Thinking Recursively

Announcements

- Assignment 2 due Wednesday
- Michael's OH changed this week to 3-5PM, Monday and Tuesday in Gates 160

Assignment 1 Emails

- Thanks for all the feedback!
  - It's very important to me that this course is a positive experience for you all.
- Common concern: debugging is a pain.
  - We don't cover debugging in lecture due to time constraints.
  - **Please check out handouts on debugging on website**
  - Debugging can be super obnoxious at times, but there are strategies you can follow that can make your life easier.

Aubrey's Debugging Process

- First step of debugging is generally to construct a simple test case that “breaks your code”
- For example, for word ladder you may know your code doesn't work for “bake” and “cook”
  - This is a tough case to debug because the correct word ladder is at least of length 4
  - First thing to do is to try to find a simpler case. Does your code work for “bot” and “cot”?

Aubrey's Debugging Process

- Next step is to find a line of code in which something happens that you know shouldn't happen.
- Common way of doing this is stepping through your code with the debugger and looking at the state of local variables
  - Learning to use the debugger will save you a lot of time!
  - Aubrey, Mike or an SL can show you how to use the debugger!

Aubrey's Debugging Process

- Final step is figure out why something “bad” is happening at this line of code.
- If it isn't immediately obvious what the line of code is doing wrong, then maybe the line is “too complicated”

```
int x = myFunction(v[i]*v[j]) + y;
```

- It may be helpful to “breakup” the code over multiple lines:

```
int t = v[i]*v[j];
int k = myFunction(t);
int x = t + k;
```

Debugging: Stuff to Avoid

- **Avoid "just staring" at your code**. This is frequently a sign of not knowing what to do.
  - Sometimes this works, but there are no guarantees
- **Avoid making "random changes" to your code**. This is also frequently a sign of not knowing what to do.
  - e.g. randomly swapping `&&` with `||`

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));
   }
}
```

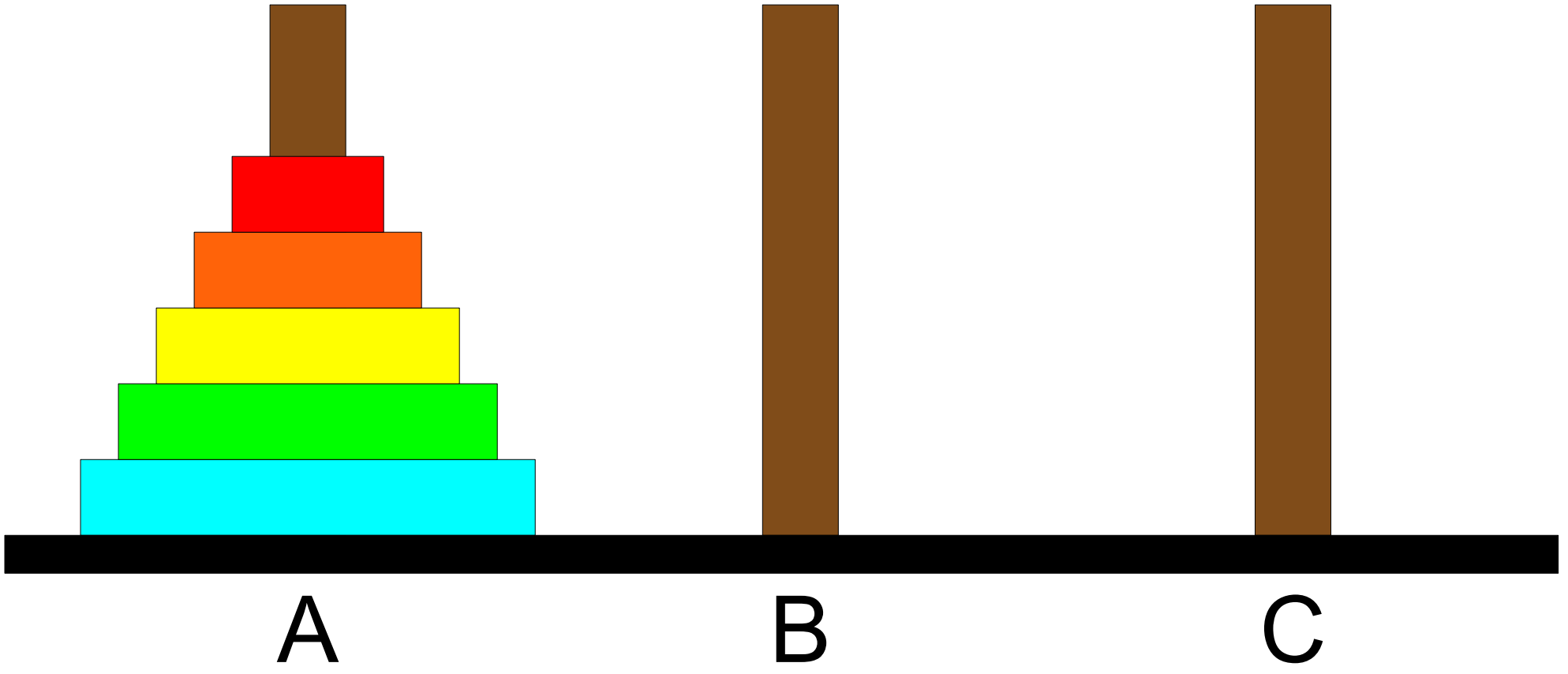
```
int maximumCoverageFor(Vector<int>& populations) {
   if (populations.size() == 0) {
      return 0;
   } else if (populations.size() == 1) {
      return populations[0];
   } else {
      Vector<int> allButFirst    = tailOf(populations);
      Vector<int> allButFirstTwo = tailOf(allButFirst);

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

What's Going On?

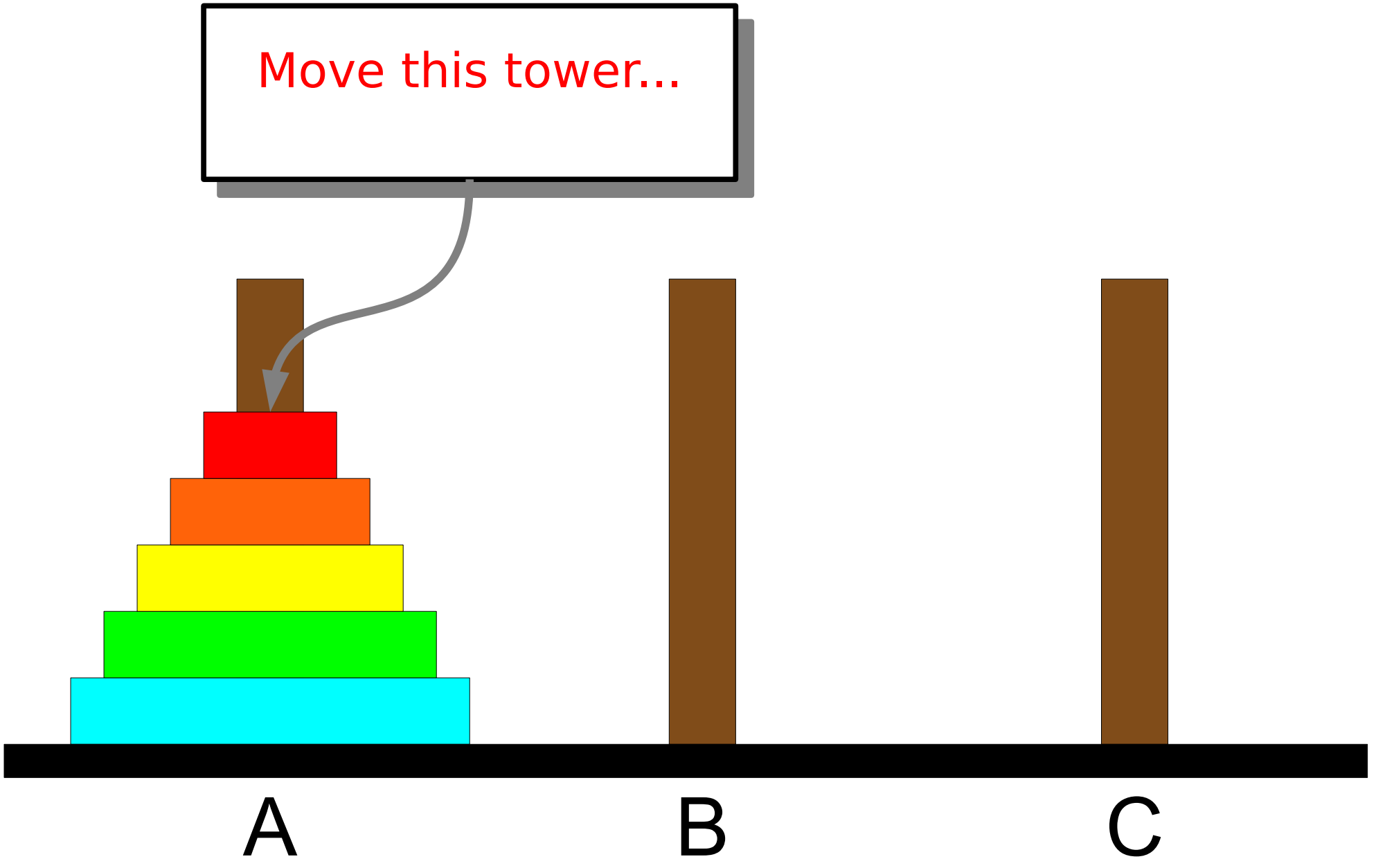
- Recursion solves a problem by continuously simplifying the problem until it becomes simple enough to be solved directly.
- The **recursive decomposition** makes the problem slightly simpler at each step.
- The **base case** is what ultimately makes the problem solvable – it guarantees that when the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

The Towers of Hanoi Problem

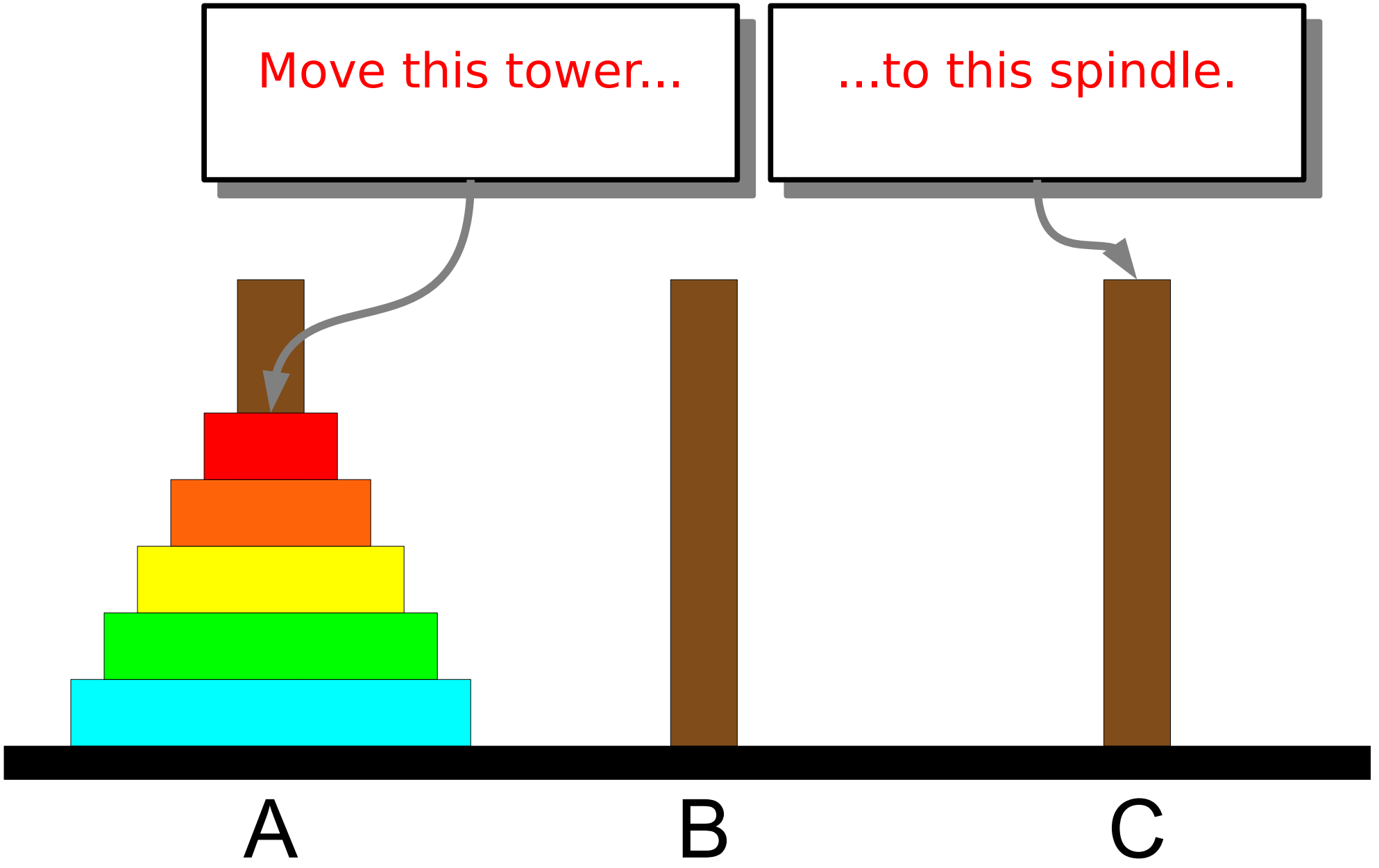
Towers of Hanoi



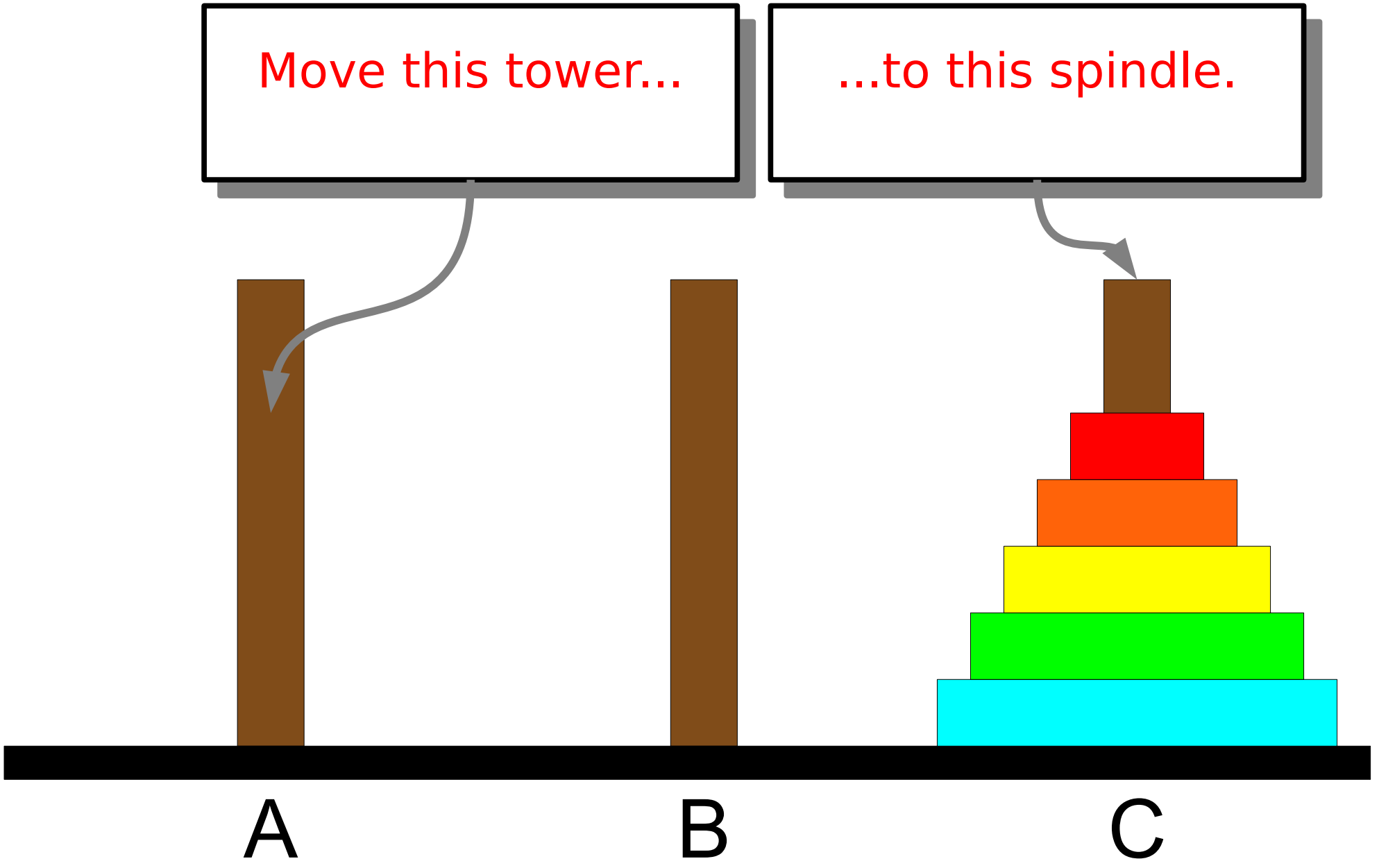
Towers of Hanoi



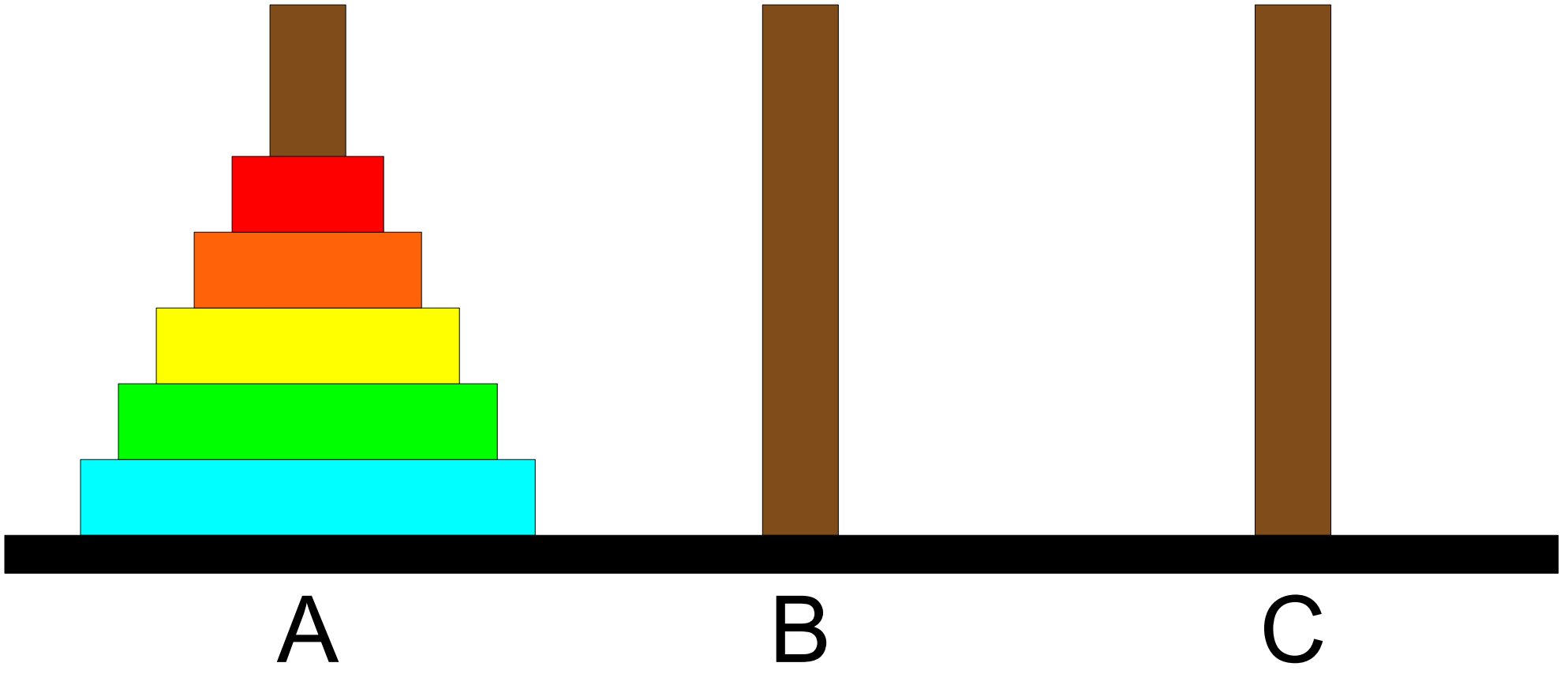
Towers of Hanoi



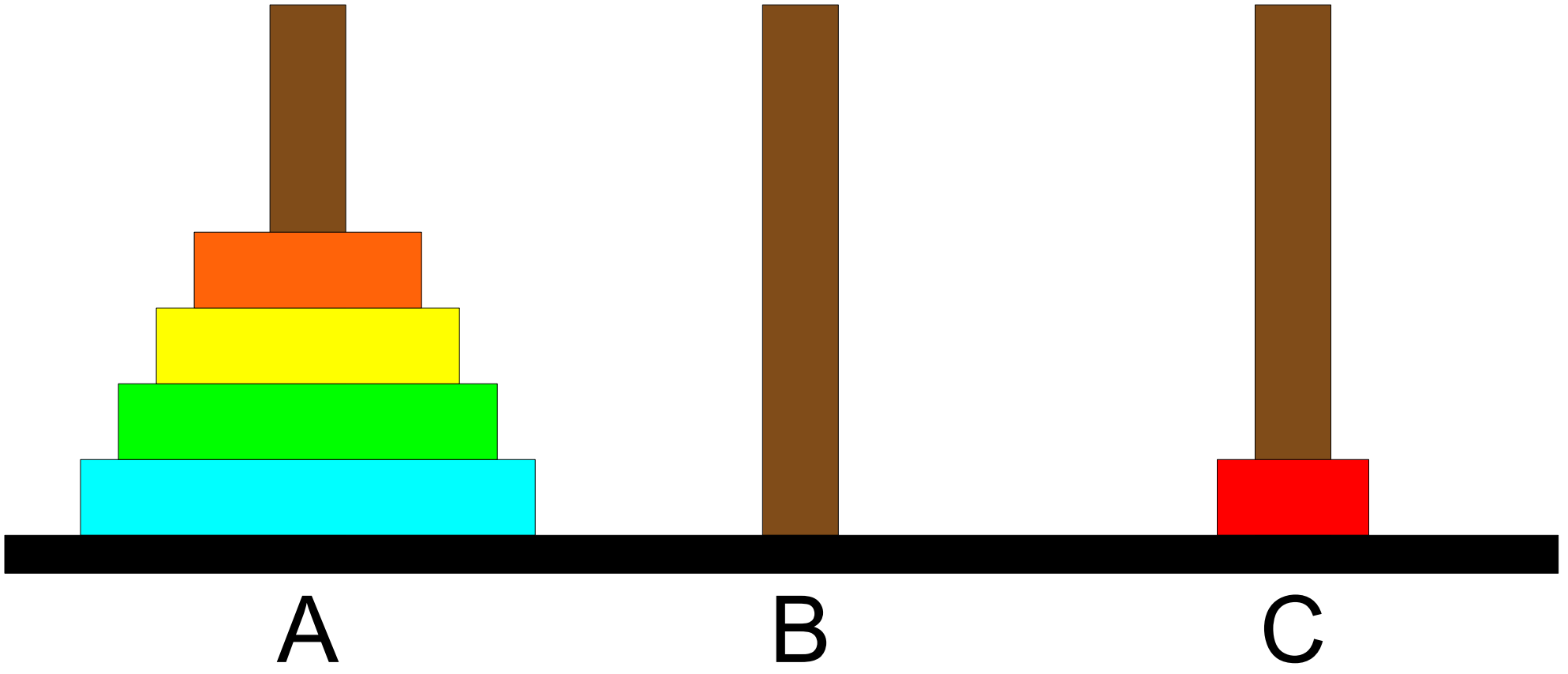
Towers of Hanoi



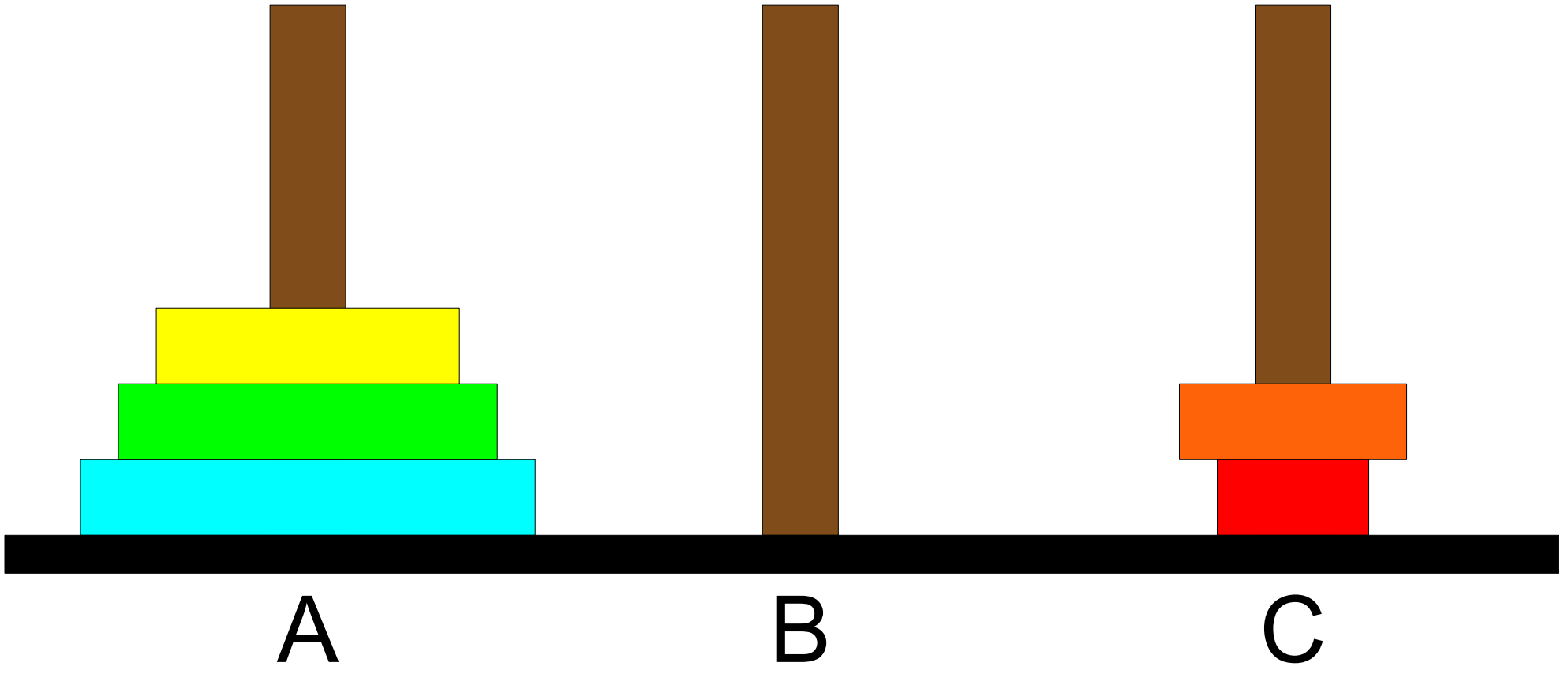
Towers of Hanoi



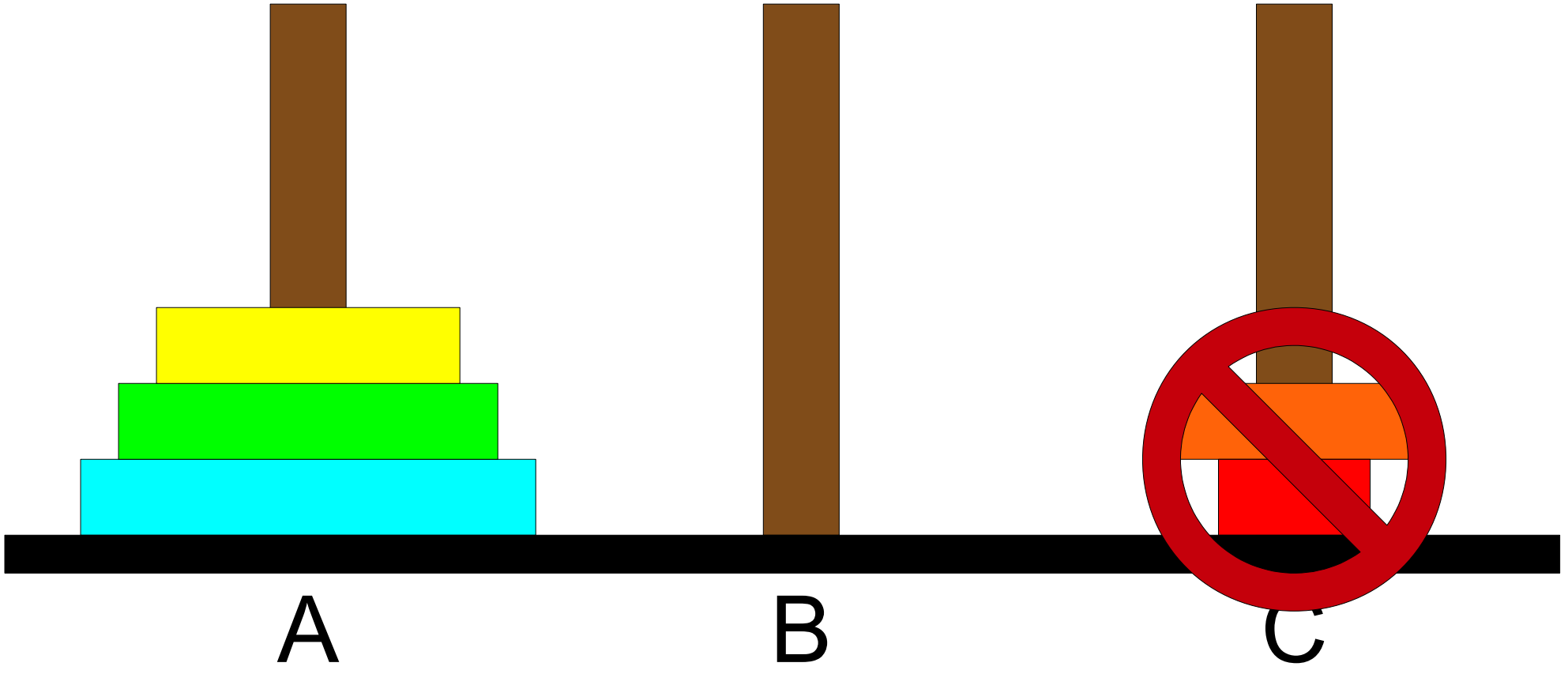
Towers of Hanoi



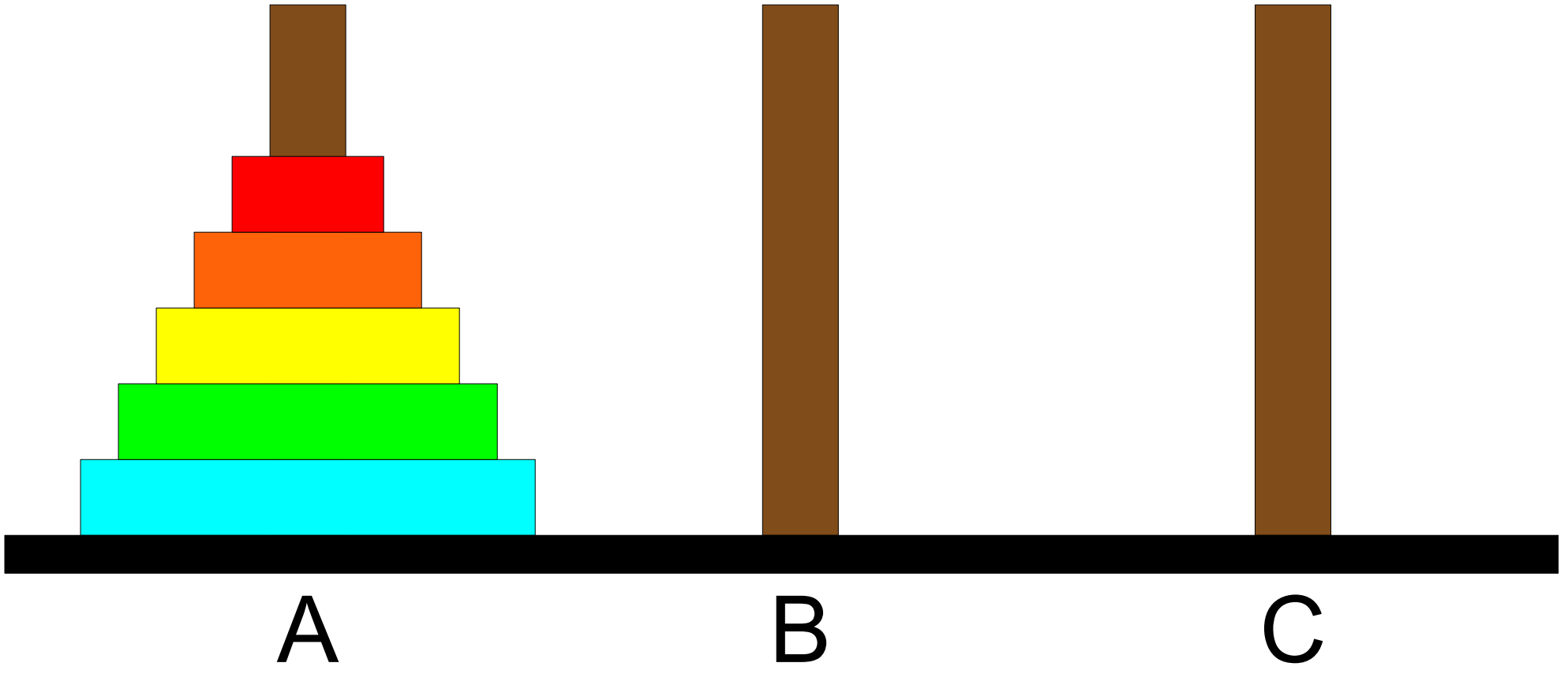
Towers of Hanoi



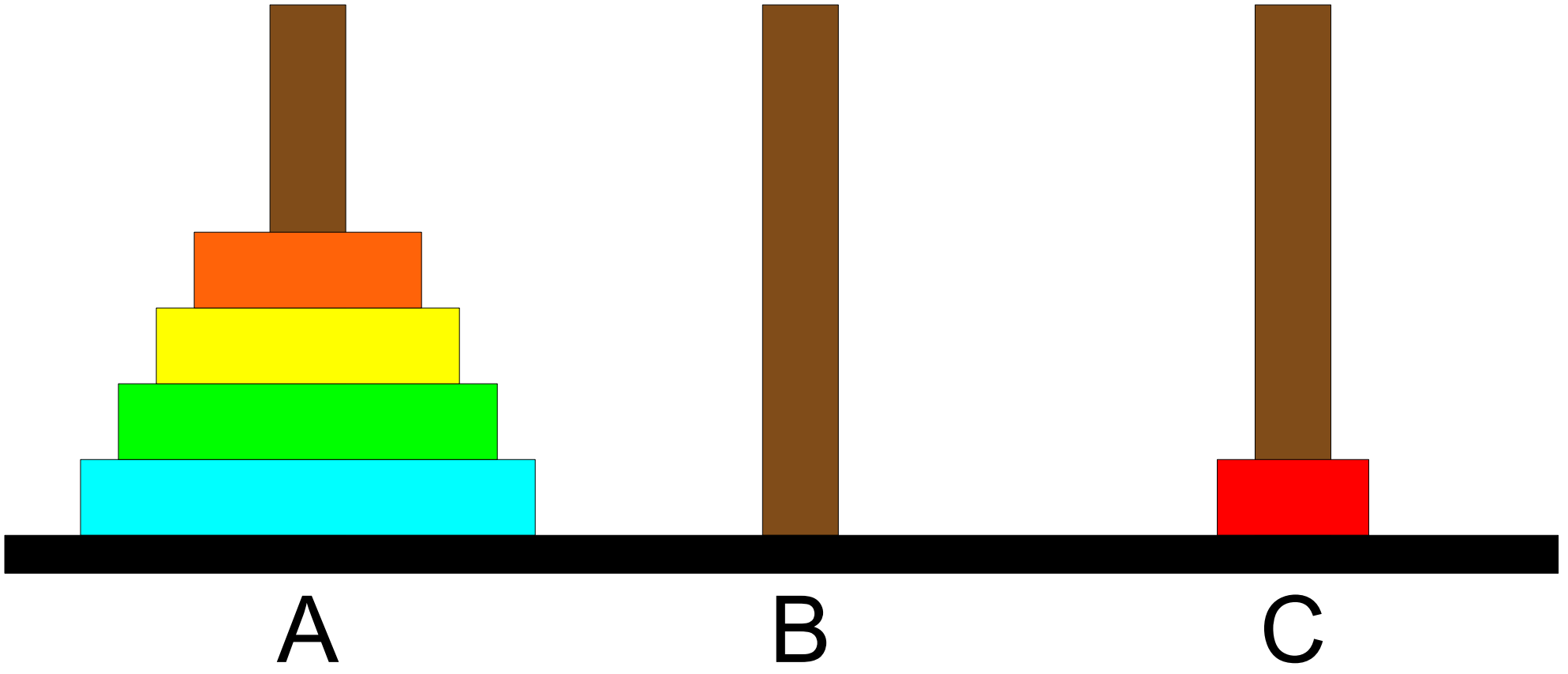
Towers of Hanoi



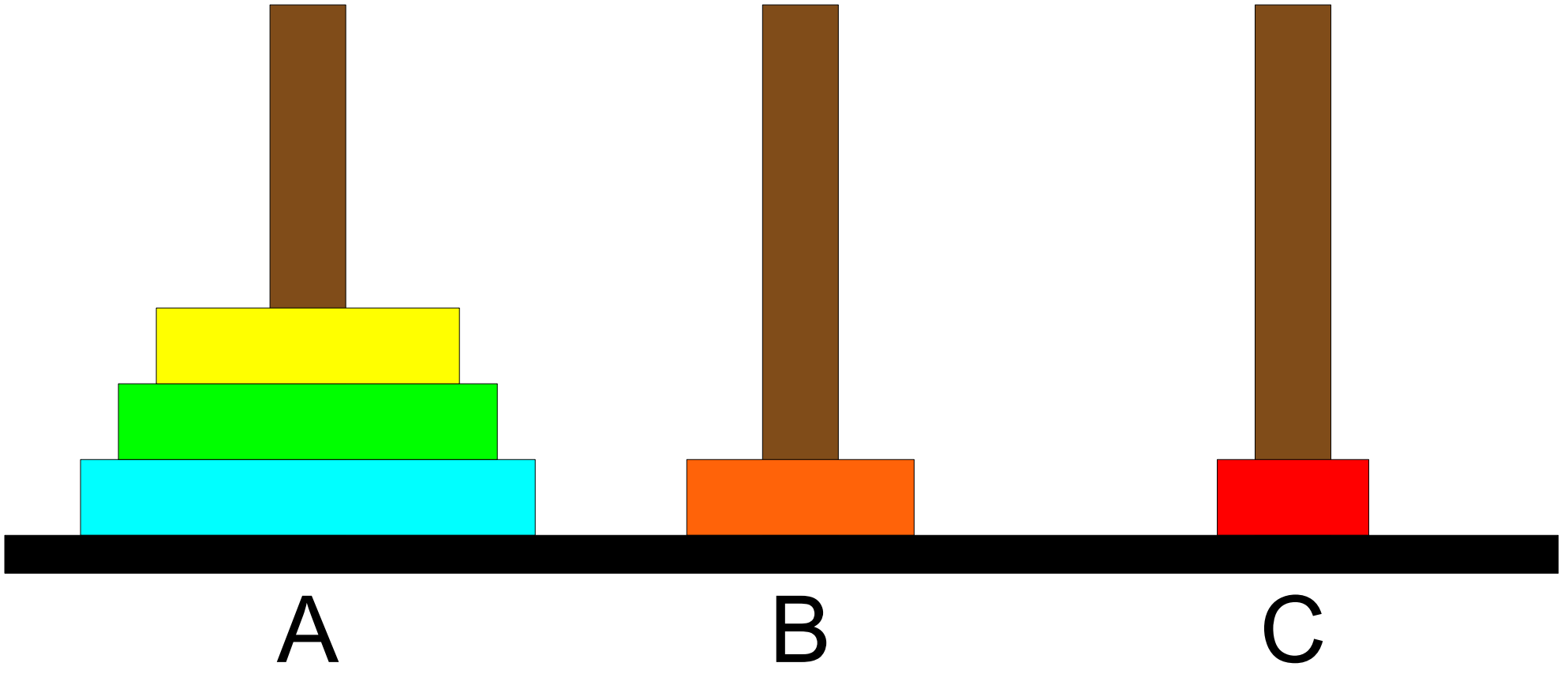
Towers of Hanoi



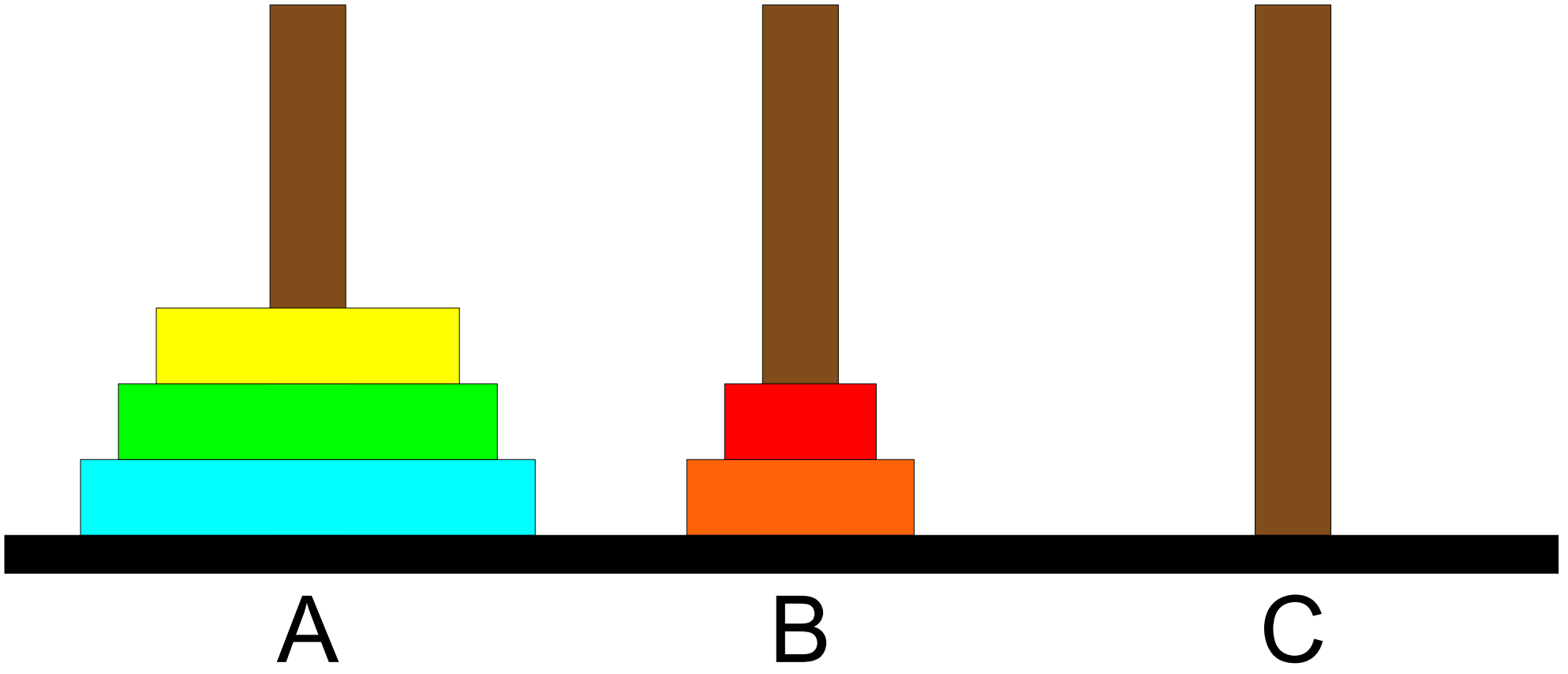
Towers of Hanoi



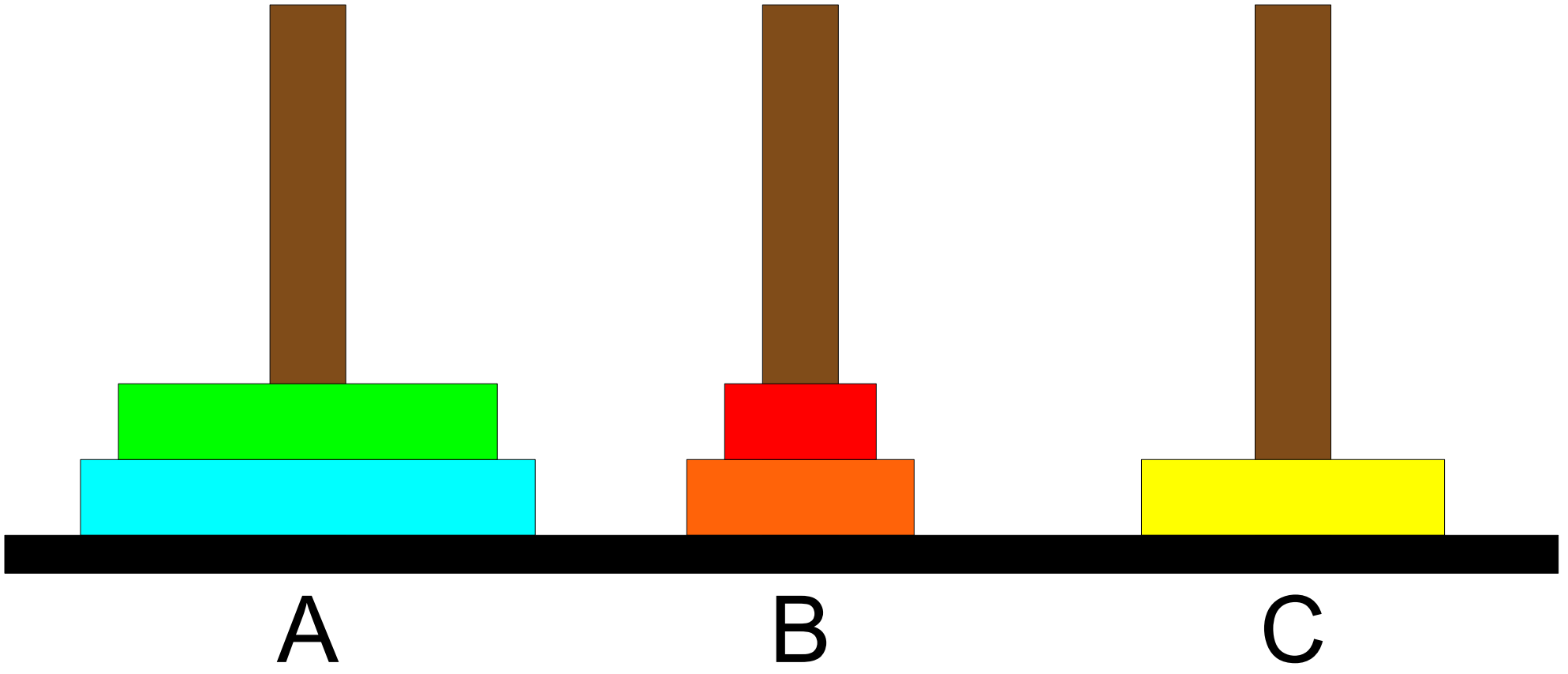
Towers of Hanoi



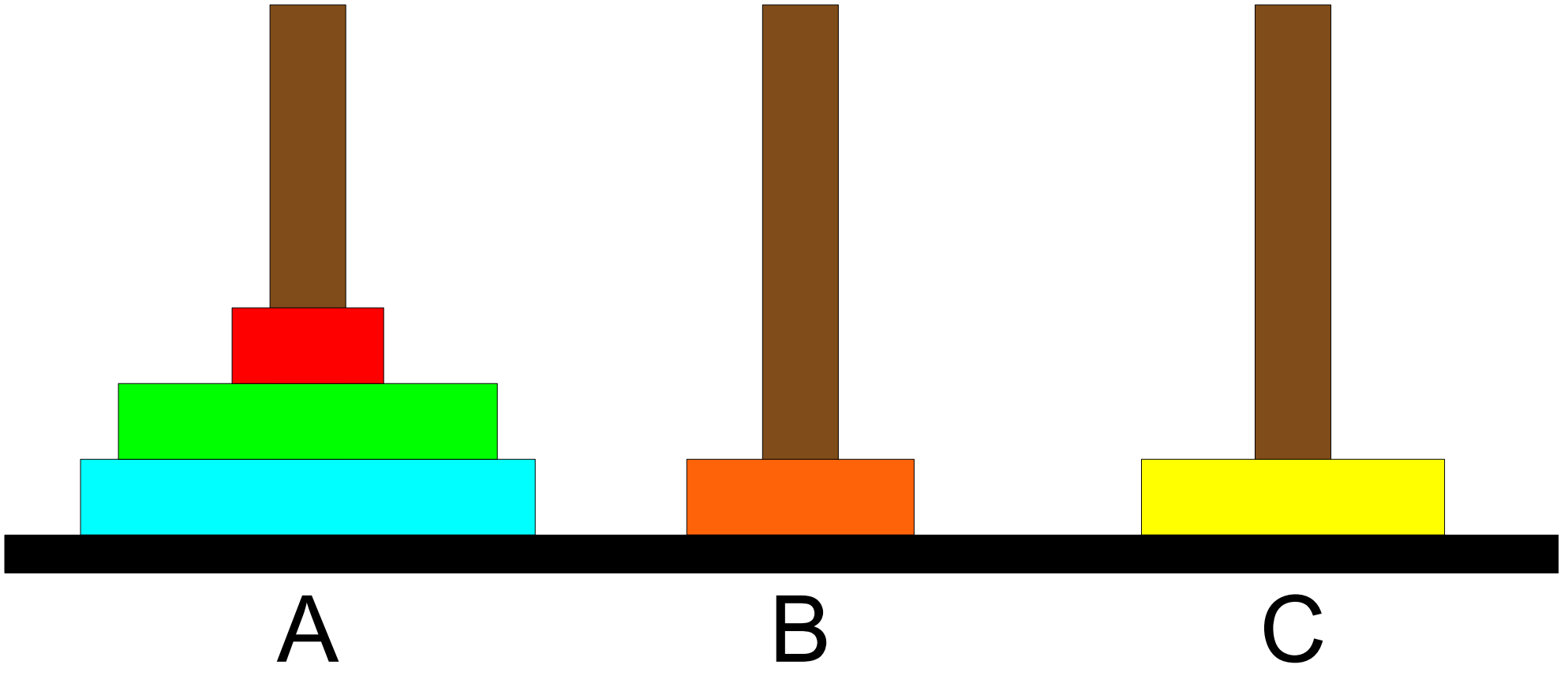
Towers of Hanoi



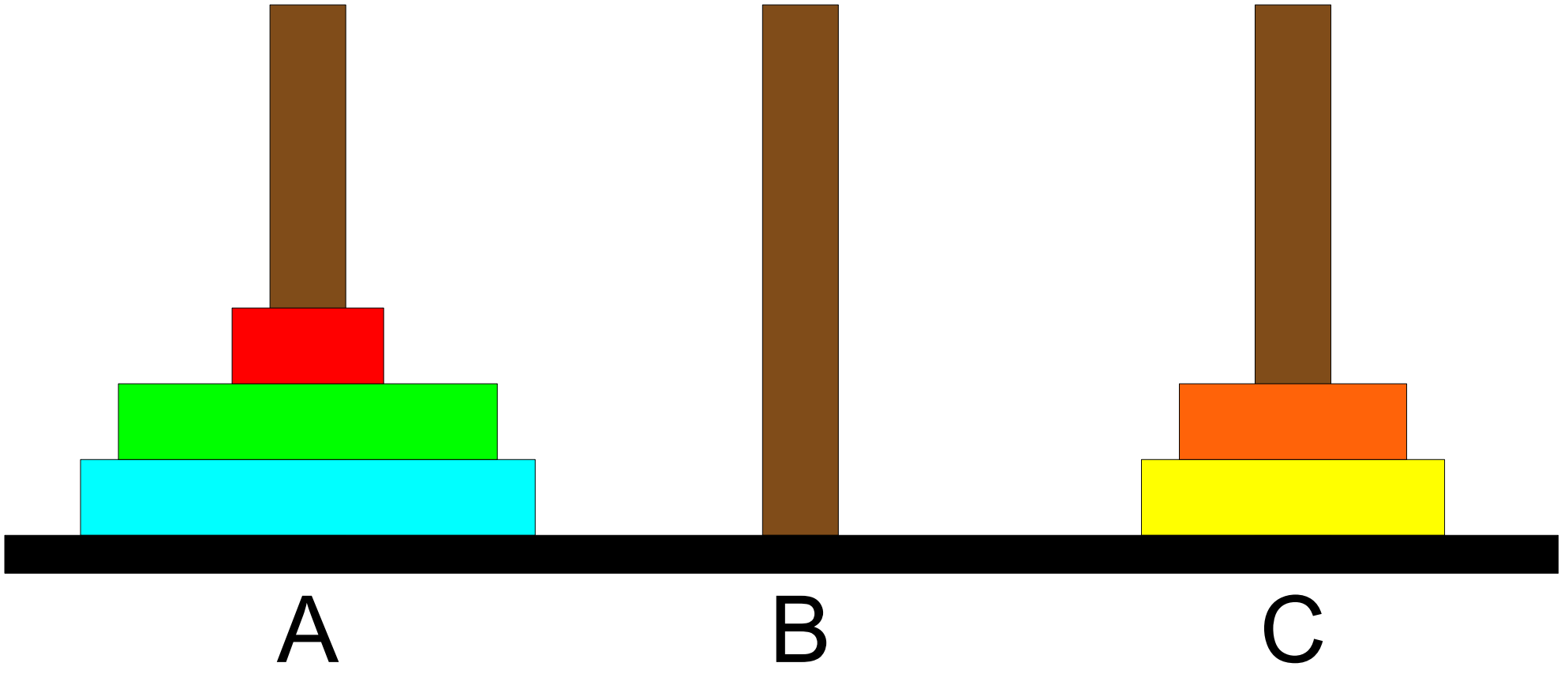
Towers of Hanoi



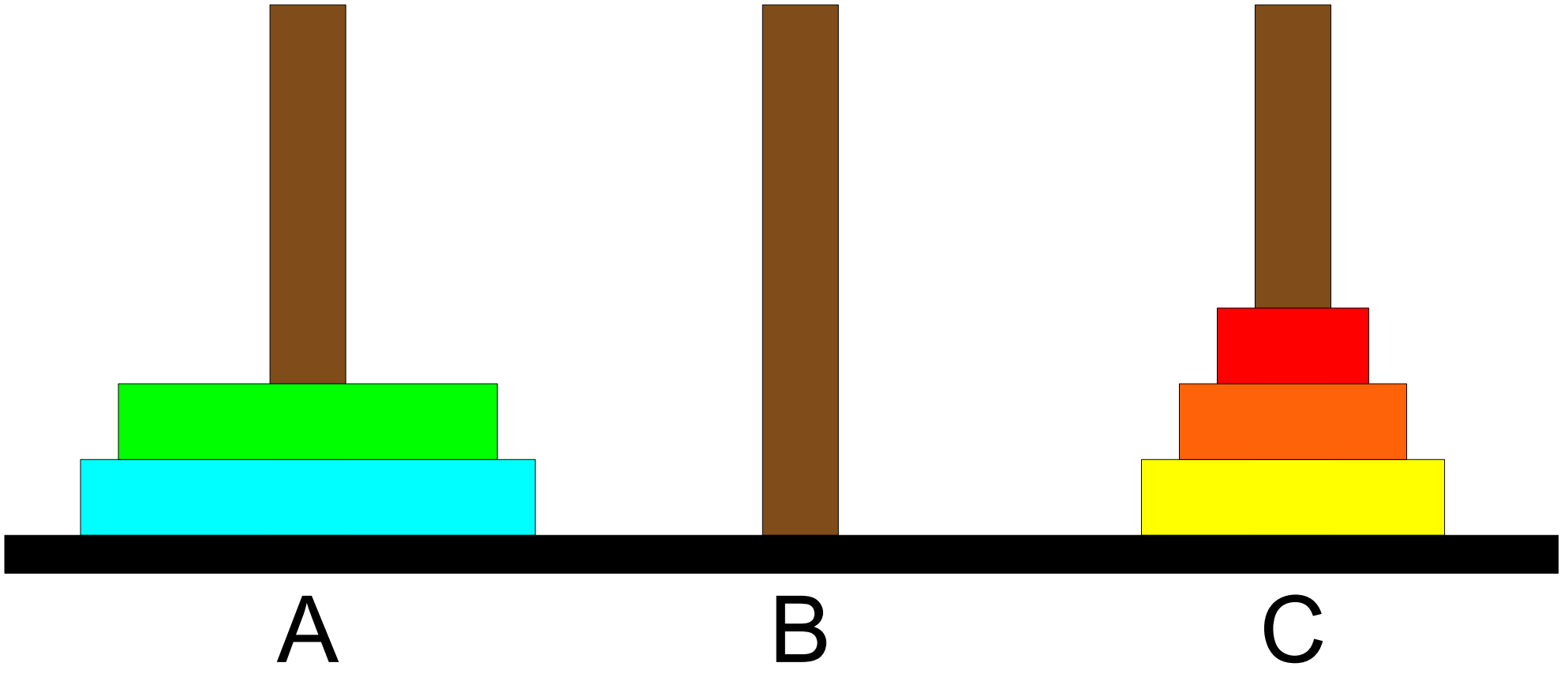
Towers of Hanoi



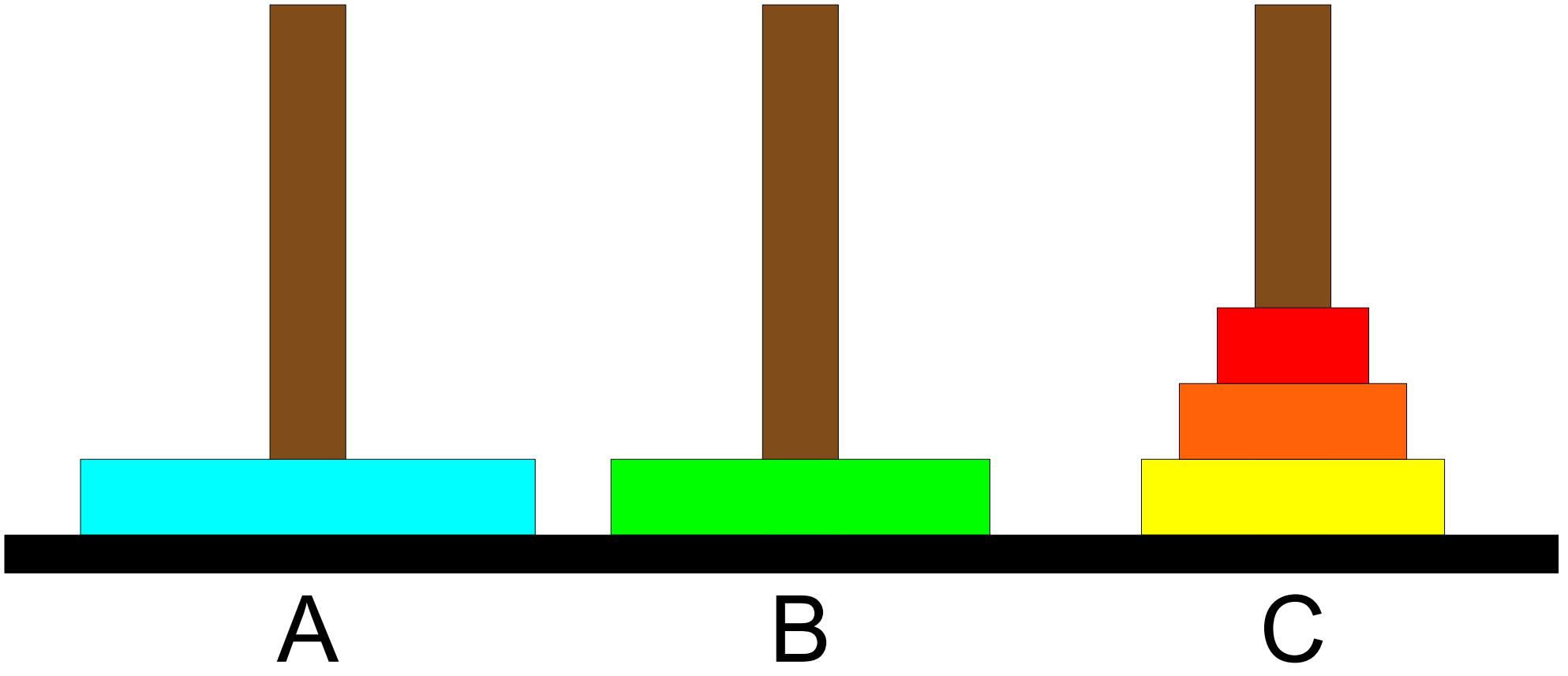
Towers of Hanoi



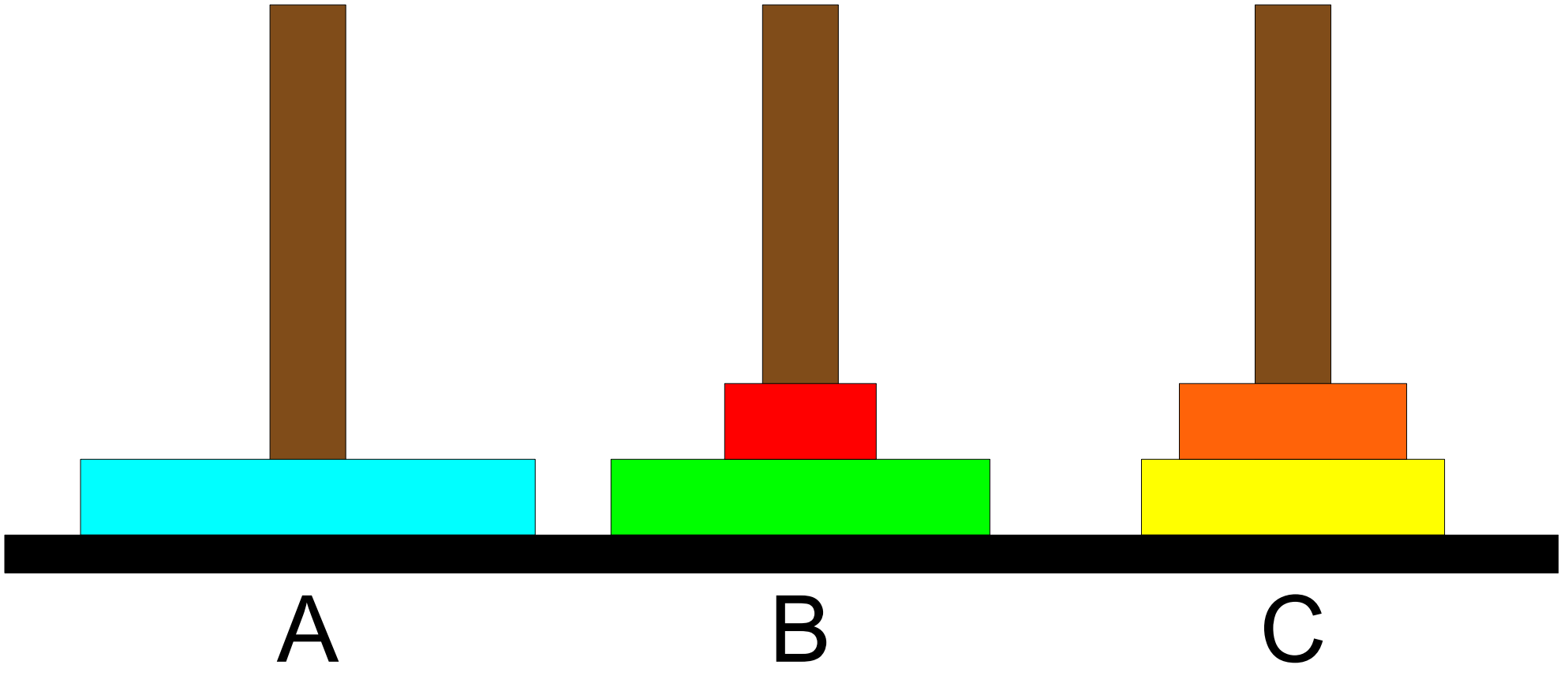
Towers of Hanoi



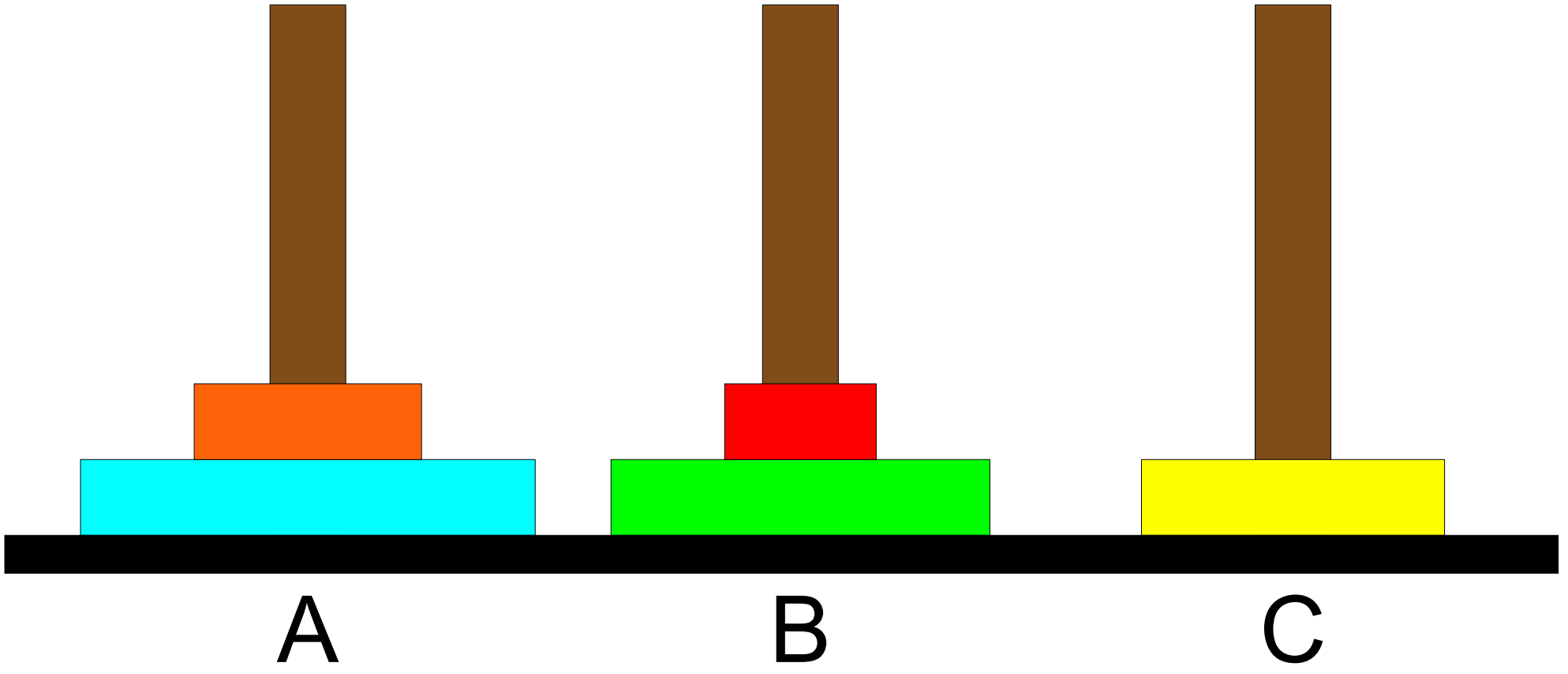
Towers of Hanoi



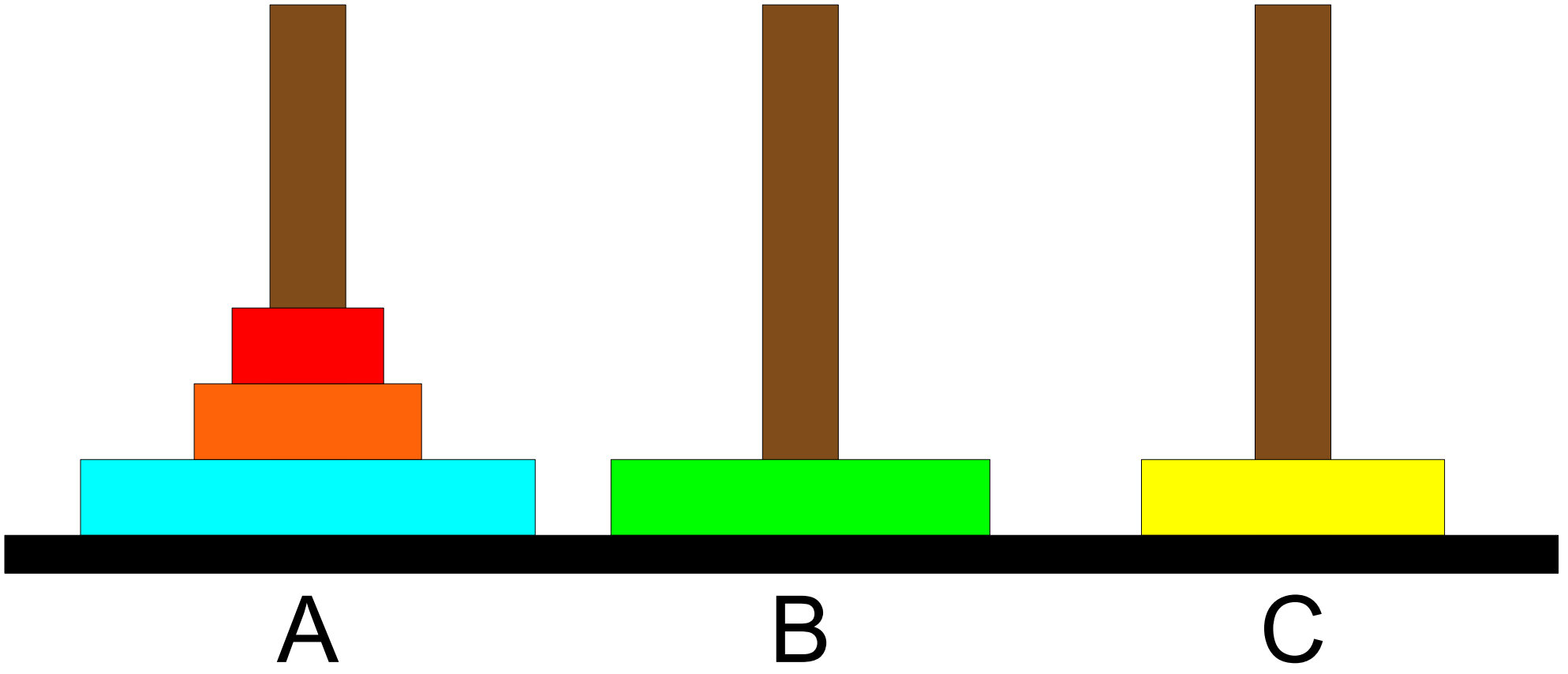
Towers of Hanoi



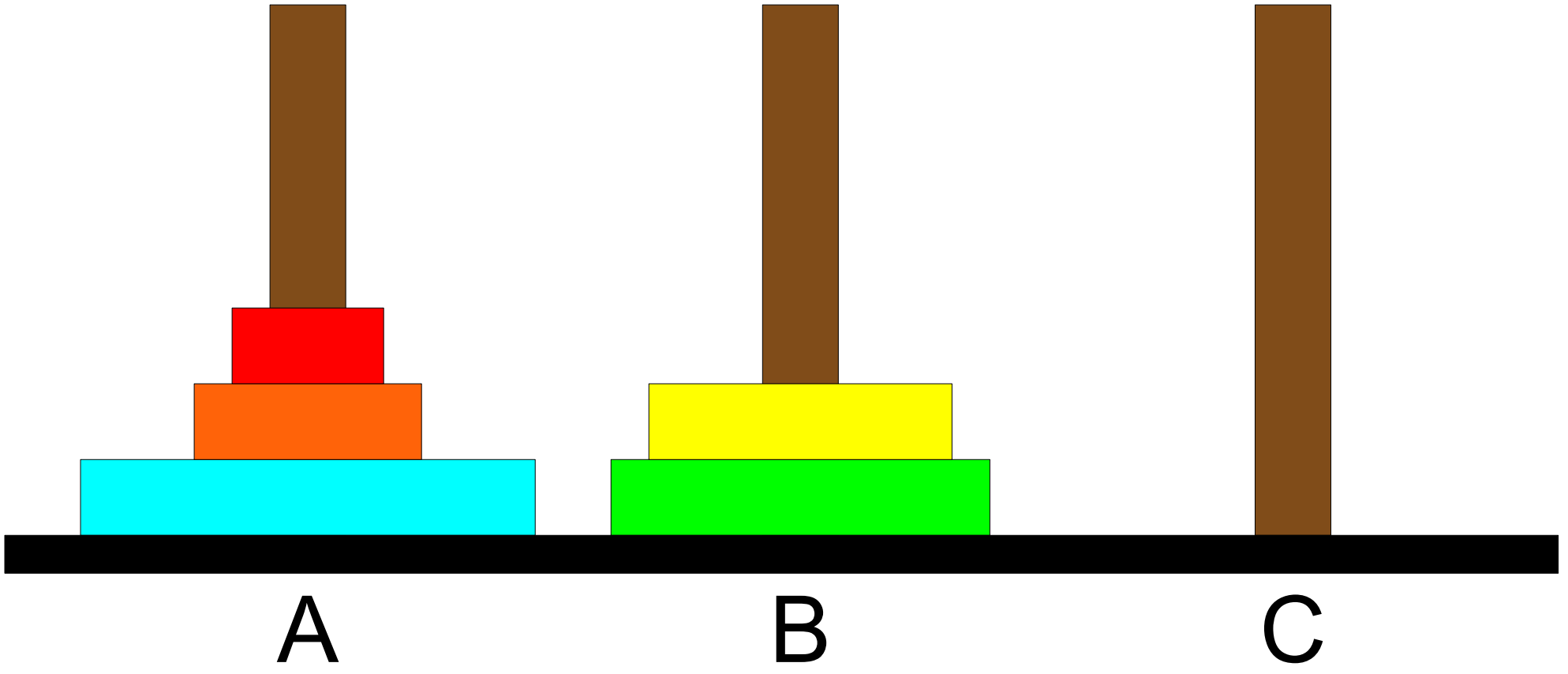
Towers of Hanoi



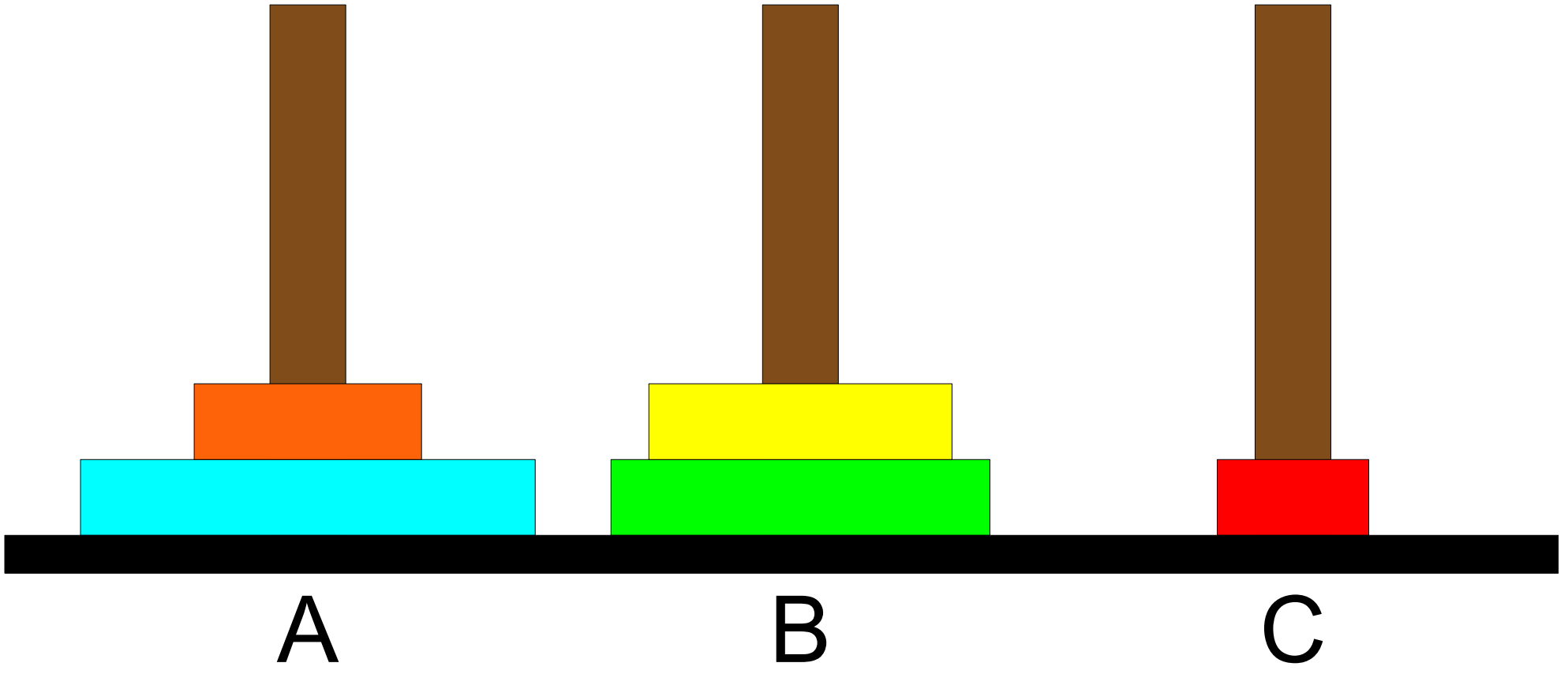
Towers of Hanoi



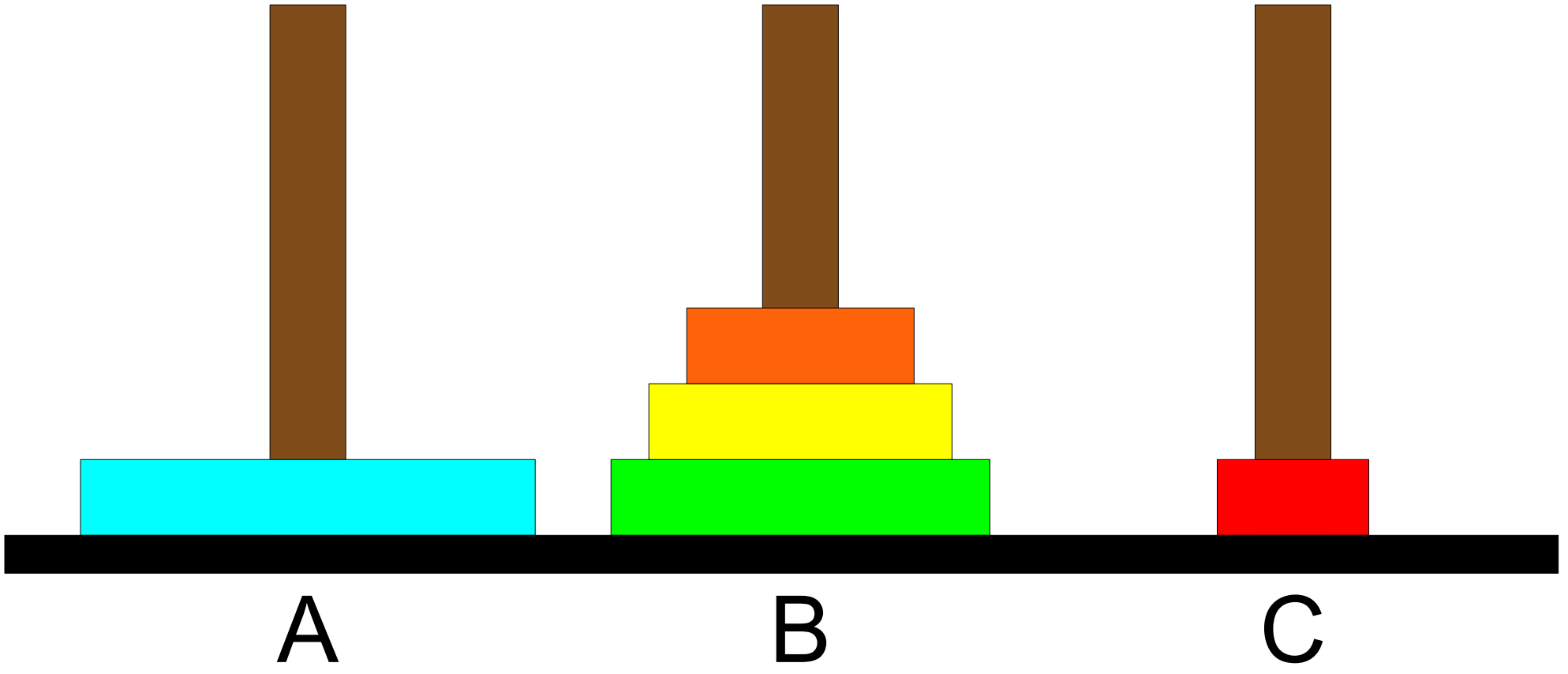
Towers of Hanoi



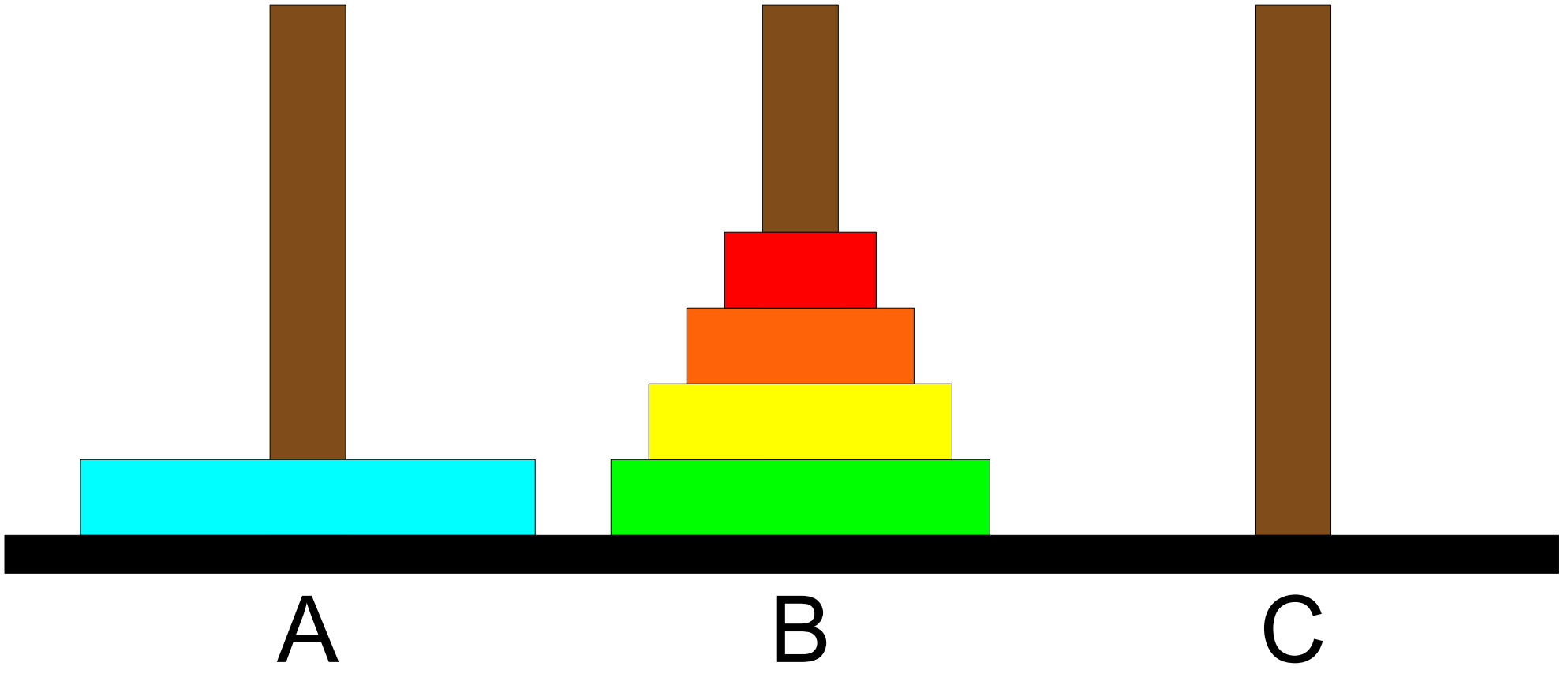
Towers of Hanoi



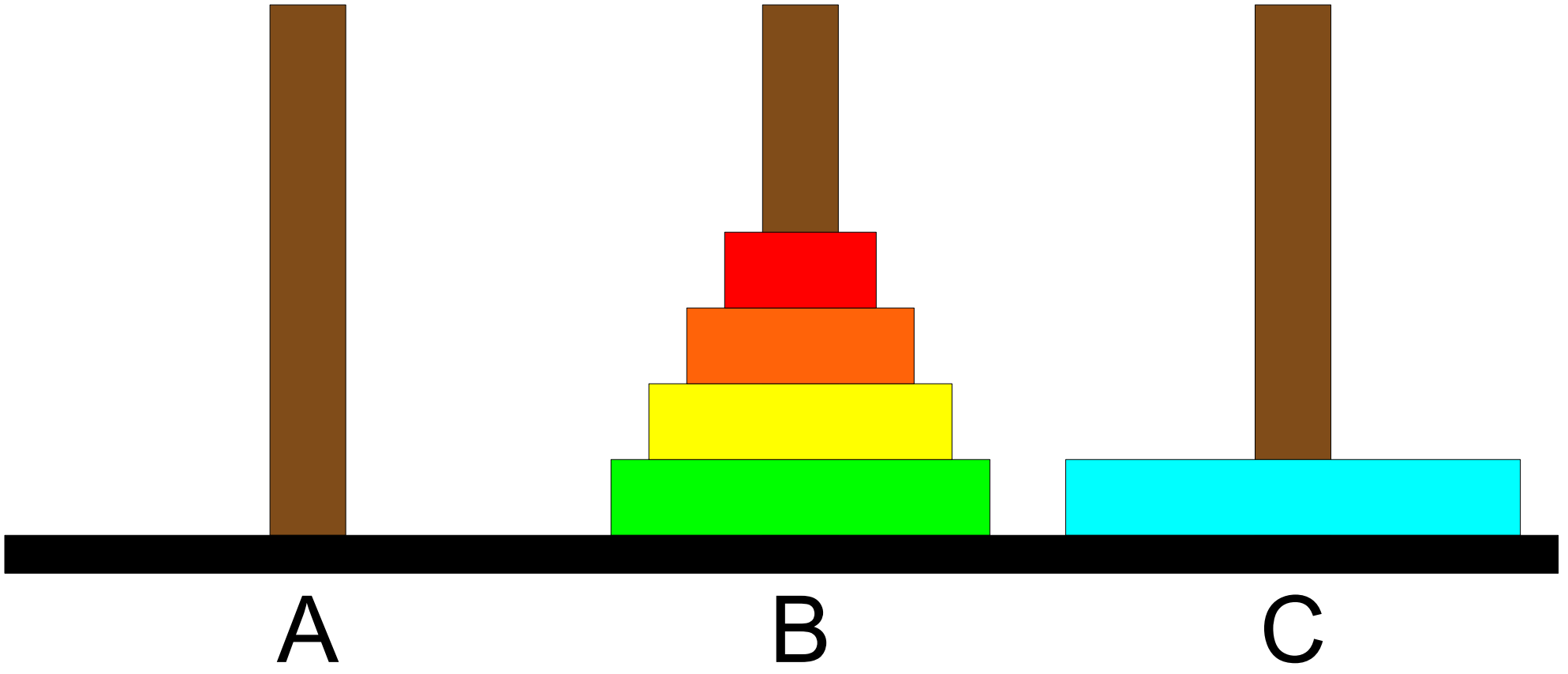
Towers of Hanoi



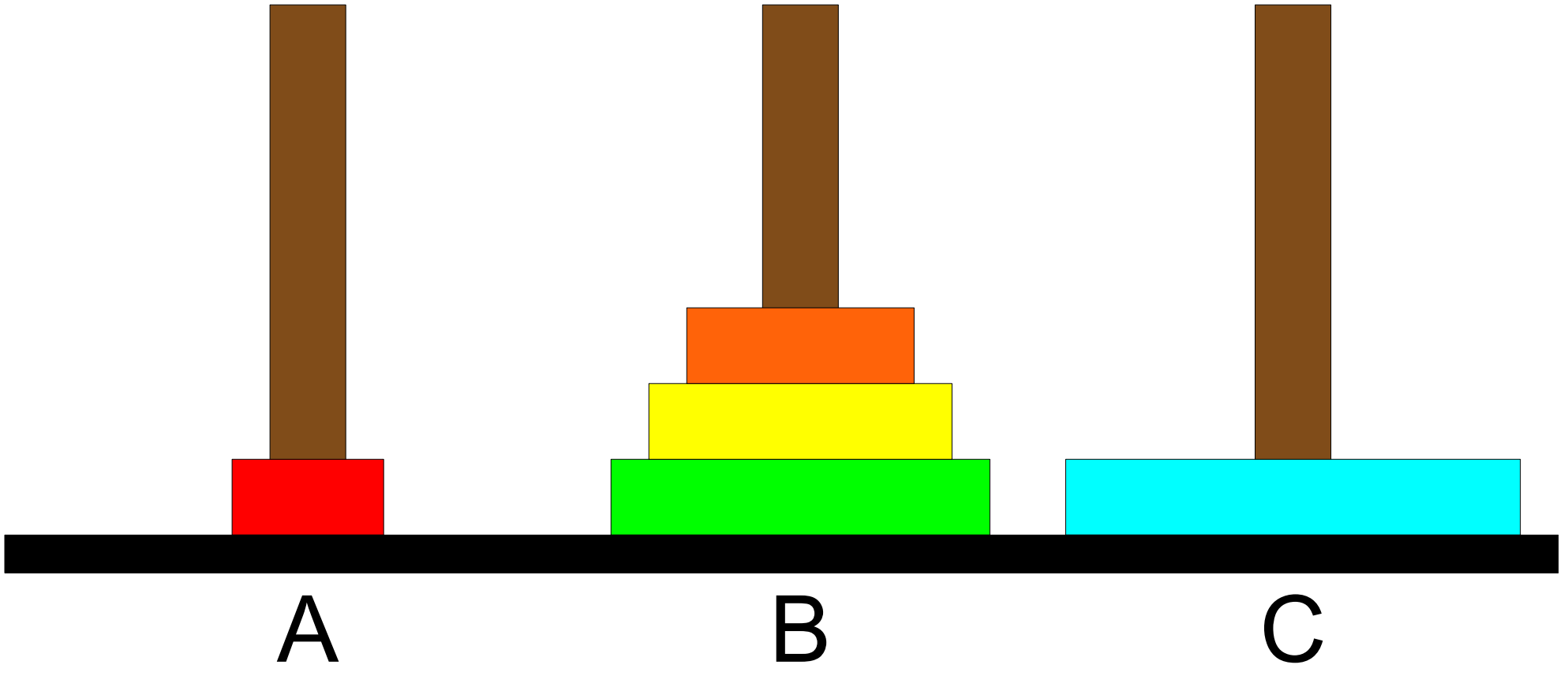
Towers of Hanoi



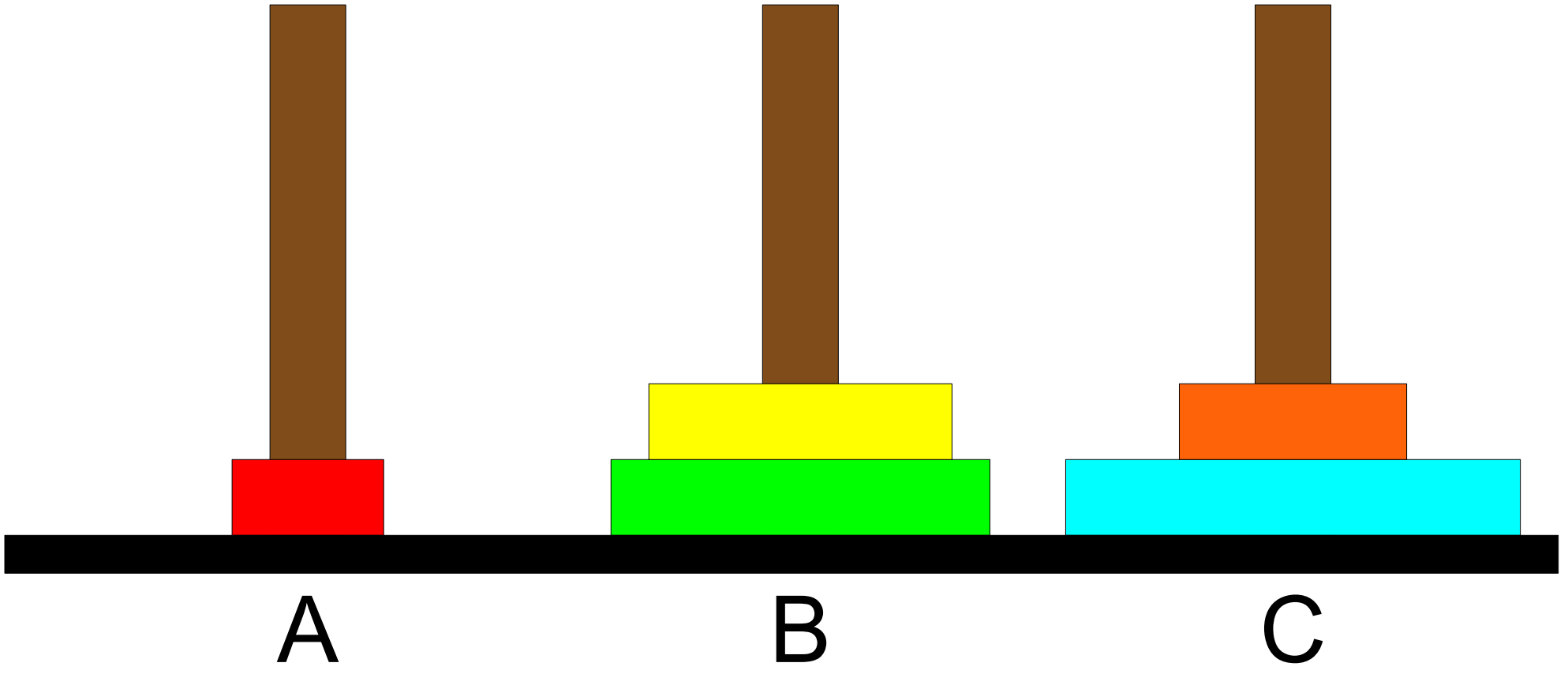
Towers of Hanoi



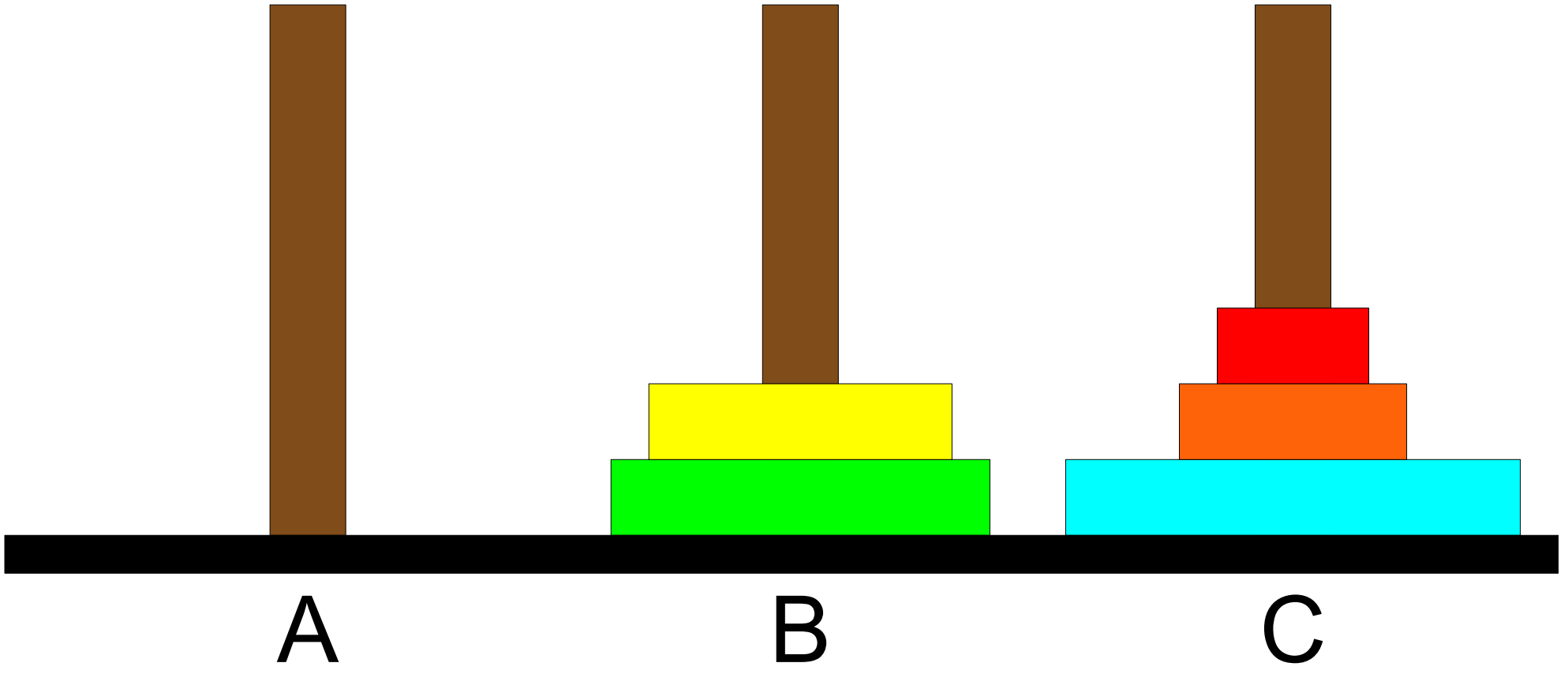
Towers of Hanoi



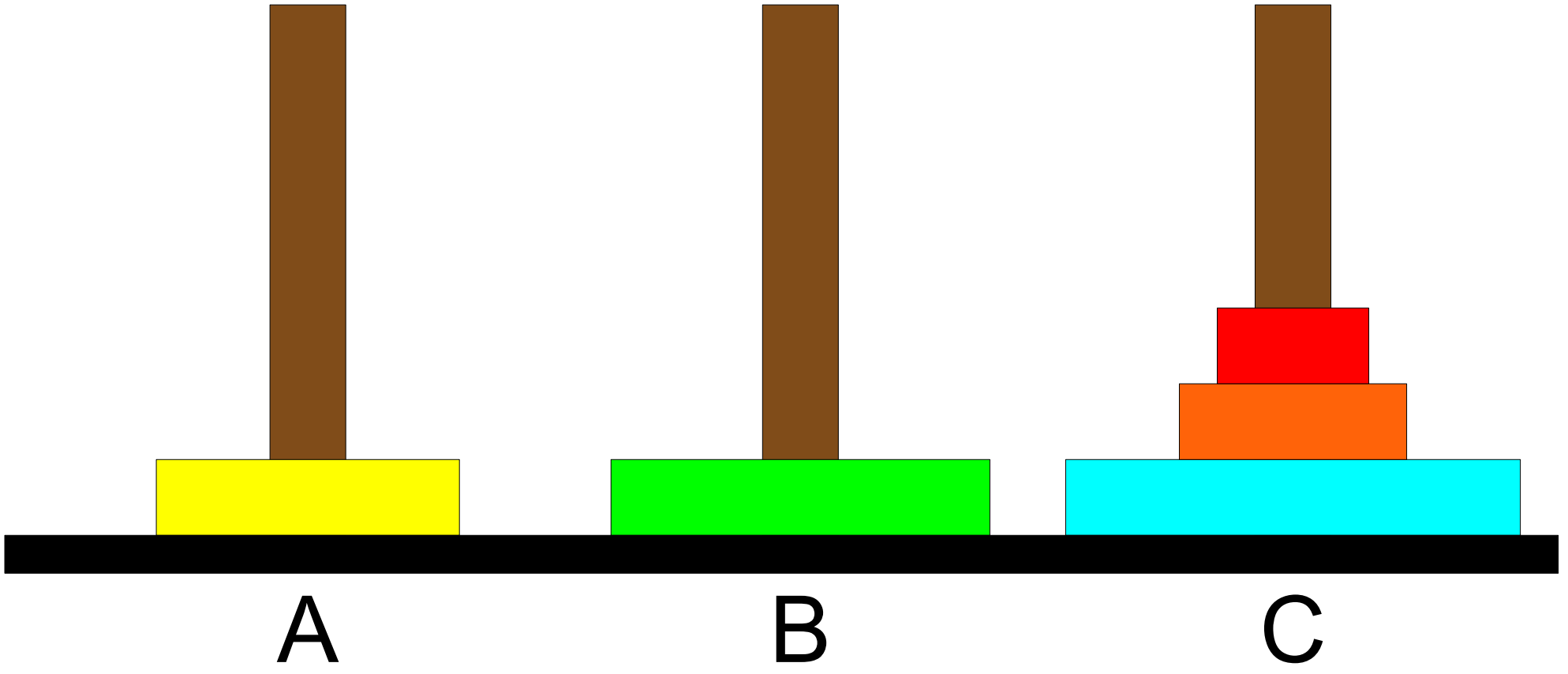
Towers of Hanoi



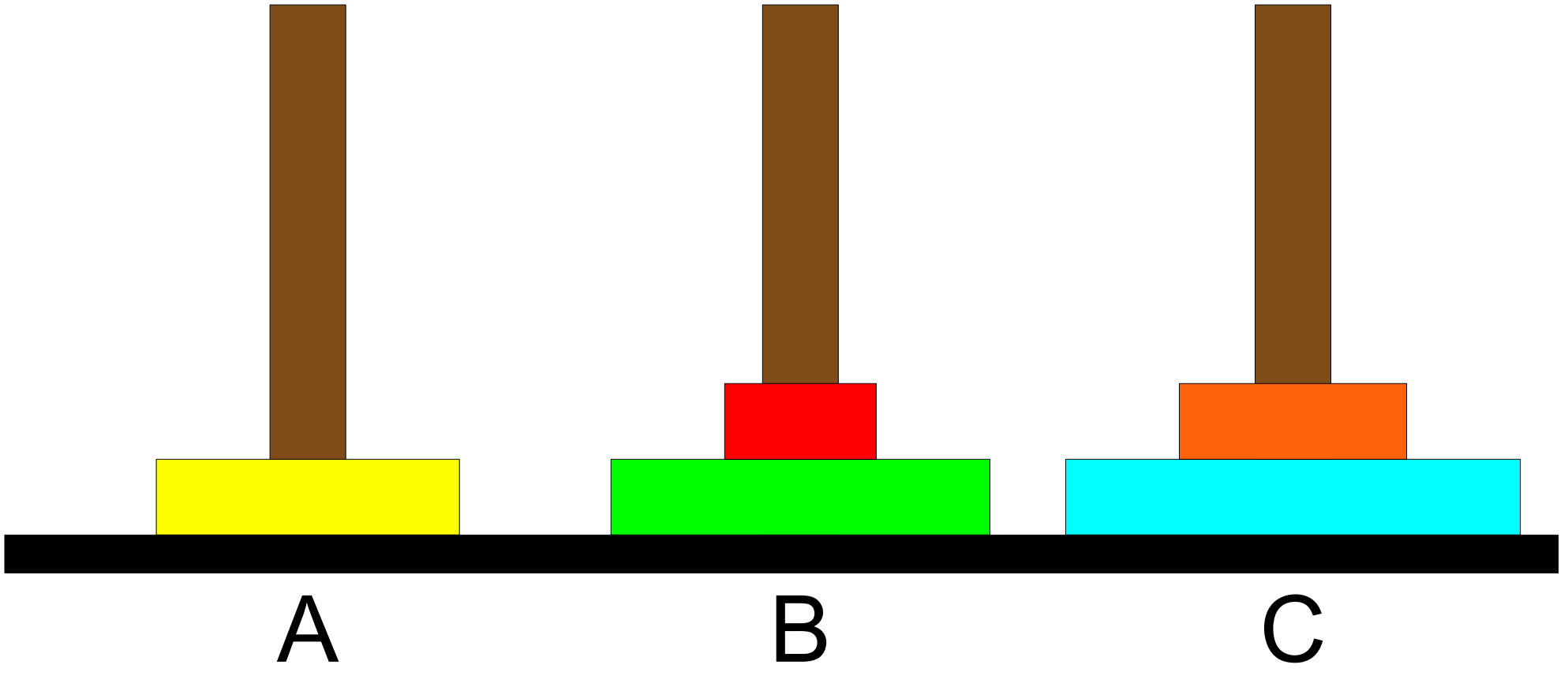
Towers of Hanoi



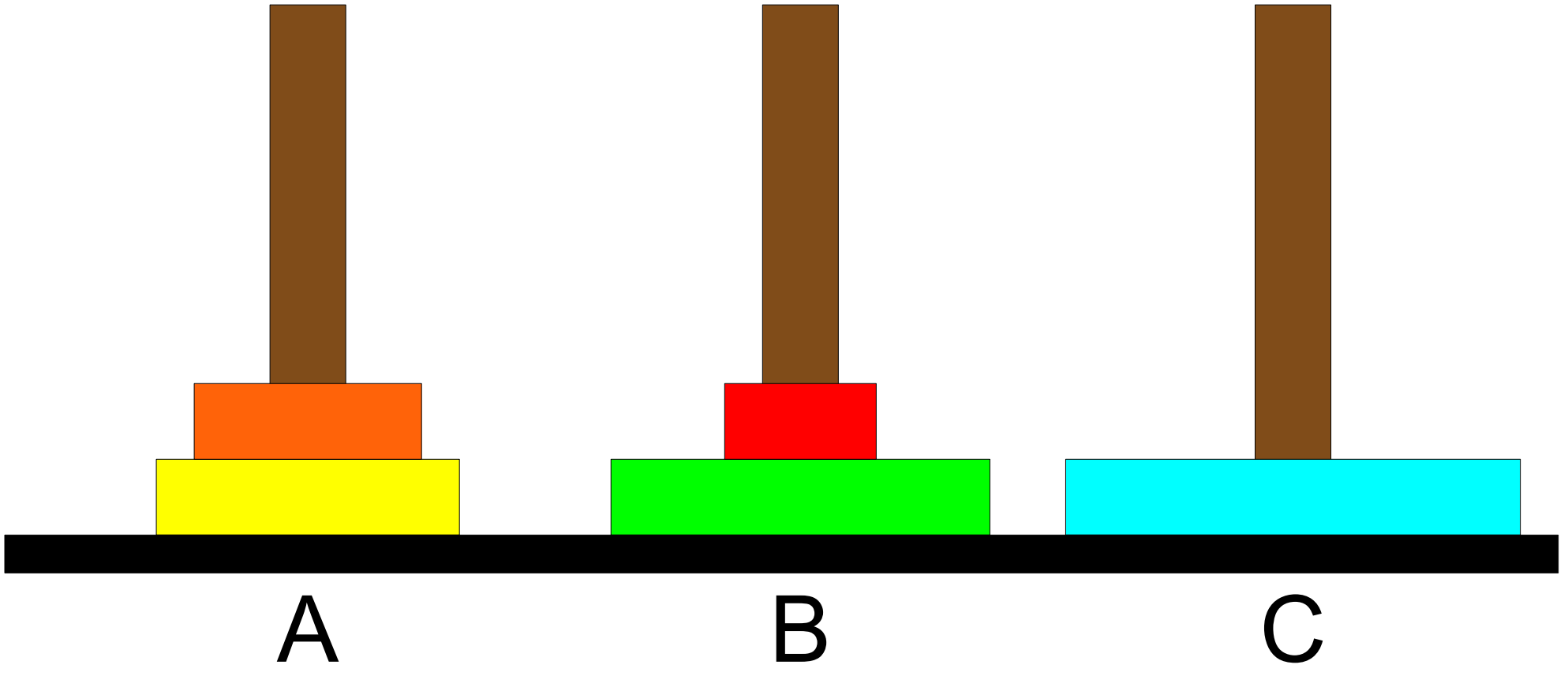
Towers of Hanoi



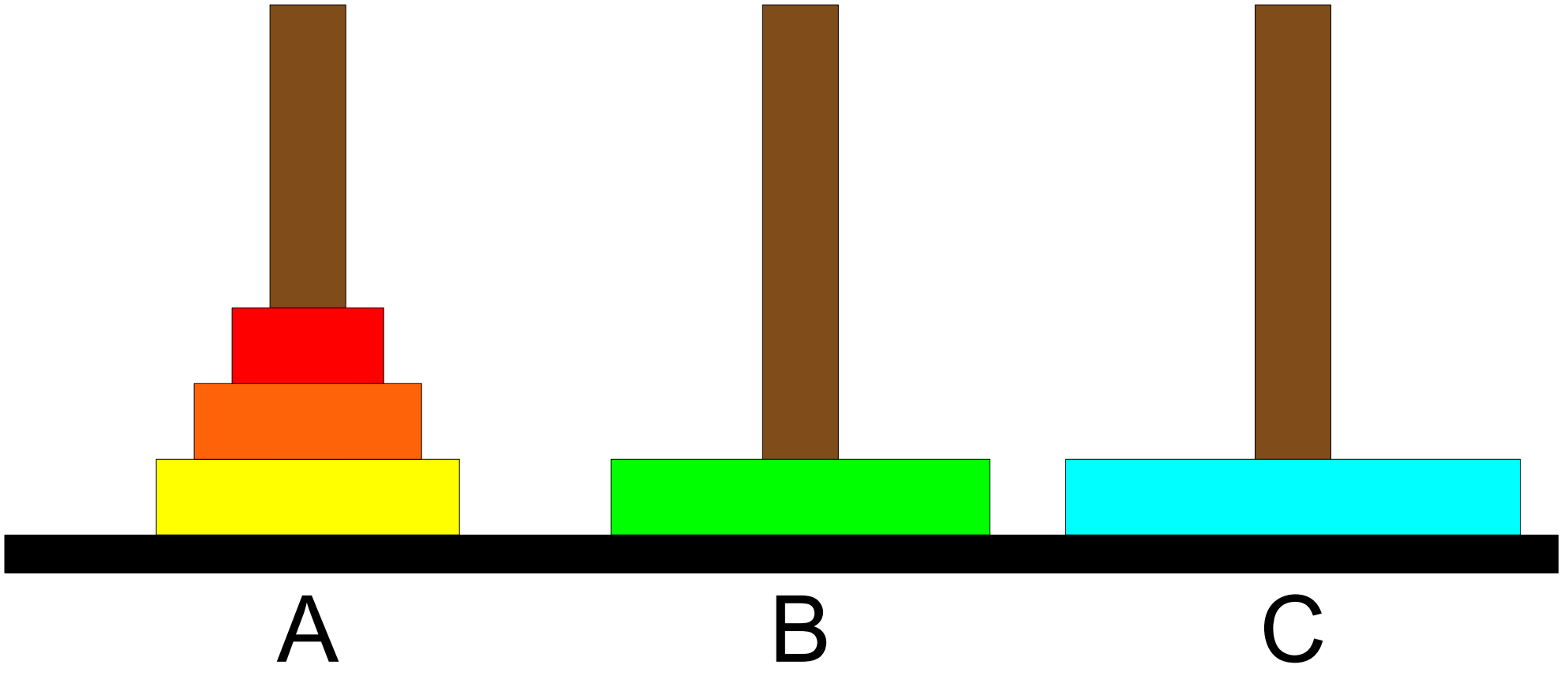
Towers of Hanoi



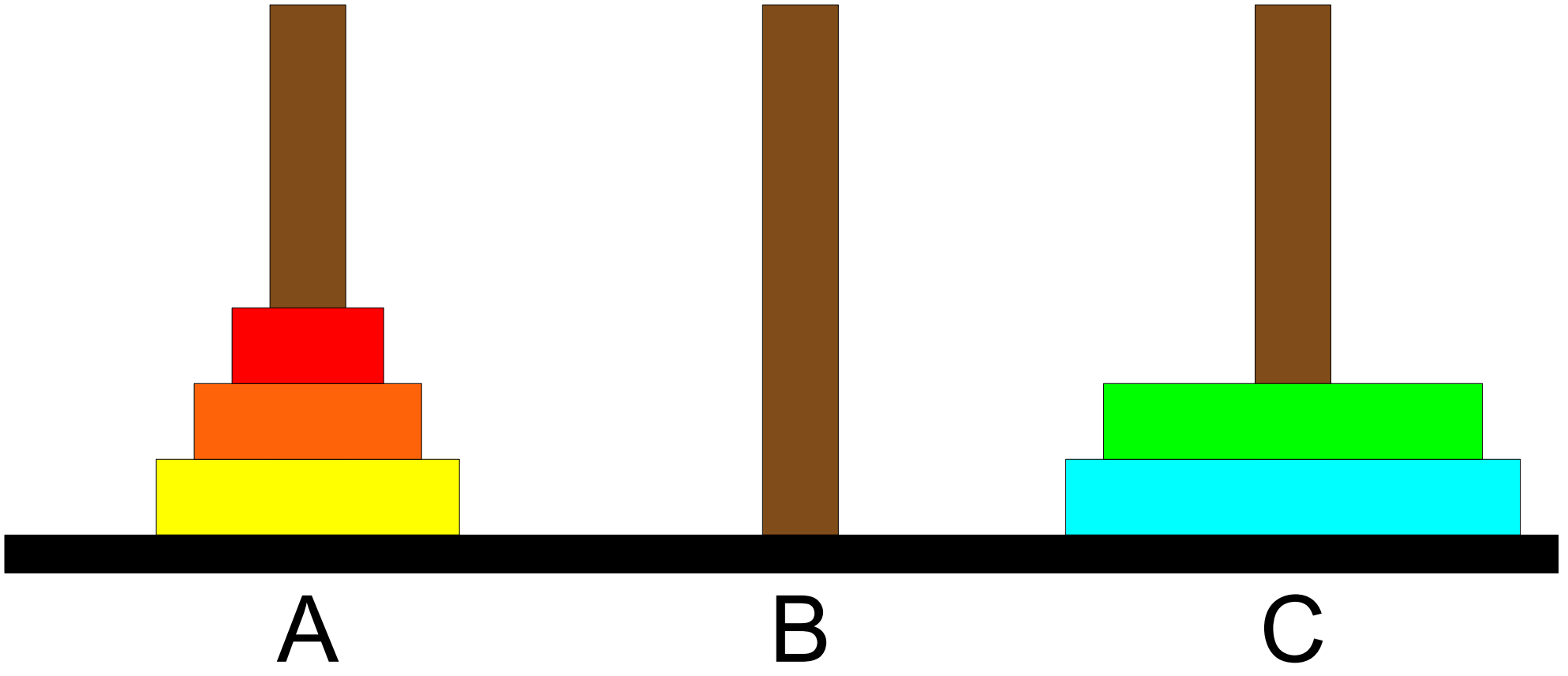
Towers of Hanoi



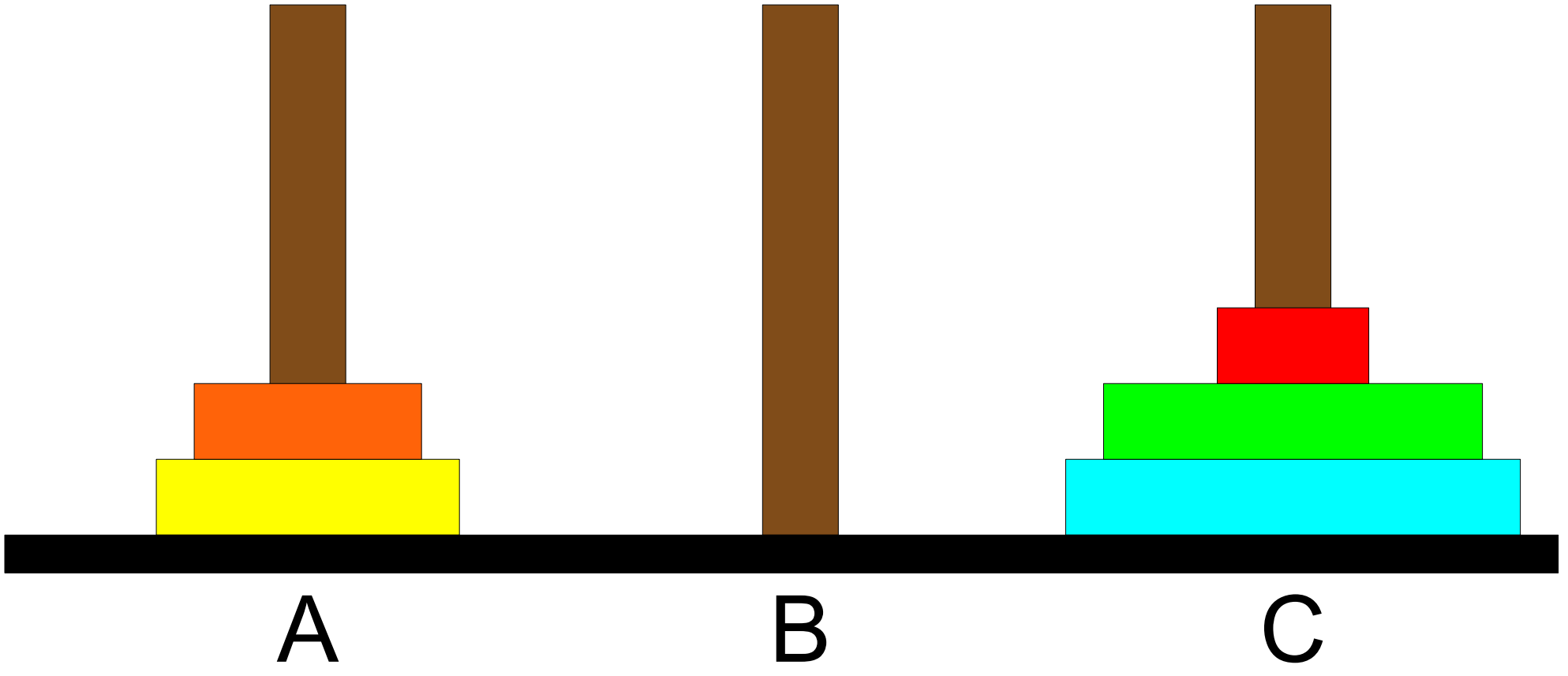
Towers of Hanoi



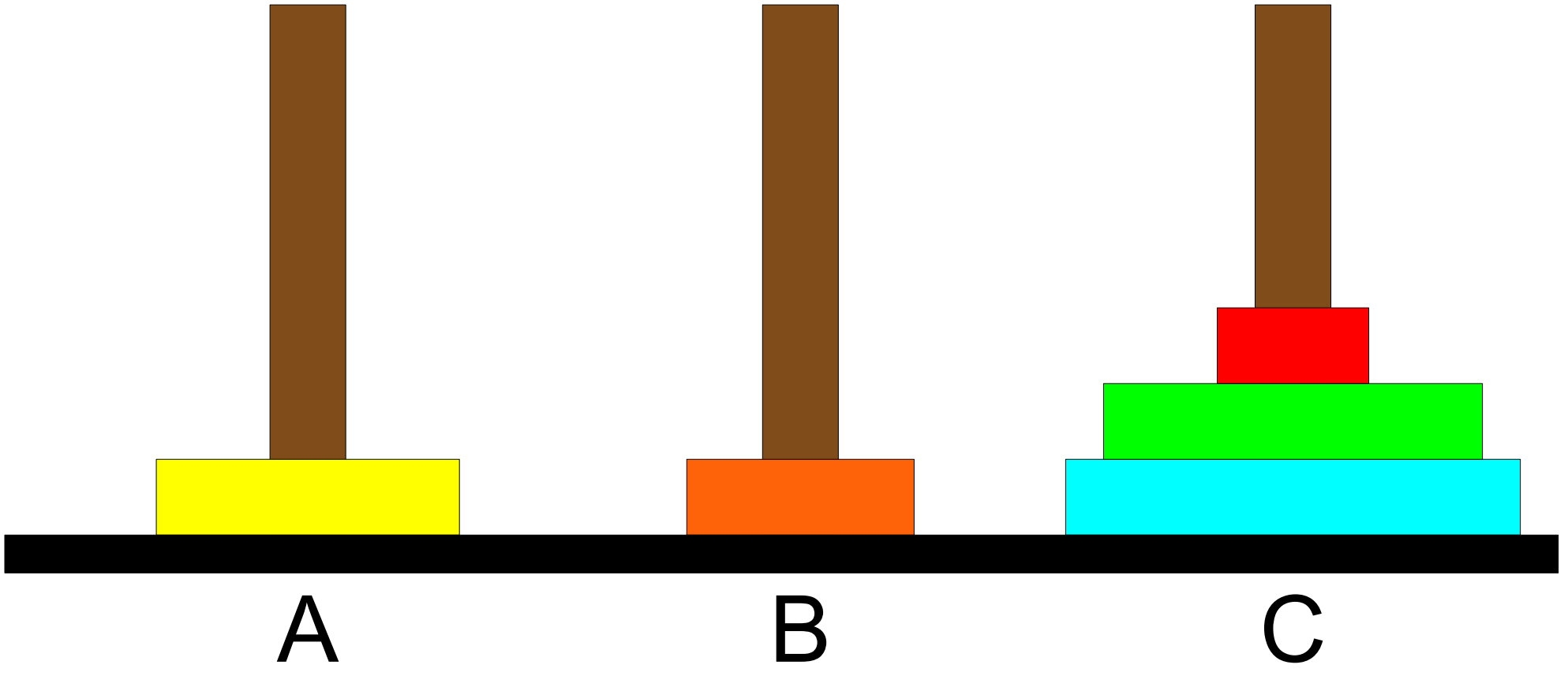
Towers of Hanoi



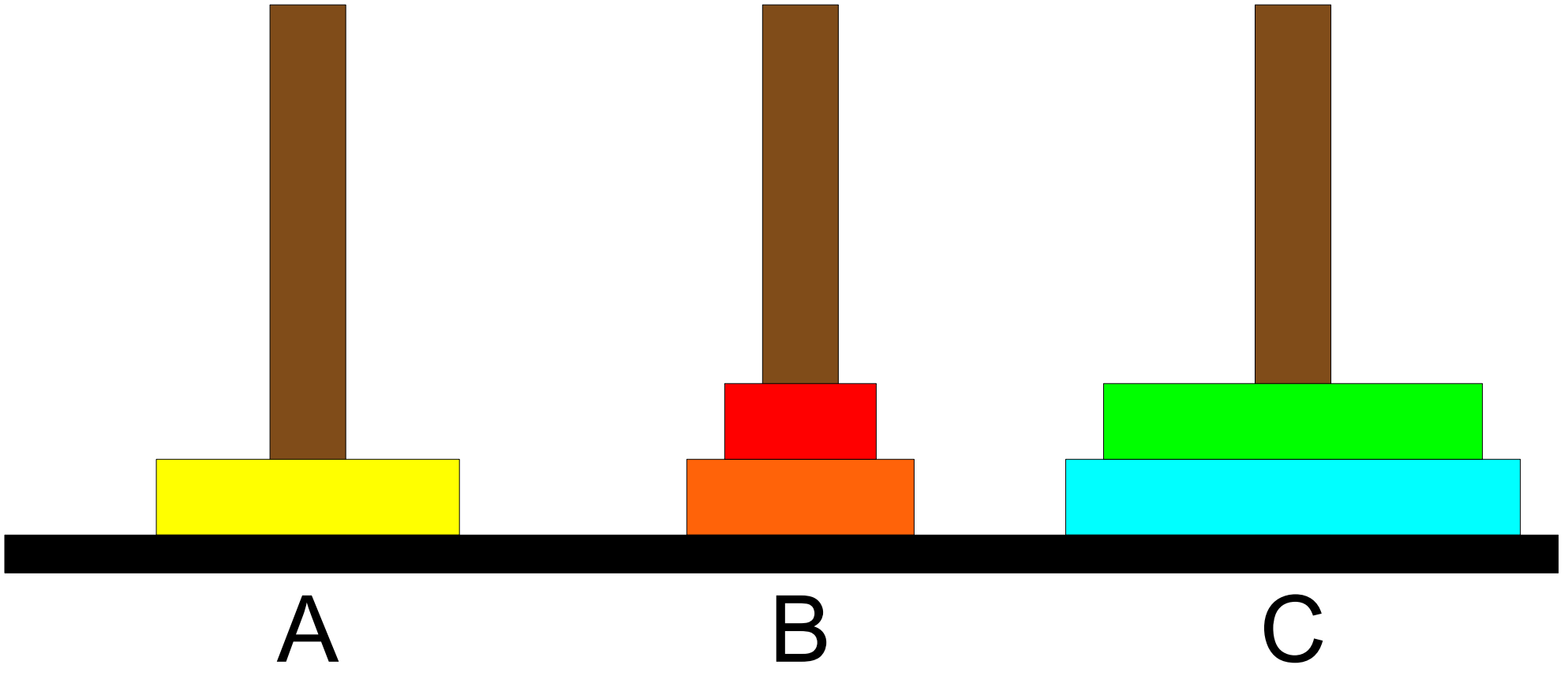
Towers of Hanoi



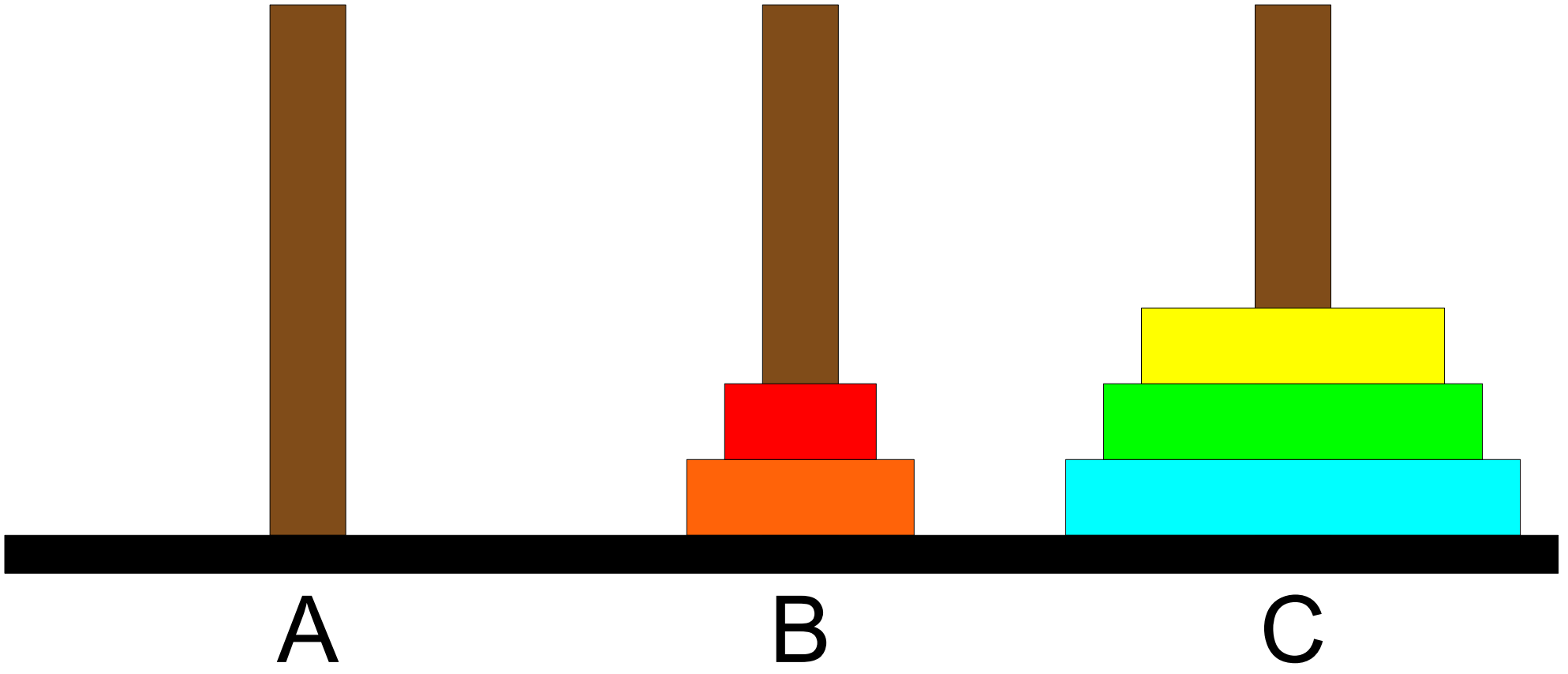
Towers of Hanoi



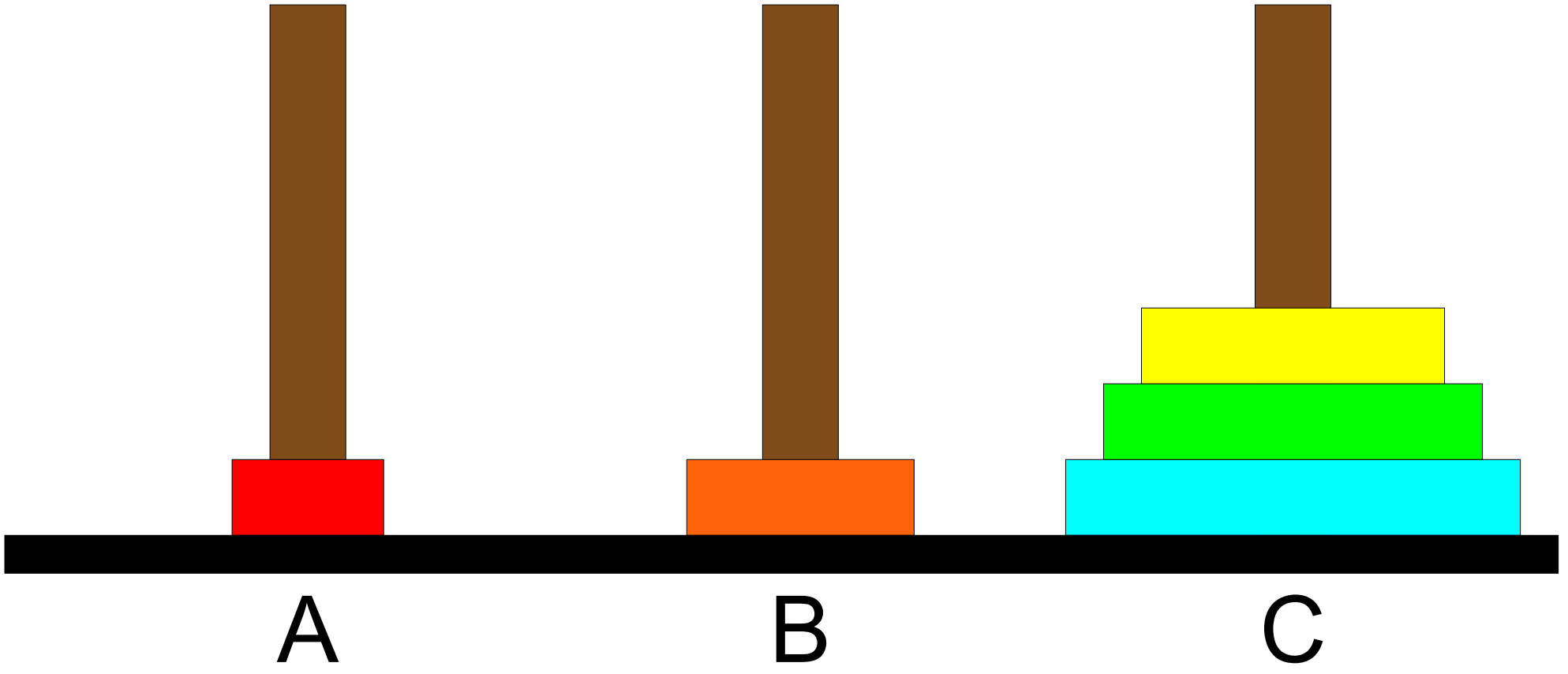
Towers of Hanoi



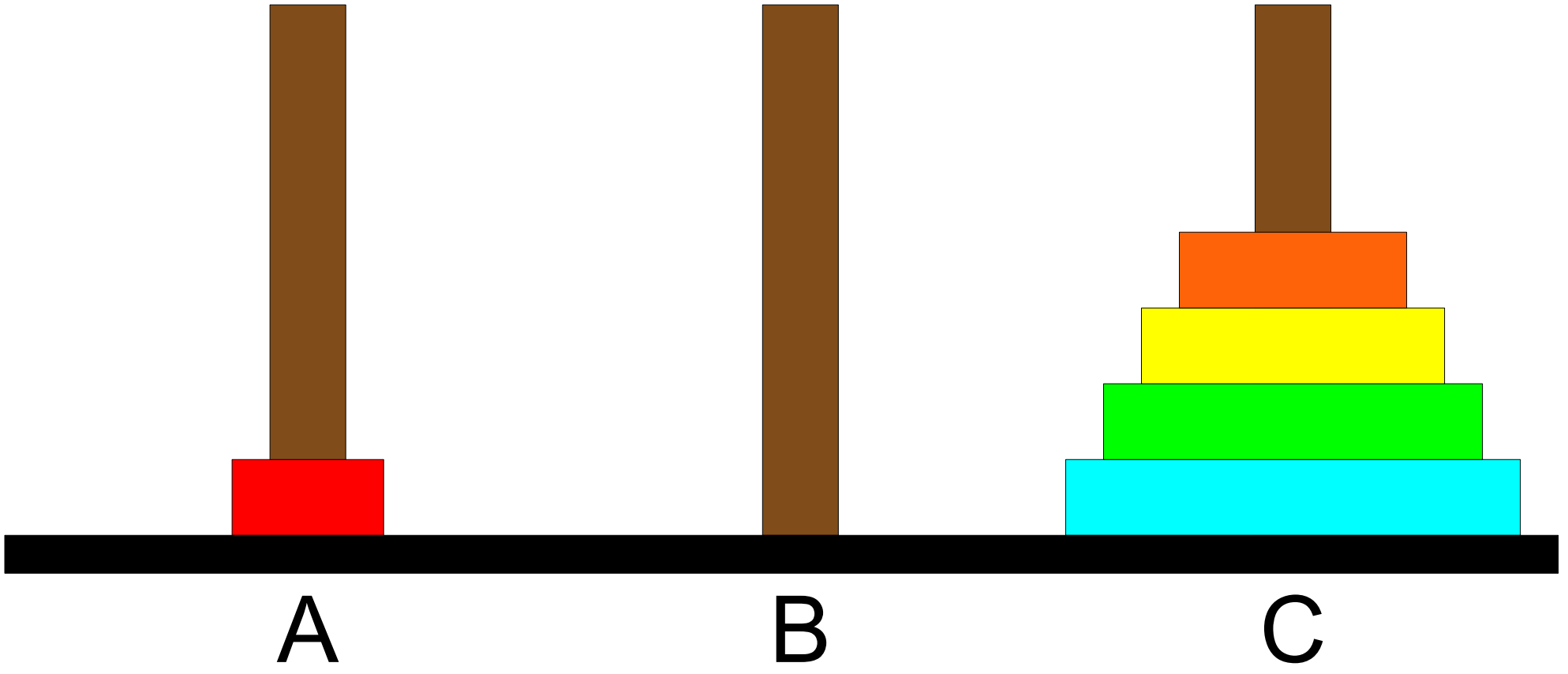
Towers of Hanoi



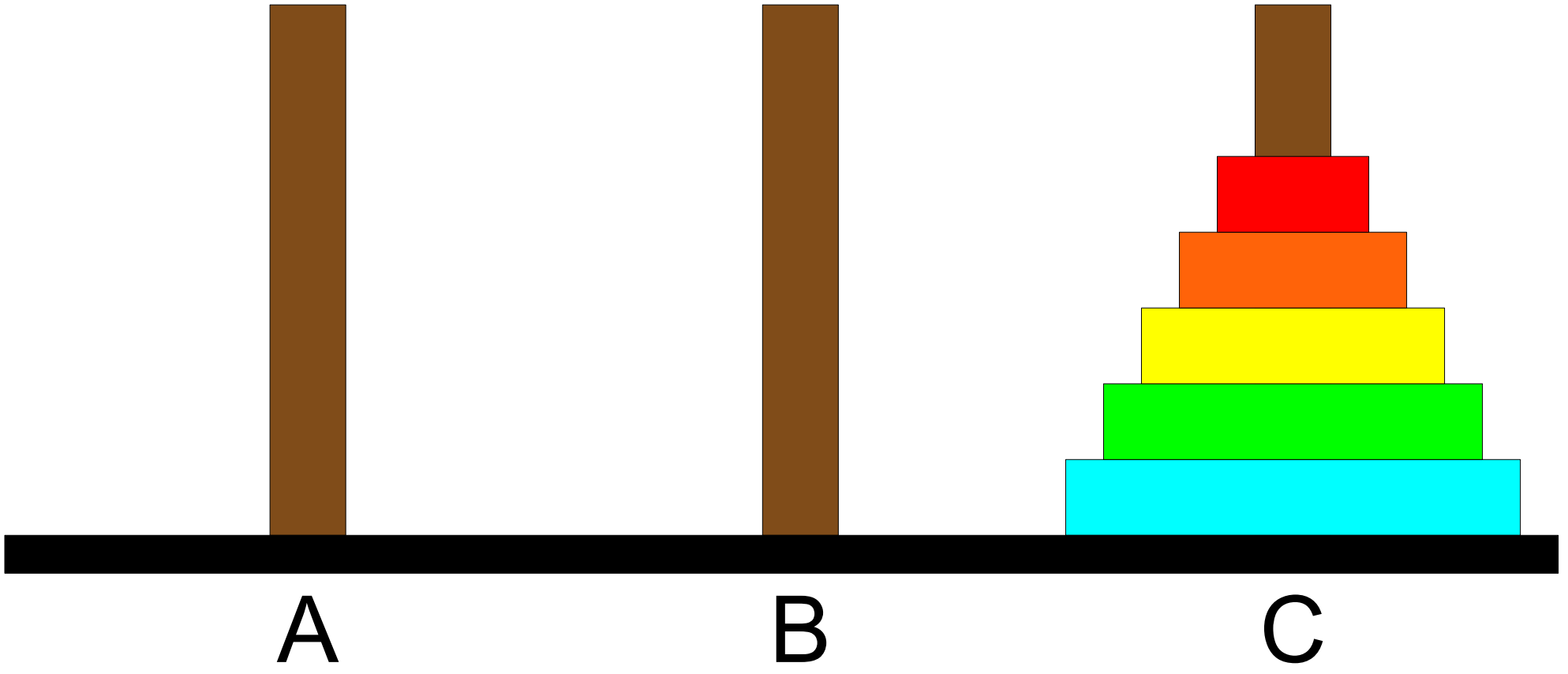
Towers of Hanoi



Towers of Hanoi

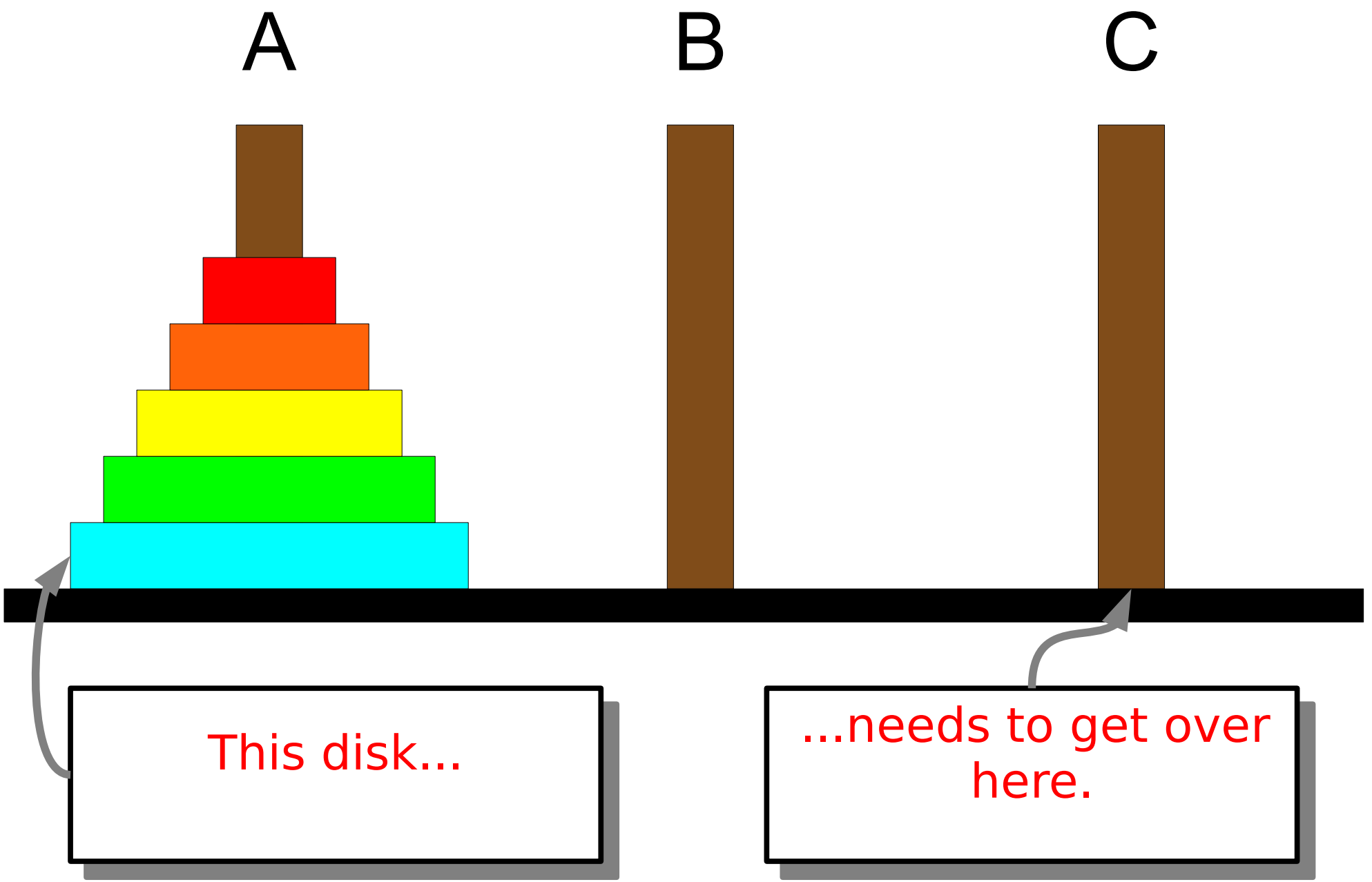


Towers of Hanoi

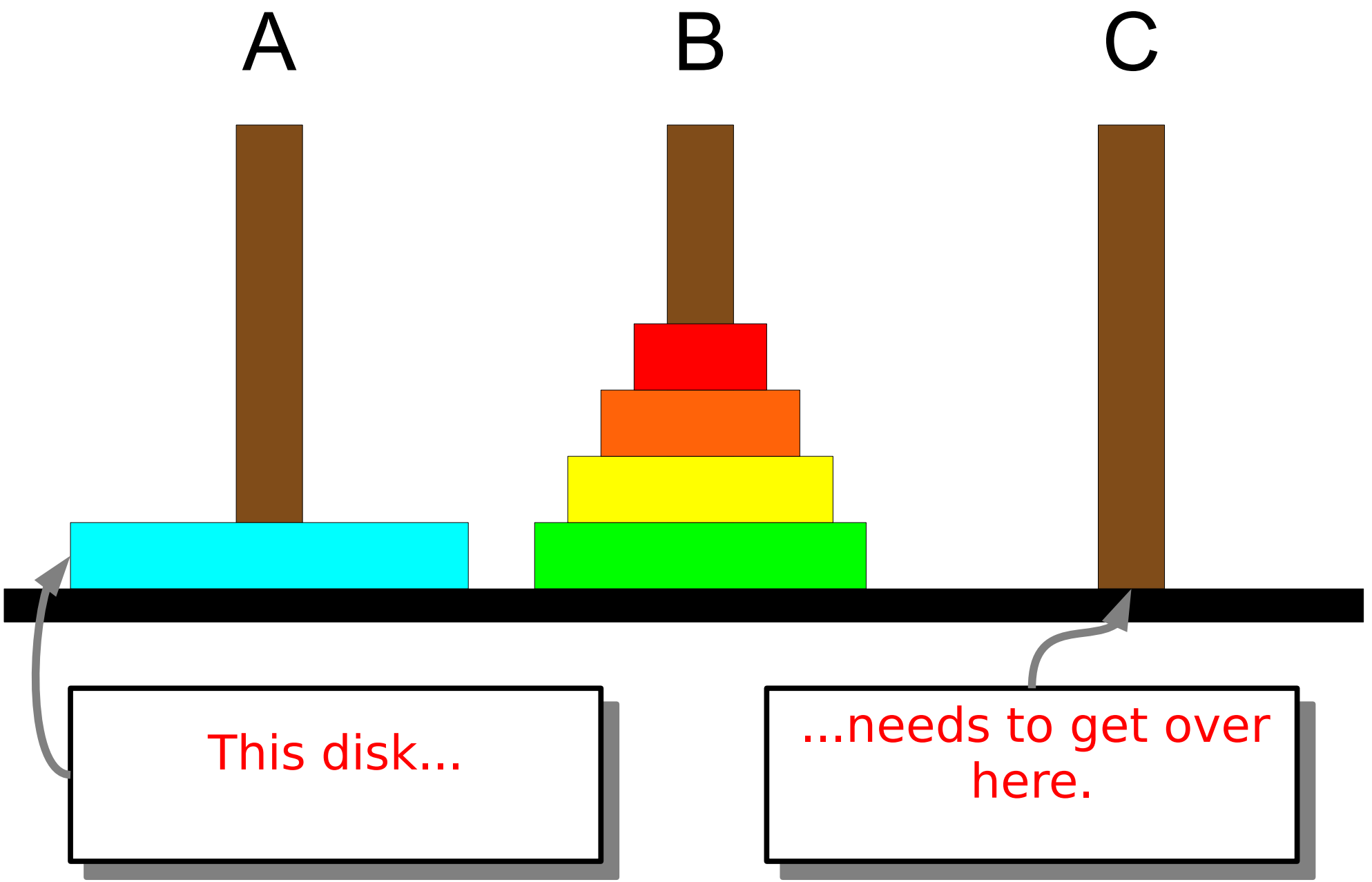


Solving the Towers of Hanoi

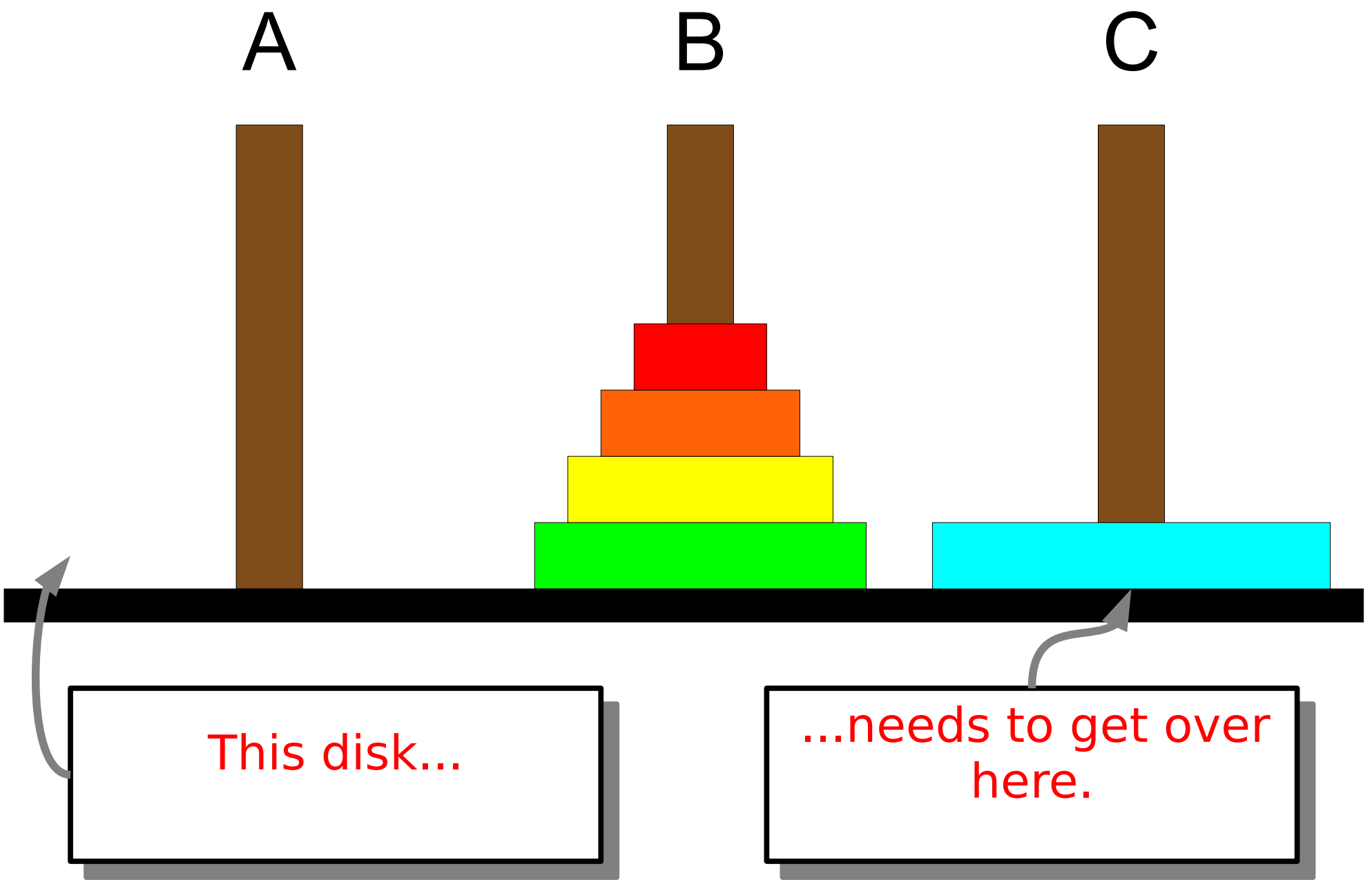
Solving the Towers of Hanoi



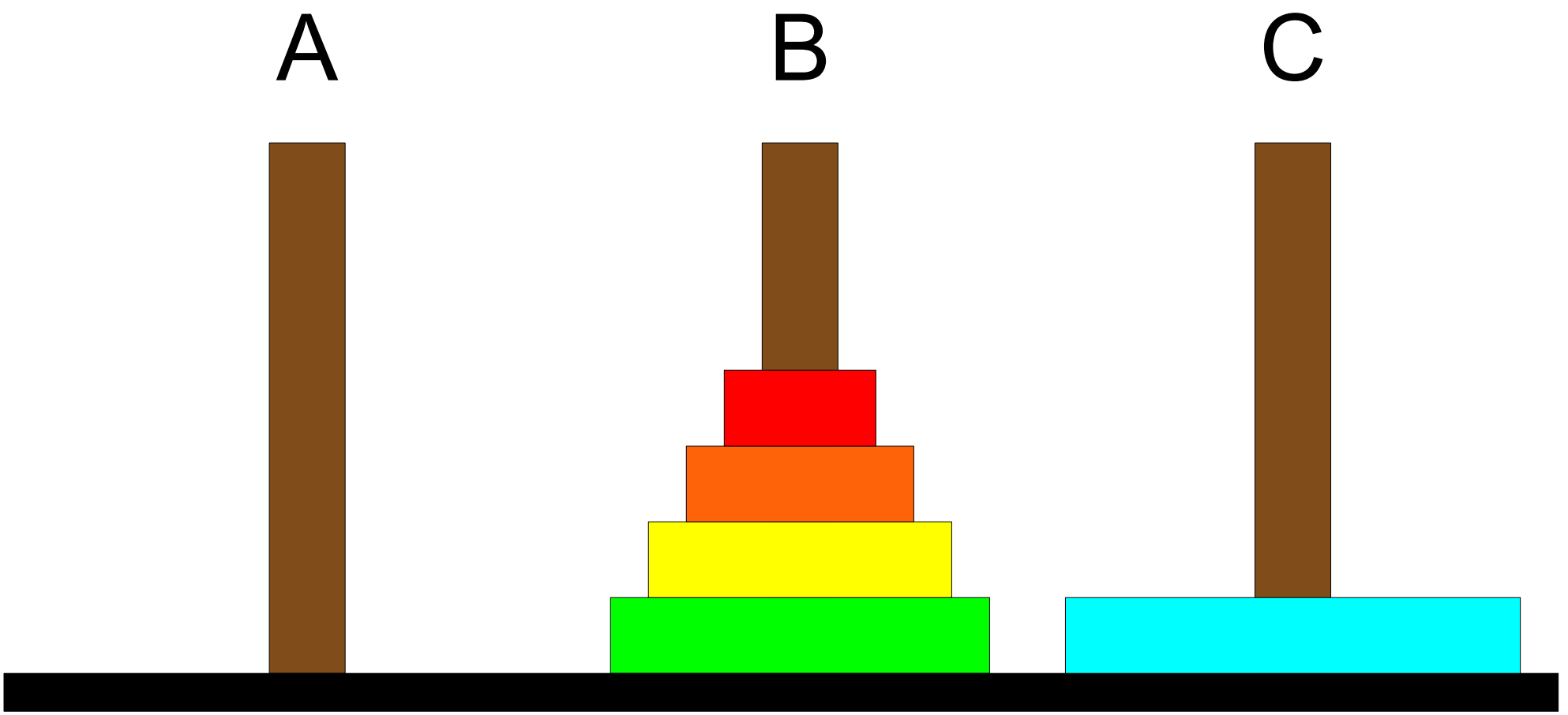
Solving the Towers of Hanoi



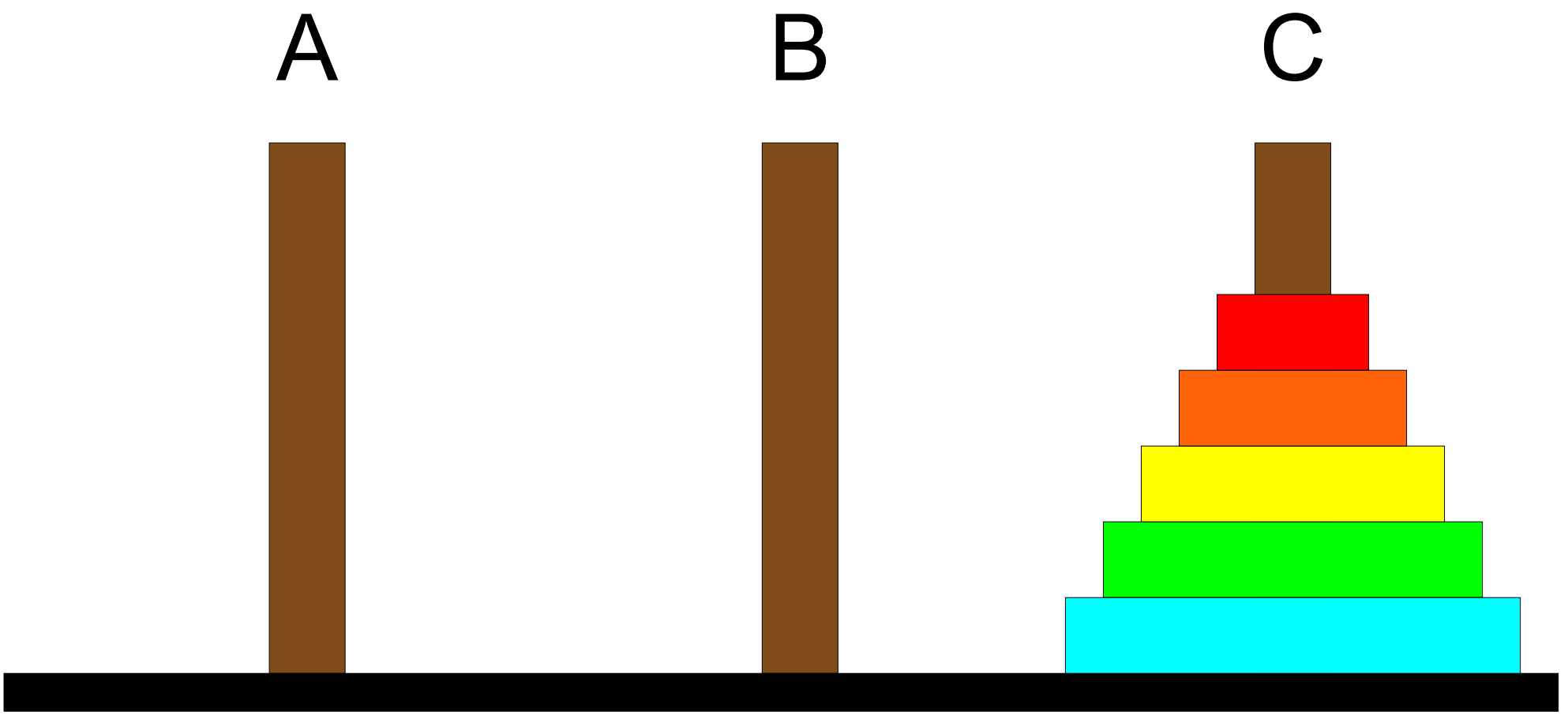
Solving the Towers of Hanoi



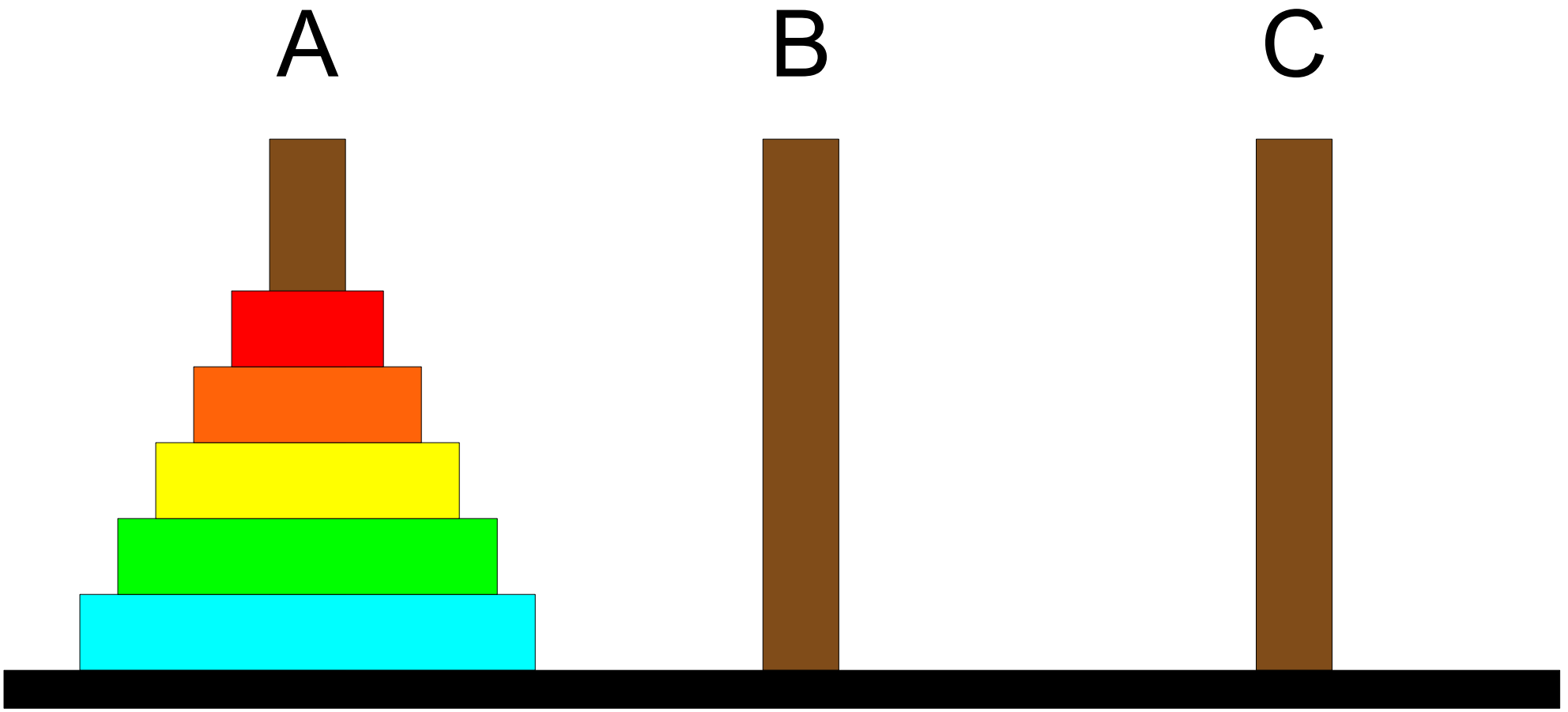
Solving the Towers of Hanoi



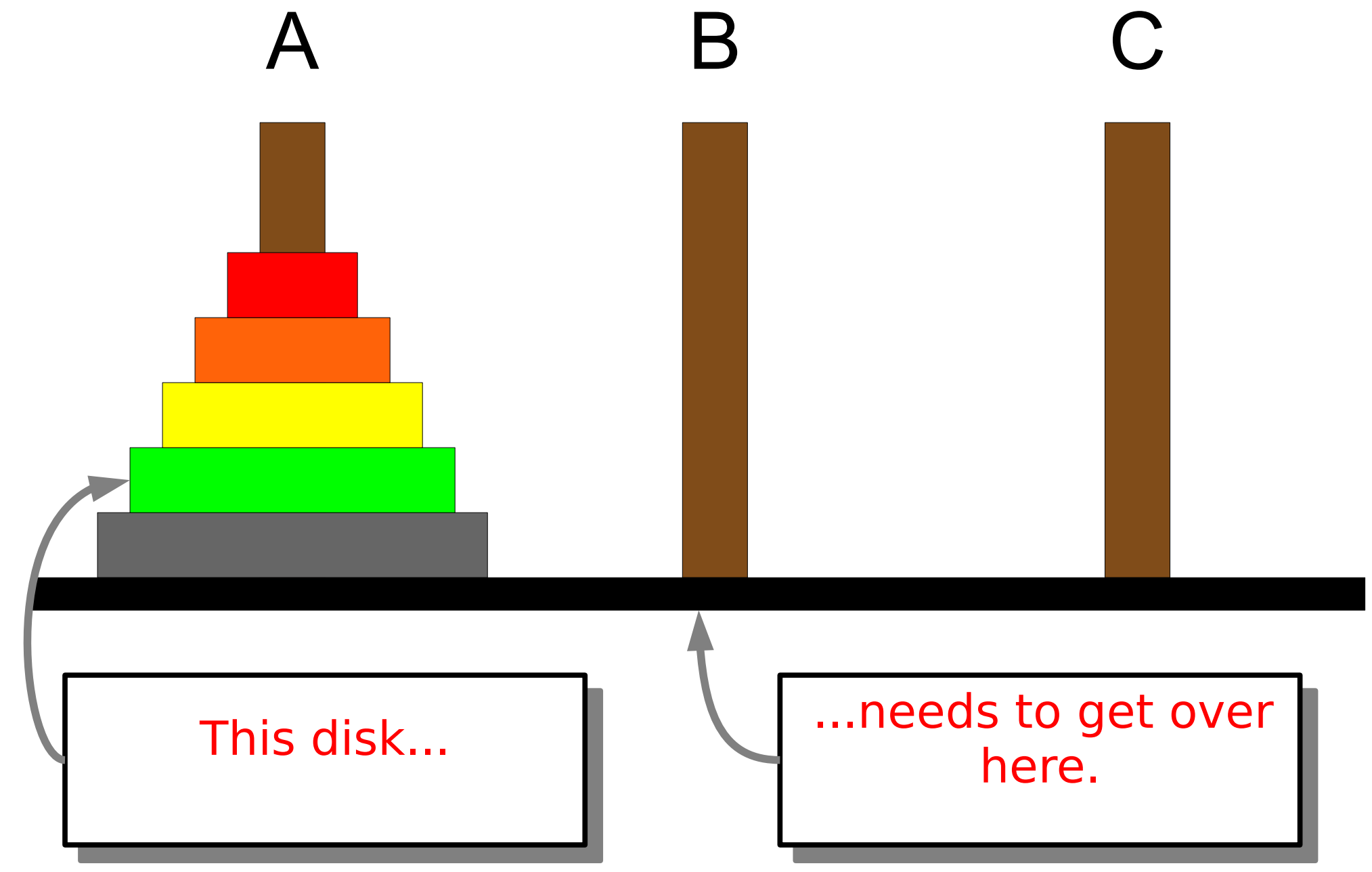
Solving the Towers of Hanoi



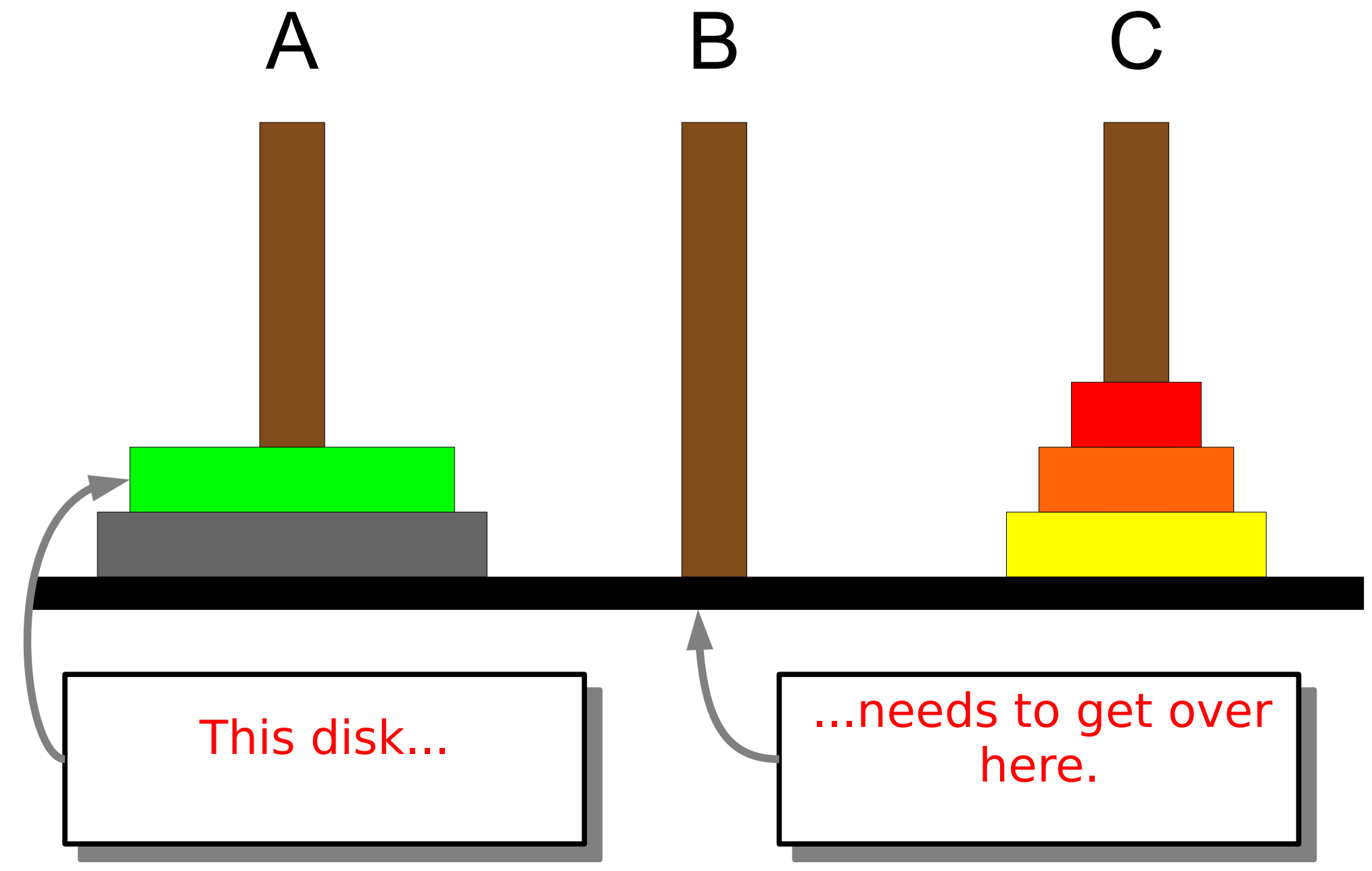
Solving the Towers of Hanoi



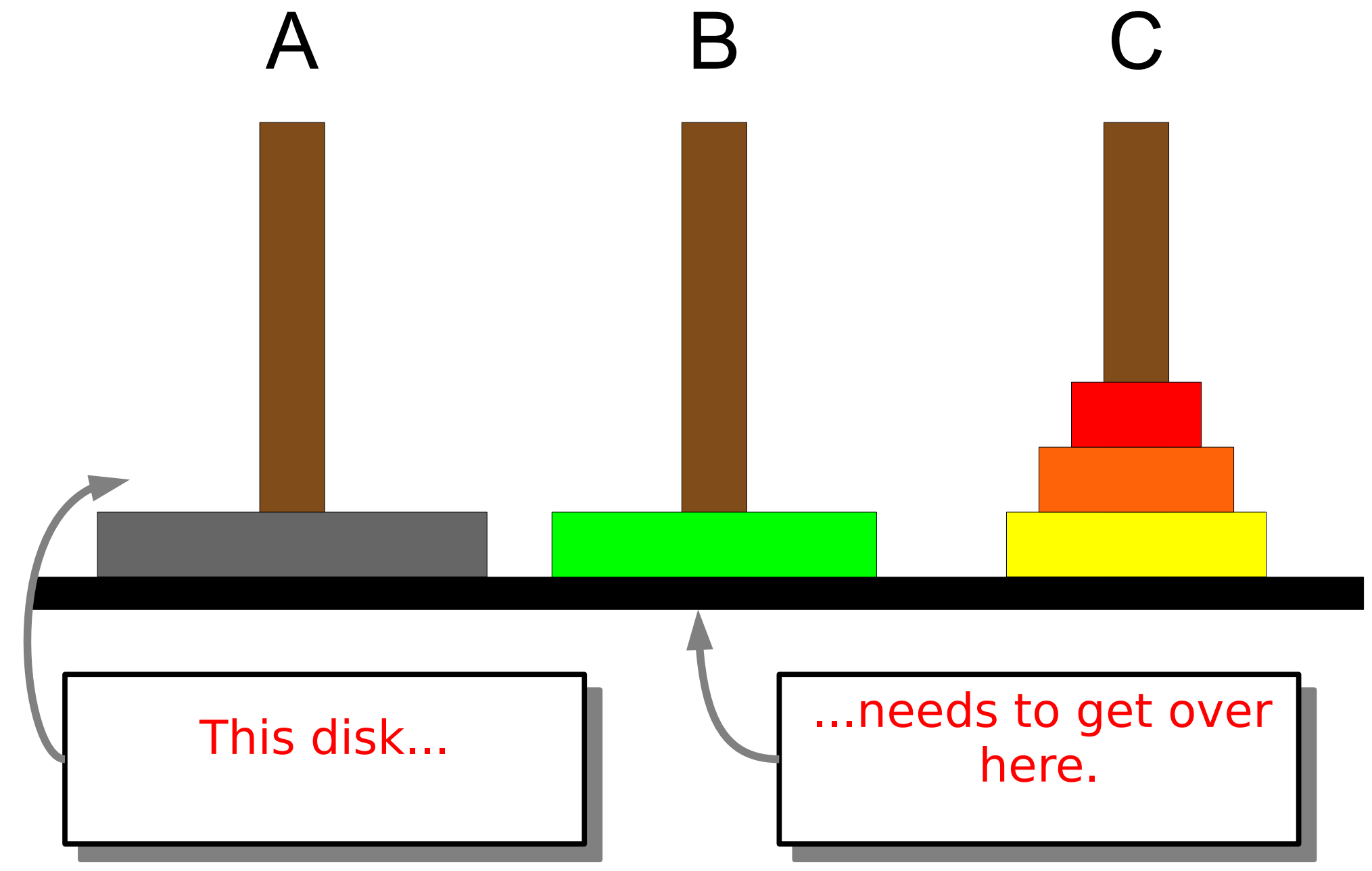
Solving the Towers of Hanoi



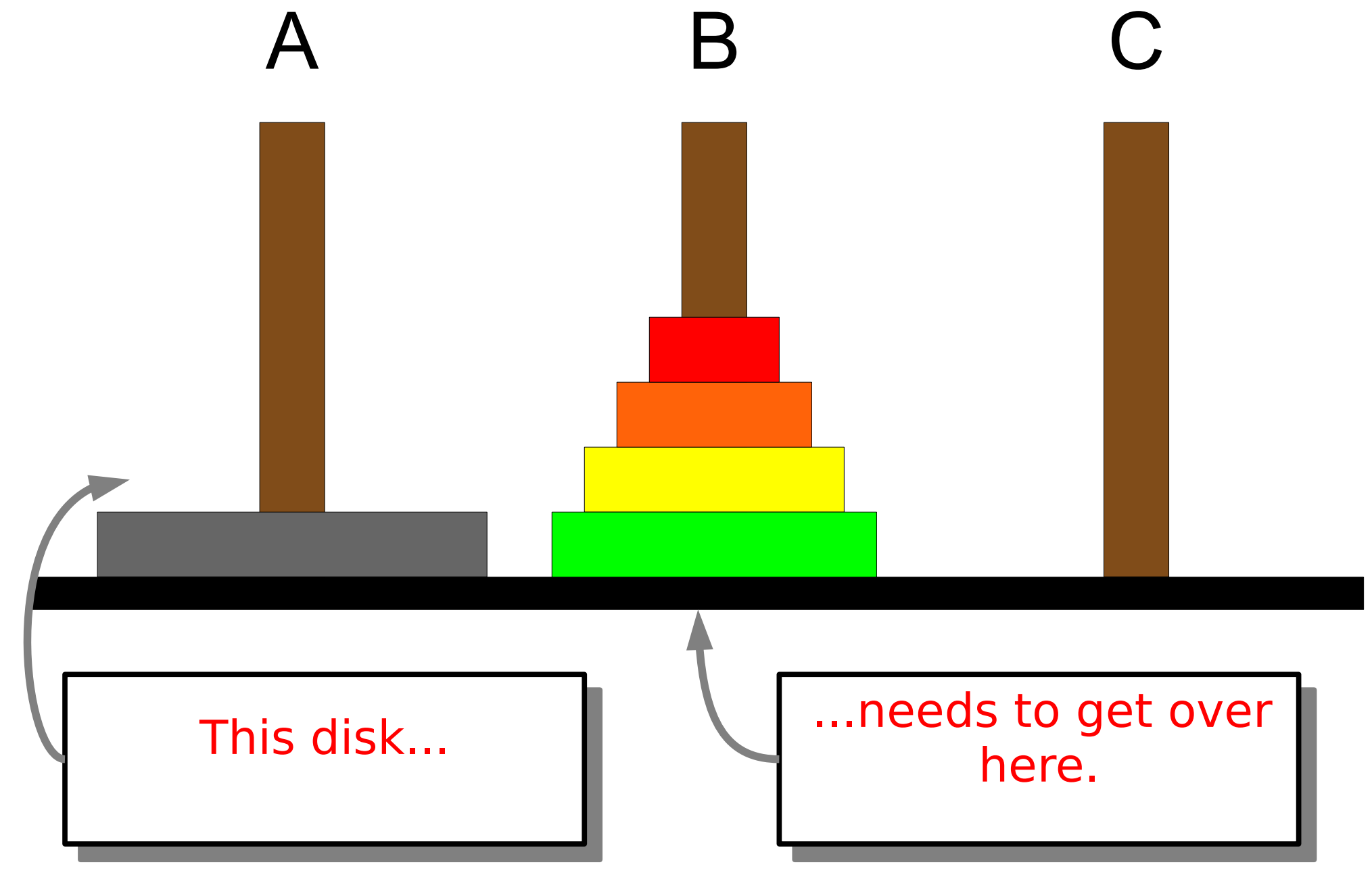
Solving the Towers of Hanoi



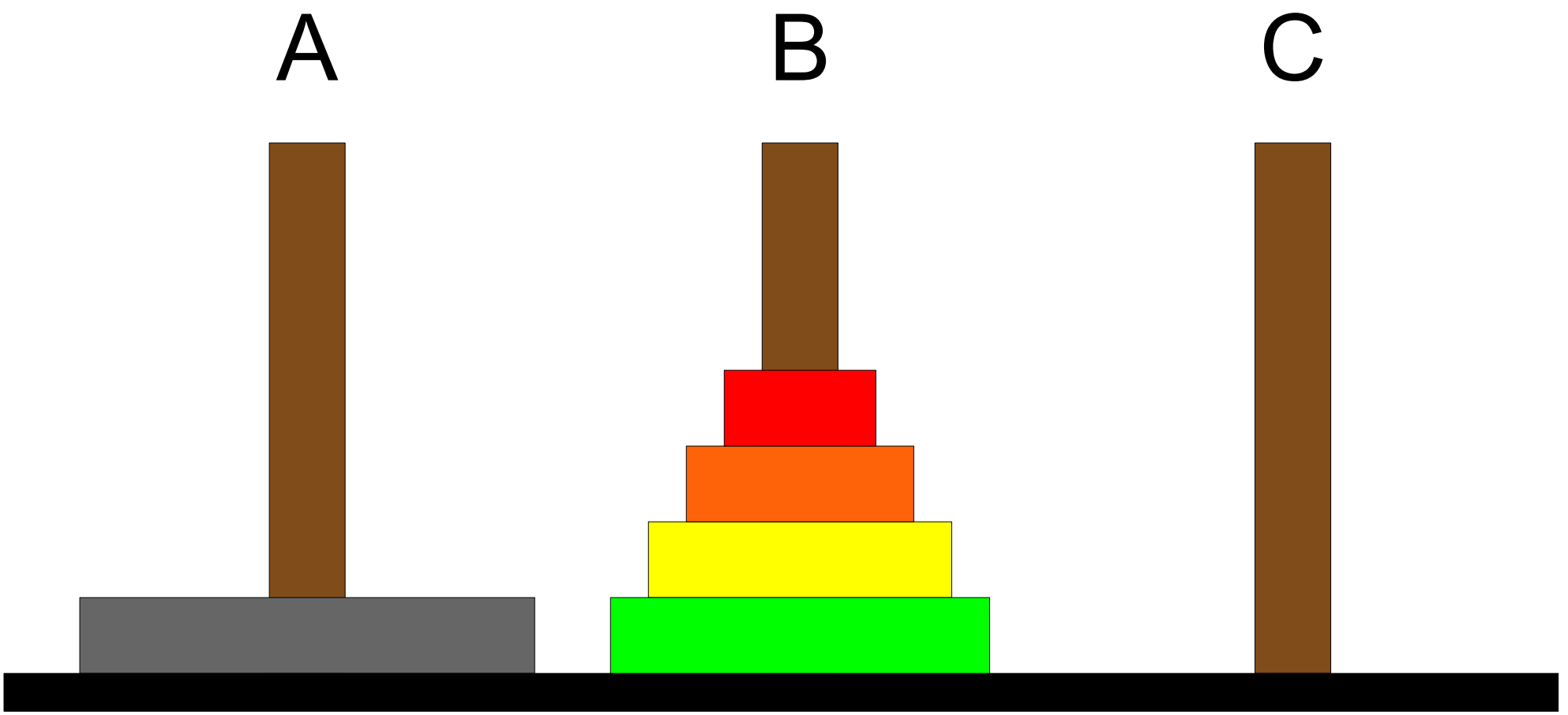
Solving the Towers of Hanoi



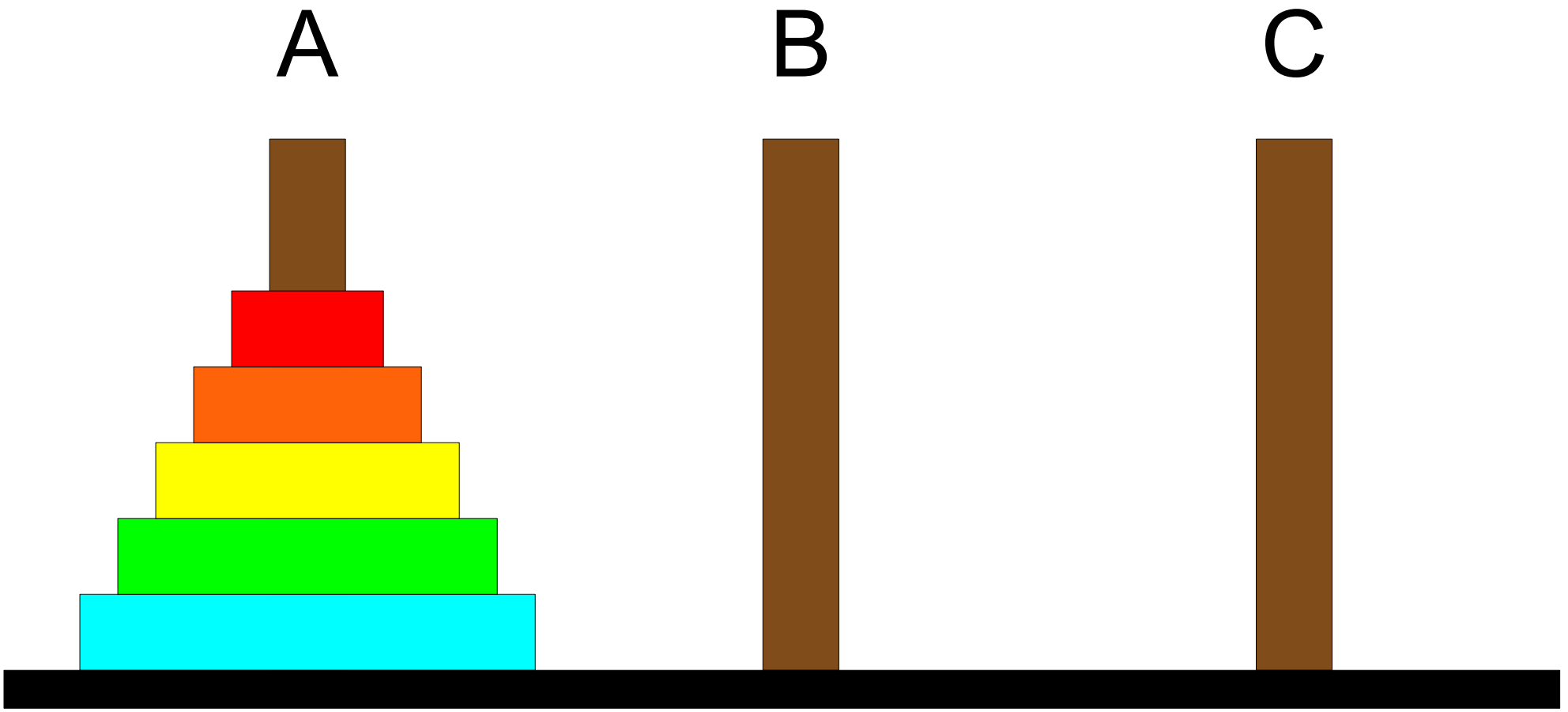
Solving the Towers of Hanoi



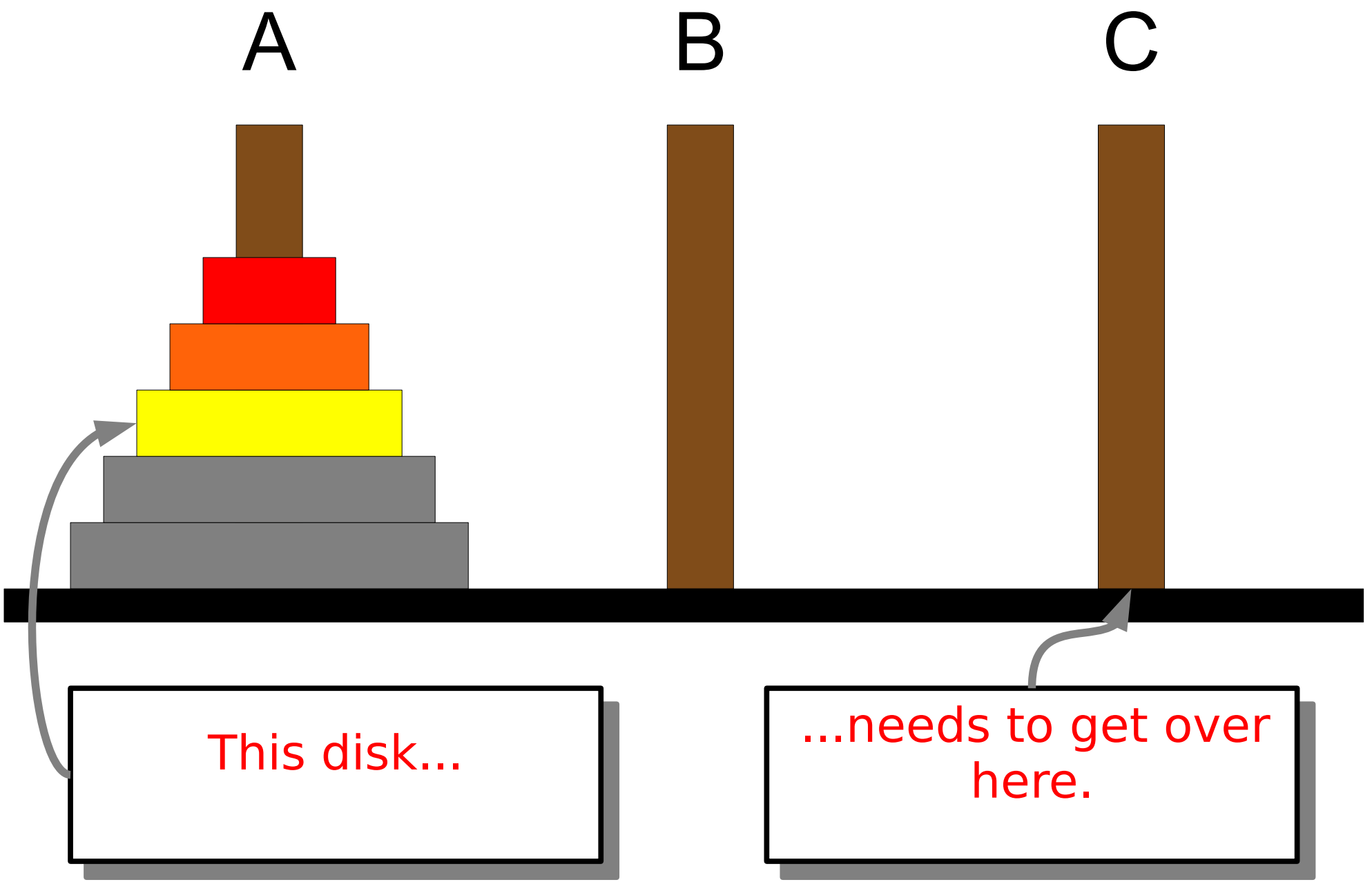
Solving the Towers of Hanoi



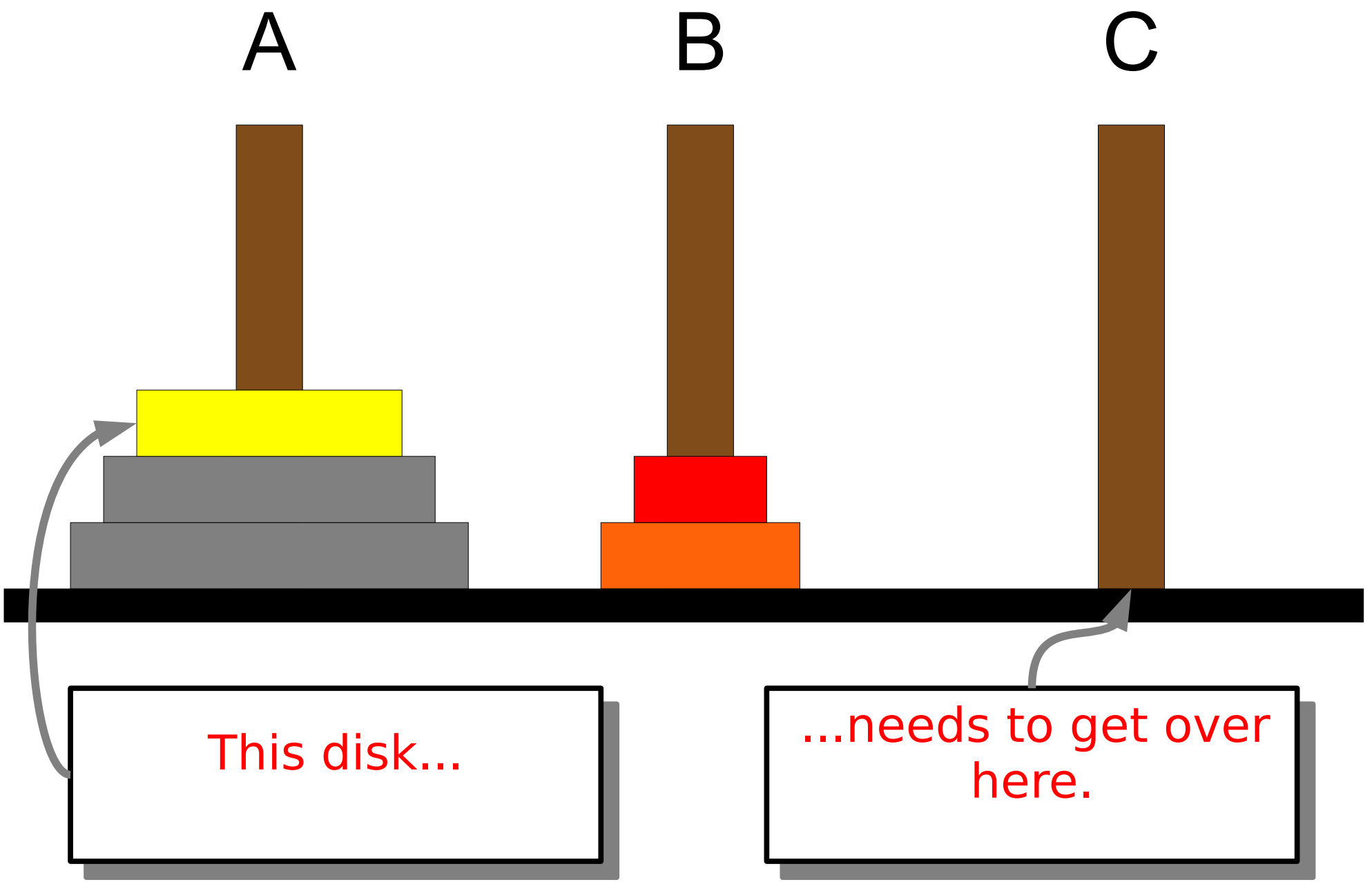
Solving the Towers of Hanoi



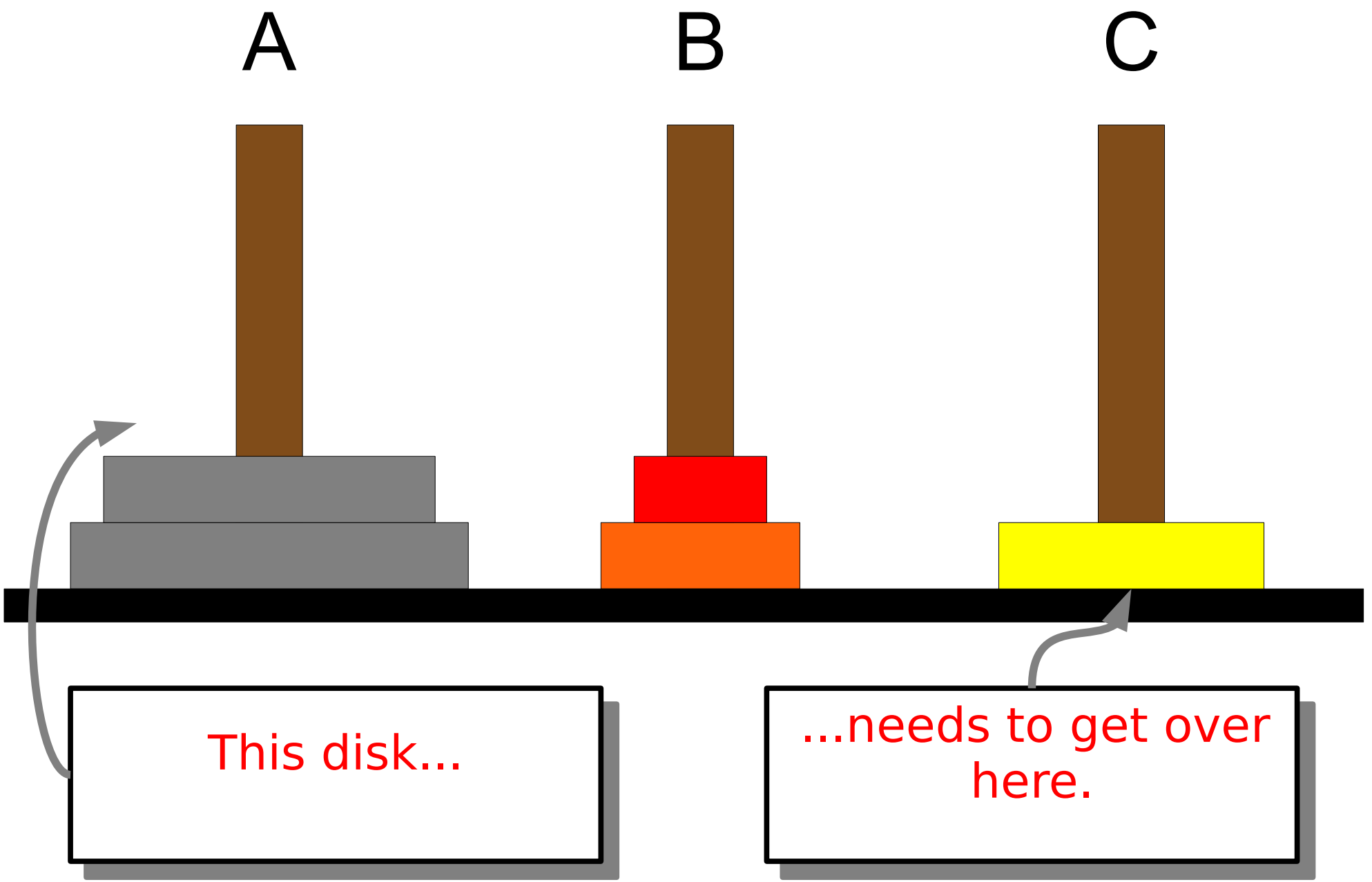
Solving the Towers of Hanoi



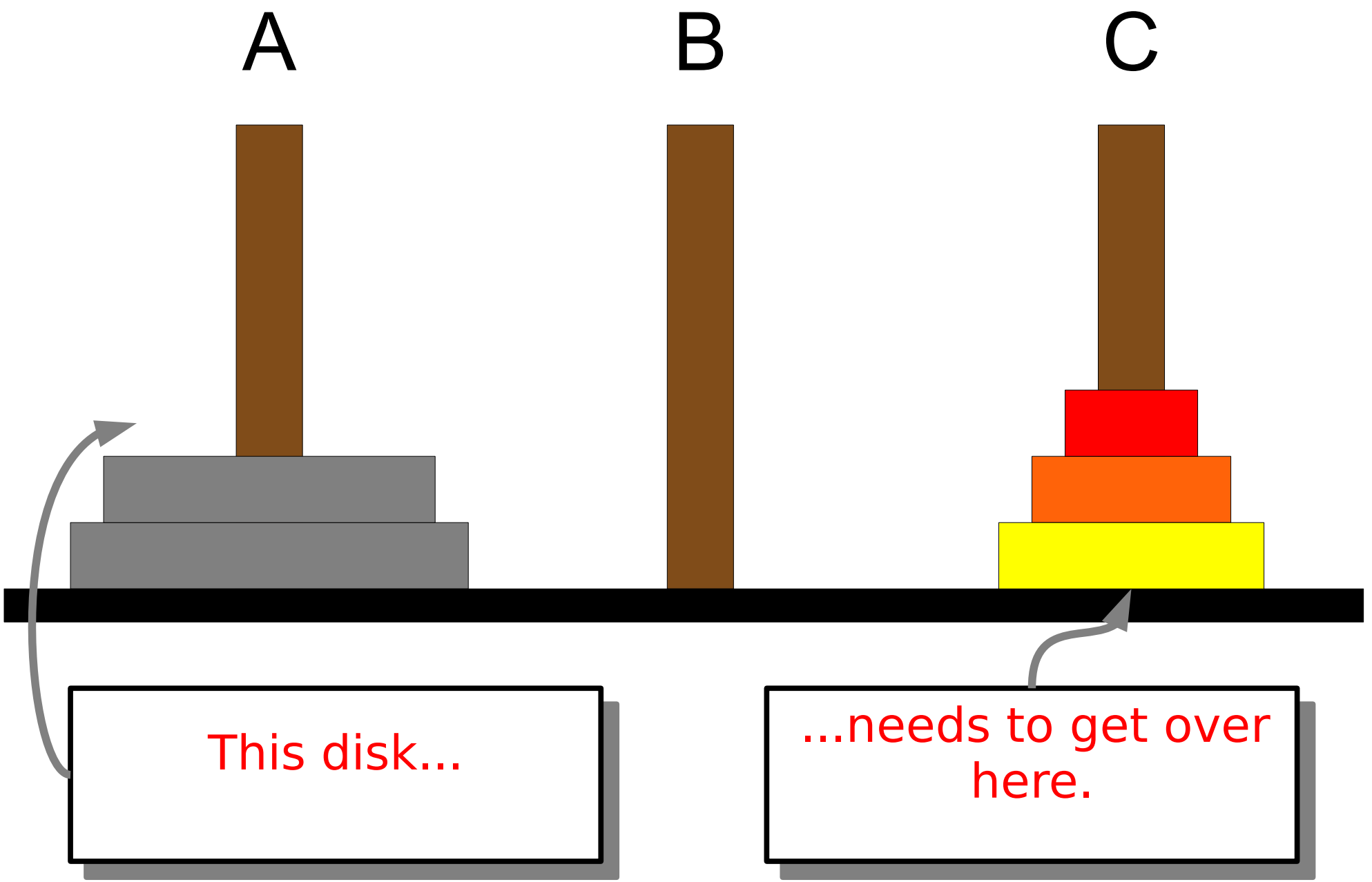
Solving the Towers of Hanoi



Solving the Towers of Hanoi



Solving the Towers of Hanoi



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.

Hanoi
(Pseudocode)

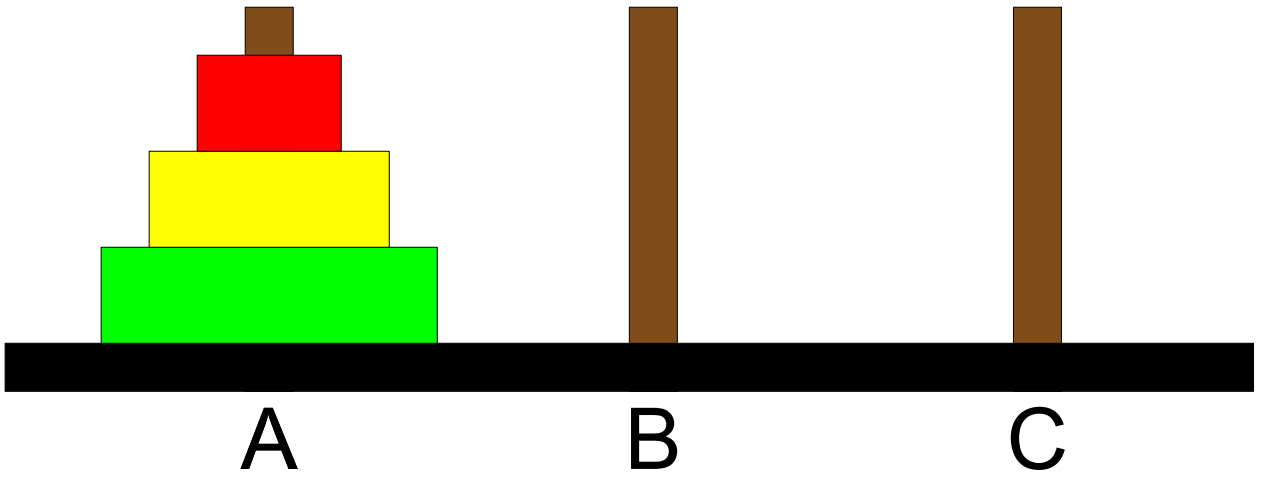
Hanoi.cpp
(Computer)

Tower of Hanoi Pseudocode

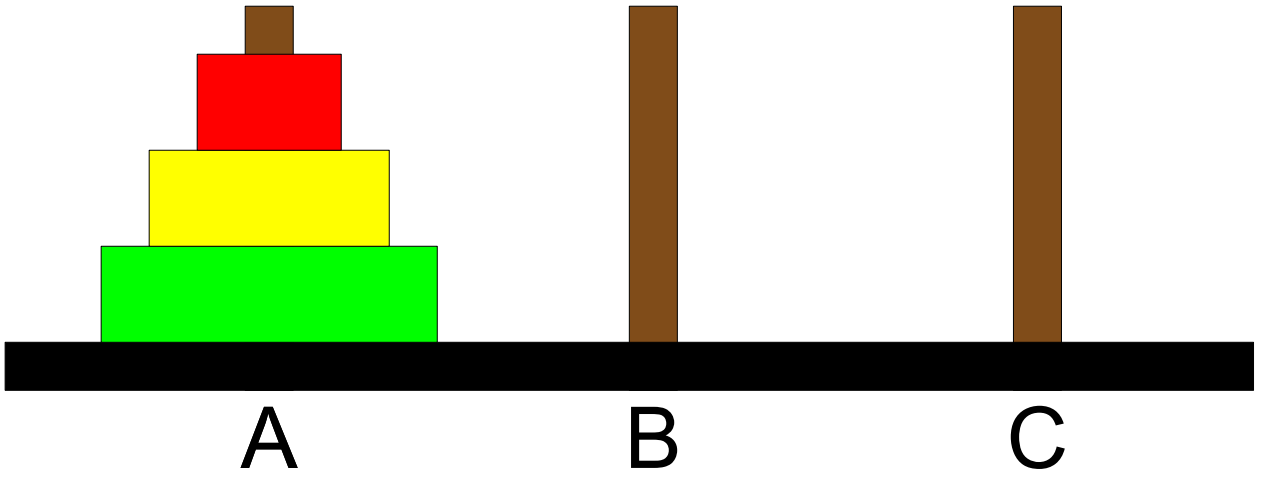
```
Input: numDisks, start, dest, temp
if numDisks == 1
    move disk from start to dest
else
    move numDisks-1 from start to temp
    move last disk from start to dest
    move numDisks-1 from temp to dest
```

```
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');
}
```



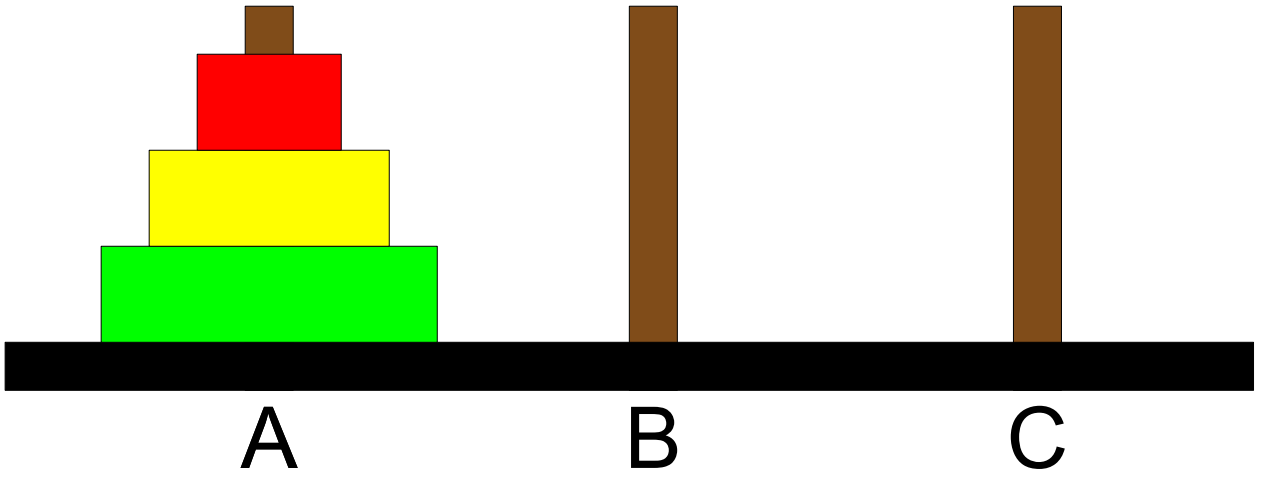
```
int main() {
    moveTower(3, 'a', 'c', 'b');
}
```



```
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);
    }
}
```

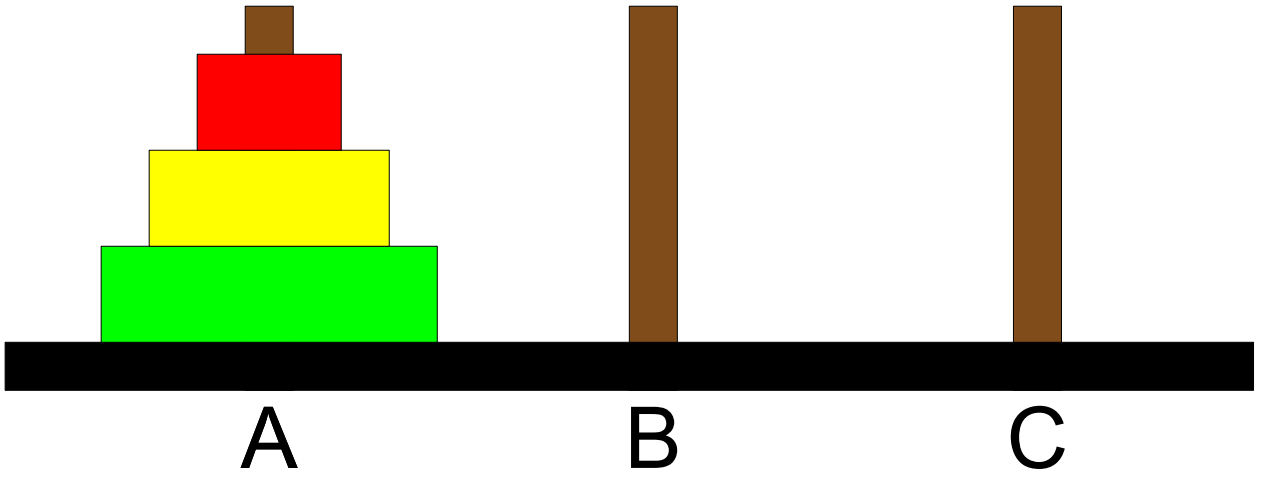
n	3	from	a	to	c	temp	b



```
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);
    }
}
```

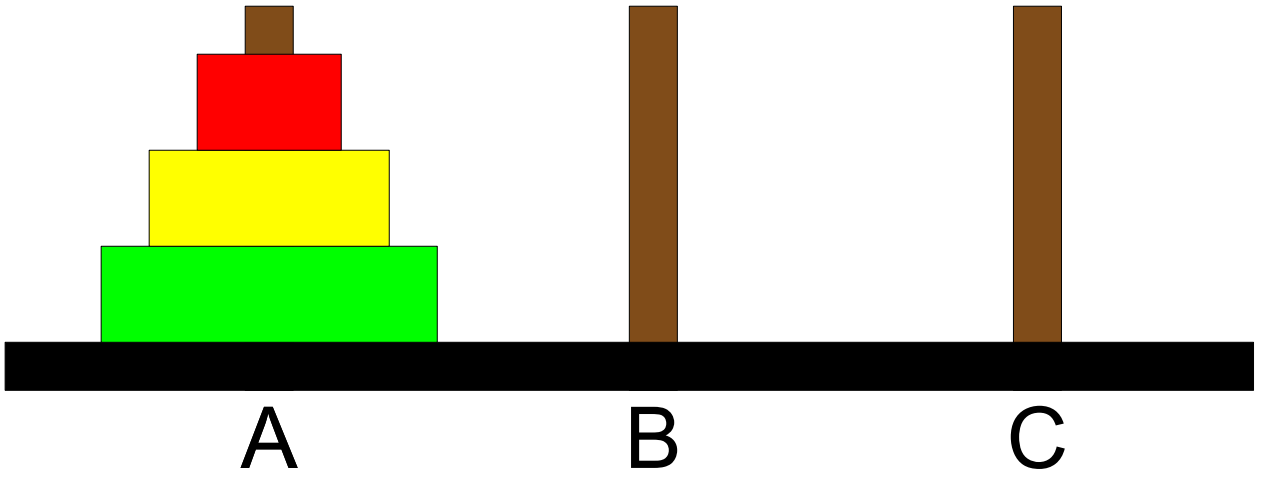
n	3	from	a	to	c	temp	b



```
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);
    }
}
```

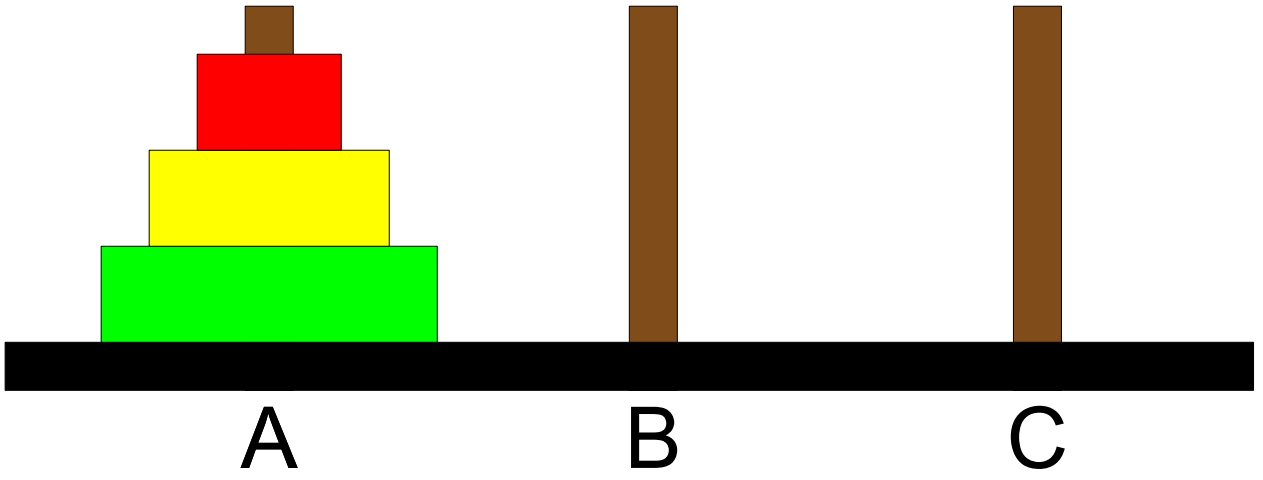
n	3	from	a	to	c	temp	b



```
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

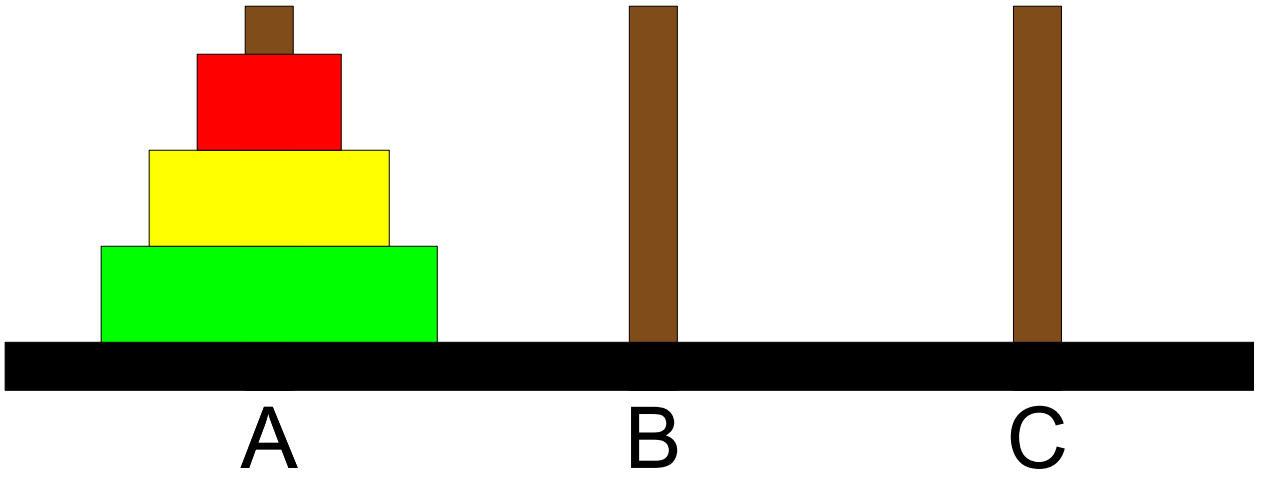


```
int main() {
}
```

```
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

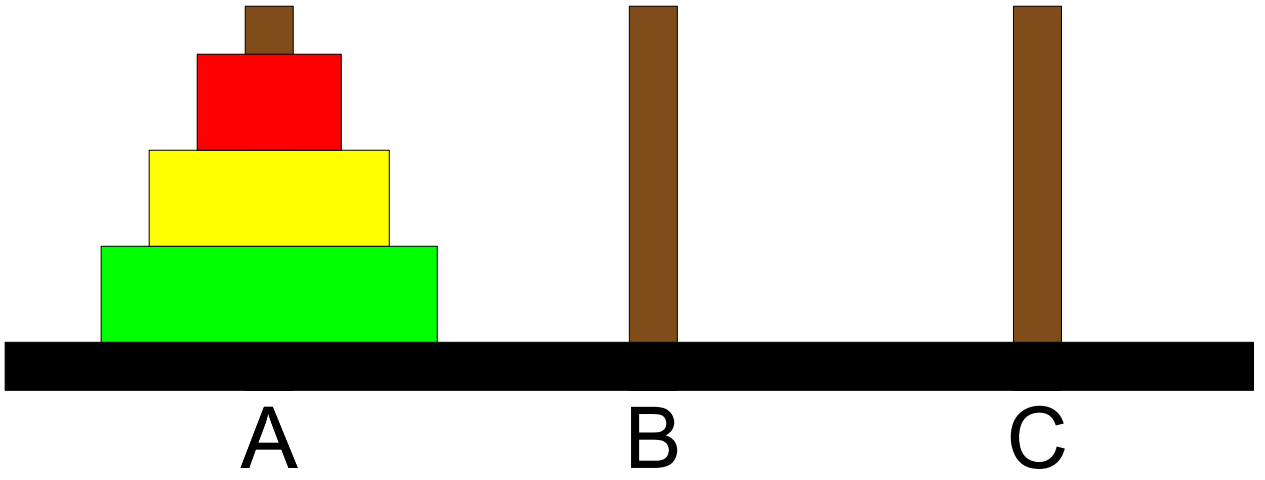


```
int main() {
}
```

```
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

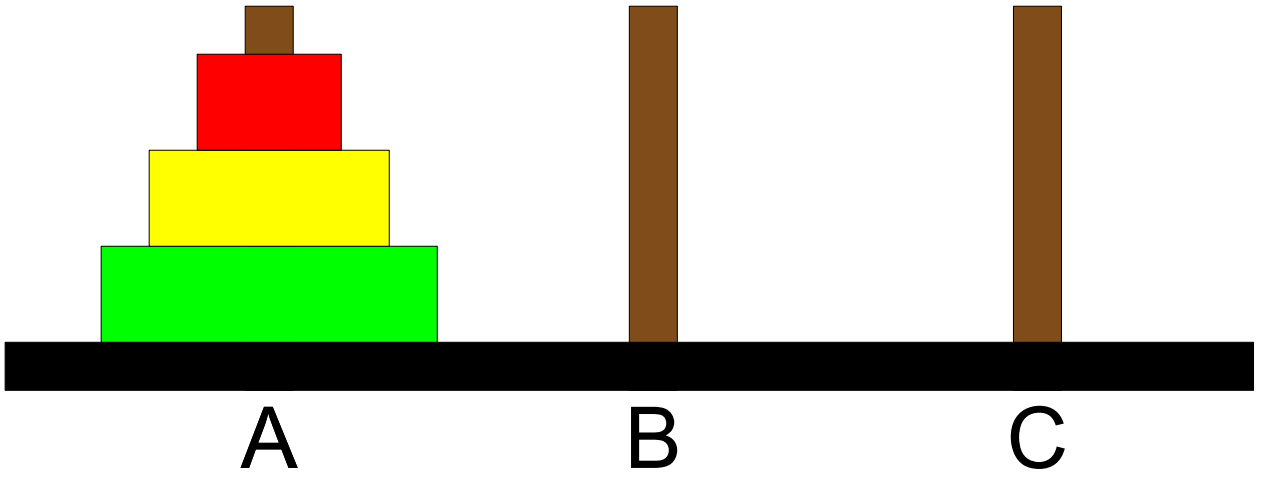


```
int main() {

}
```

```
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

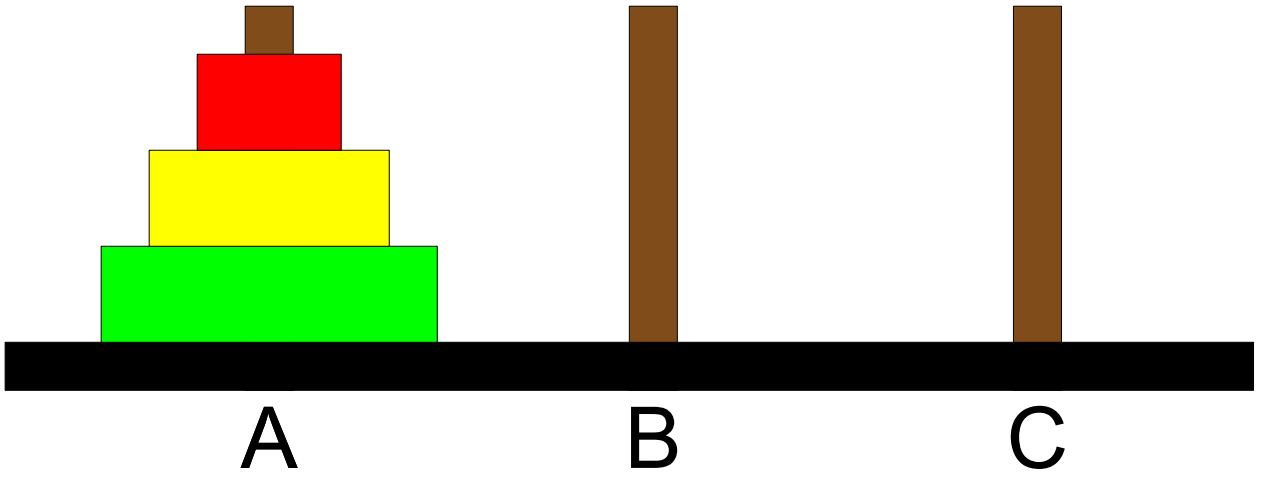


```
int main() {
}
```

```
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



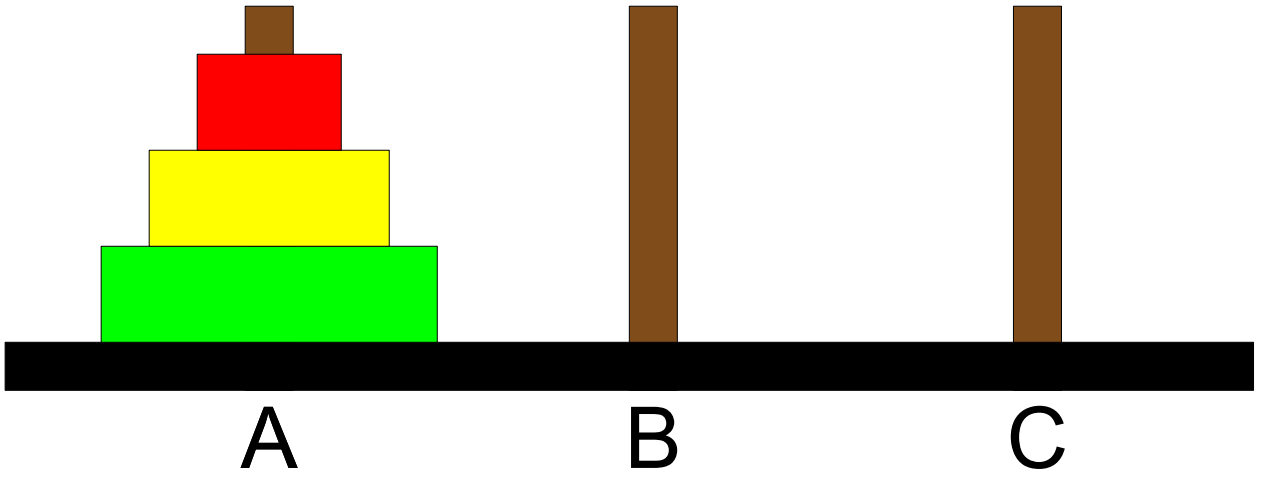
```
int main() {
}
```

```
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



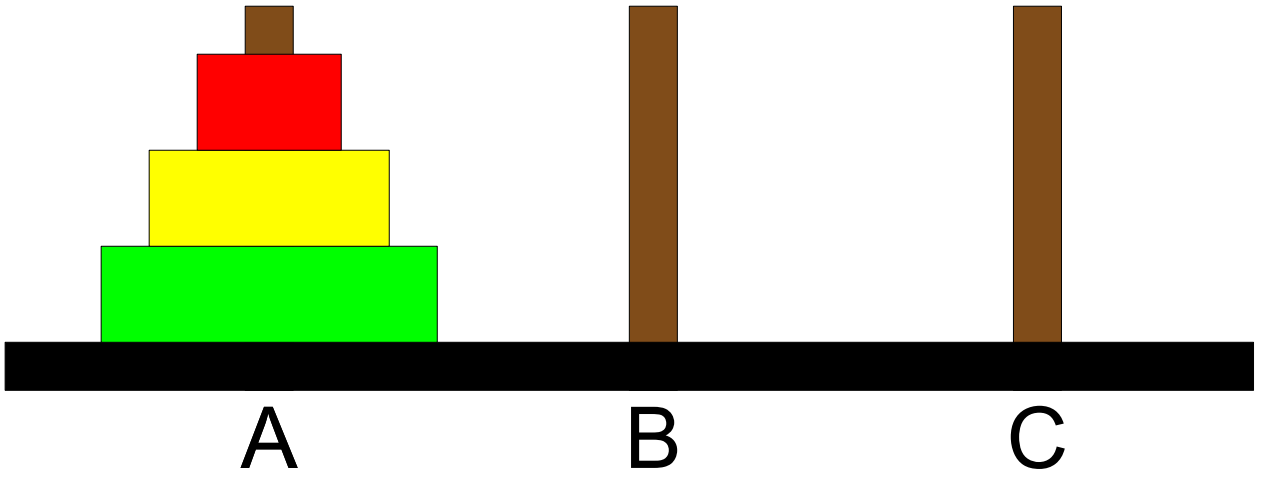
```
int main() {
}
```

```
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



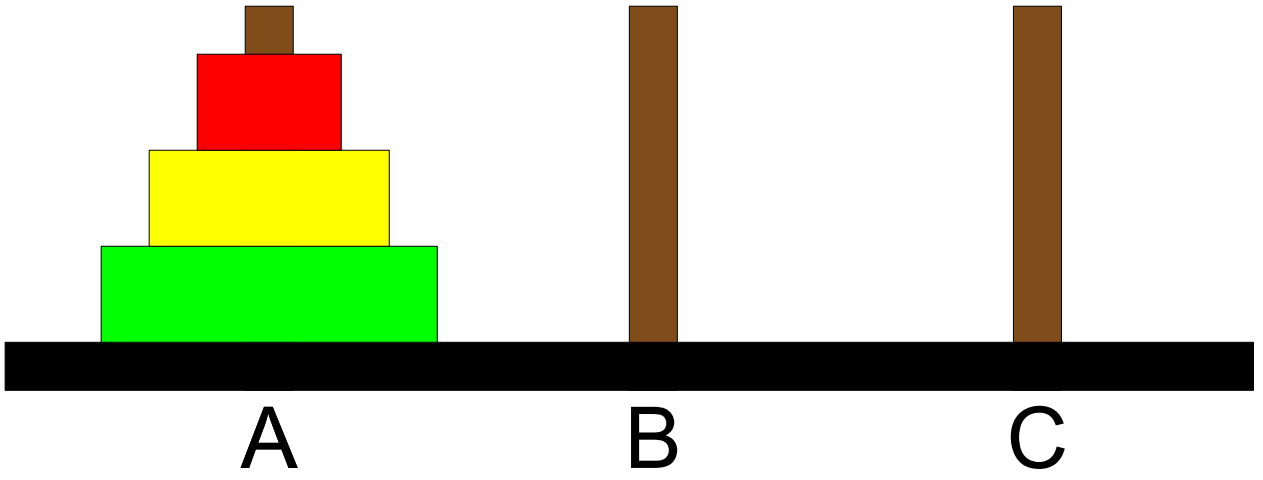
```
int main() {
}
```

```
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



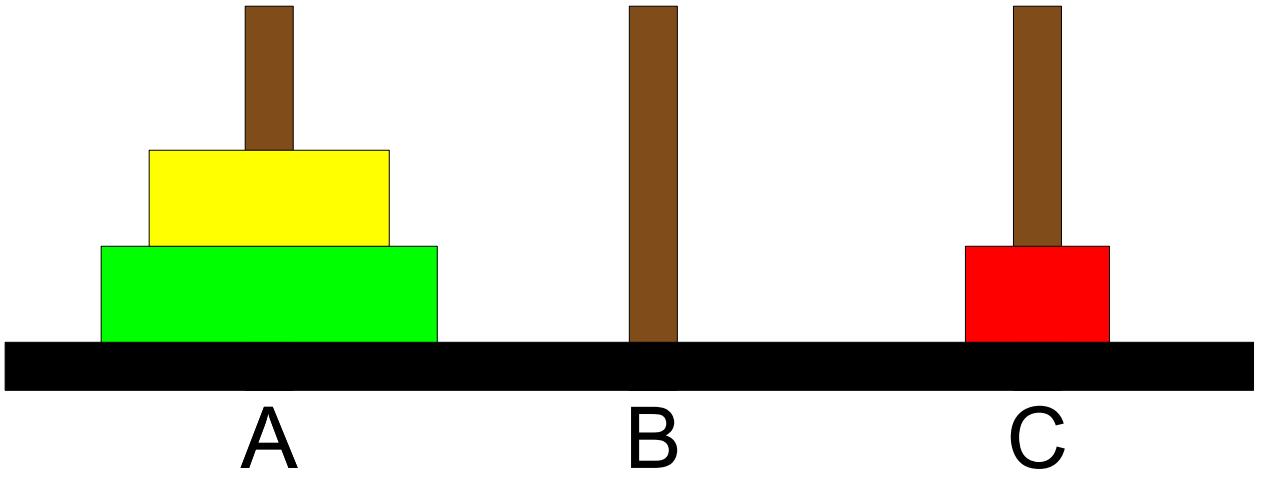
```
int main() {
}
```

```
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

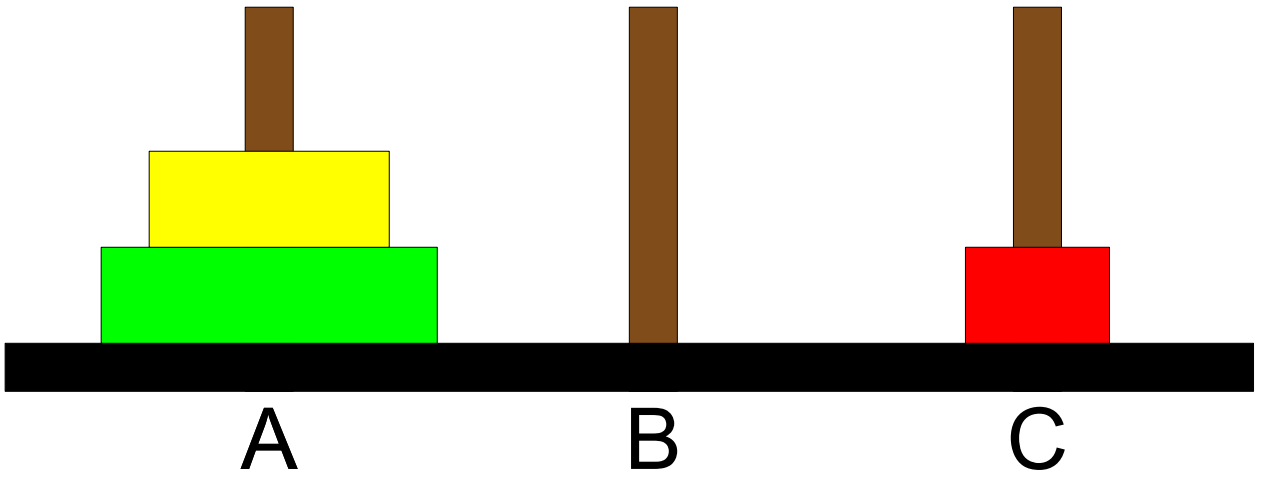


```
int main() {
}
```

```
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

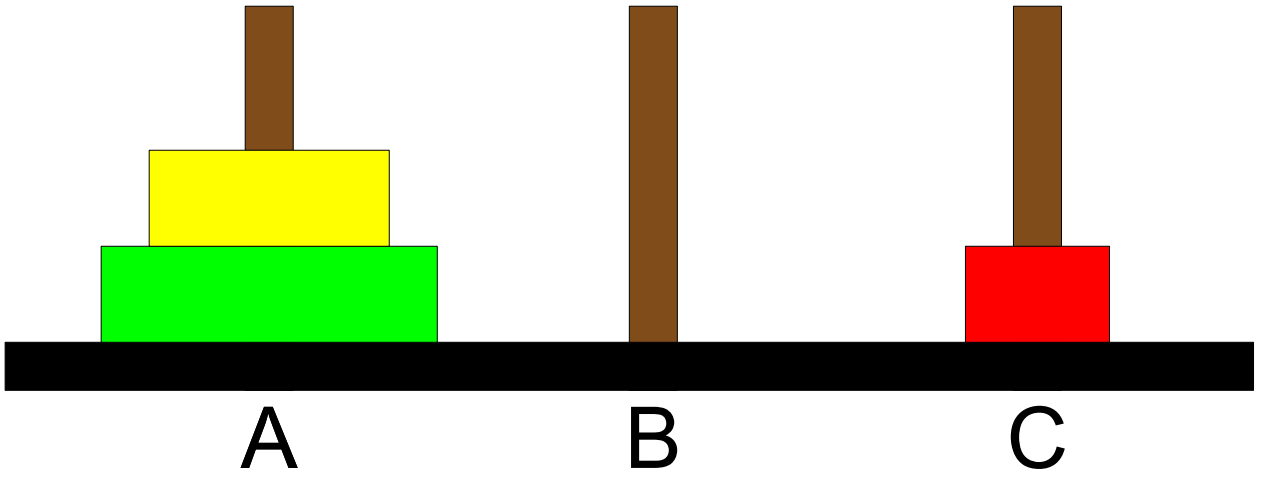


```
int main() {
}
```

```
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

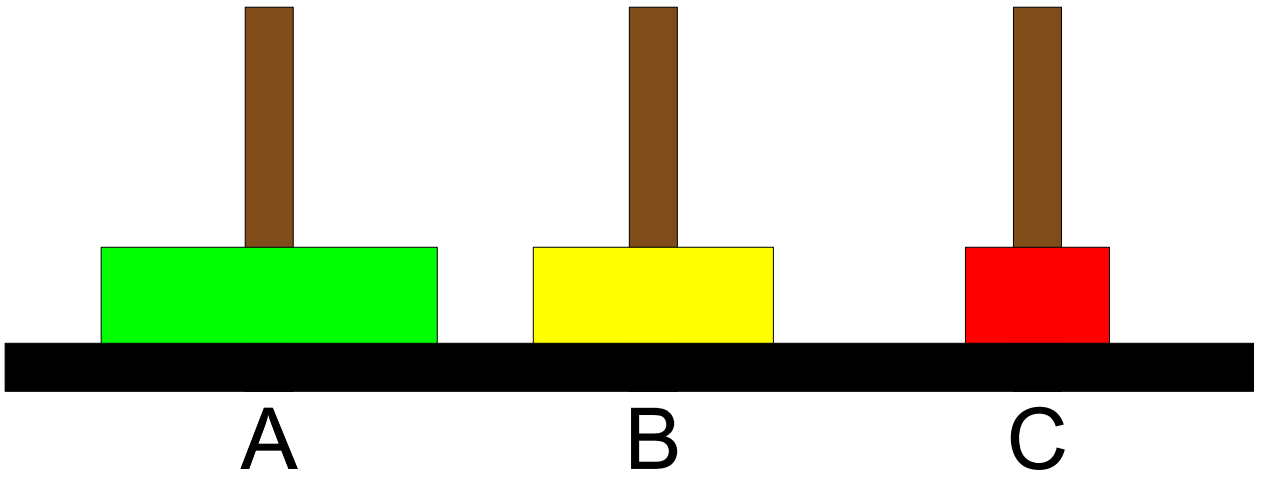


```
int main() {

}
```

```
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

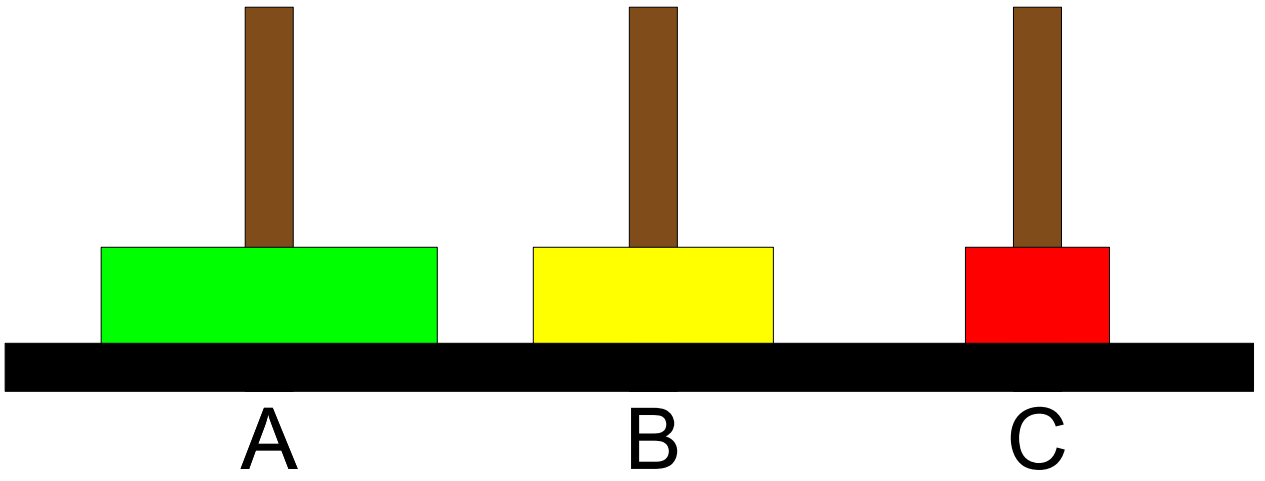


```
int main() {
}
```

```
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



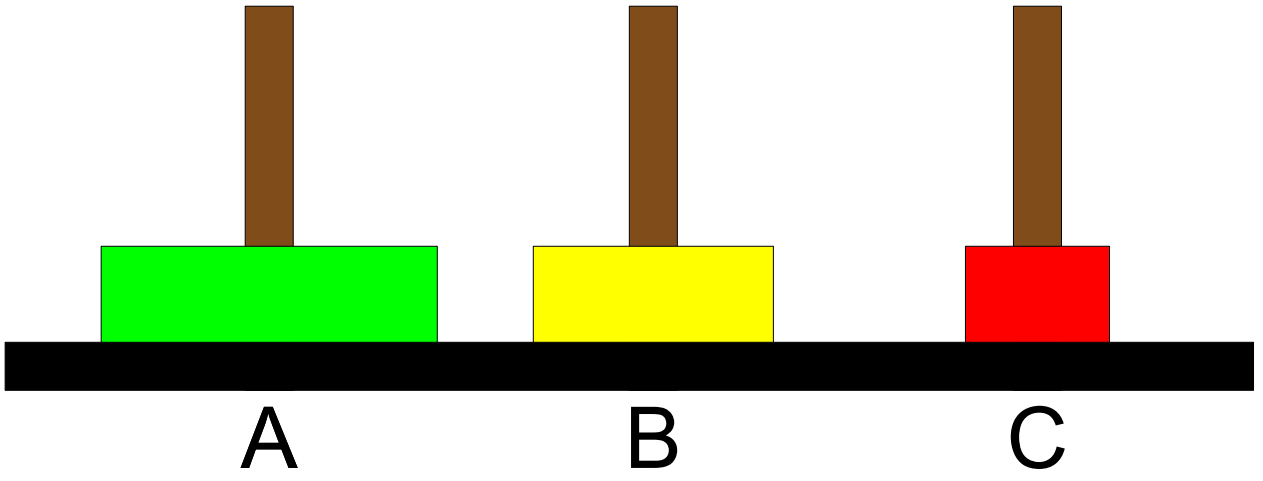
```
int main() {
}
```

```
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



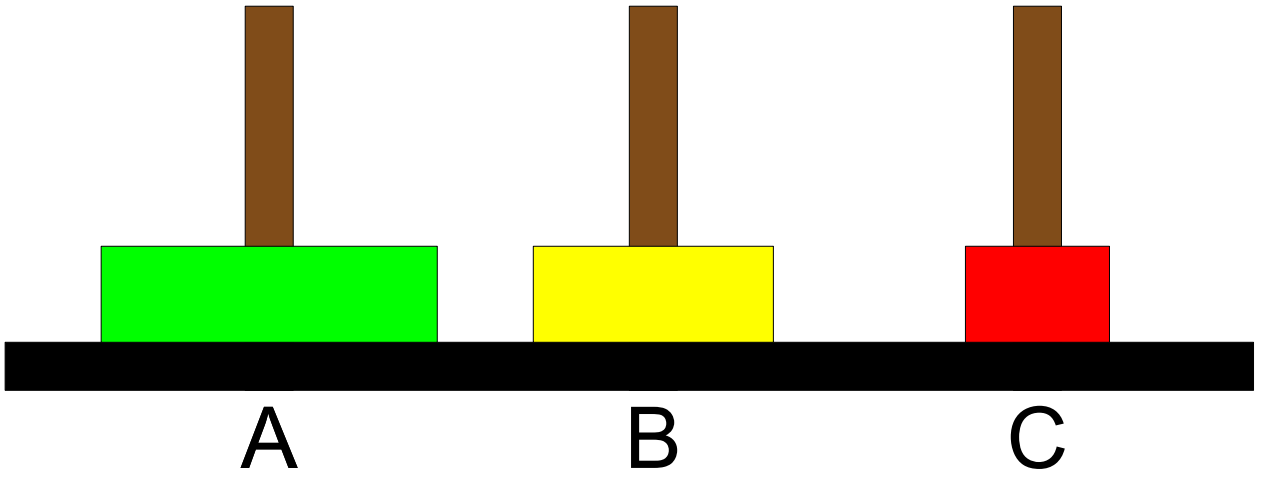
```
int main() {
}
```

```
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



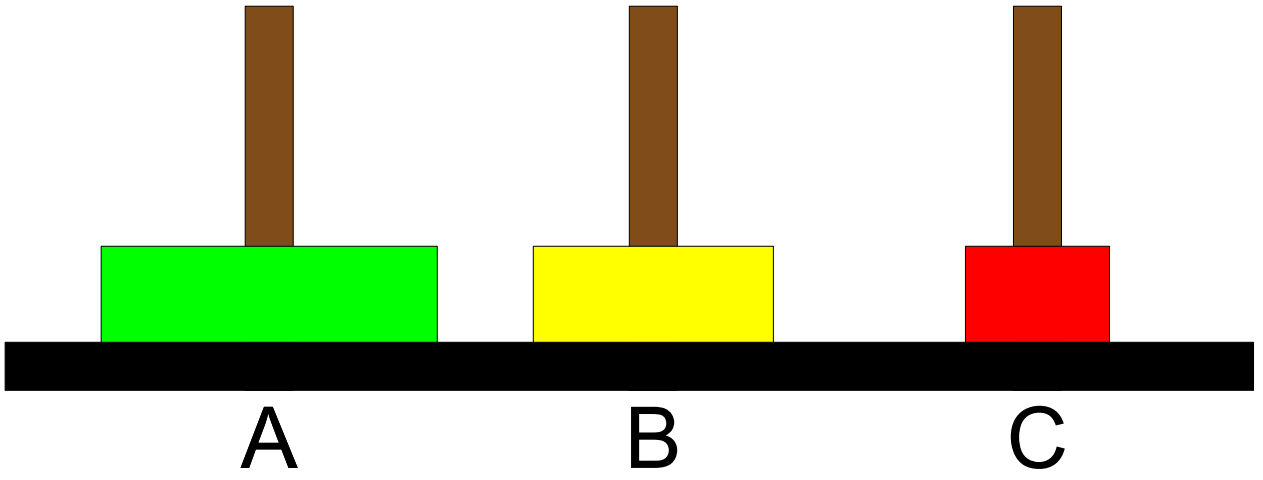
```
int main() {
}
```

```
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



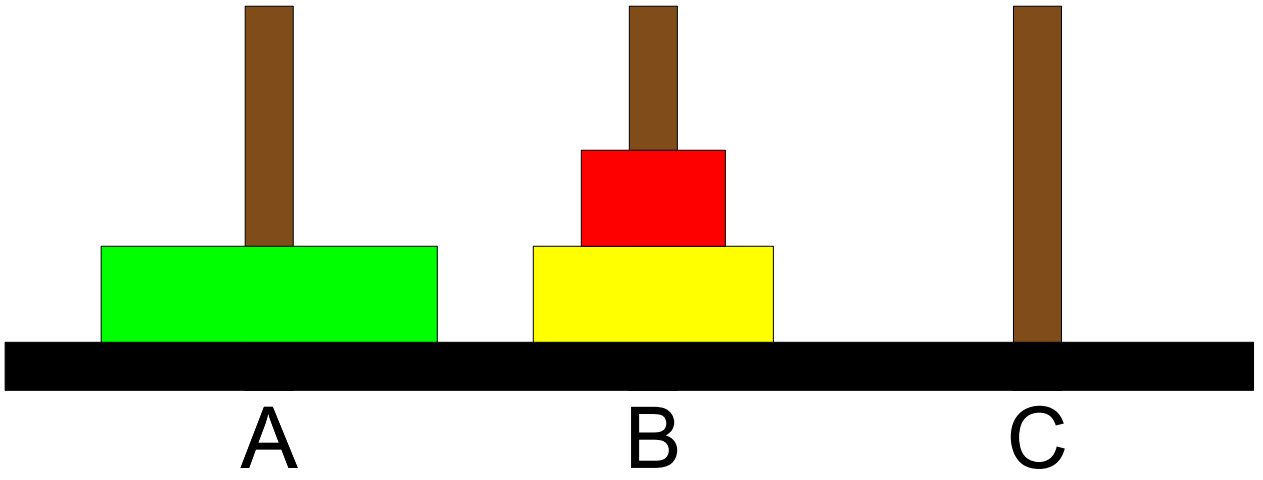
```
int main() {
}
```

```
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

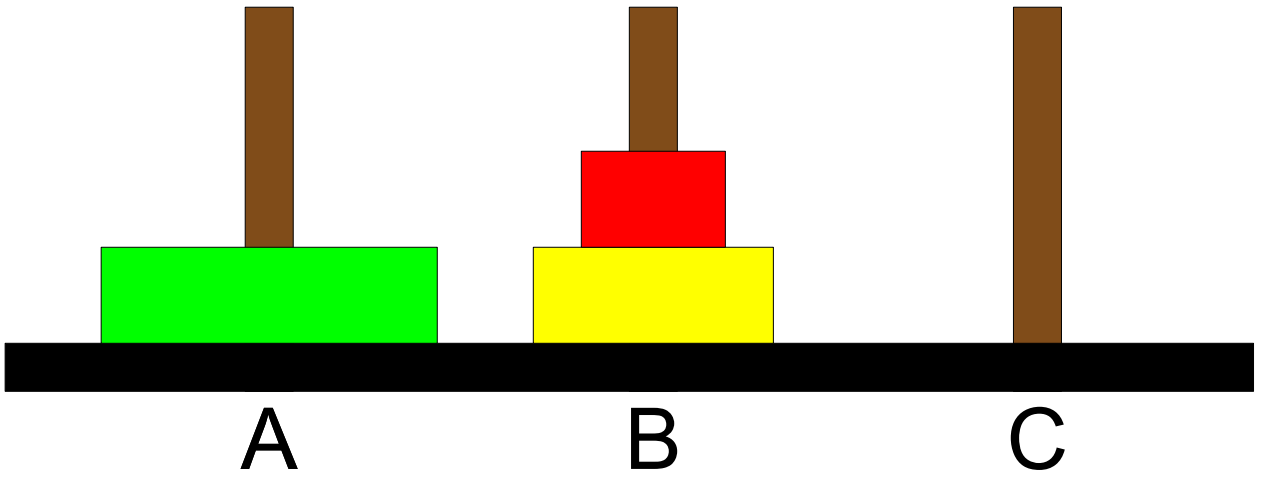


```
int main() {
}
```

```
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);
    }
}
```

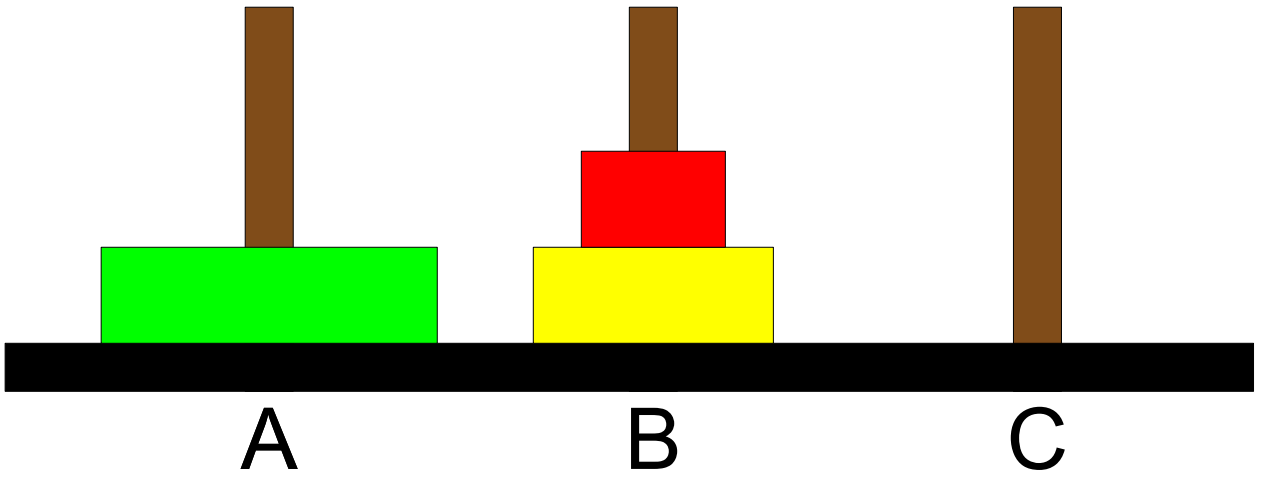
n	2	from	a	to	b	temp	c



```
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);
    }
}
```

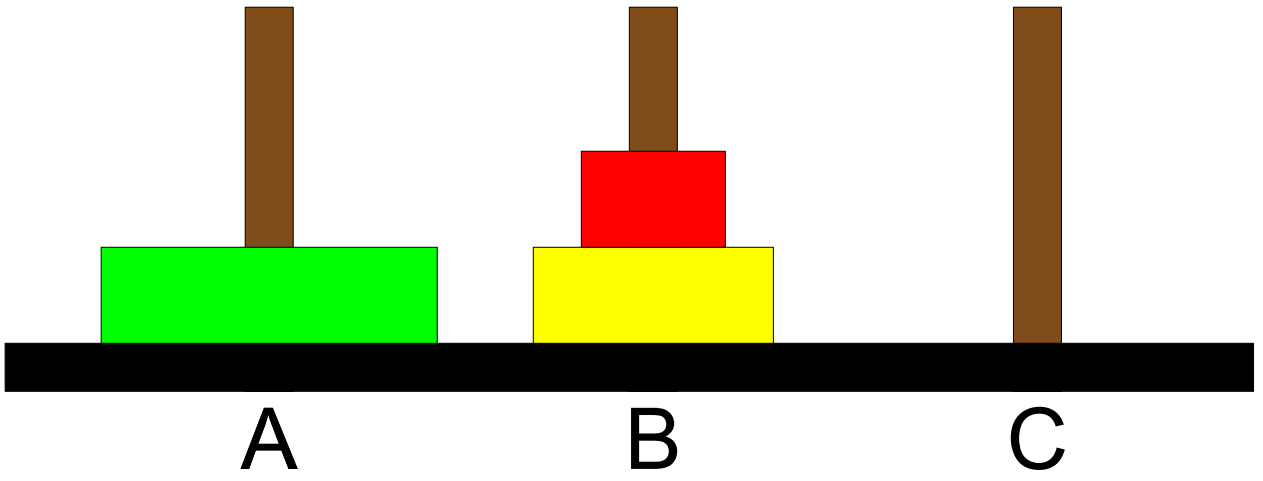
n	3	from	a	to	c	temp	b



```
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);
    }
}
```

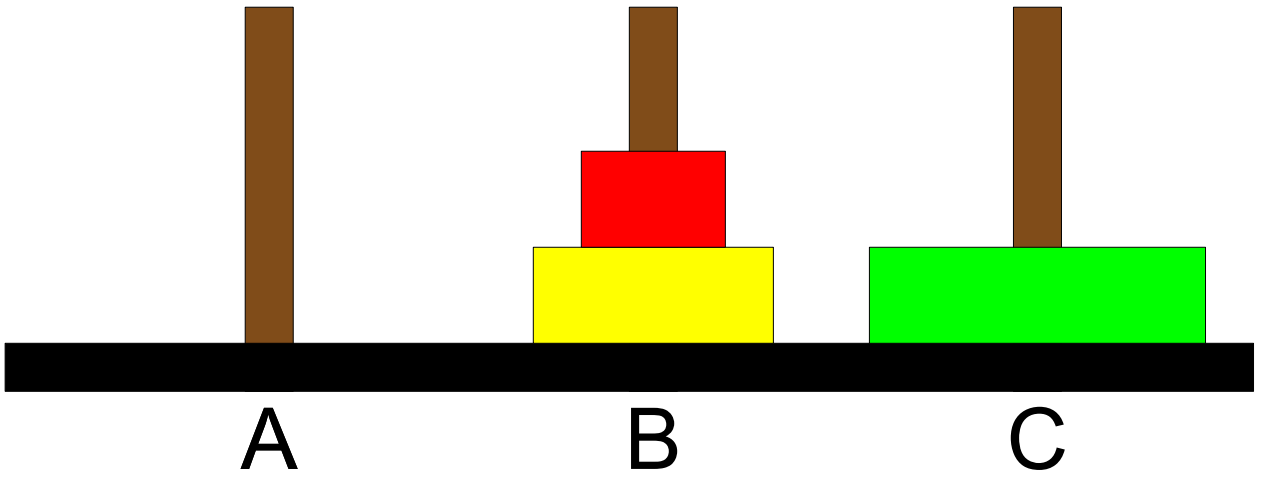
n	3	from	a	to	c	temp	b



```
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);
    }
}
```

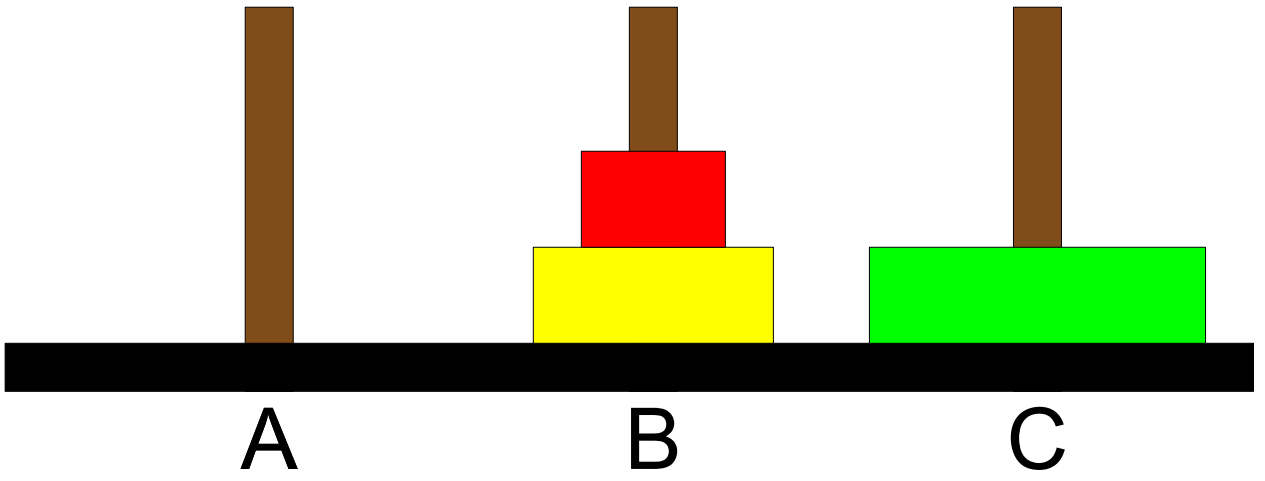
n	3	from	a	to	c	temp	b



```
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

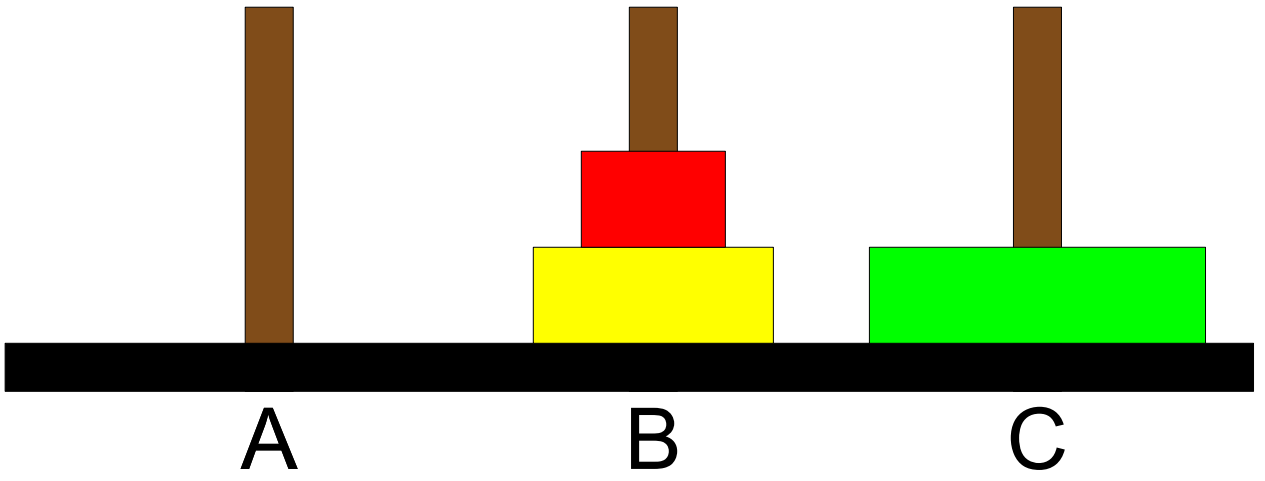


```
int main() {
}
```

```
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

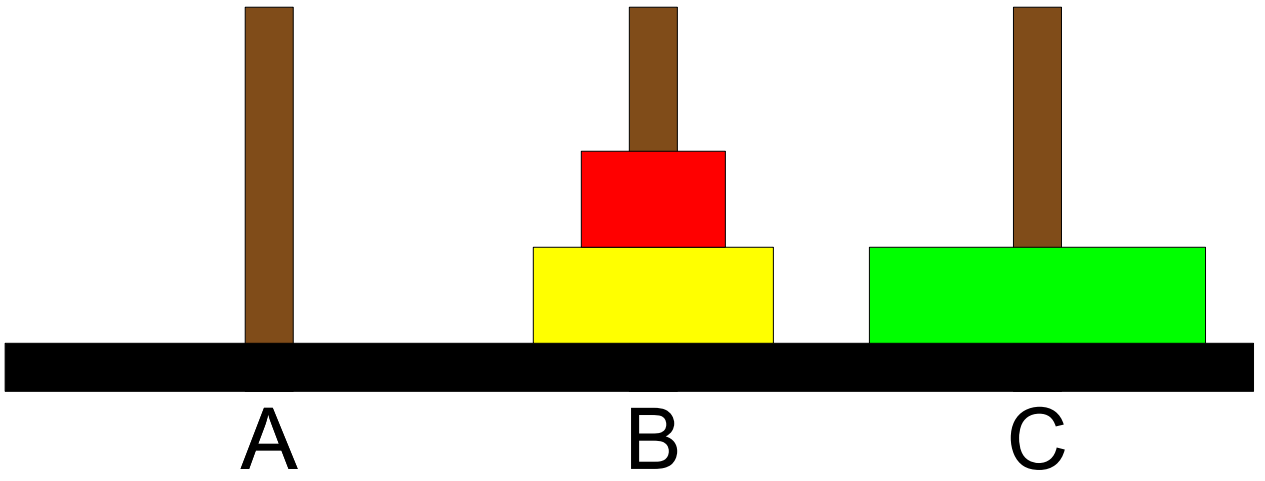


```
int main() {

}
```

```
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

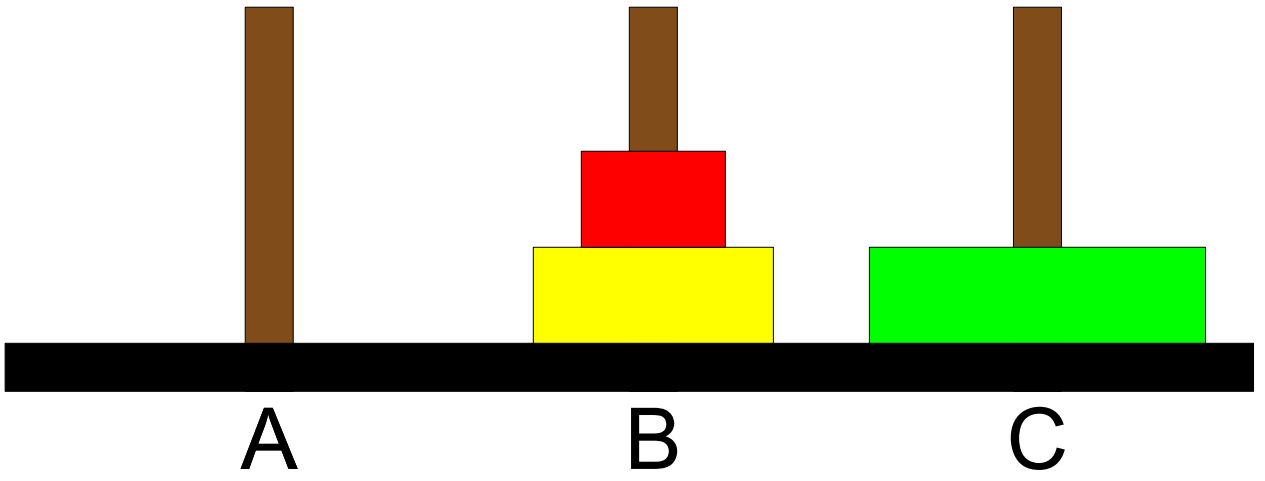


```
int main() {
}
```

```
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

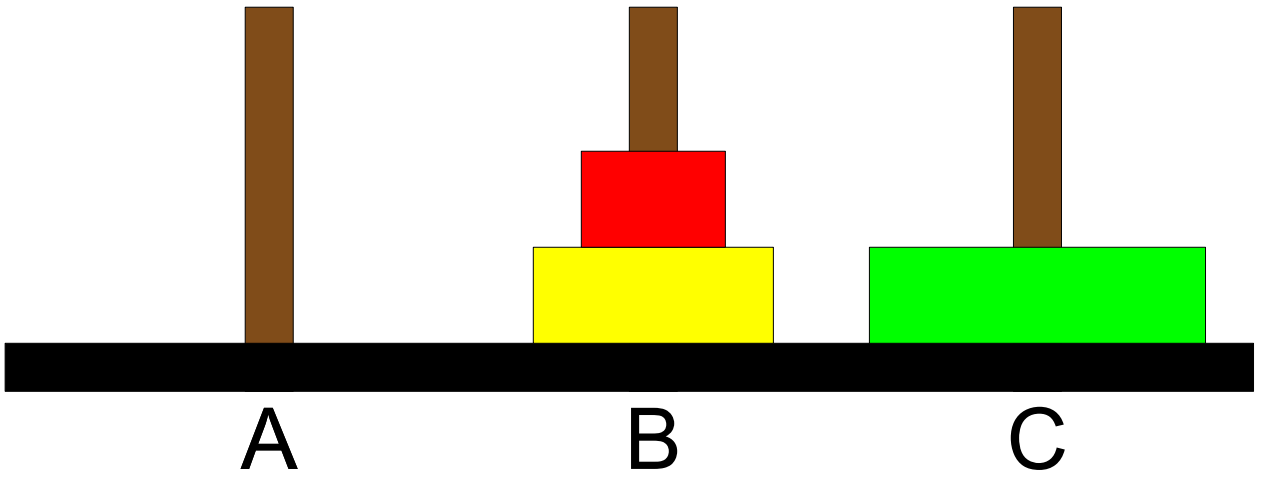


```
int main() {
}
```

```
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



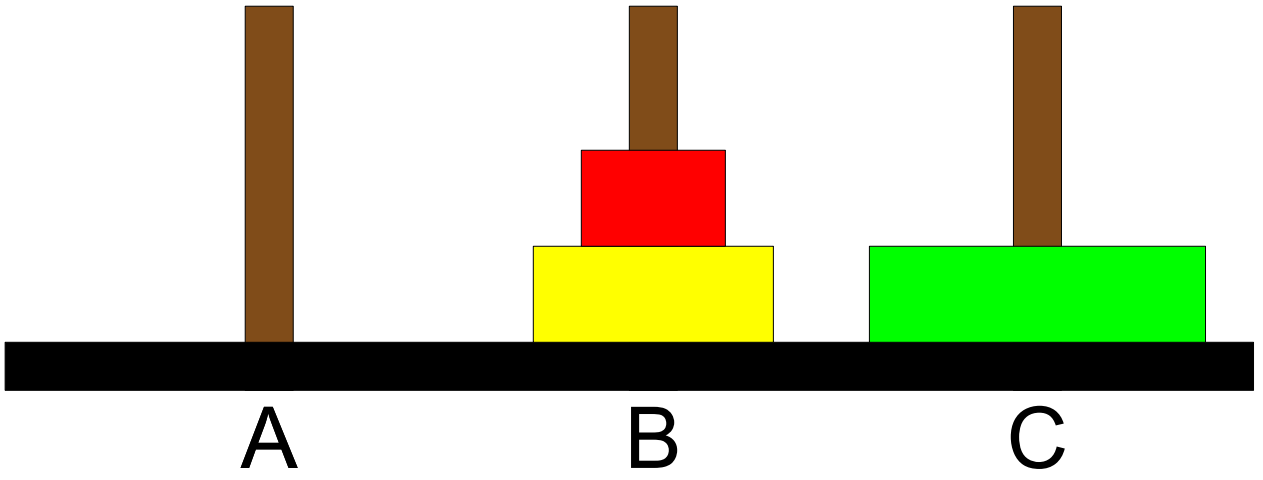
```
int main() {
}
```

```
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



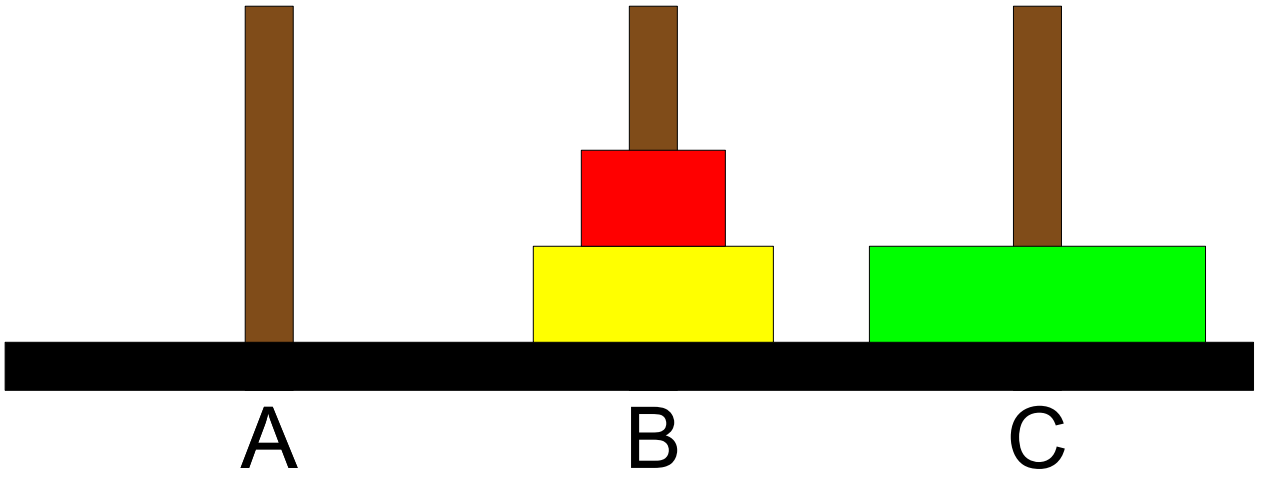
```
int main() {
}
```

```
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



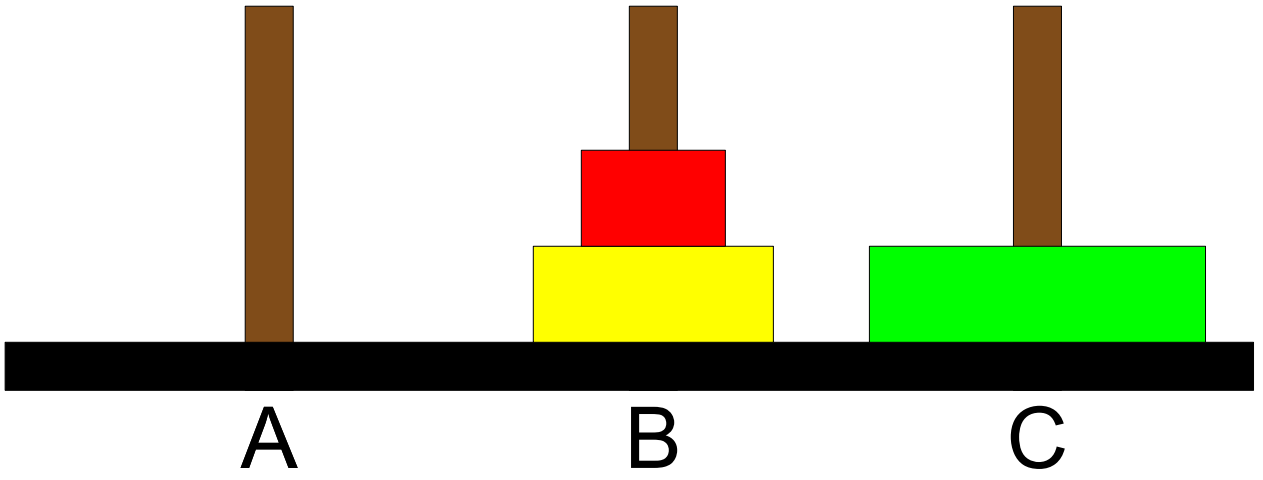
```
int main() {
}
```

```
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



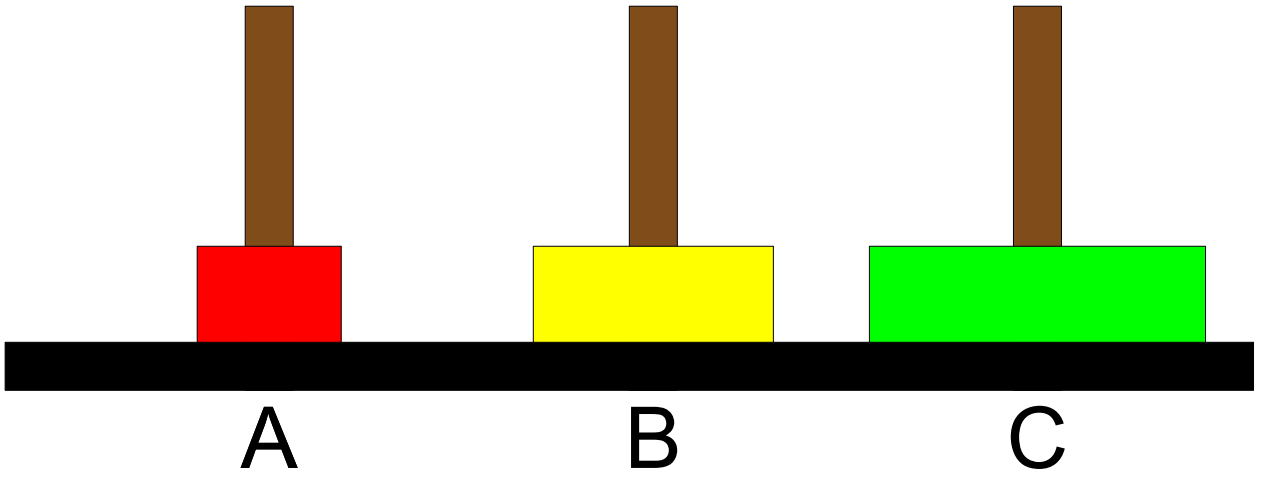
```
int main() {
}
```

```
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

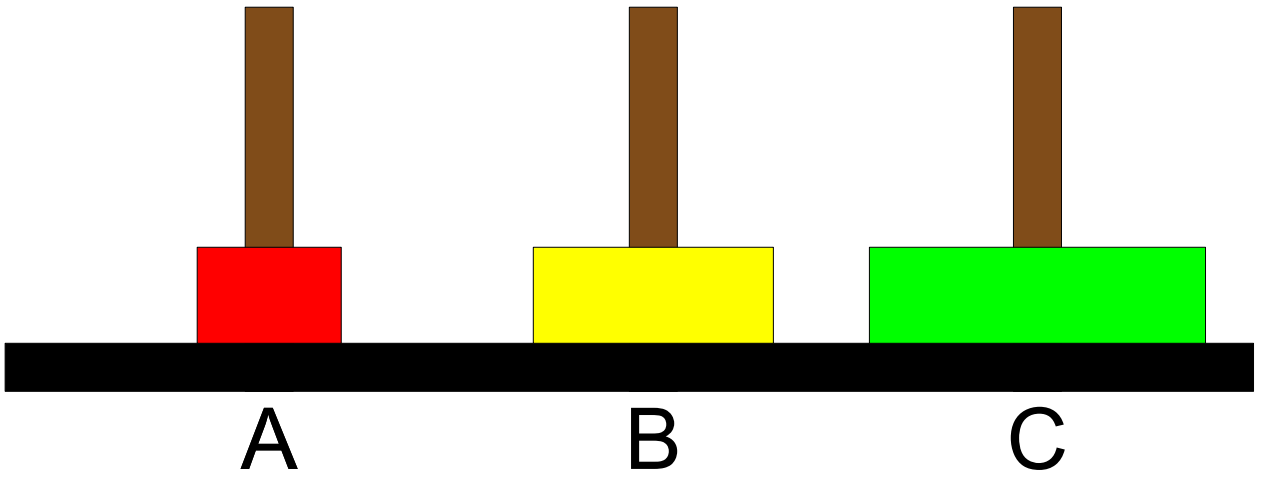


```
int main() {
}
```

```
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

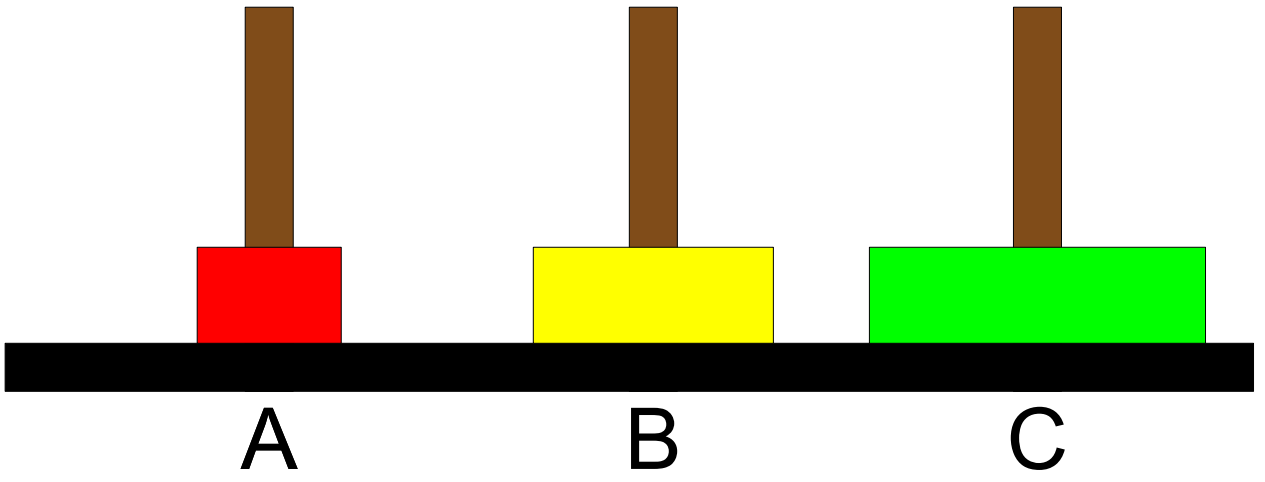


```
int main() {
}
```

```
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

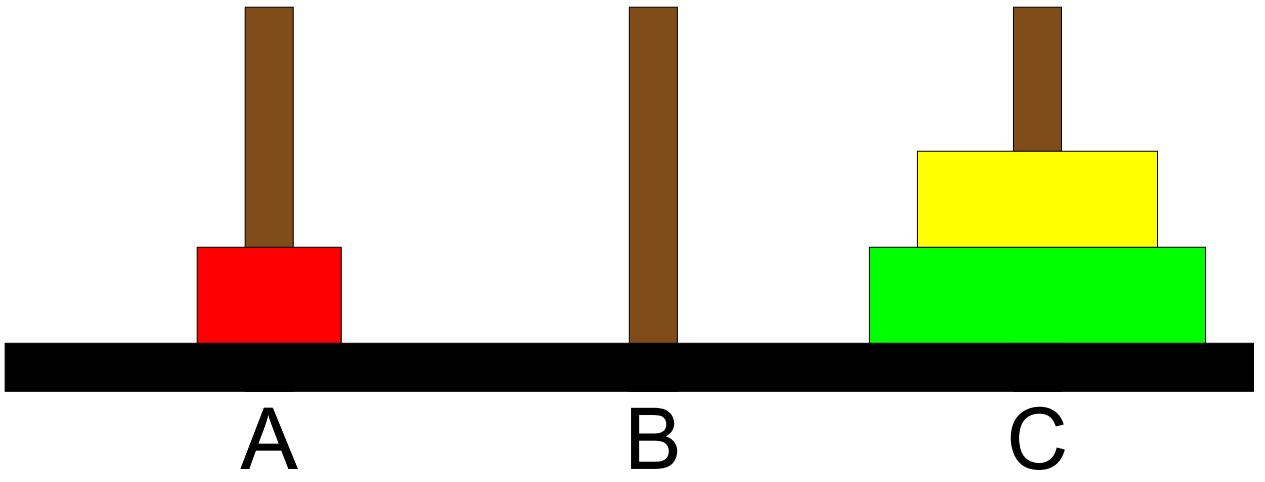


```
int main() {
}
```

```
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

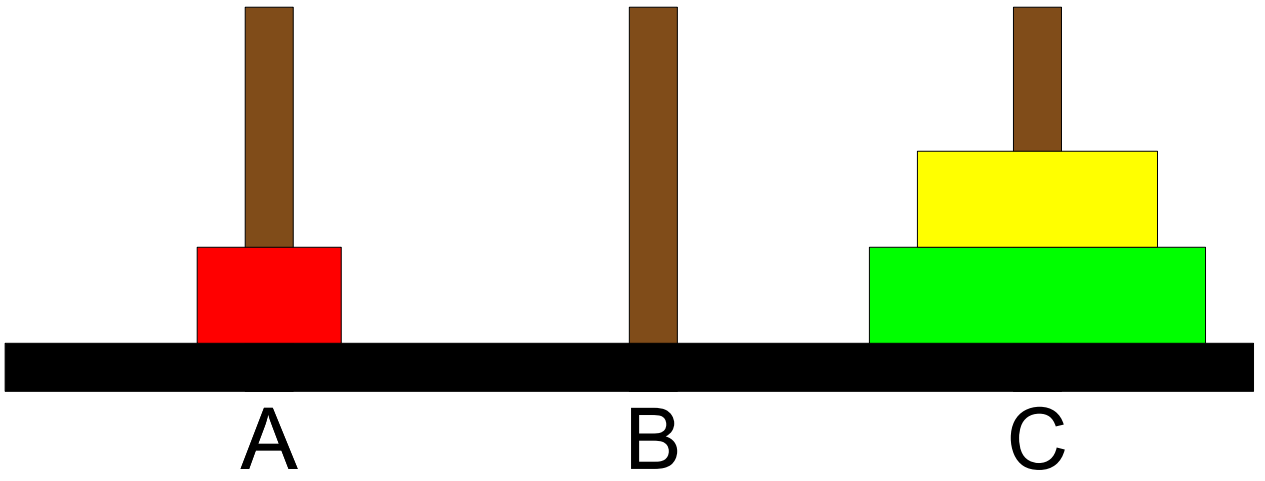


```
int main() {
}
```

```
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



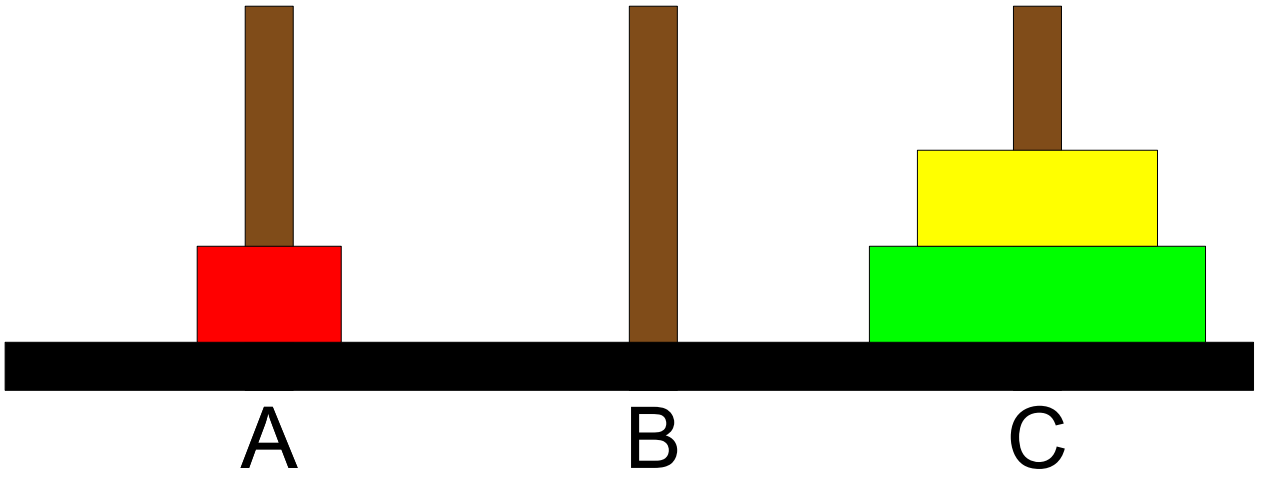
```
int main() {
}
```

```
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



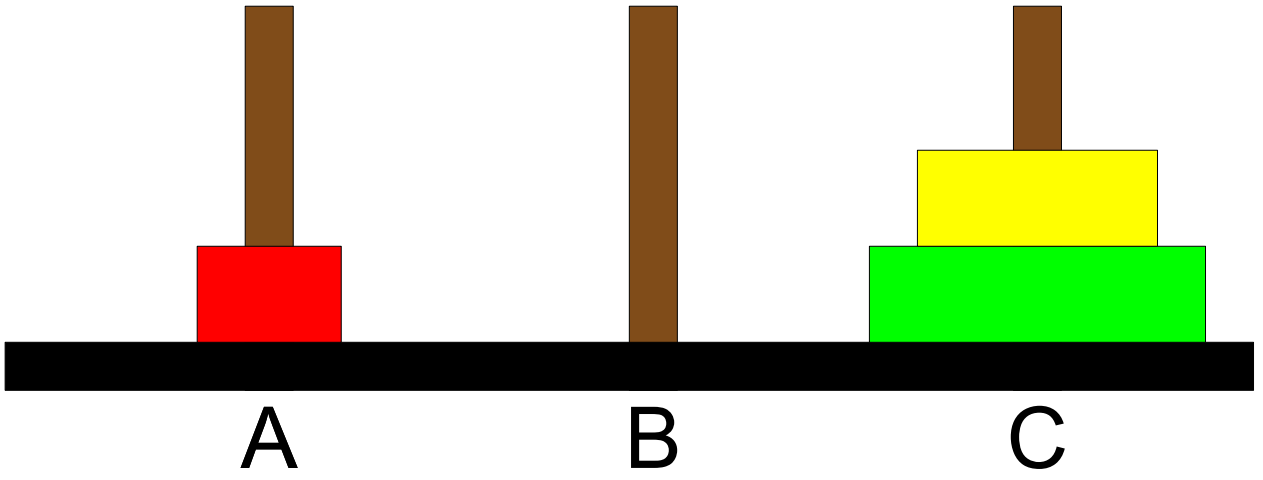
```
int main() {
}
```

```
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



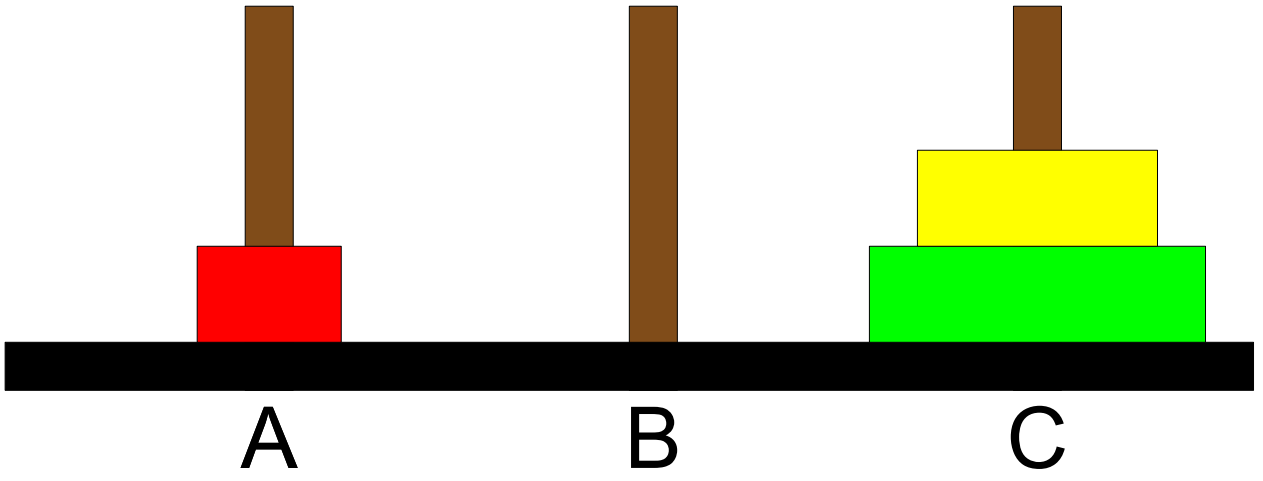
```
int main() {
}
```

```
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



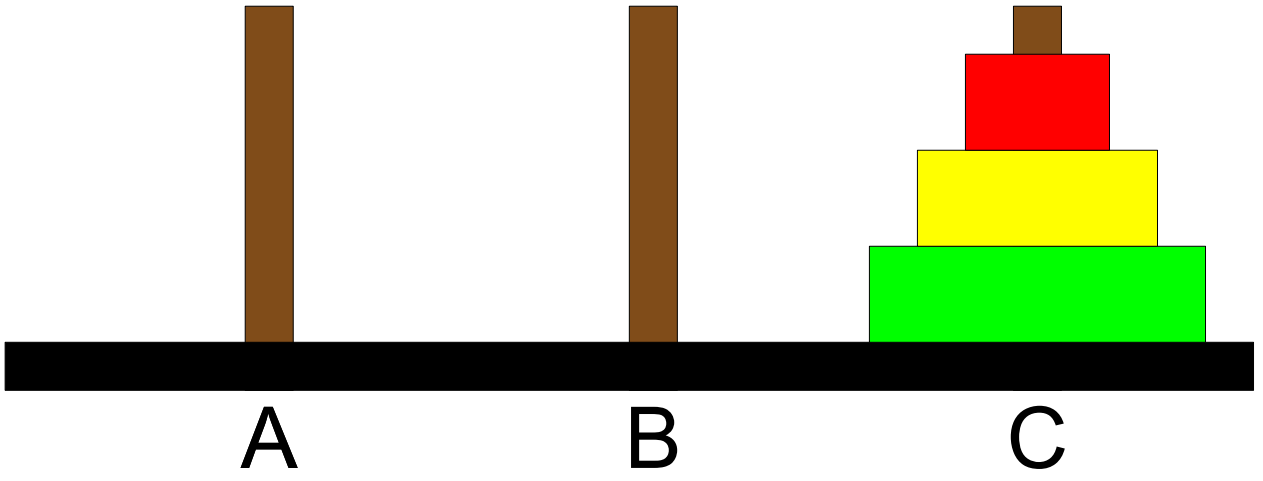
```
int main() {
}
```

```
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

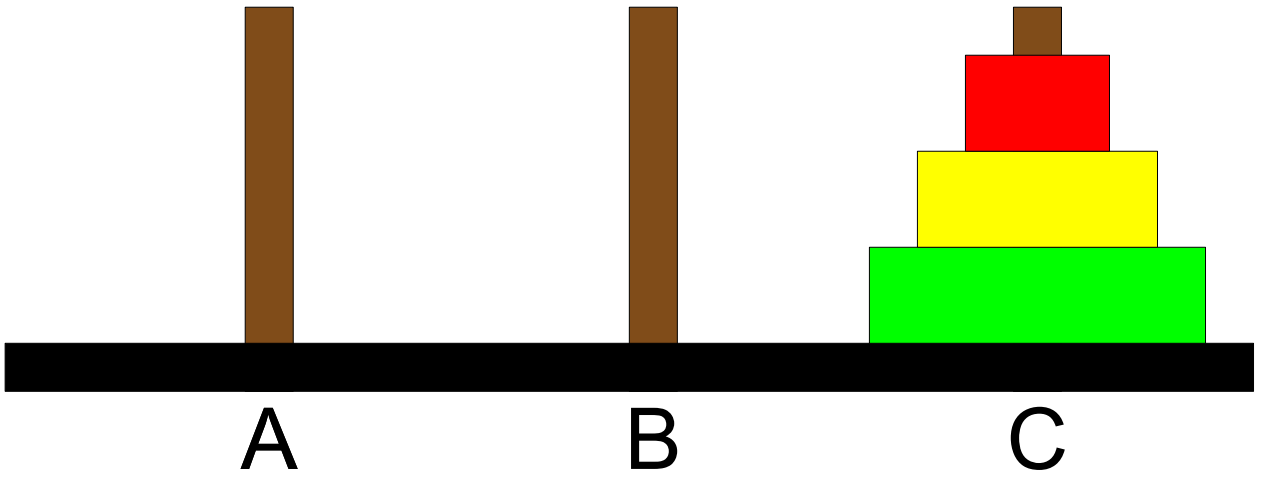


```
int main() {

}
```

```
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);
    }
}
```

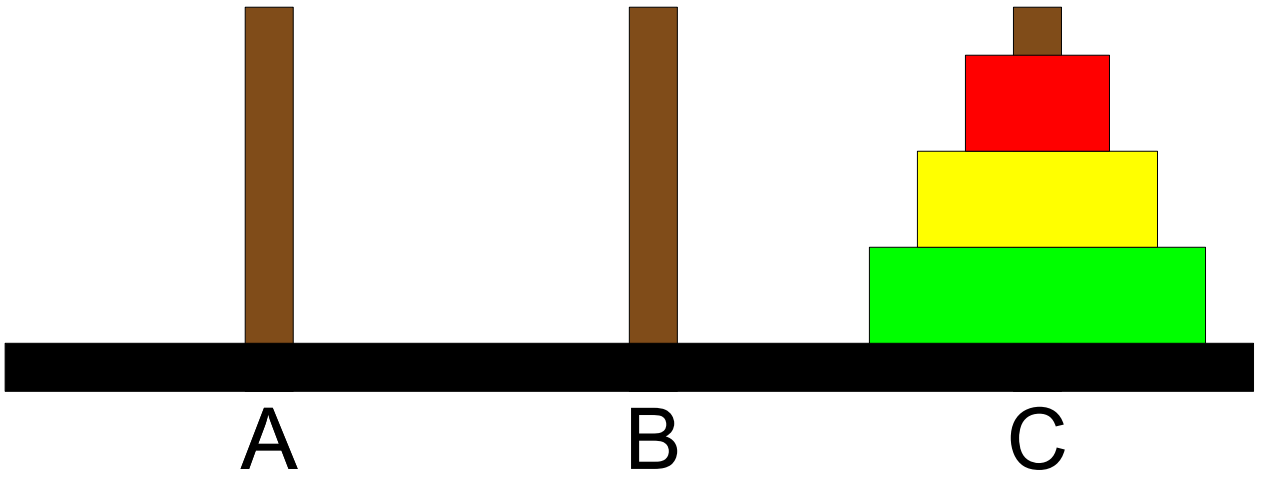
n	2	from	b	to	c	temp	a



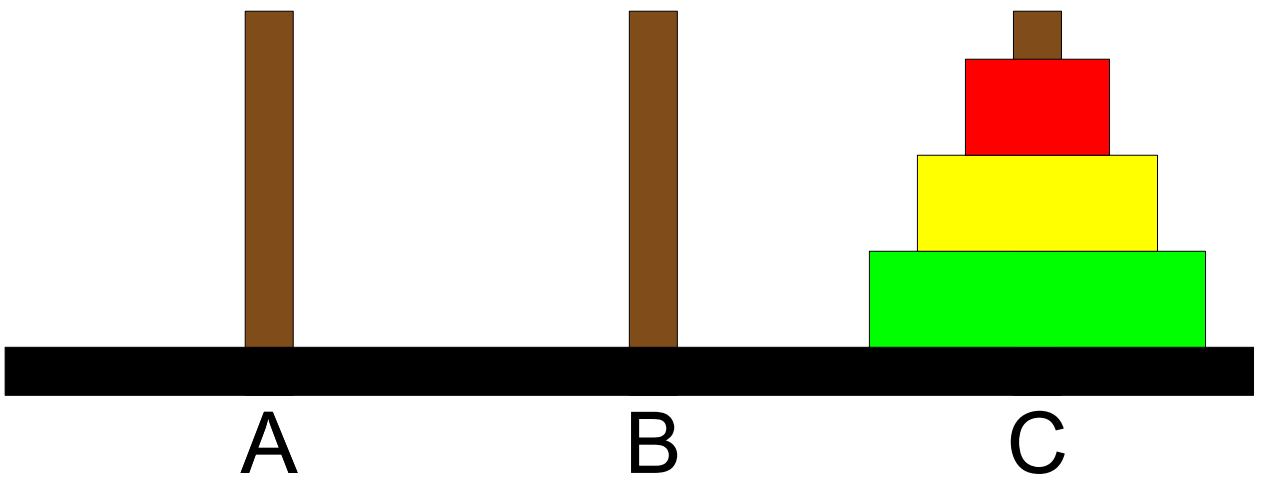
```
int main() {

}
```

```
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');
}
```



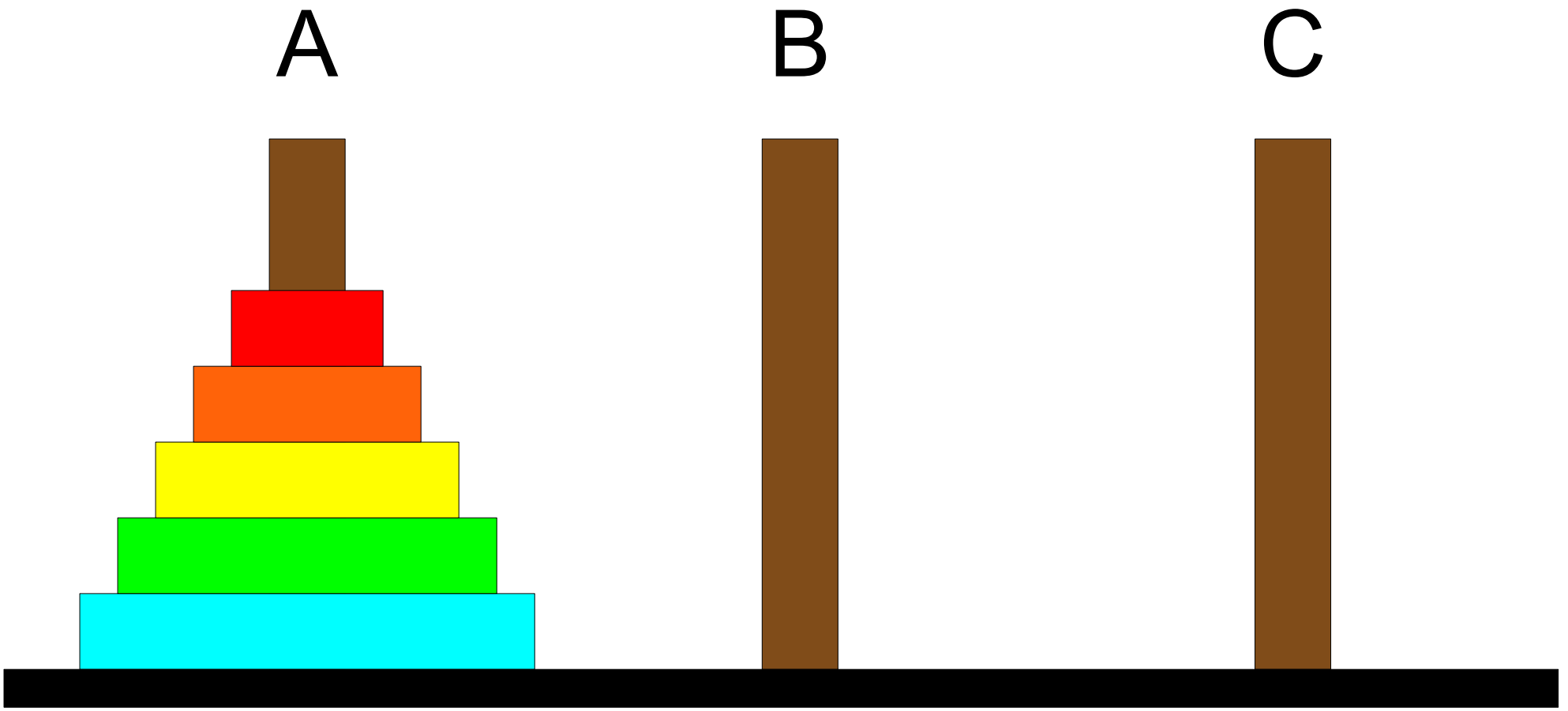
Why Recursion can be Challenging

- In most iterative programs you can “think through” what the program will do in your head
  - Effectively you're simulating the program in your brain.
- It can be much more difficult to do this with recursive programs.
- Question: In Tower of Hanoi, how many times will `moveTower()` be called?
  - The recursive decomposition of `moveTower`() calls `moveTower`() twice.
  - For a tower of height n, `moveTower()` will be called ≈ 2^n times!
    - No way you can simulate that many calls in your head

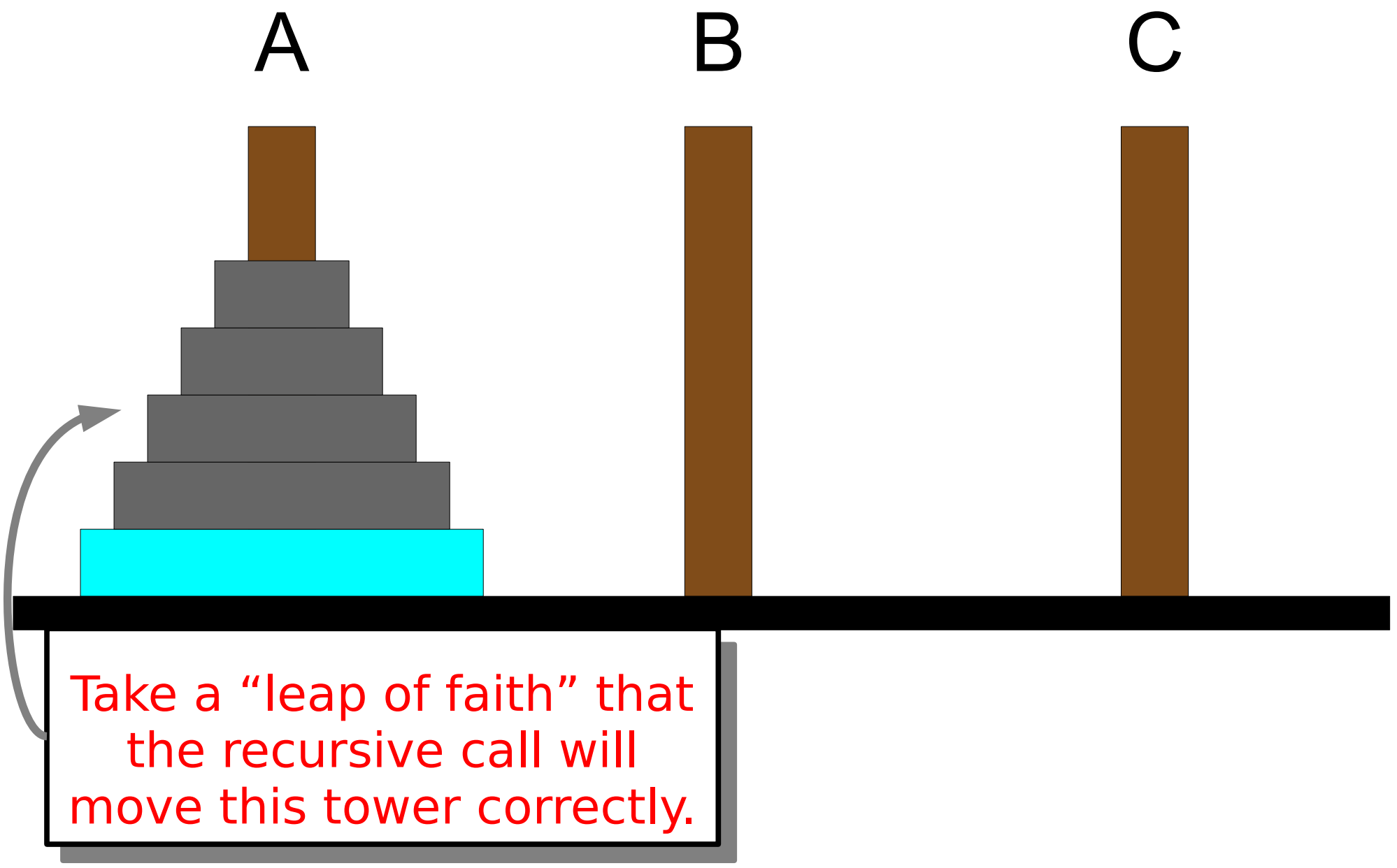
Writing Recursive Functions

- 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.

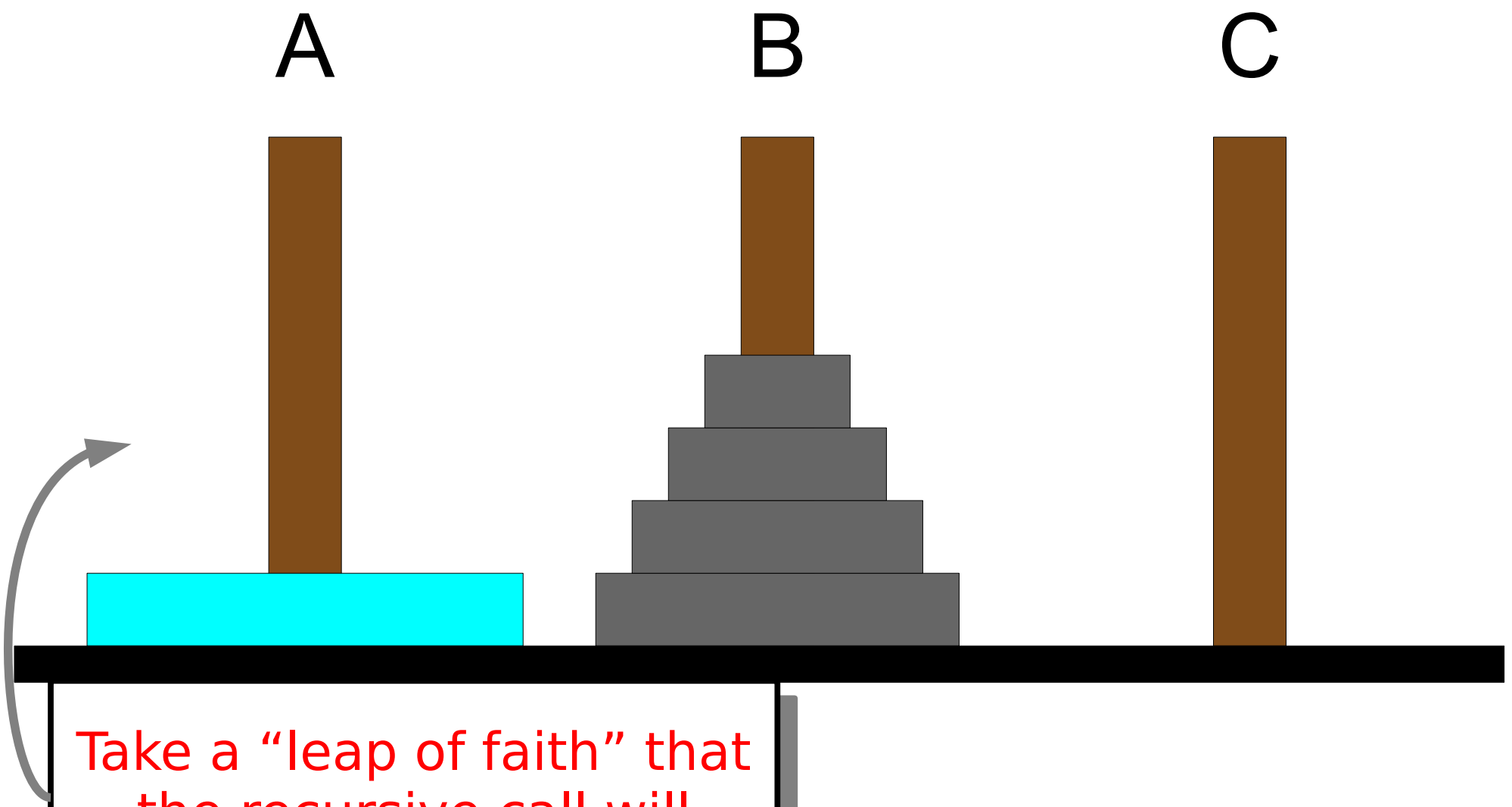
Recursive “Leap of Faith”



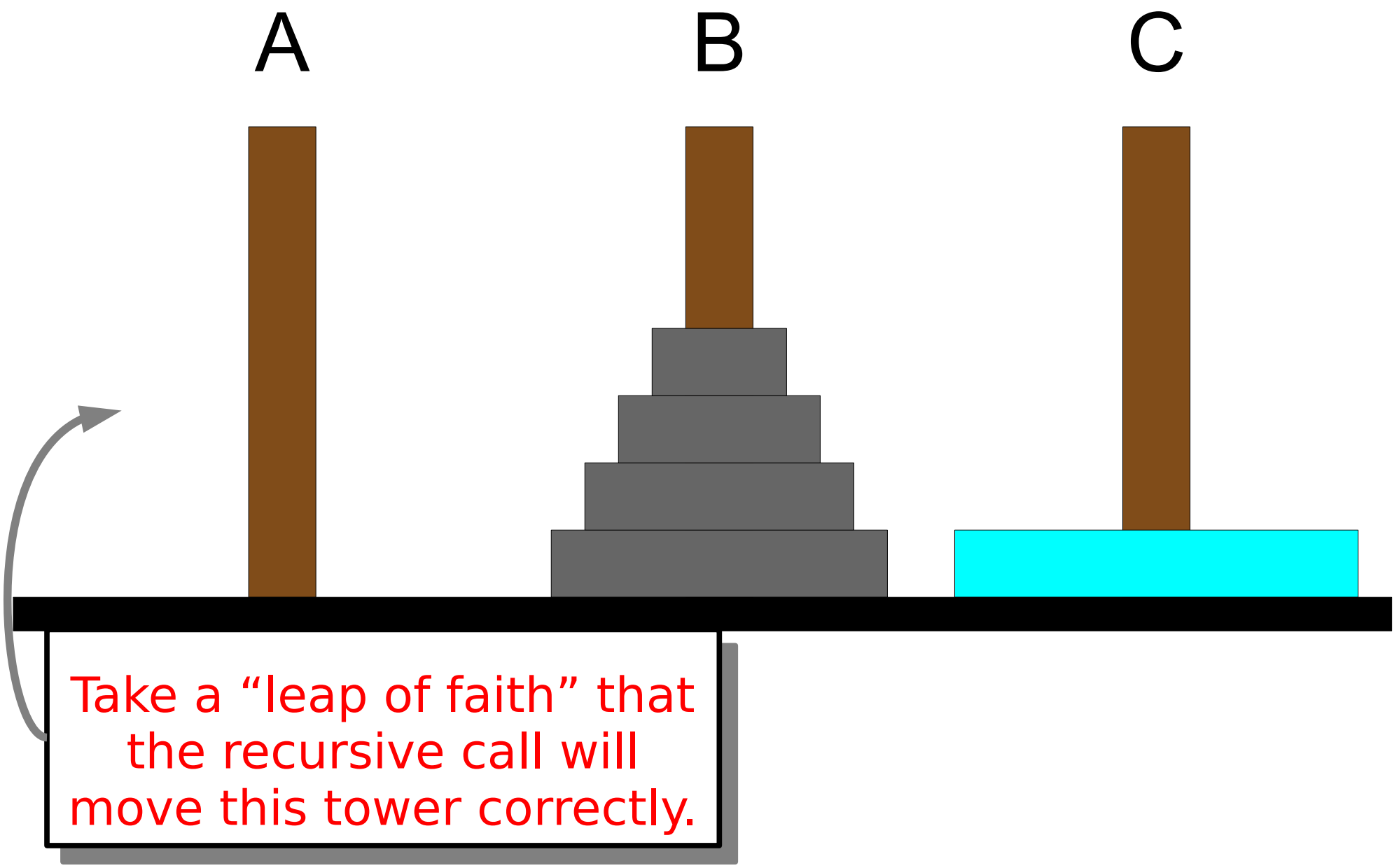
Recursive “Leap of Faith”



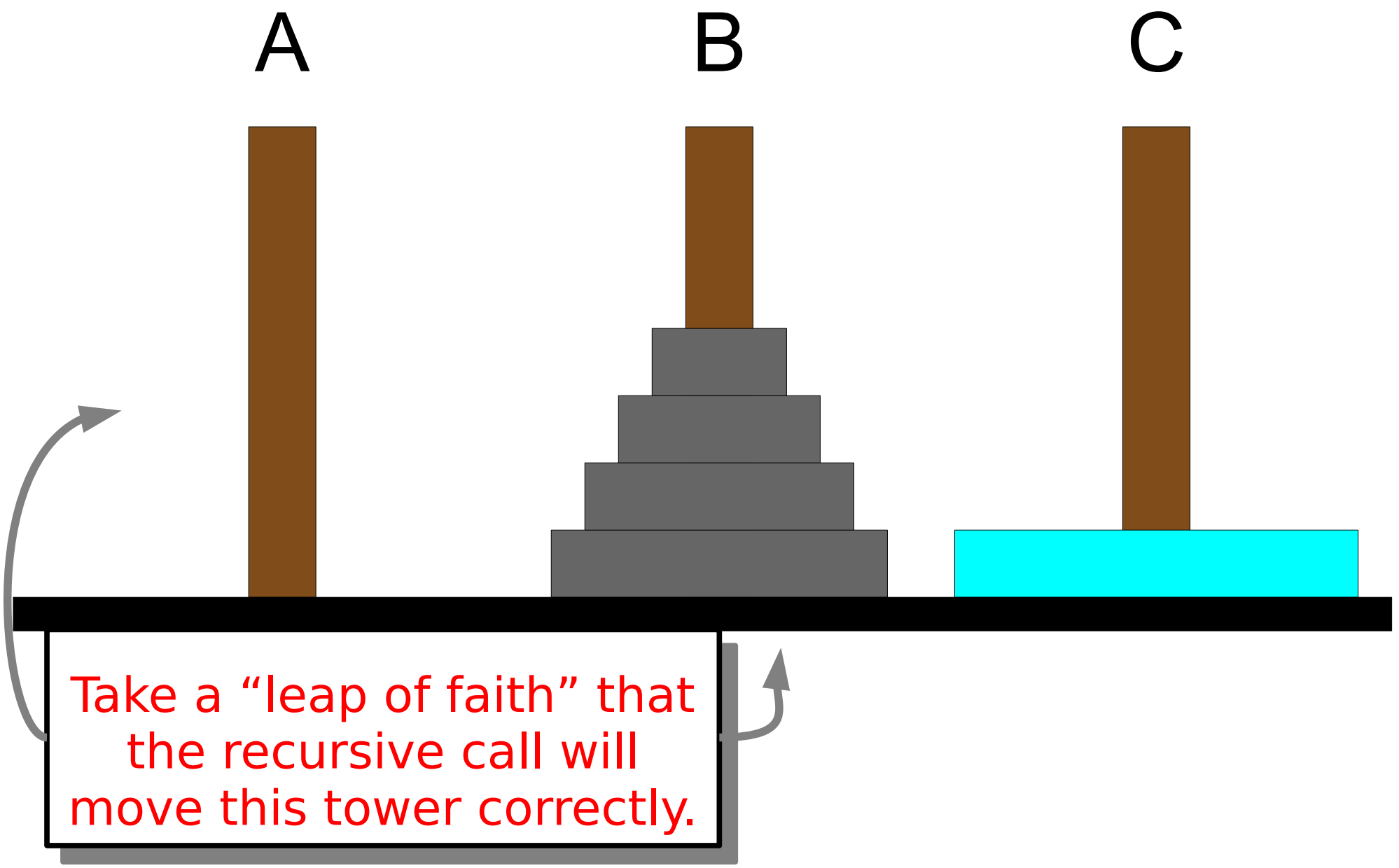
Recursive “Leap of Faith”



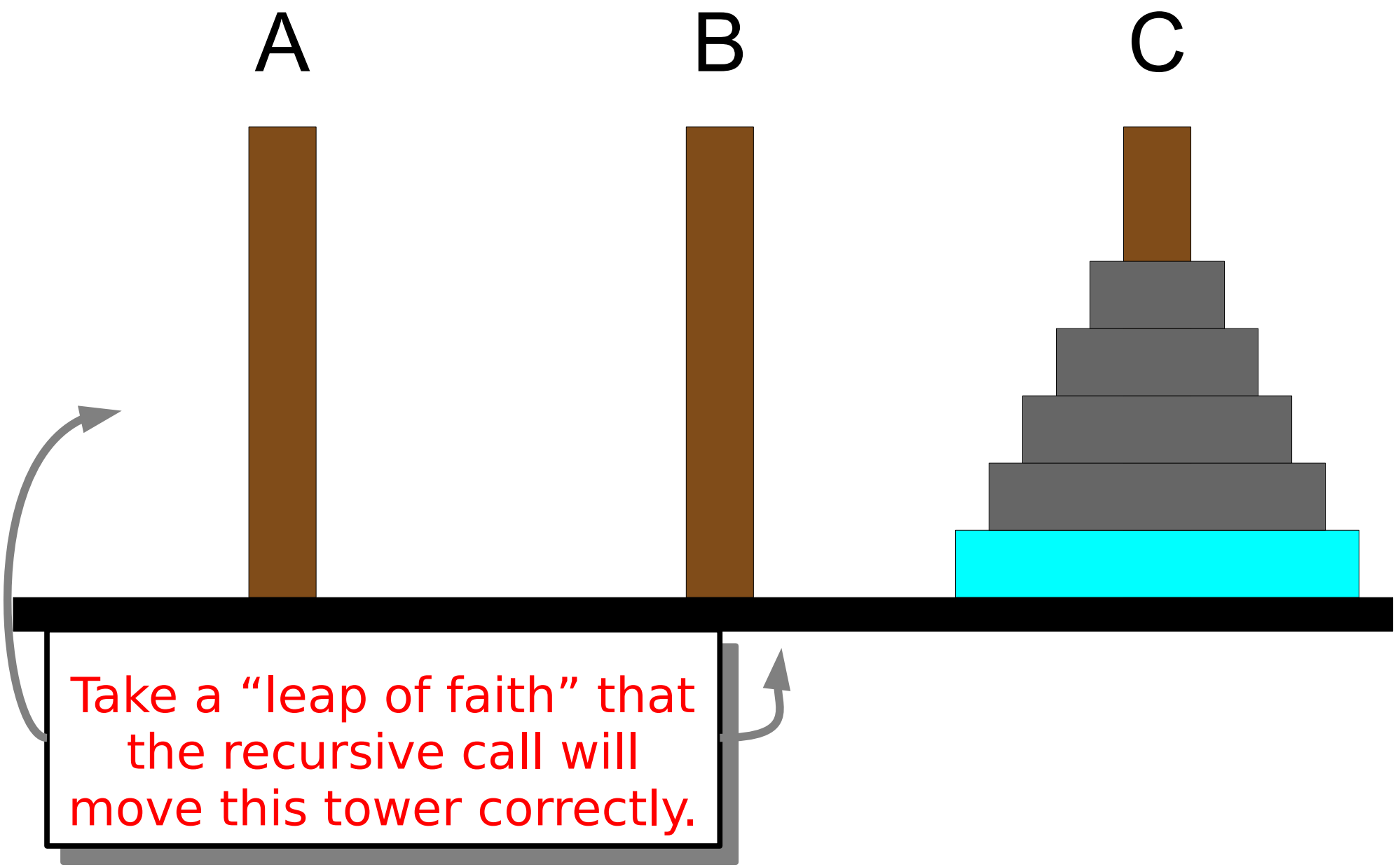
Recursive “Leap of Faith”



Recursive “Leap of Faith”



Recursive “Leap of Faith”



Writing Recursive Functions

- Another way of putting this: when writing recursive functions it can be very helpful to **assume** that the recursive call will do the right thing.
- This is helpful because it's very hard to think through an **exponential** number of recursive calls.

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.

```
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);
    }
}
```

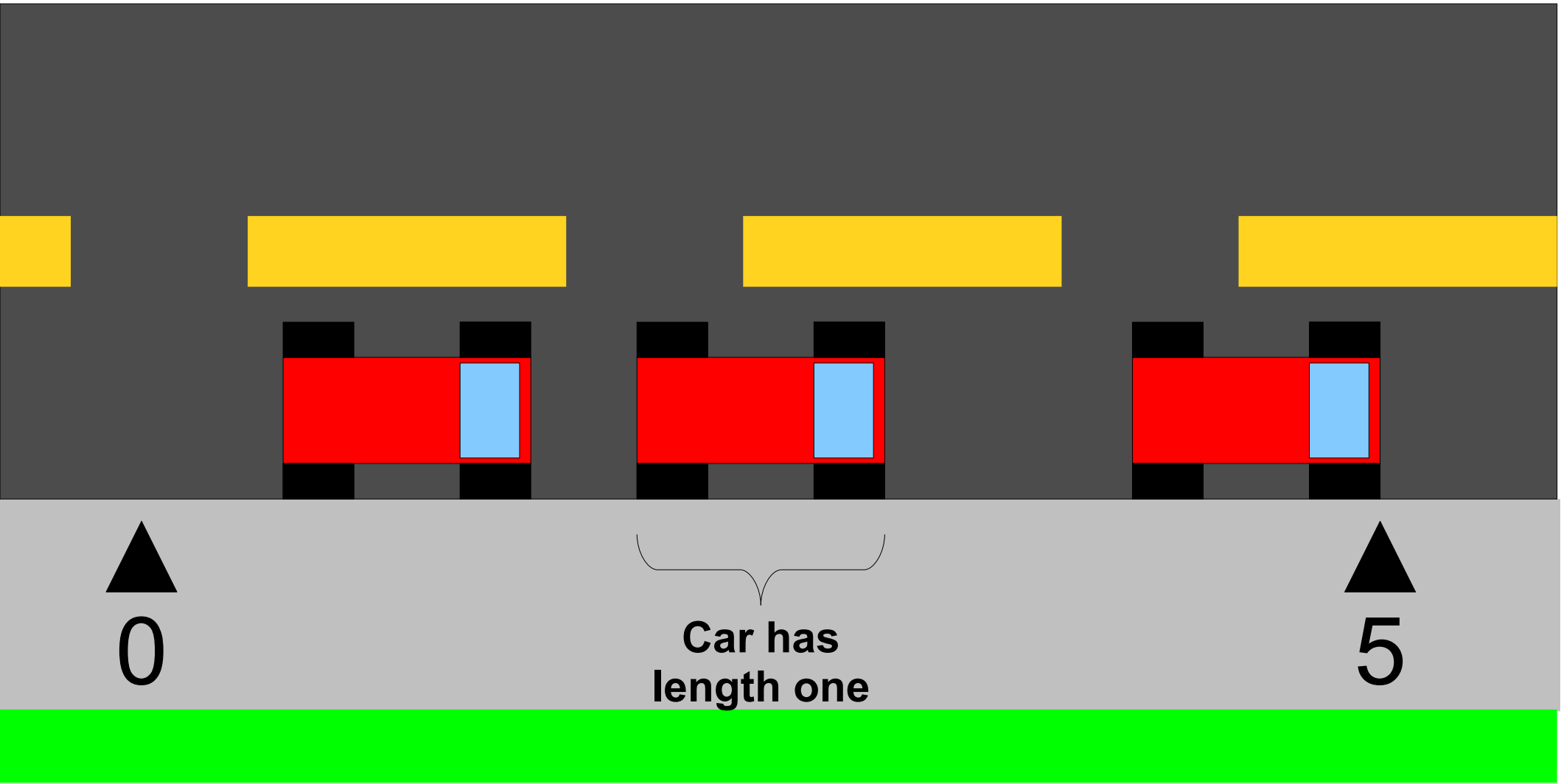
```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
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);
    }
}
```

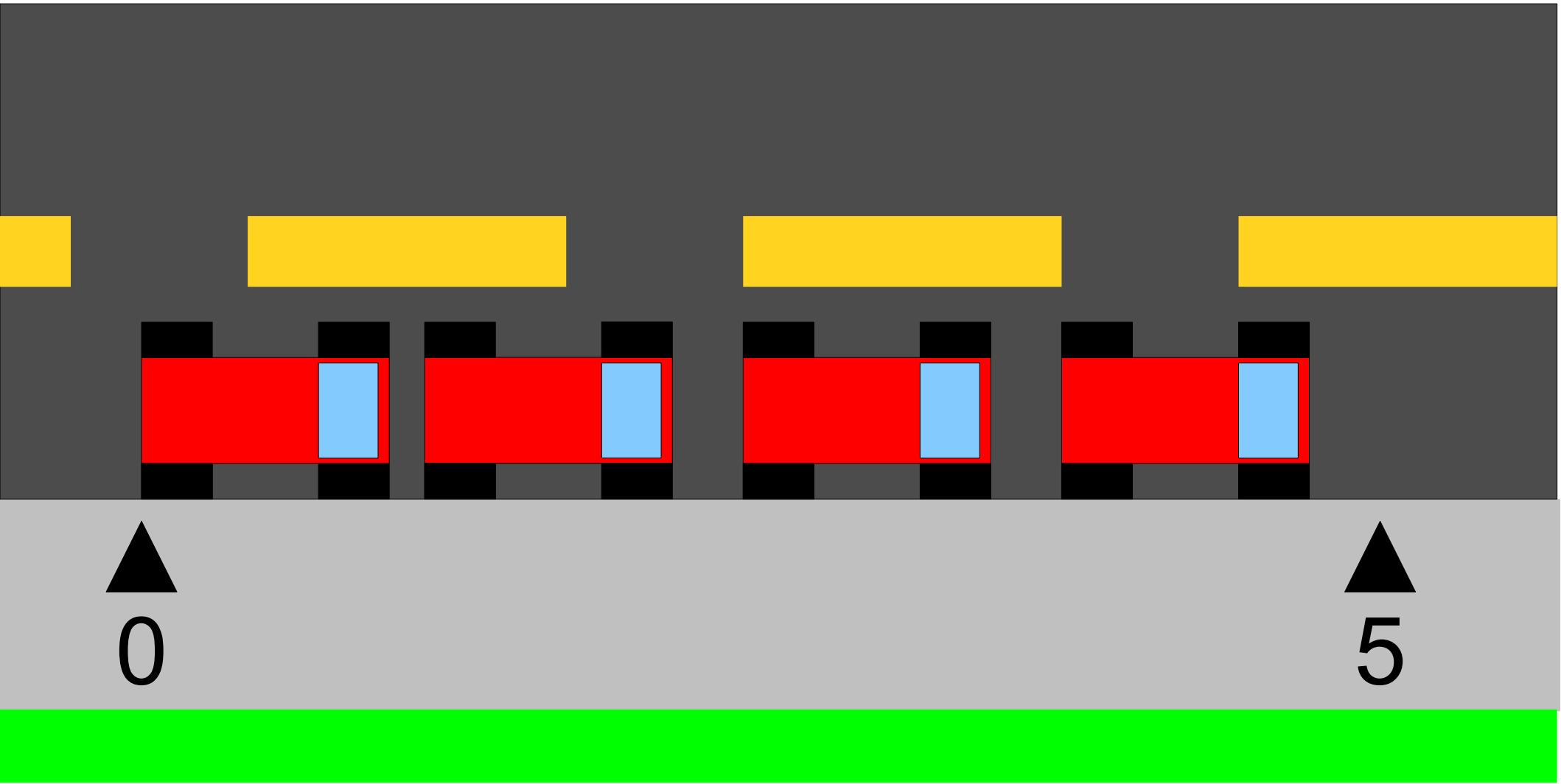
```
void moveTower(int n, char from, char to, char temp) {
    if (n != 0) {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
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);
    }
}
```

Parking Randomly



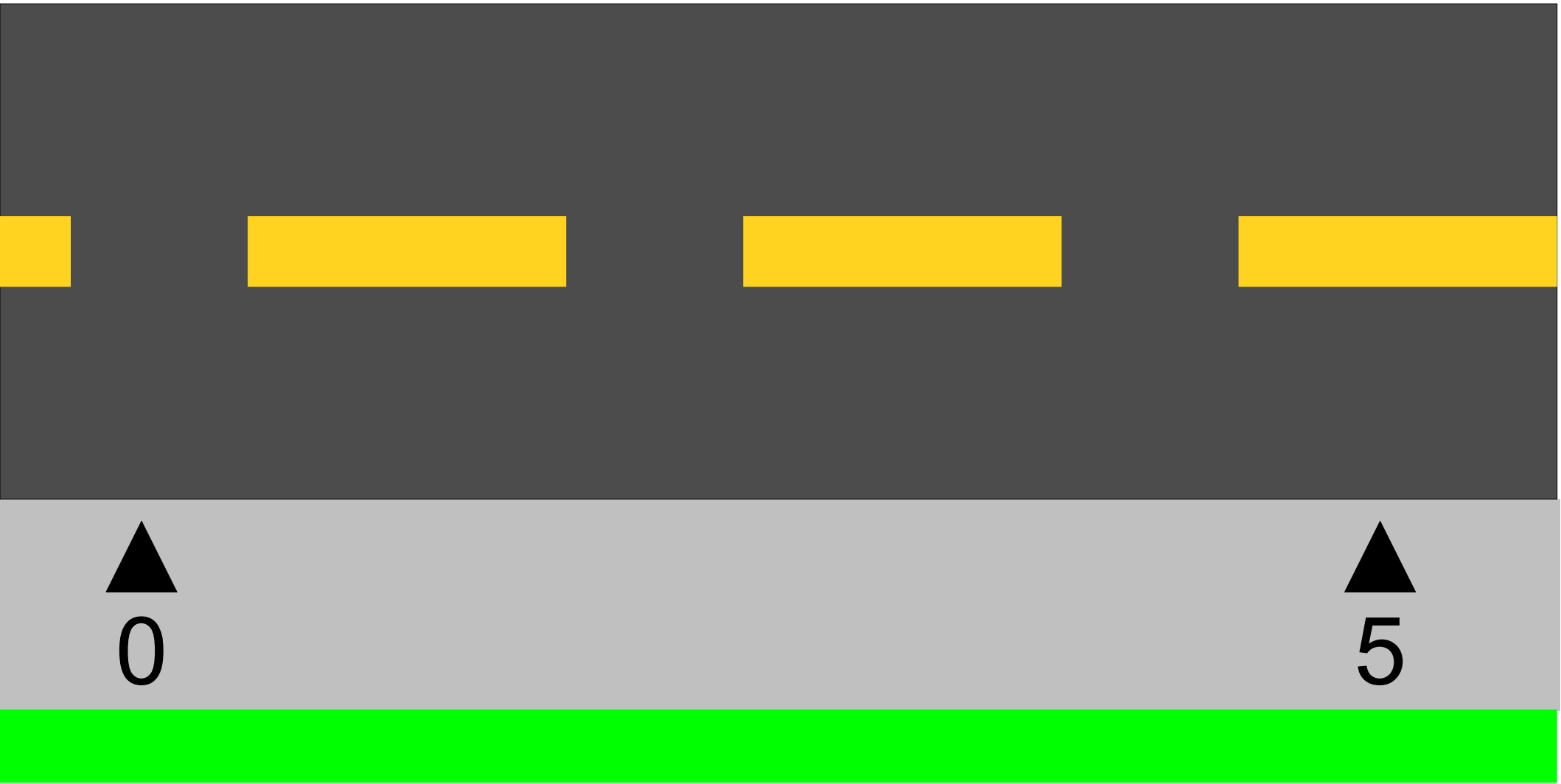
Parking Randomly



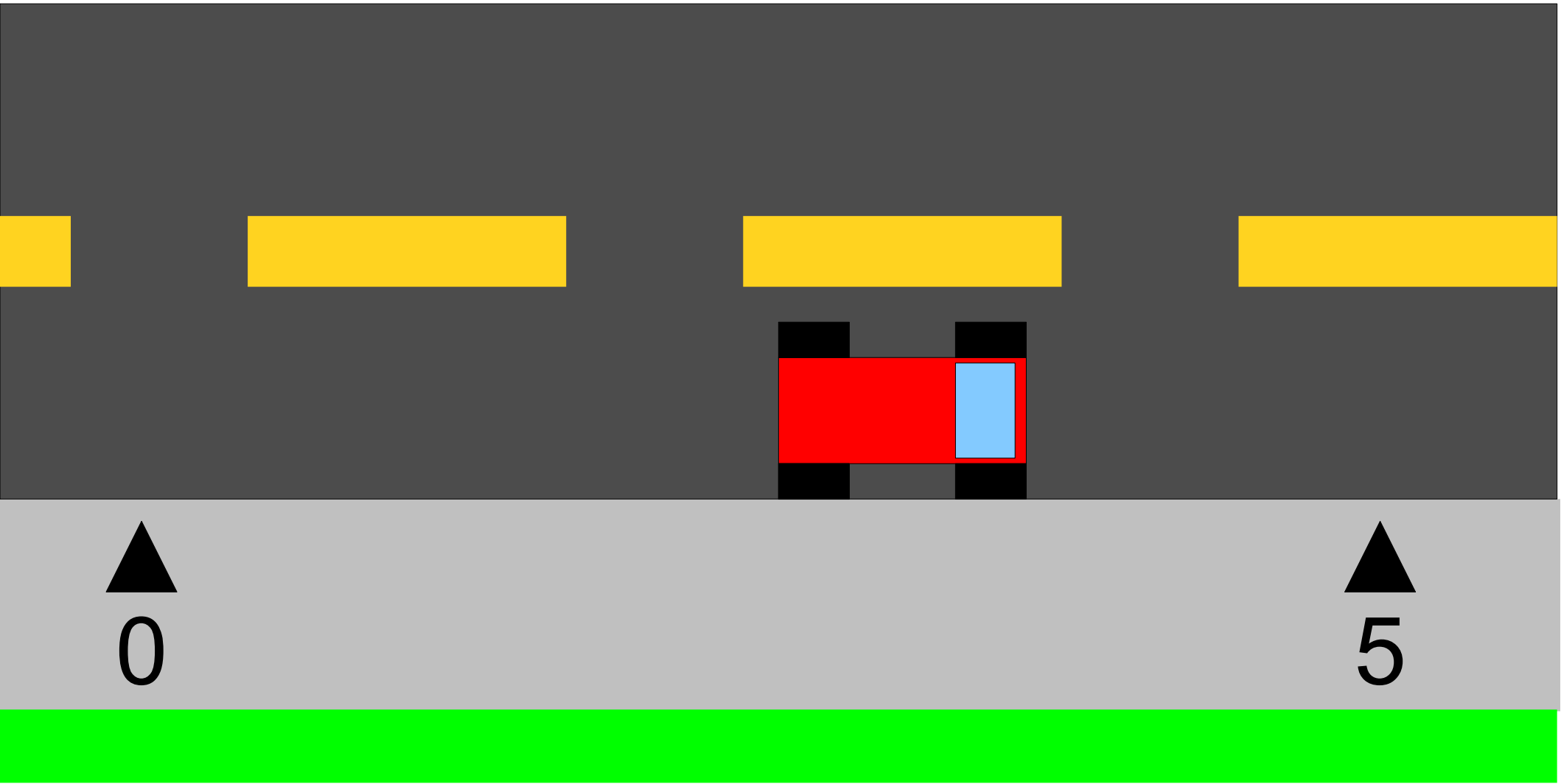
Parking Randomly

- 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?

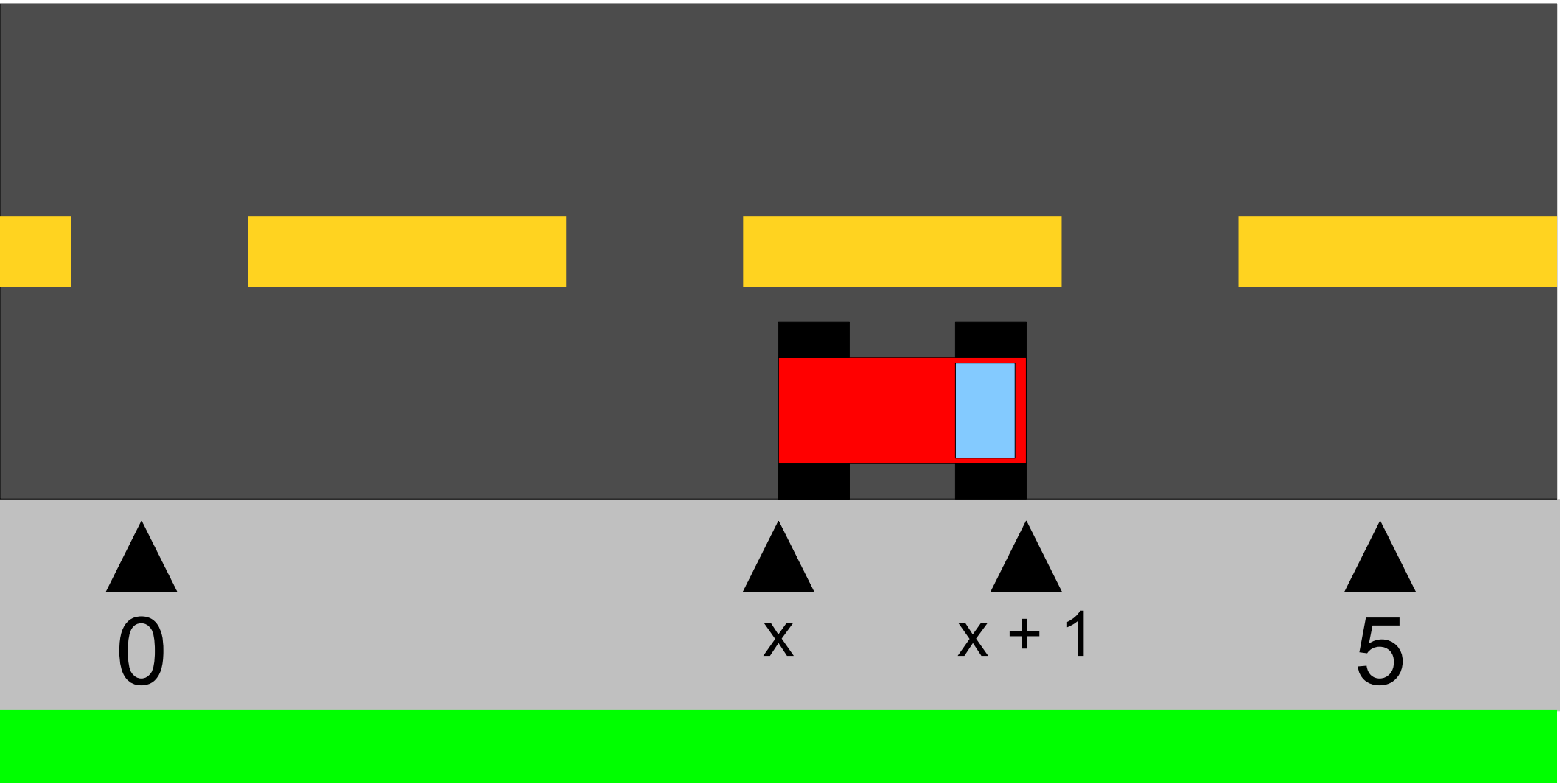
Parking Randomly



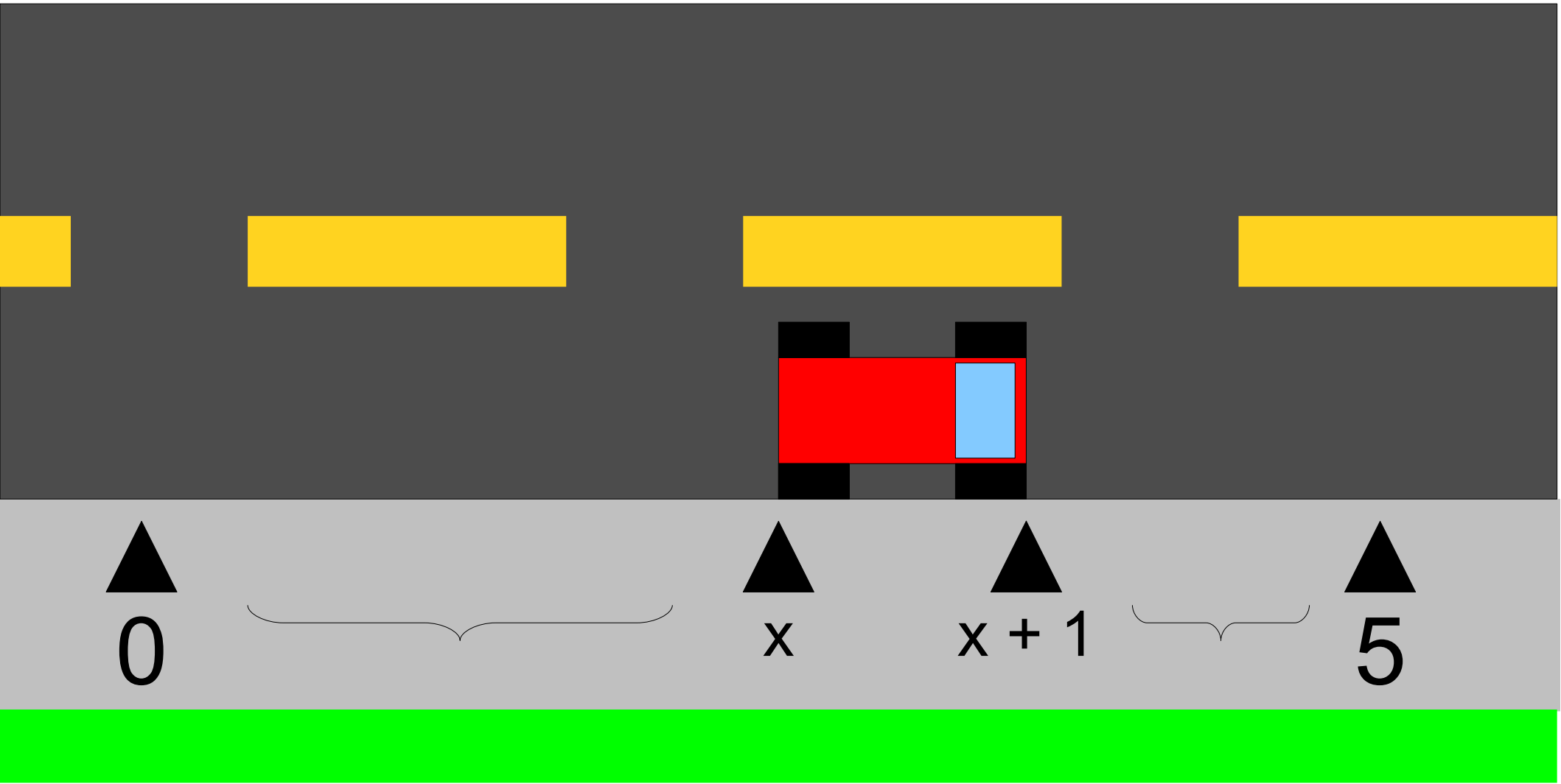
Parking Randomly



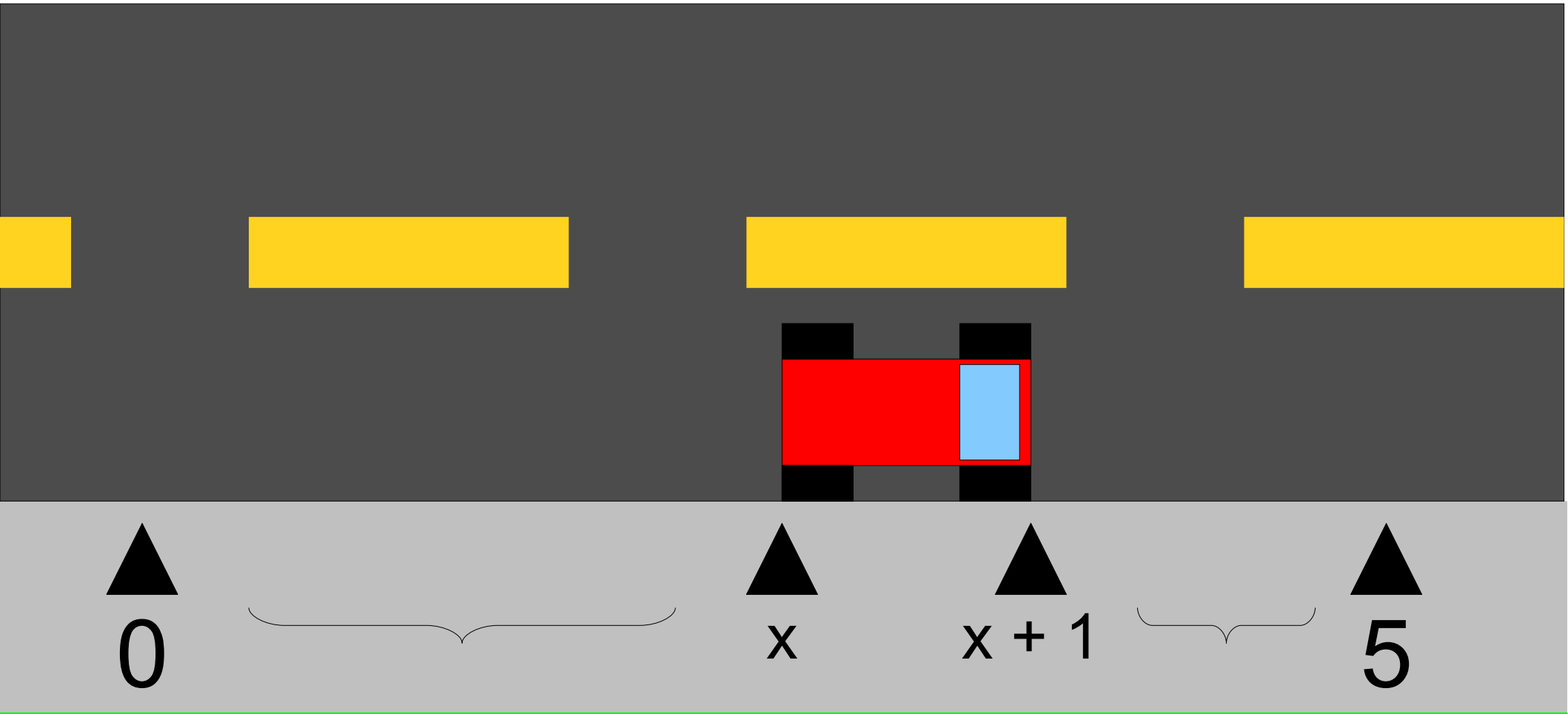
Parking Randomly



Parking Randomly



Parking Randomly



Place cars randomly in these ranges!

`ParkingRandomly`
(Pseudocode)

`ParkingRandomly.cpp`
(Computer)

Parking Randomly Pseudocode

```
Input: low, high
If there isn't room for a car
    Return 0
“place a car” randomly
X = recurse on left
Y = recurse on right
Return 1 + X + Y
```

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... 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.
  - Spoiler: Fast sorting algorithms

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.