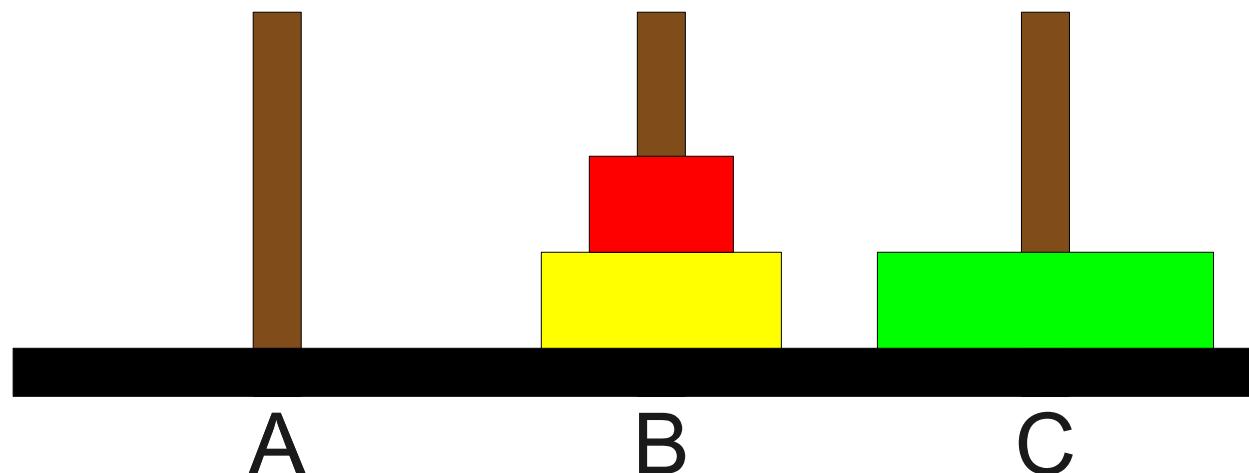
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n      1      from      b      to      a      temp      c



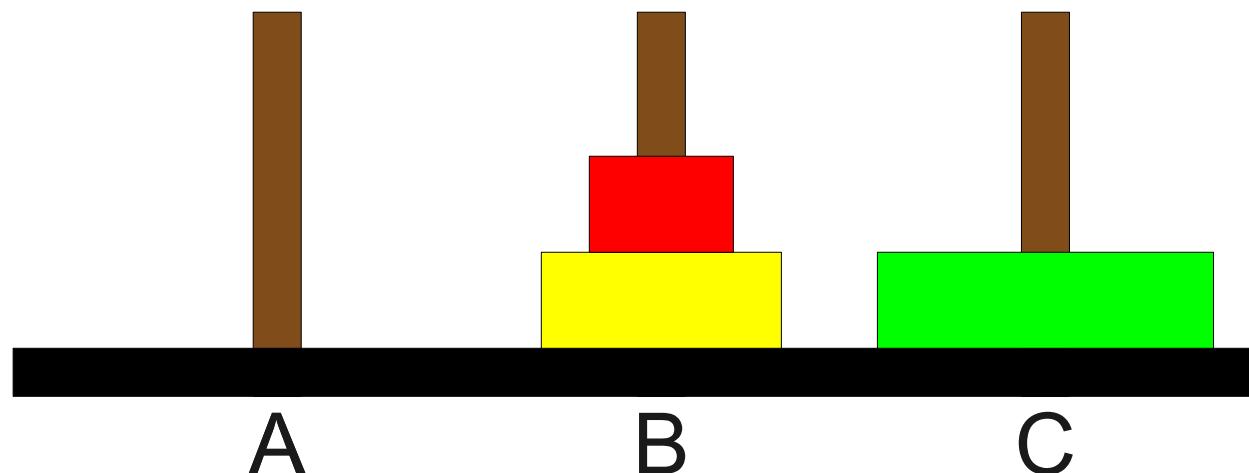
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n      1      from      b      to      a      temp      c



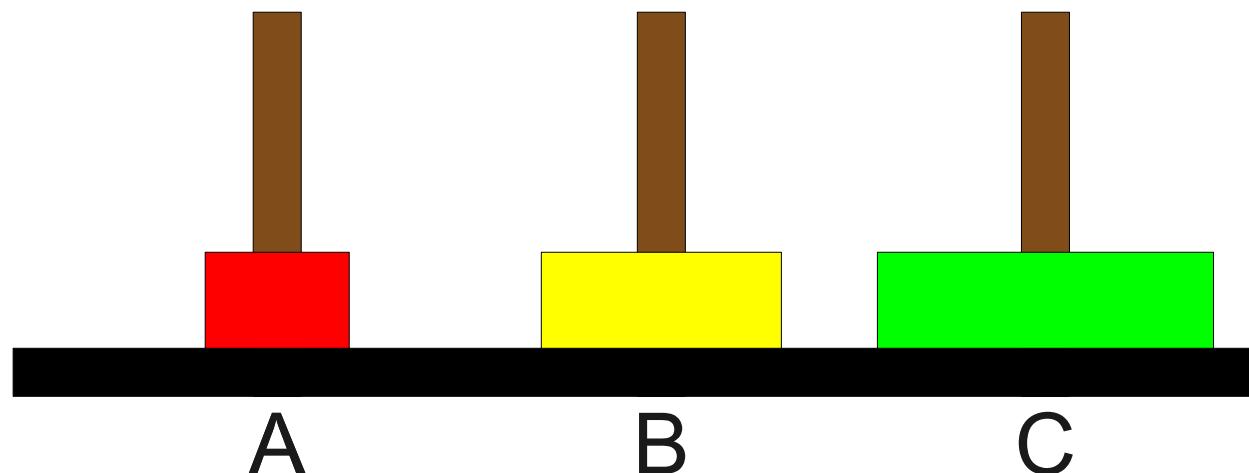
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n      1      from      b      to      a      temp      c



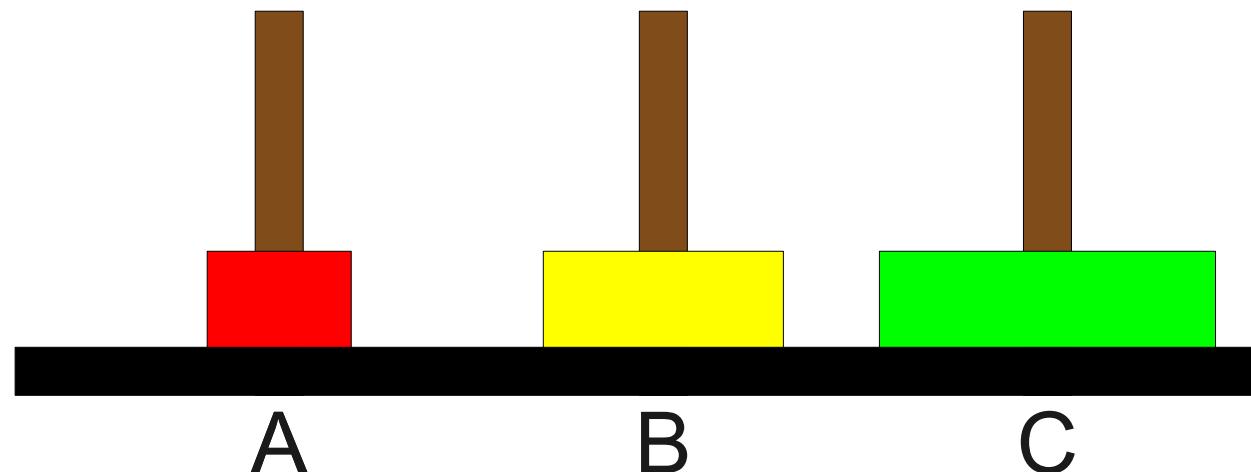
```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n      1      from      b      to      a      temp      c



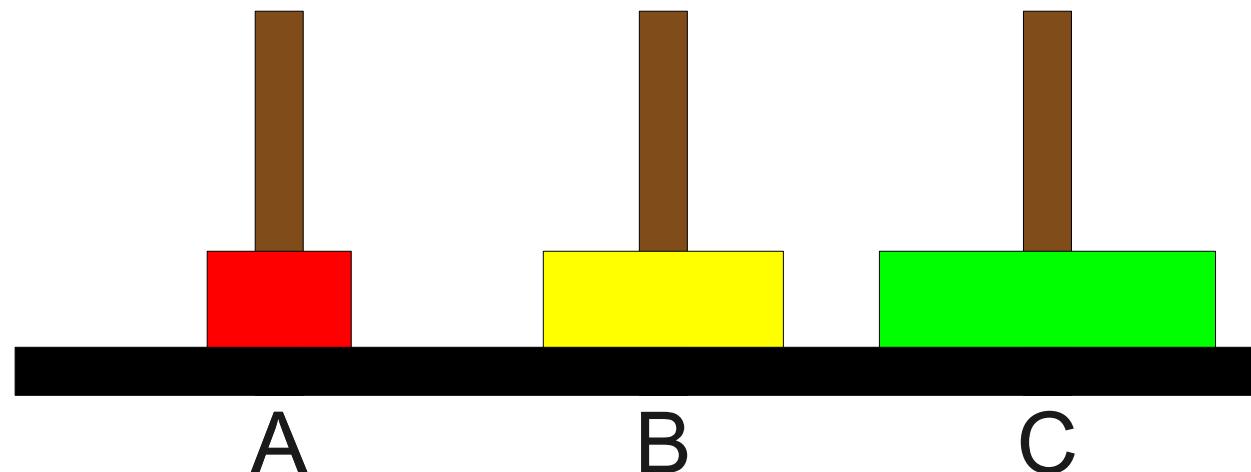
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a



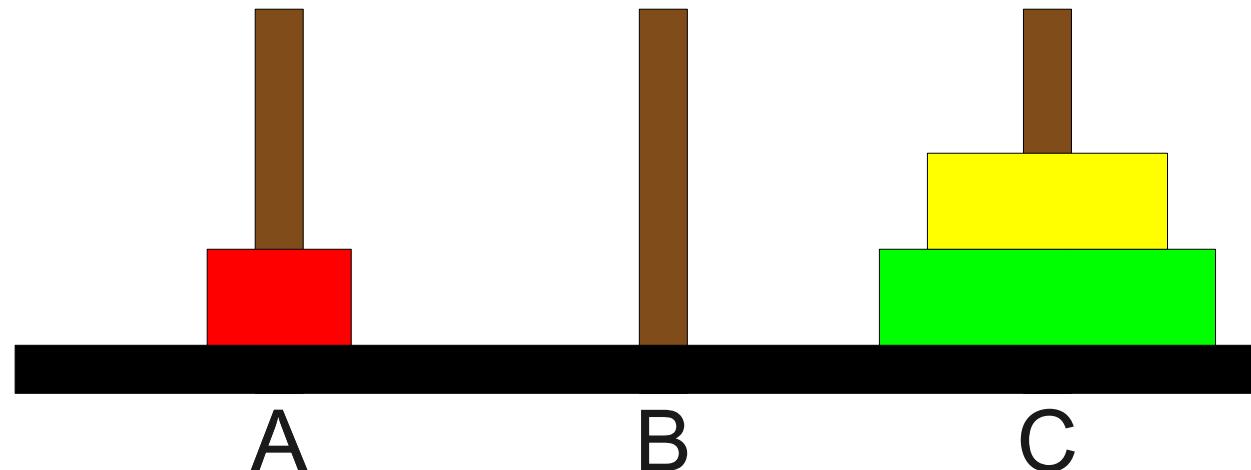
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a



```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
            moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a

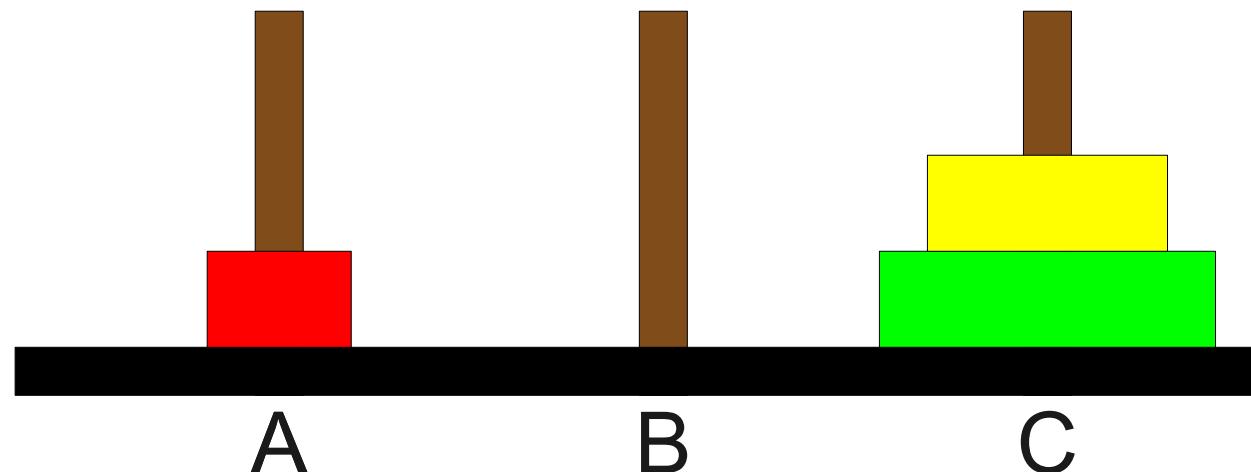


```

int main() {
}
void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        if (n == 1) {
            moveSingleDisk(from, to);
        } else {
            moveTower(n - 1, from, temp, to);
            moveSingleDisk(from, to);
moveTower(n - 1, temp, to, from);
        }
    }
}

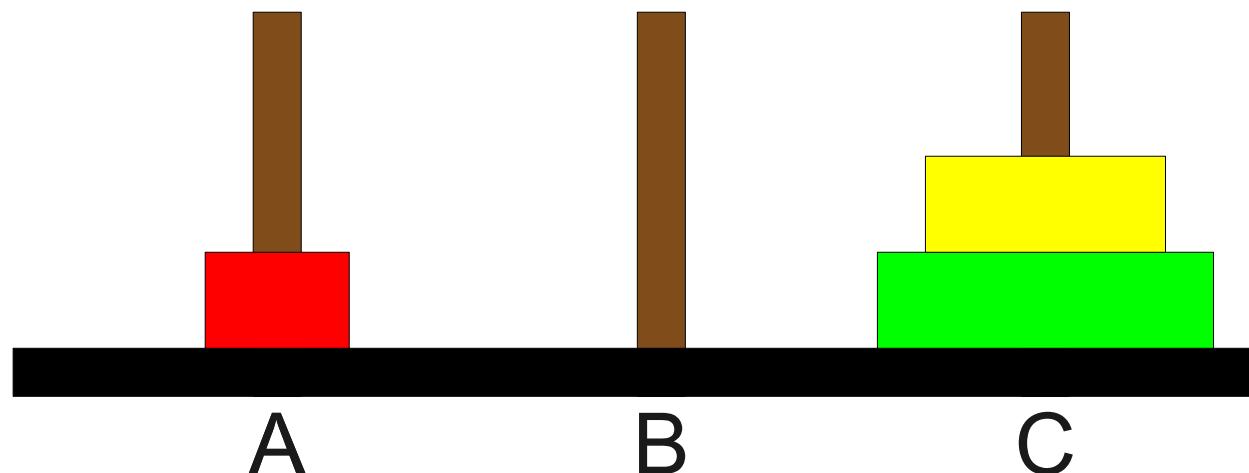
```

n 2    from b    to c    temp a



```
int main() {  
}  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        void moveTower(int n, char from, char to, char temp) {  
            if (n == 1) {  
                moveSingleDisk(from, to);  
            } else {  
                moveTower(n - 1, from, temp, to);  
                moveSingleDisk(from, to);  
                moveTower(n - 1, temp, to, from);  
            }  
        }  
    }  
}
```

n    1        from    a        to    c        temp    b



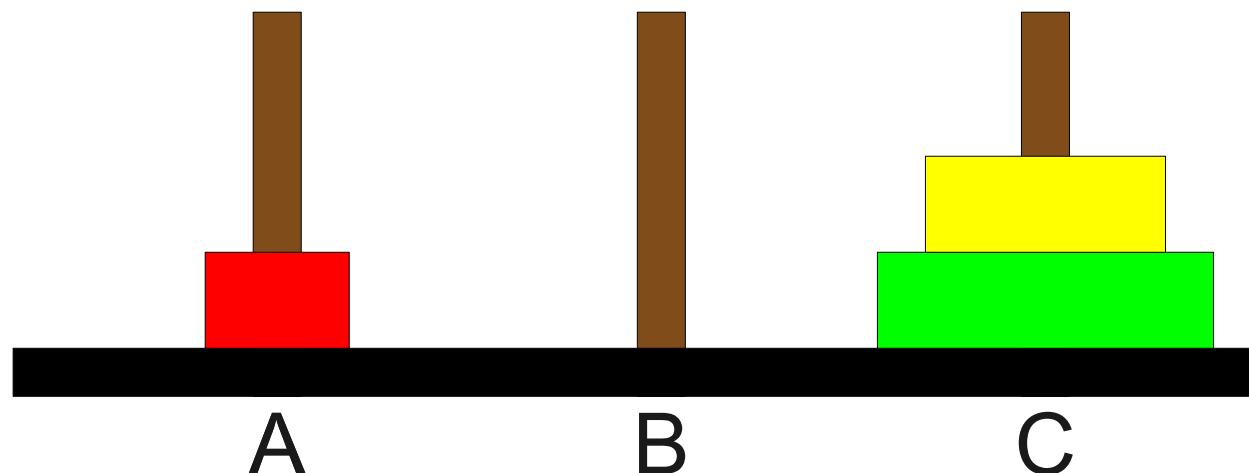
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

```

n    1    from    a    to    c    temp    b



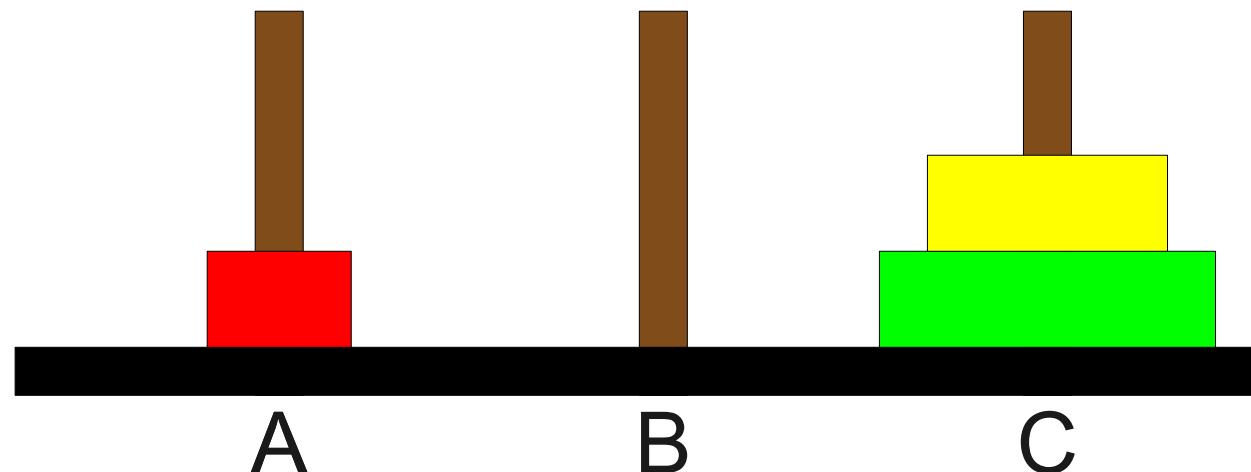
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

```

n      1      from      a      to      c      temp      b



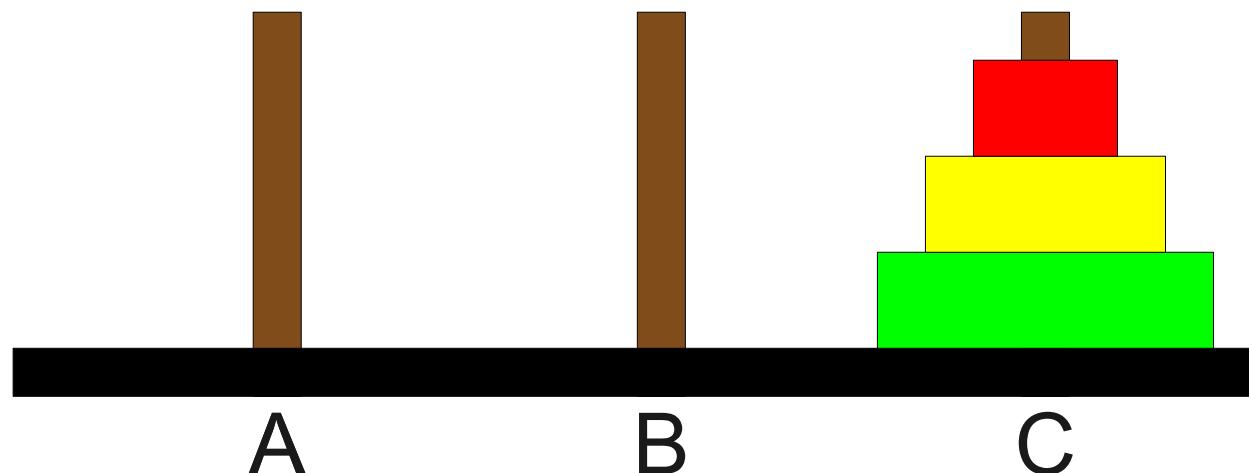
```

int main() {
}

void moveTower(int n, char from, char to, char temp) {
    void moveTower(int n, char from, char to, char temp) {
        void moveTower(int n, char from, char to, char temp) {
            if (n == 1) {
                moveSingleDisk(from, to);
            } else {
                moveTower(n - 1, from, temp, to);
                moveSingleDisk(from, to);
                moveTower(n - 1, temp, to, from);
            }
        }
    }
}

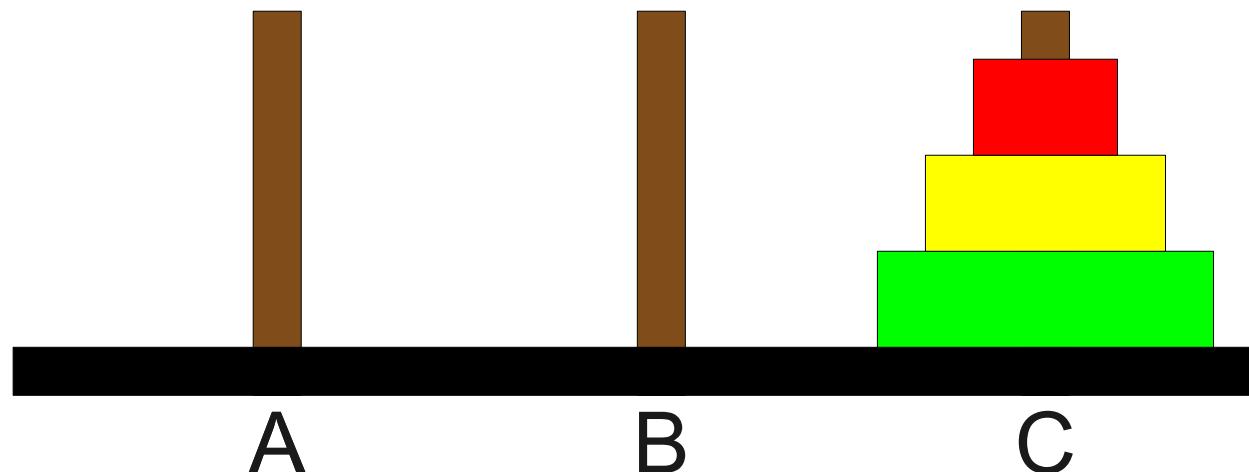
```

n      1      from      a      to      c      temp      b



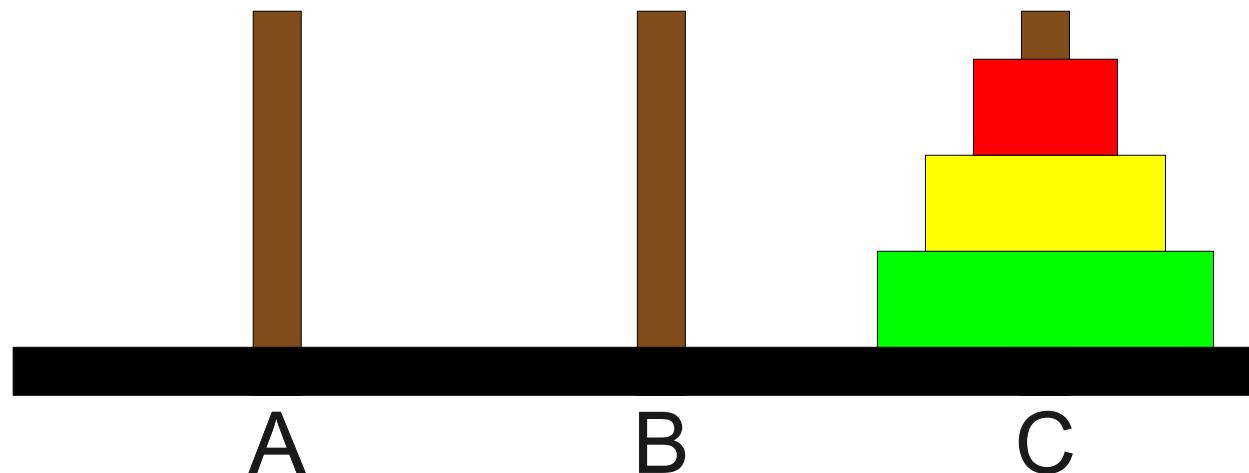
```
int main() {  
} void moveTower(int n, char from, char to, char temp) {  
    void moveTower(int n, char from, char to, char temp) {  
        if (n == 1) {  
            moveSingleDisk(from, to);  
        } else {  
            moveTower(n - 1, from, temp, to);  
            moveSingleDisk(from, to);  
moveTower(n - 1, temp, to, from);  
        }  
    }  
}
```

n 2    from b    to c    temp a

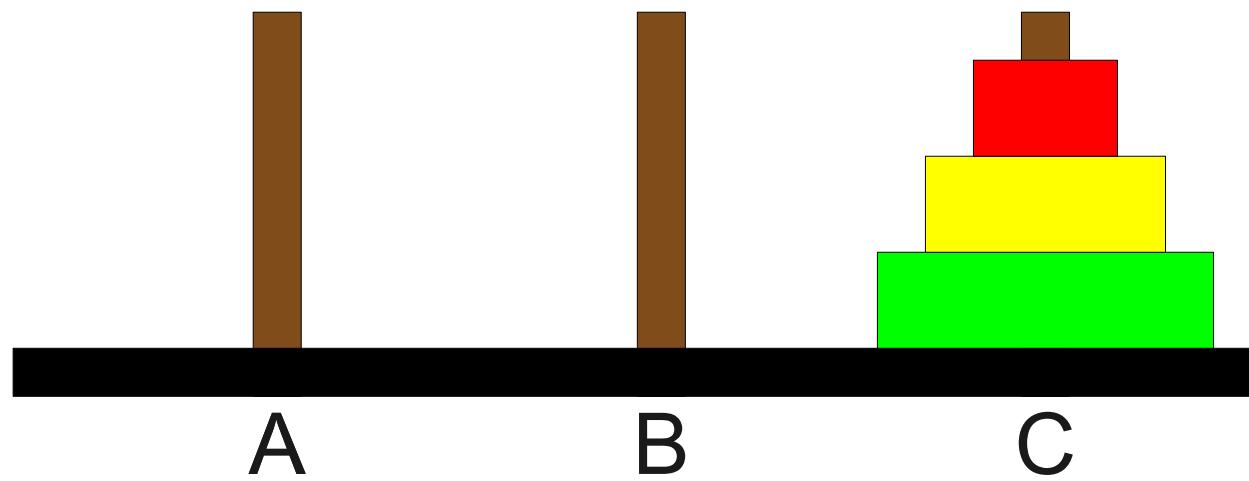


```
int main() {  
}  
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

n 3    from a    to b    temp c



```
int main() {
    moveTower(3, 'a', 'b', 'c');
}
```



# Emergent Behavior

- Even though each function call does very little work, the overall behavior of the function is to solve the Towers of Hanoi.
- It's often tricky to think recursively because of this **emergent behavior**:
  - No one function call solves the entire problem.
  - Each function does only a small amount of work on its own and delegates the rest.

# Writing Recursive Functions

- Although it is good to be able to trace through a set of recursive calls to understand how they work, you will need to build up an intuition for recursion to use it effectively.
- You will need to learn to trust that your recursive calls – which are to the function that you are currently writing! – will indeed work correctly.
  - Eric Roberts calls this the “**Recursive Leap of Faith.**”
- ***Everyone can learn to think recursively.*** If this seems confusing now, **don't panic**. You'll start picking this up as we continue forward.

```
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

```
void moveTower(int n, char from, char to, char temp) {  
    if (n == 1) {  
        moveSingleDisk(from, to);  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

```
void moveTower(int n, char from, char to, char temp) {  
    if (n == 0) {  
  
    } else {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

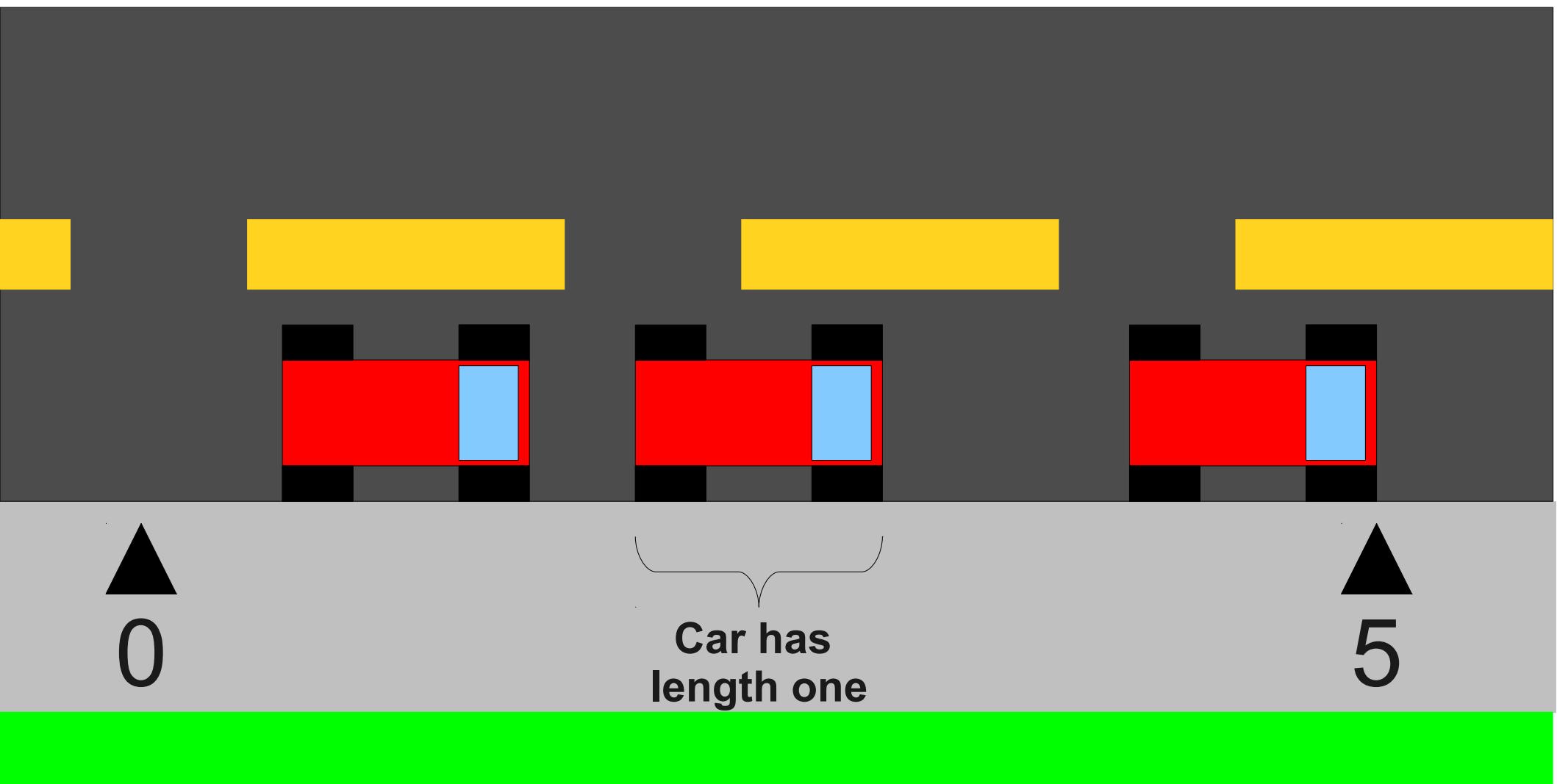
```
void moveTower(int n, char from, char to, char temp) {  
    if (n != 0) {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

```
void moveTower(int n, char from, char to, char temp) {  
    if (n != 0) {  
        moveTower(n - 1, from, temp, to);  
        moveSingleDisk(from, to);  
        moveTower(n - 1, temp, to, from);  
    }  
}
```

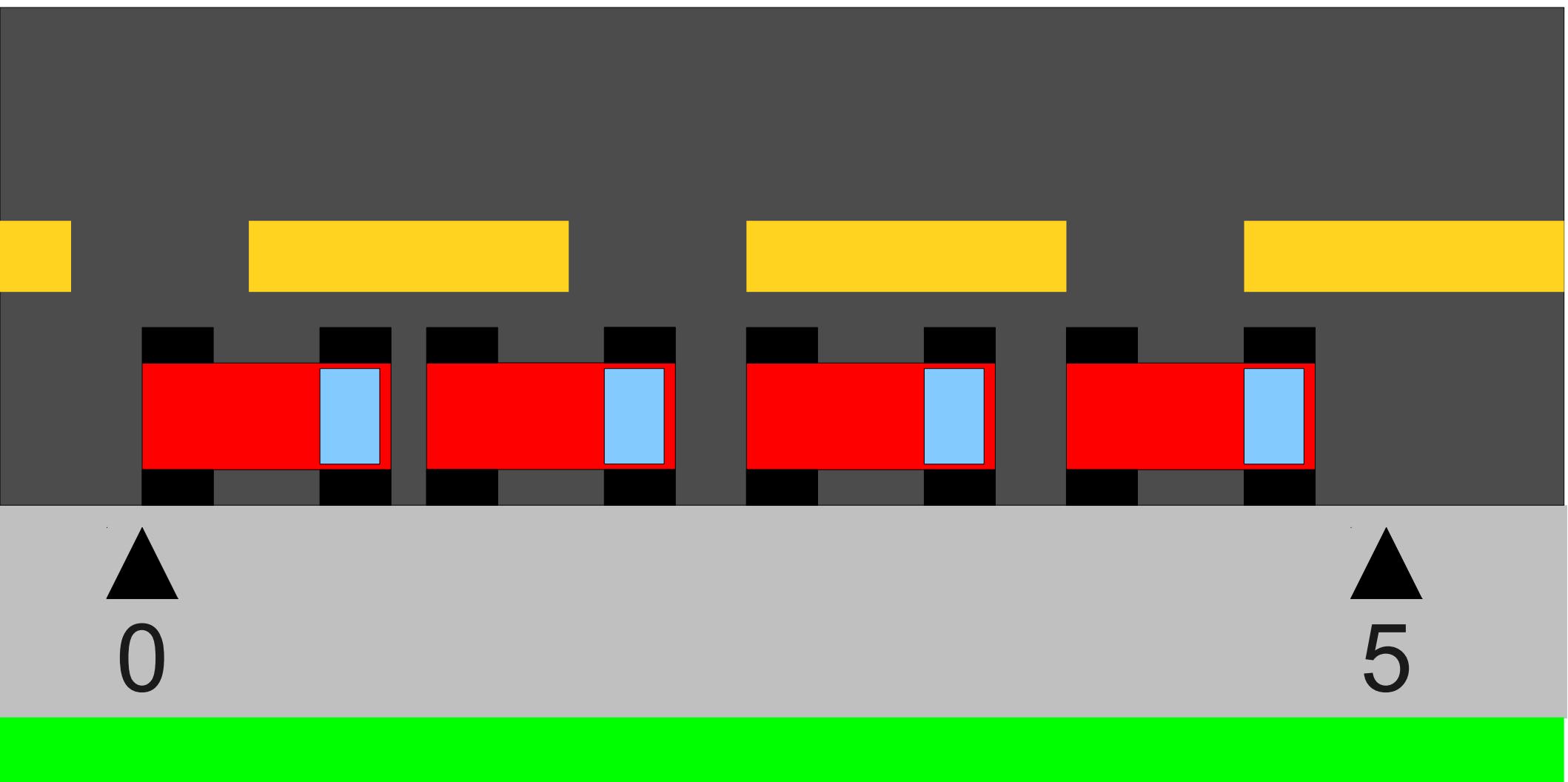
# Picking a Base Case

- When choosing base cases, you should always try to pick the absolute smallest case possible.
- The simplest case is often so simple that it appears silly.
  - Solve Towers of Hanoi with no disks.
  - Add up no numbers.
  - List all strings of no characters.
- This is a skill you'll build up with practice.

# Parking Randomly



# Parking Randomly



# Parking Randomly

- Given a curb of length five, how many cars, on average, can park on the curb?
- We can get an approximate value through random simulation:
  - Simulate random parking a large number of times.
  - Output the average number of cars that could park.
- **Question:** How do we simulate parking cars on the curb?

# Parking Randomly

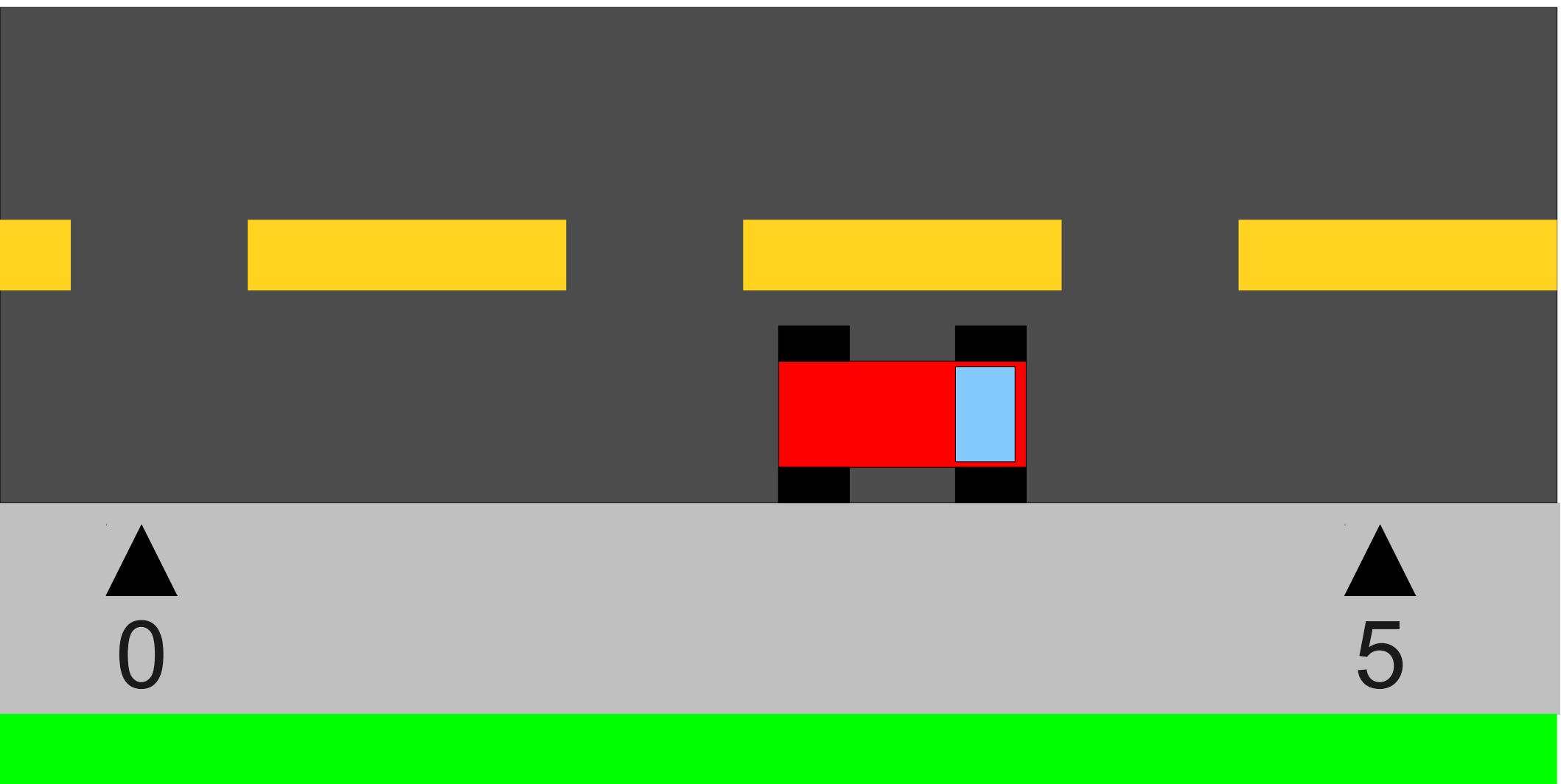


0

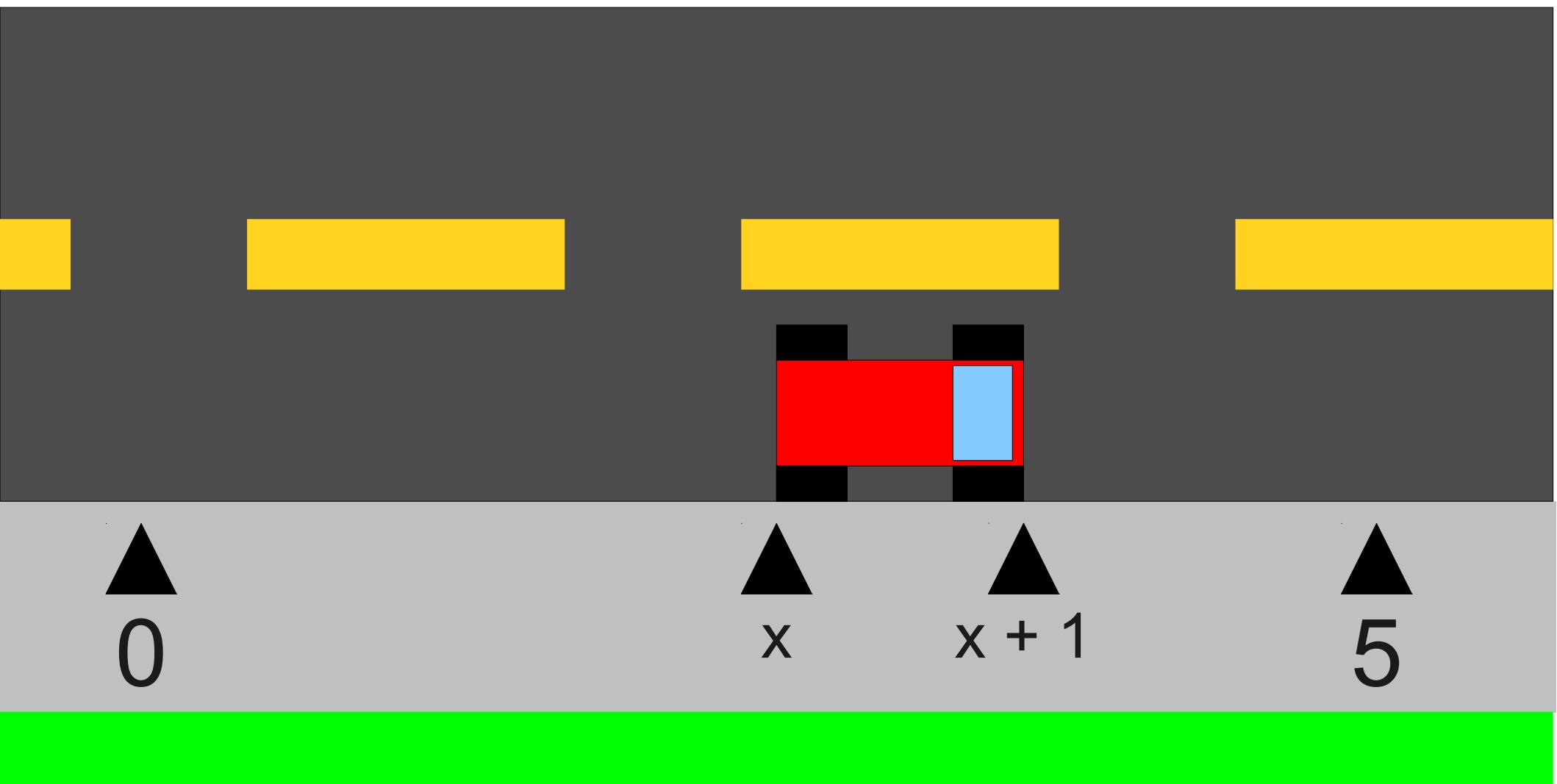


5

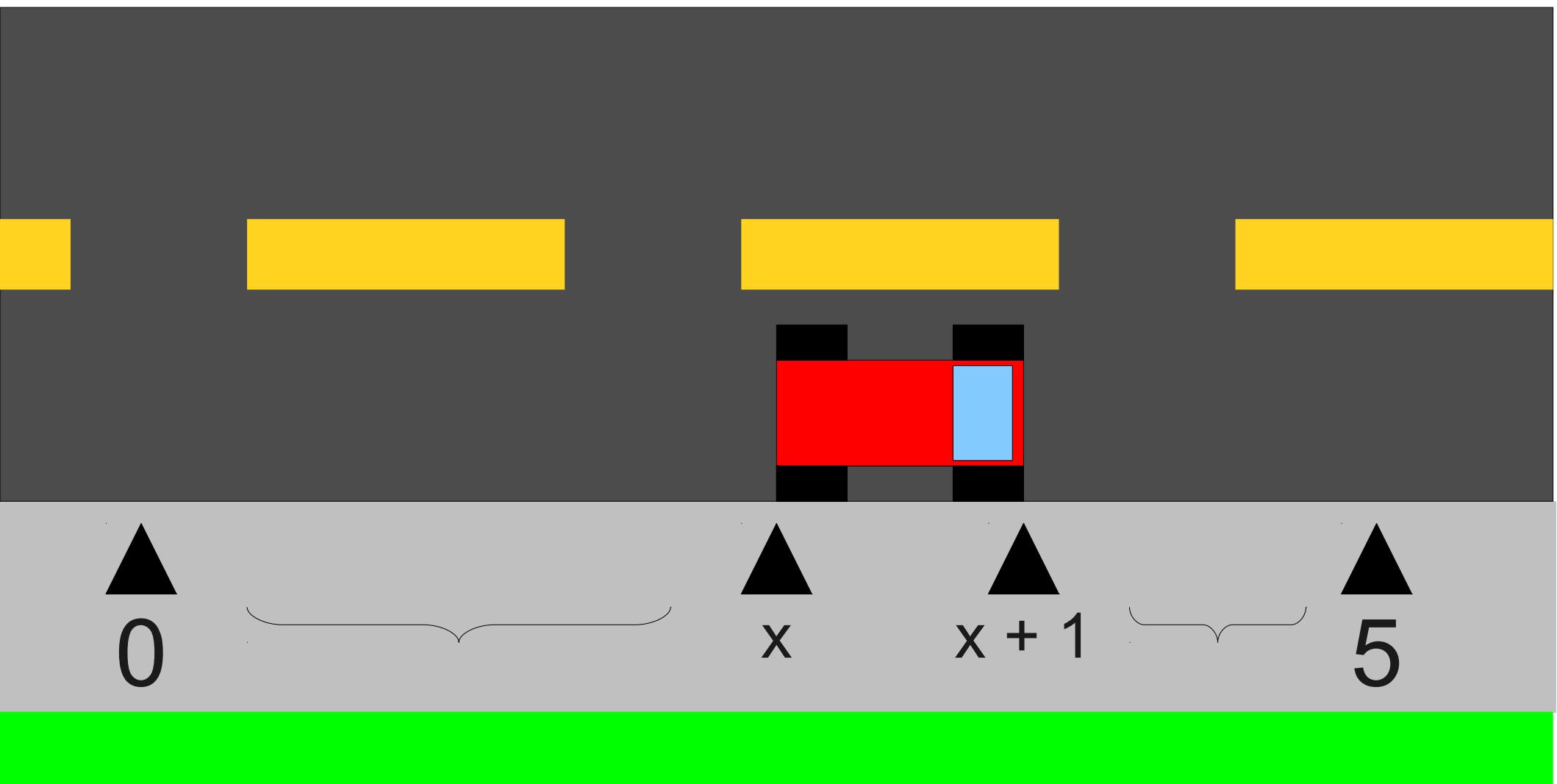
# Parking Randomly



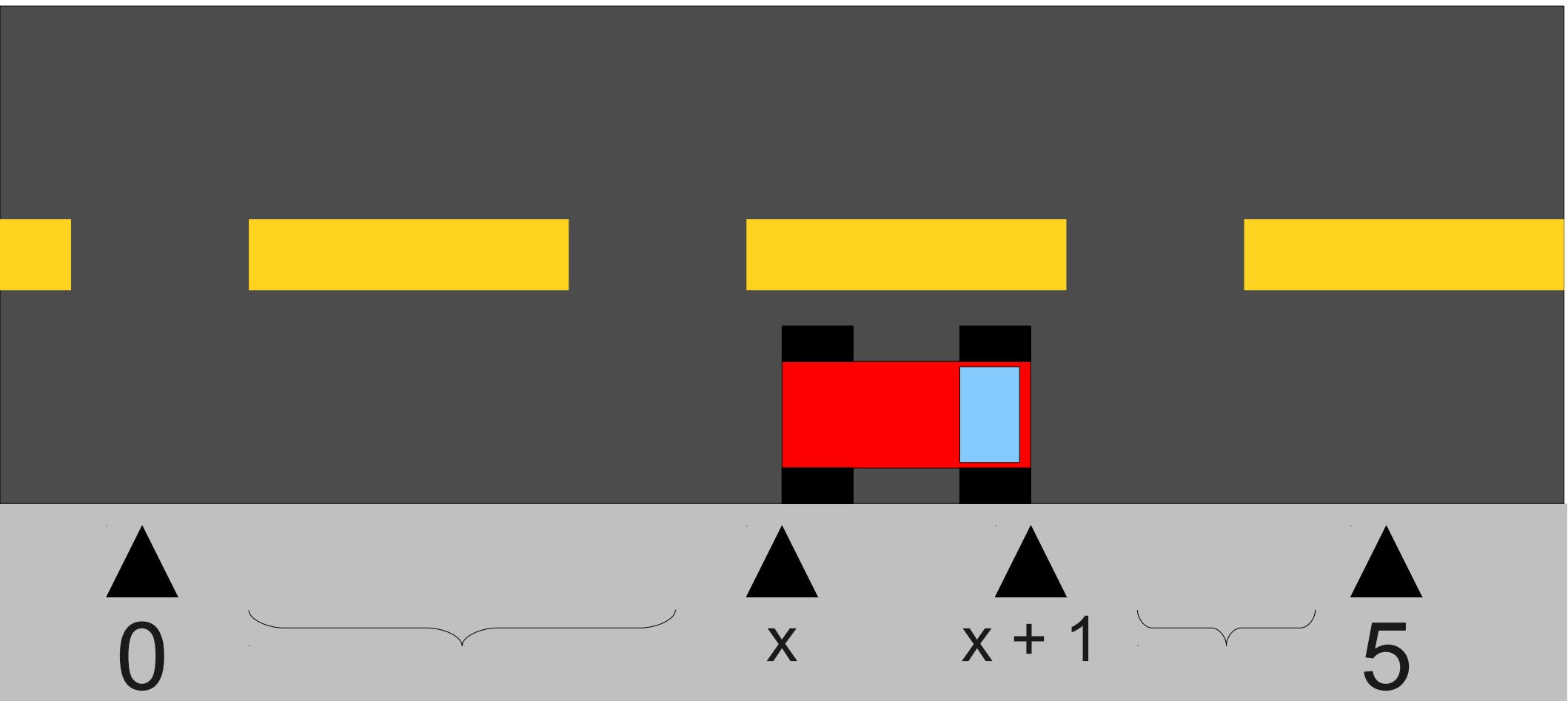
# Parking Randomly



# Parking Randomly



# Parking Randomly



Place cars randomly in these ranges!

# Parking Randomly

```
int parkRandomly(double low, double high) {  
    if (high - low < 1.0) {  
        return 0;  
    } else {  
        double x = randomReal(low, high - 1.0);  
        return 1 + parkRandomly(low, x) +  
            parkRandomly(x + 1.0, high);  
    }  
}
```

# The Parking Ratio

- The average number of cars that can be parked in a range of width  $w$  for sufficiently large  $w$  is approximately
$$0.7475972 w$$
- The constant  $0.7475972\dots$  is called **Rényi's Parking Constant**.
- For more details, visit <http://mathworld.wolfram.com/RenyisParkingConstants.html>.

# So What?

- The beauty of our algorithm is the following recursive insight:

**Split an area into smaller, independent pieces and solve each piece separately.**

- Many problems can be solved this way.

# Next Time

- **Graphical Recursion**
  - How do you draw a self-similar object?
- **Exhaustive Recursion**
  - How do you generate all objects of some type?
  - Algorithms for subsets, permutations, and combinations.