Thinking Recursively

An Interesting Read

“How to Train Your Robot”

http://drtechniko.wordpress.com/2012/04/09/how-to-train-your-robot/

Teaching kids to program by having them program their parents!

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));
   }
}
```

An Interesting Listen

This American Life:
"Take The Money and Run For Office"

http://www.thisamericanlife.org/radio-archives/episode/461/take-the-money-and-run-for-office

Federal Campaign Limits

	To each candidate or candidate committee per election	To national party committee per calendar year	To state, district & local party committee per calendar year	To any other political committee per calendar year[1]	Special Limits
Individual may give	$2,500*	$30,800*	$10,000 (combined limit)	$5,000	$117,000* overall biennial limit: • $46,200* to all candidates • $70,800* to all PACs and parties[2]

Source: **http://www.fec.gov/pages/brochures/contriblimits.shtml**

Federal Campaign Limits

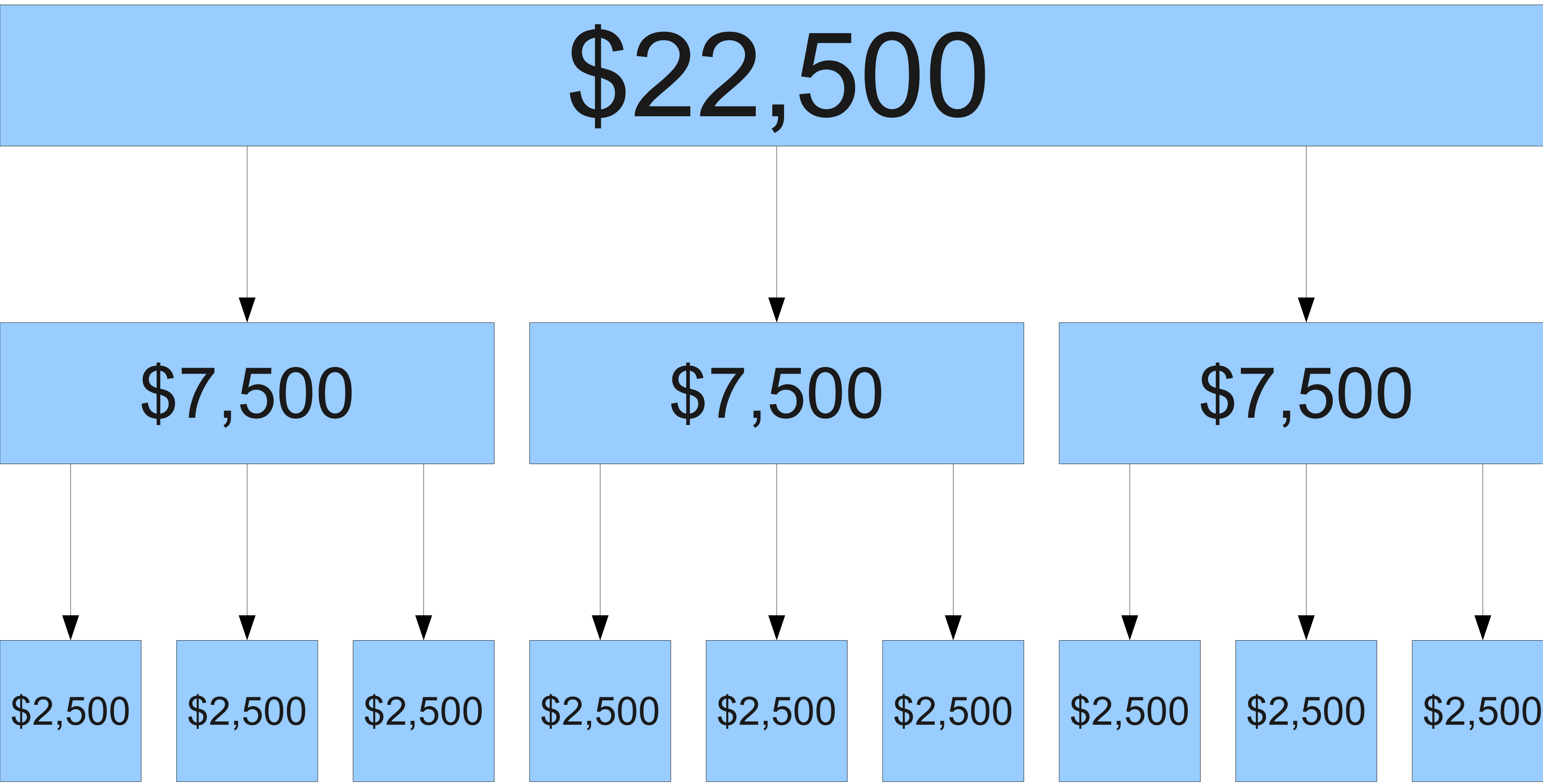
	To each candidate or candidate committee per election	To national party committee per calendar year	To state, district & local party committee per calendar year	To any other political committee per calendar year[1]	Special Limits
Individual may give	$2,500*	$30,800*	$10,000 (combined limit)	$5,000	$117,000* overall biennial limit: • $46,200* to all candidates • $70,800* to all PACs and parties[2]

Source: **http://www.fec.gov/pages/brochures/contriblimits.shtml**

Raising Money

- According to several sources (Bloomberg, the Wall Street Journal, Politico, etc.), the 2008 Presidential Election cost about $1.5 billion.
- How can you raise that much money from private donors?

Raising Money



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.
}
```

Recursive Problem-Solving

```
if (n is at most $2,500) {
    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.
}
```

Recursive Problem-Solving

```
if (n is at most $2,500) {
    Donate $n


} 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.
}
```

Recursive Problem-Solving

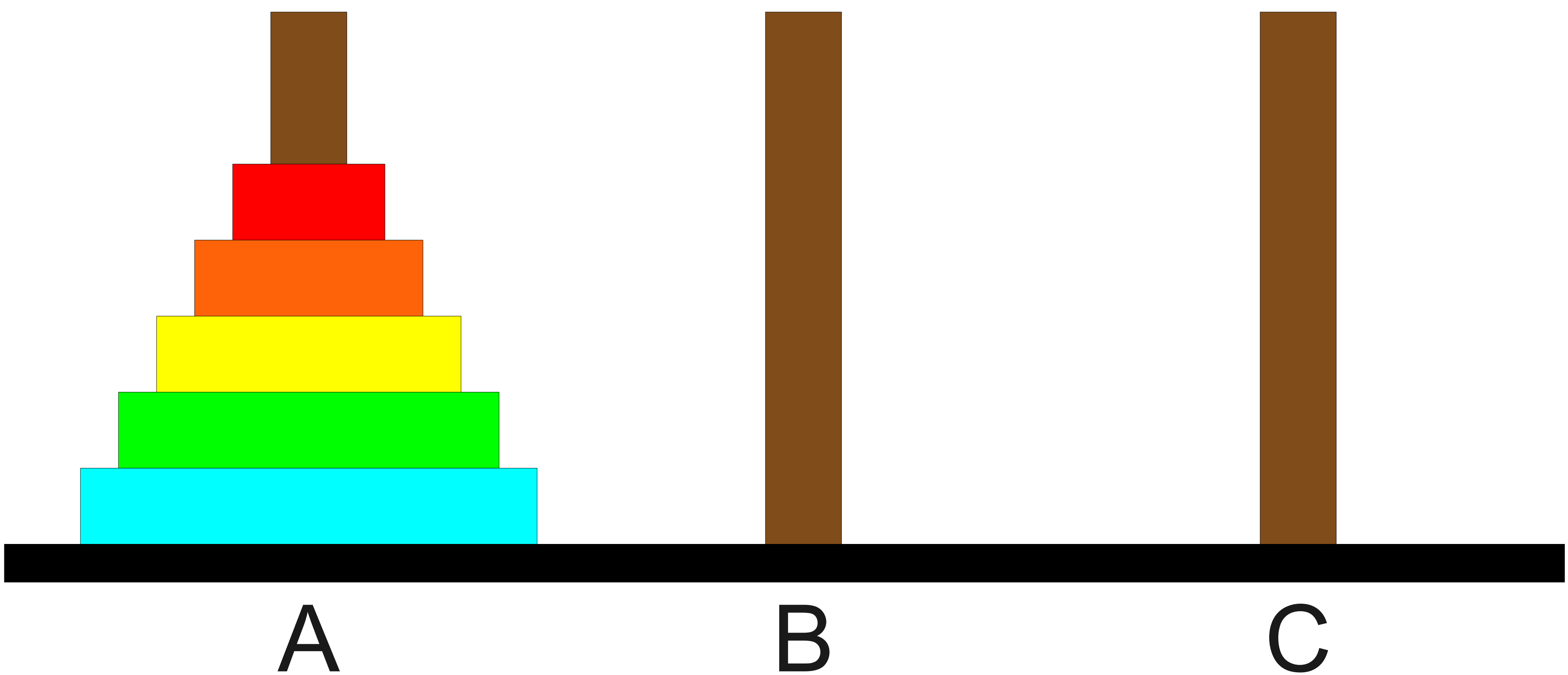
```
if (n is at most $2,500) {
    Donate $n

} else {
    Find three other people.
    Tell them each to raise $n / 3.

}
```

What's Going On?

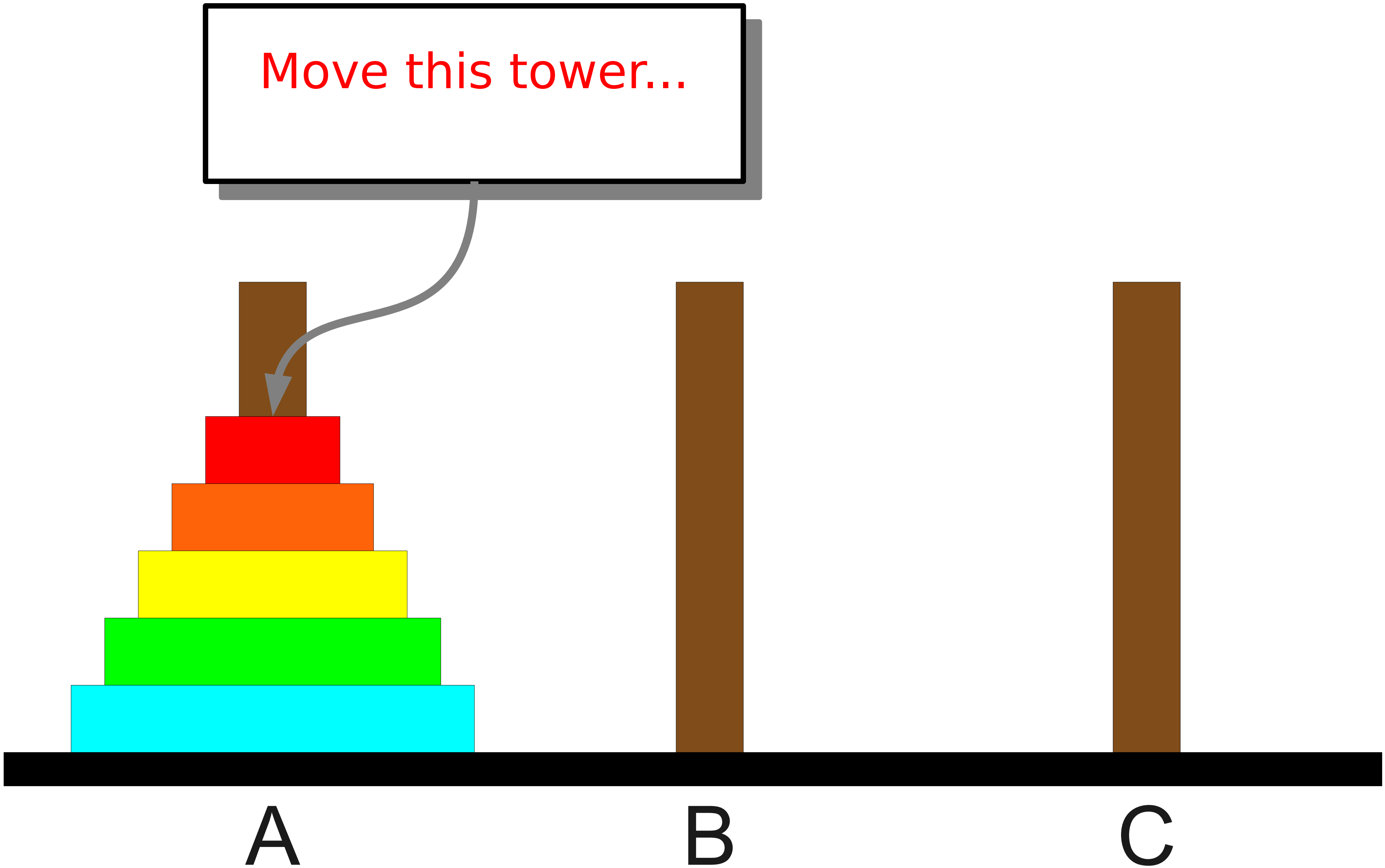
- Recursion solves a problem by continuously simplifying the problem until it becomes simple enough to be solved directly.
- The **recursive decomposition** makes the problem slightly simpler at each step.
- The **base case** is what ultimately makes the problem solvable – it guarantees that when the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

The Towers of Hanoi Problem

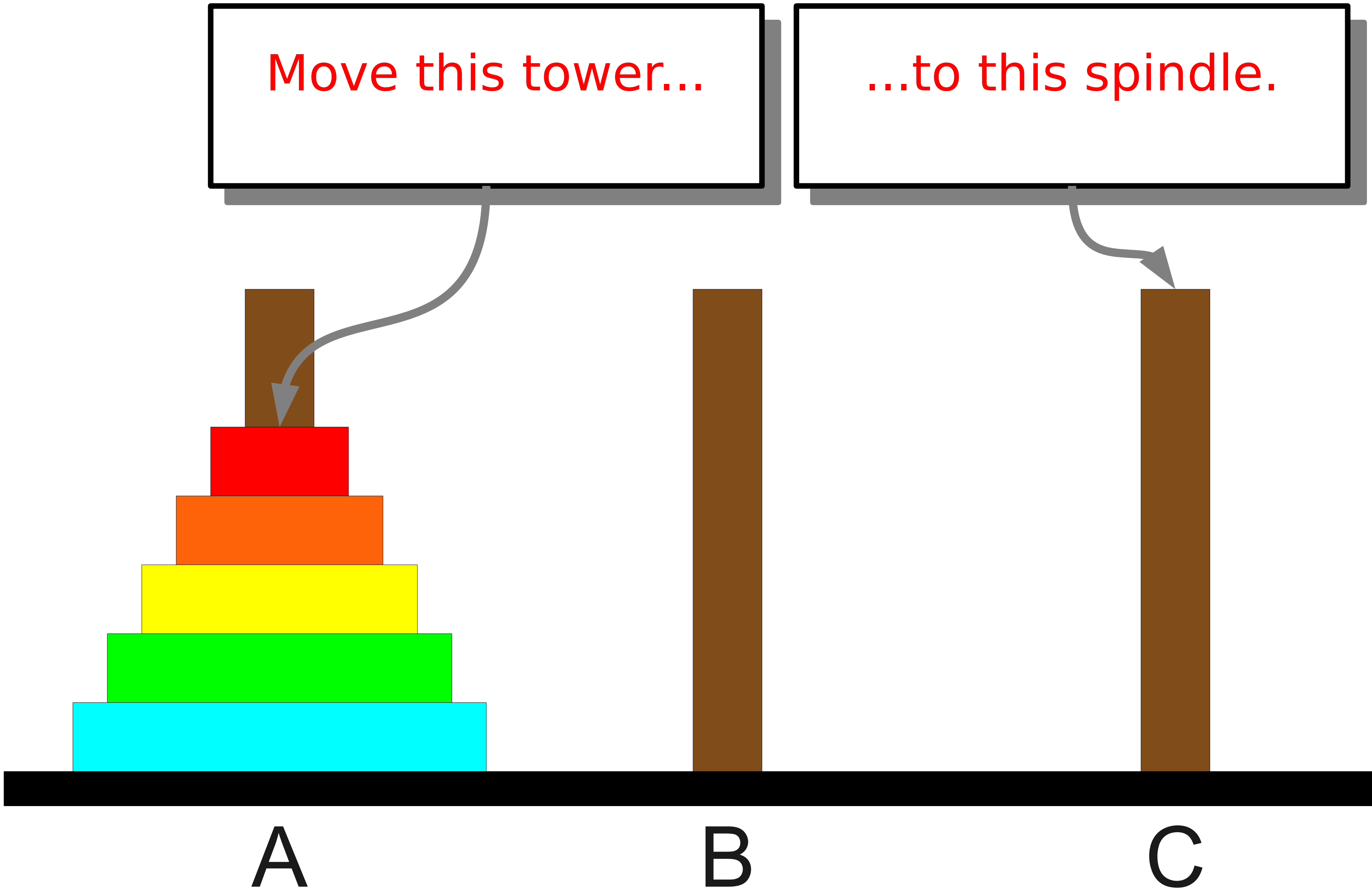
Towers of Hanoi



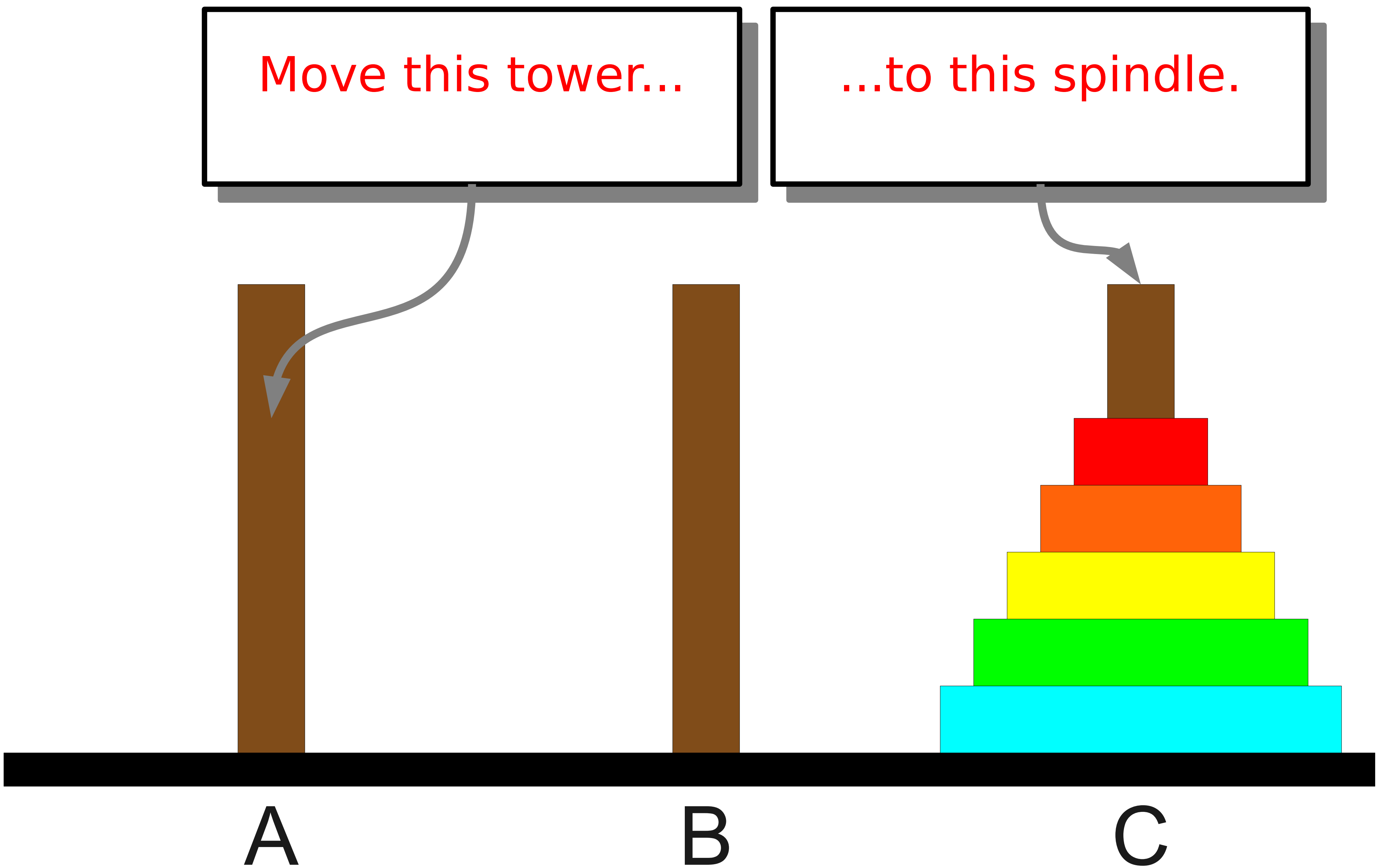
Towers of Hanoi



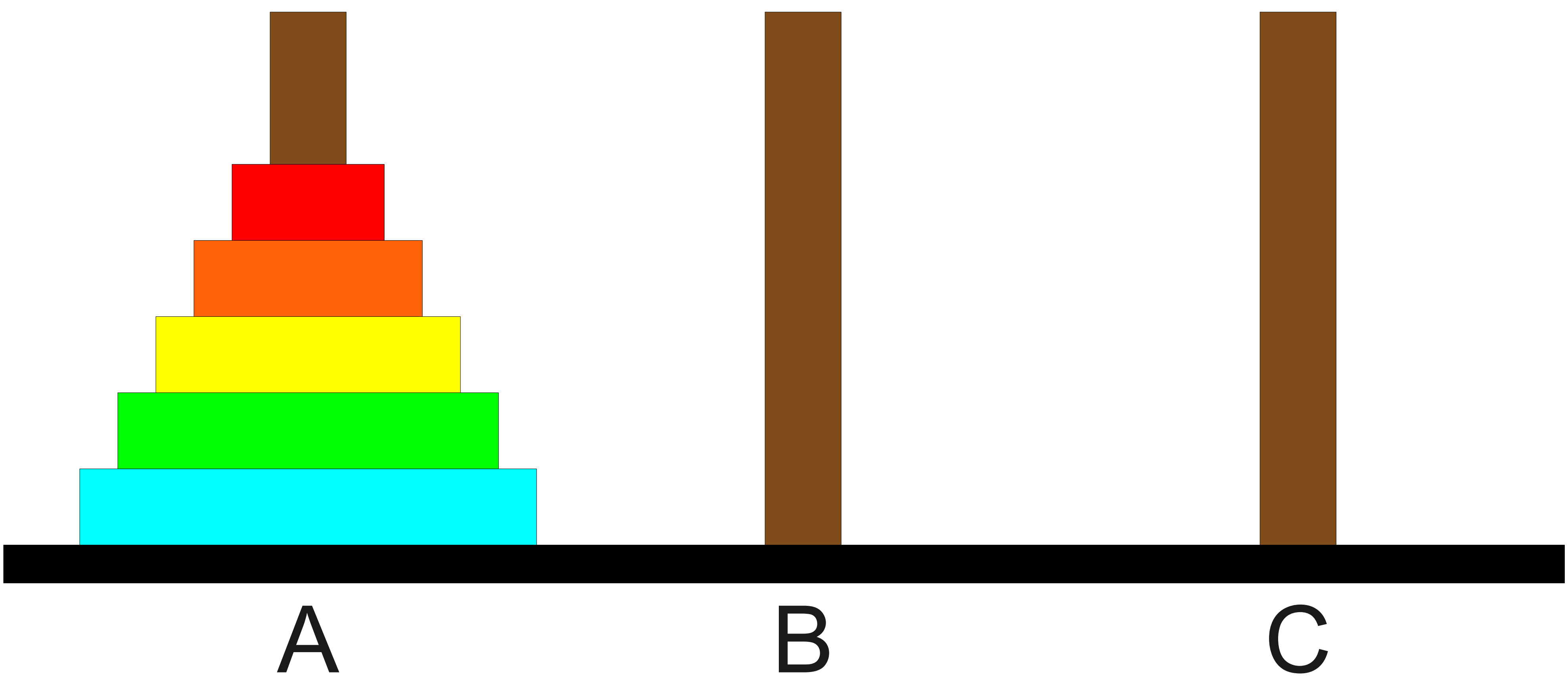
Towers of Hanoi



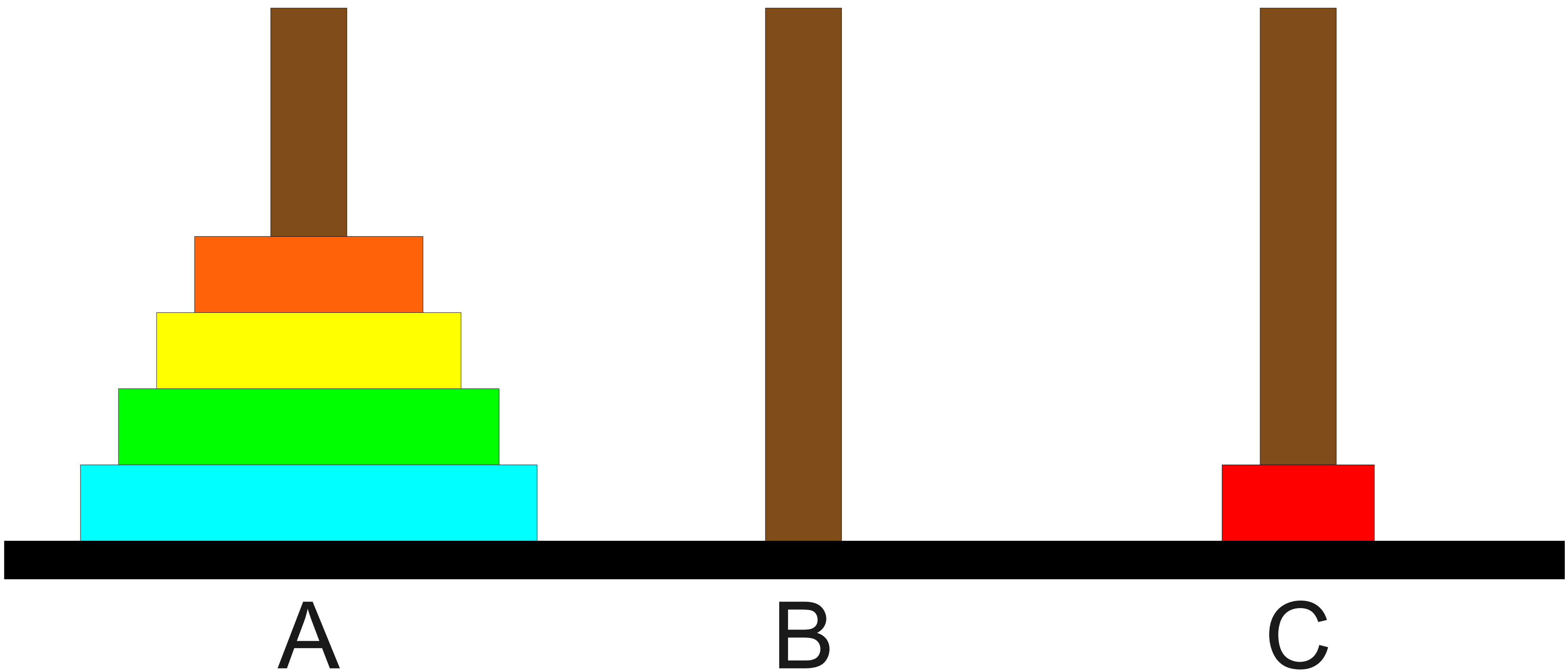
Towers of Hanoi



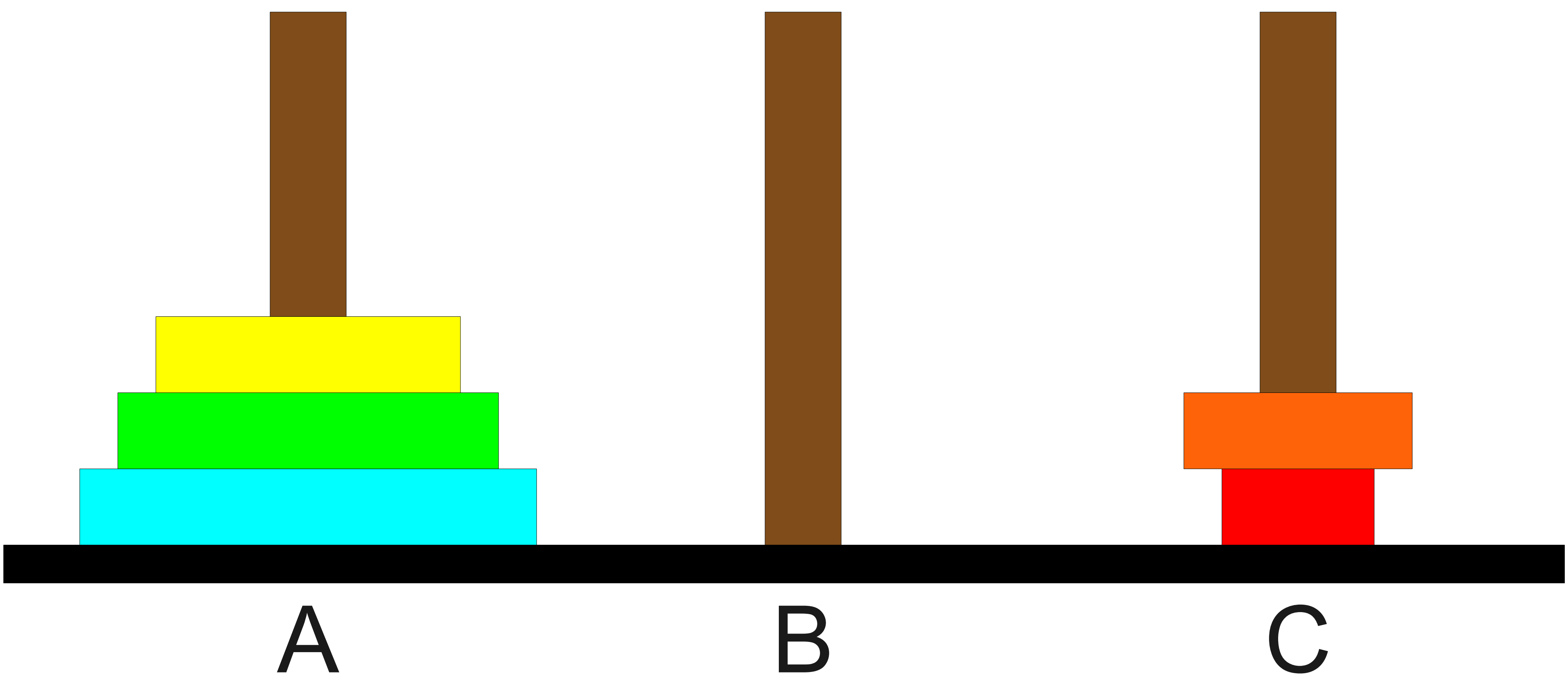
Towers of Hanoi



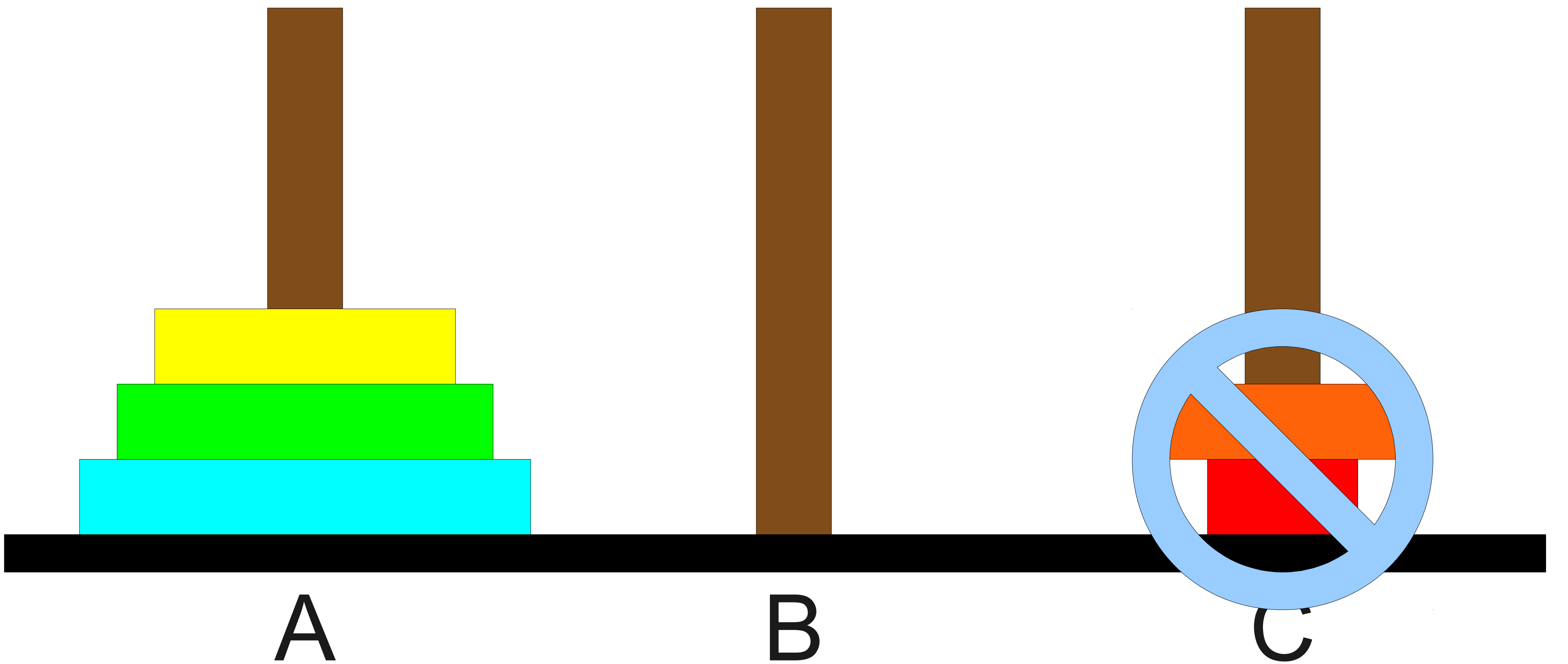
Towers of Hanoi



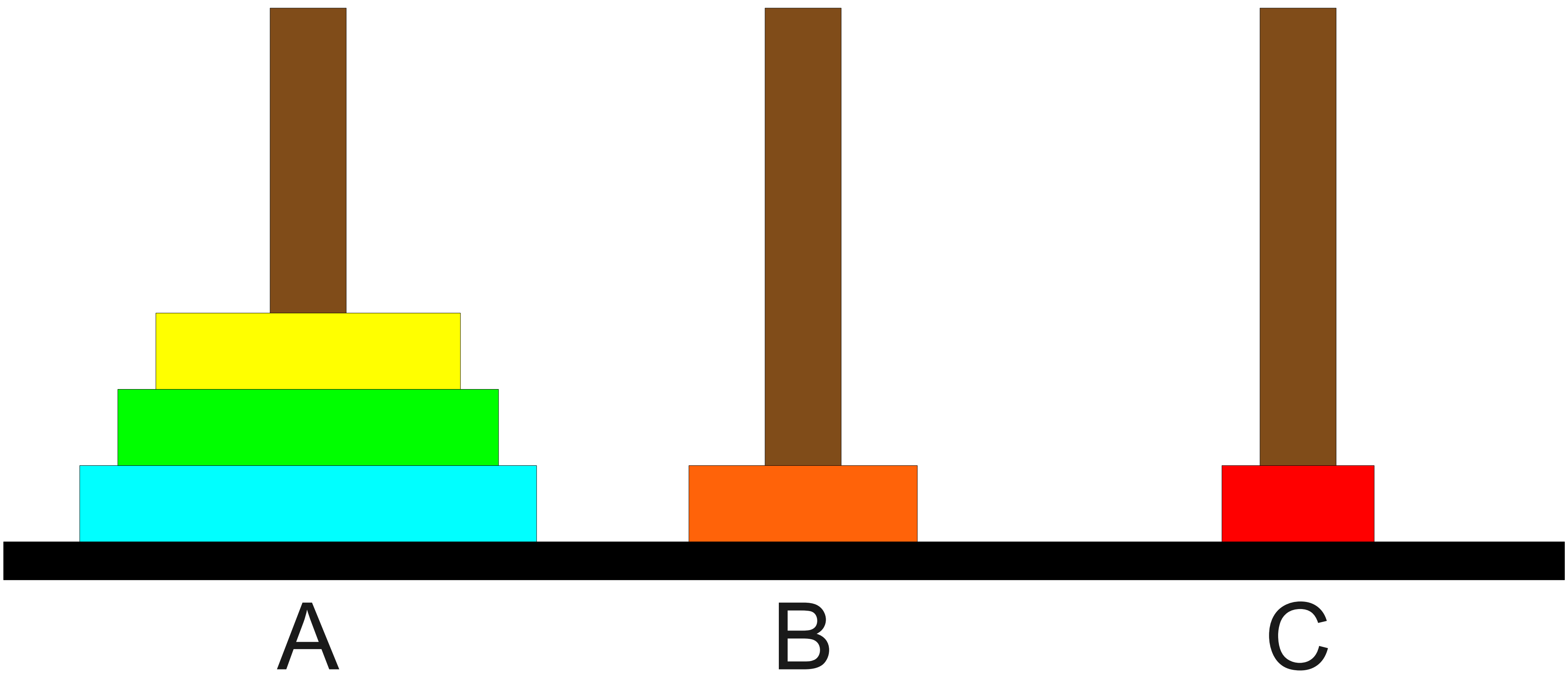
Towers of Hanoi



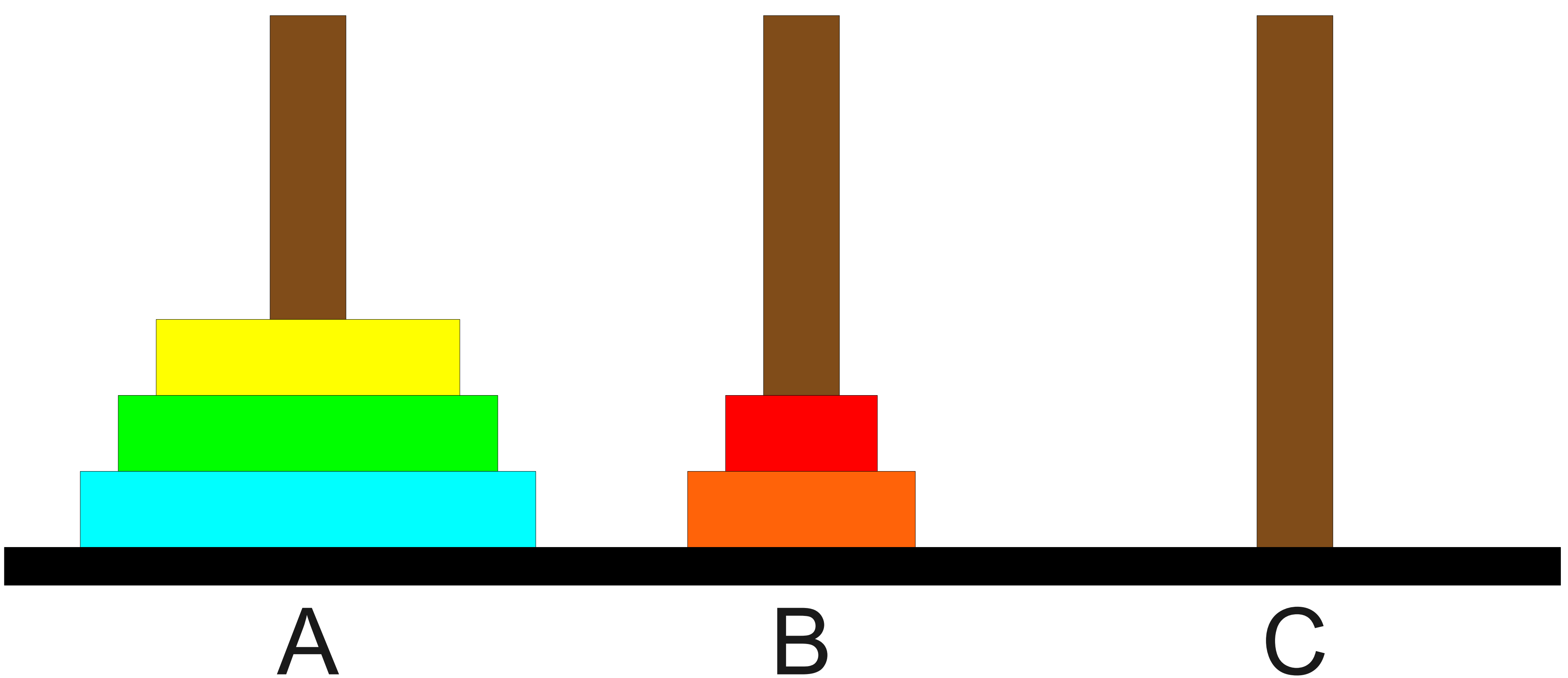
Towers of Hanoi



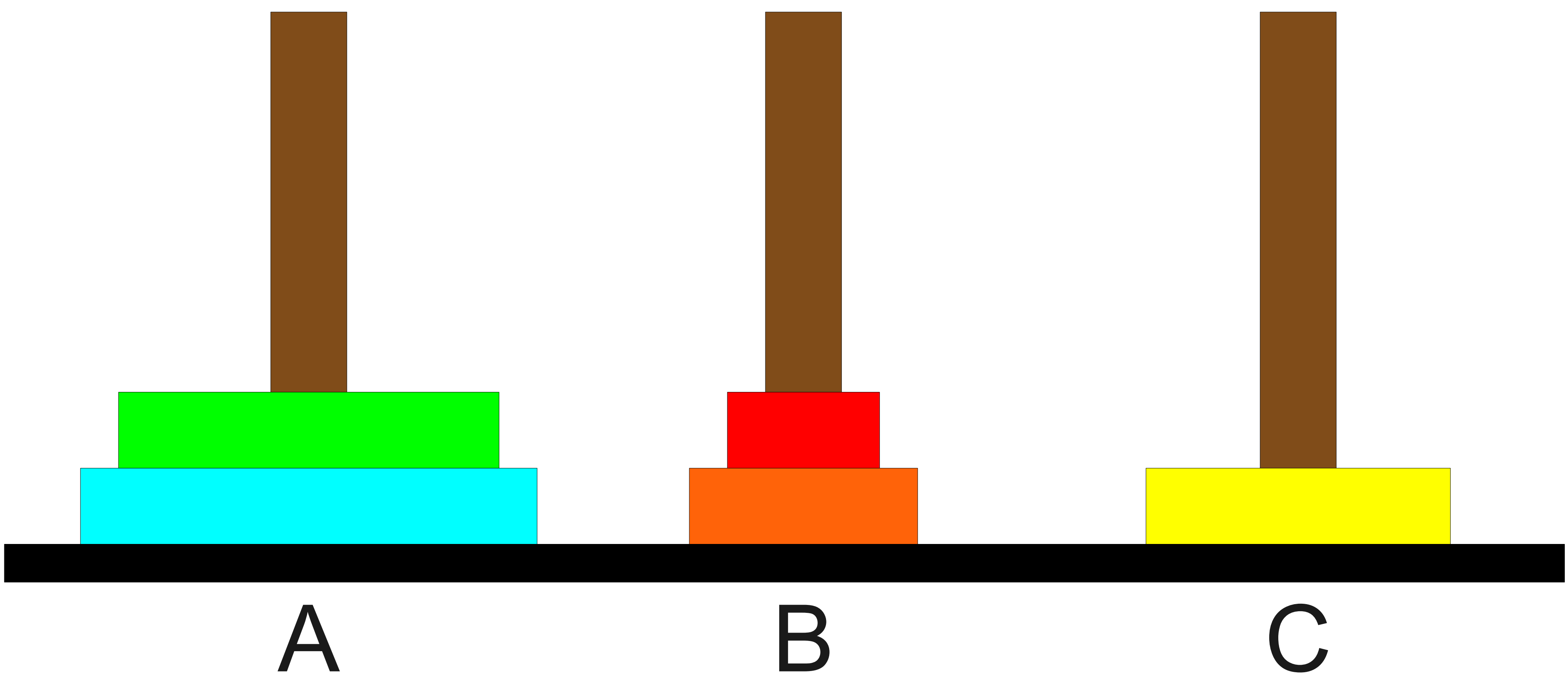
Towers of Hanoi



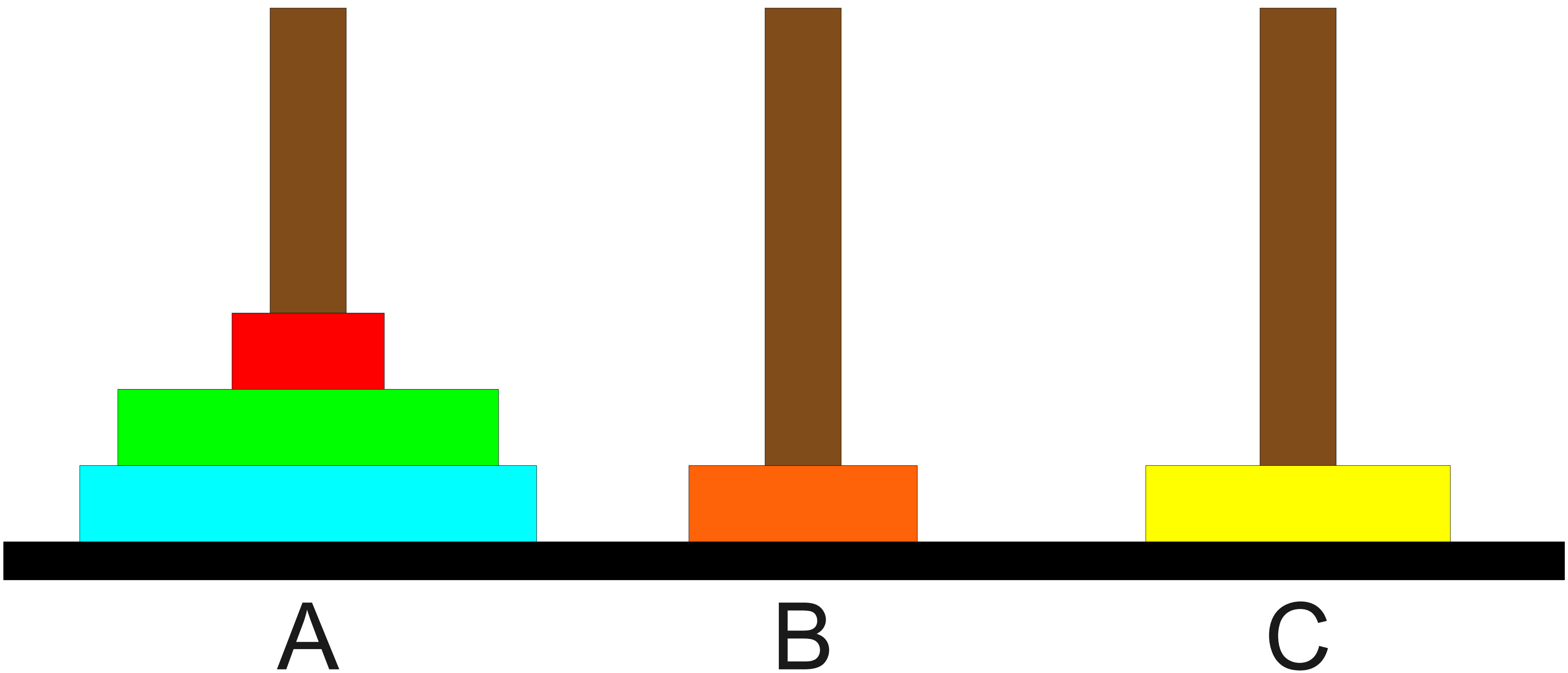
Towers of Hanoi



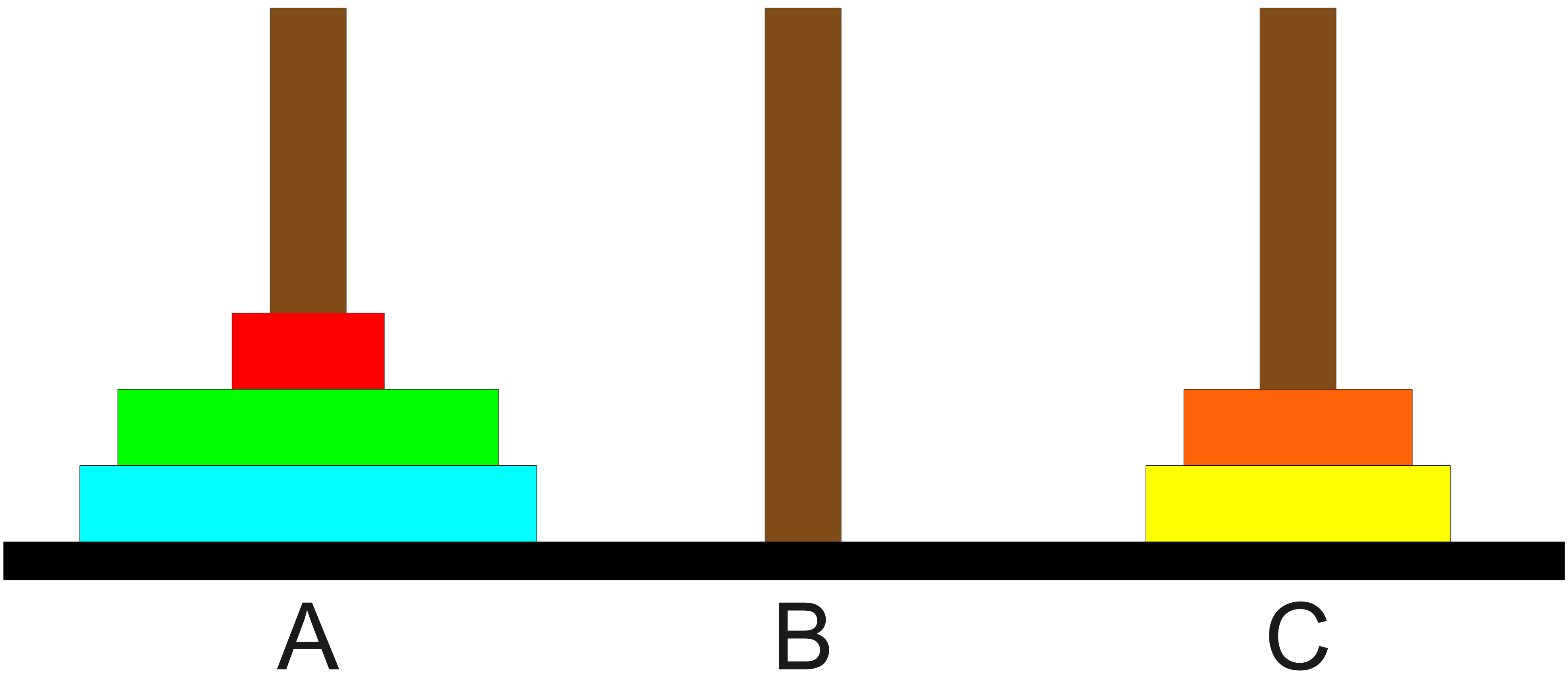
Towers of Hanoi



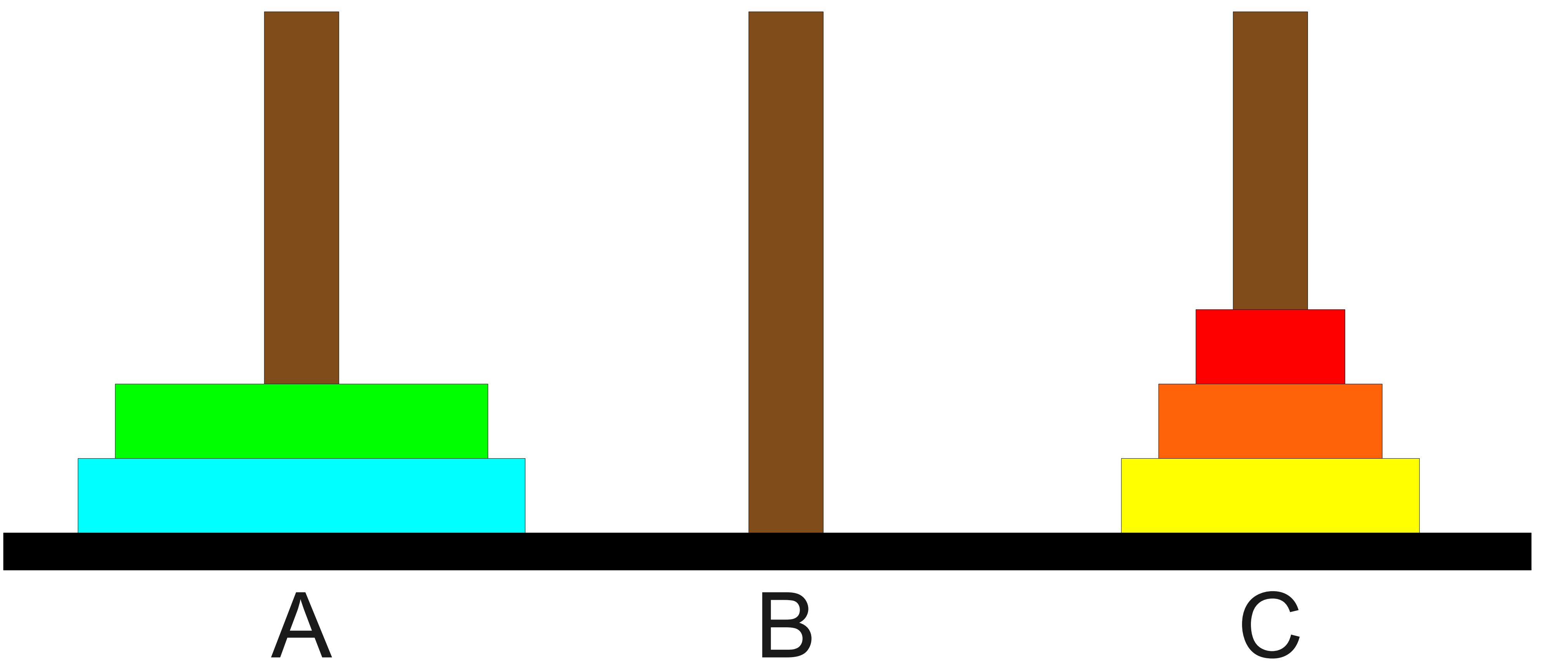
Towers of Hanoi



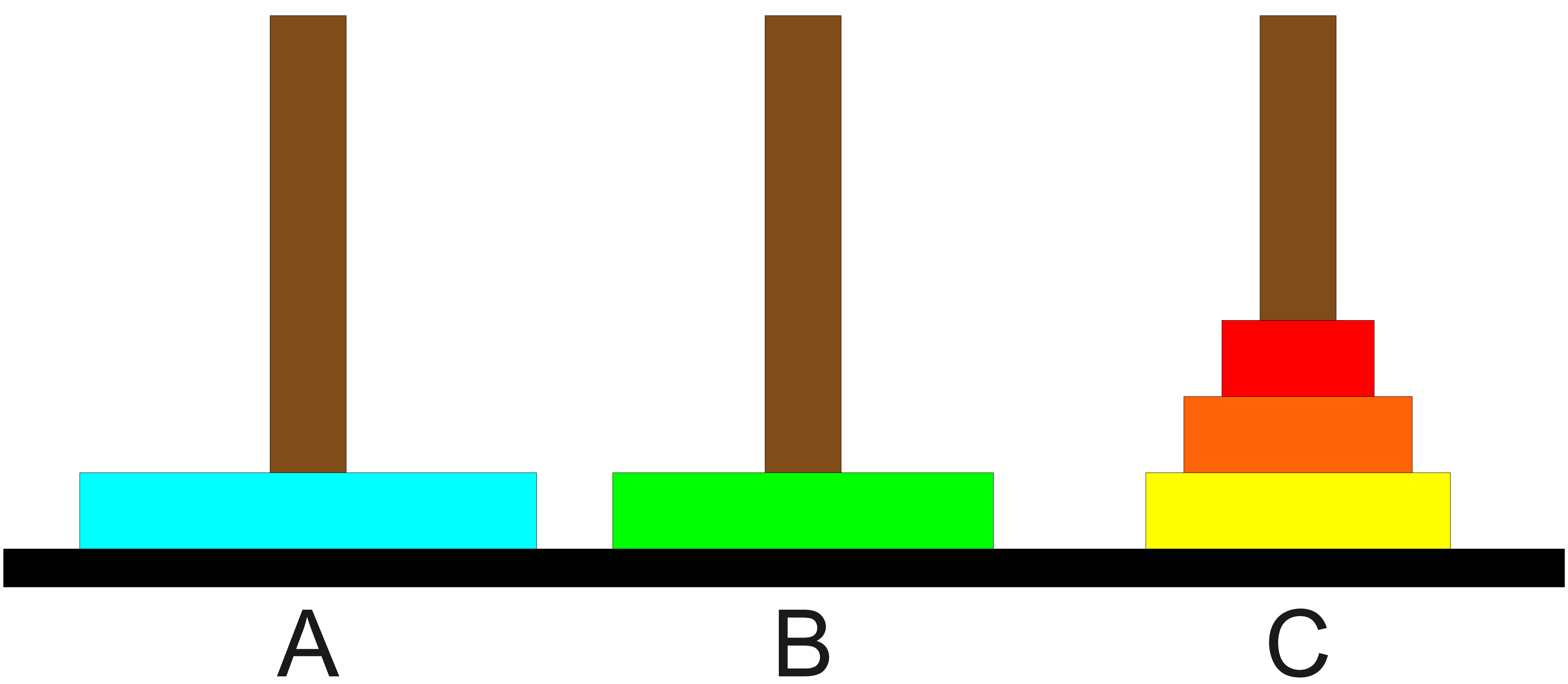
Towers of Hanoi



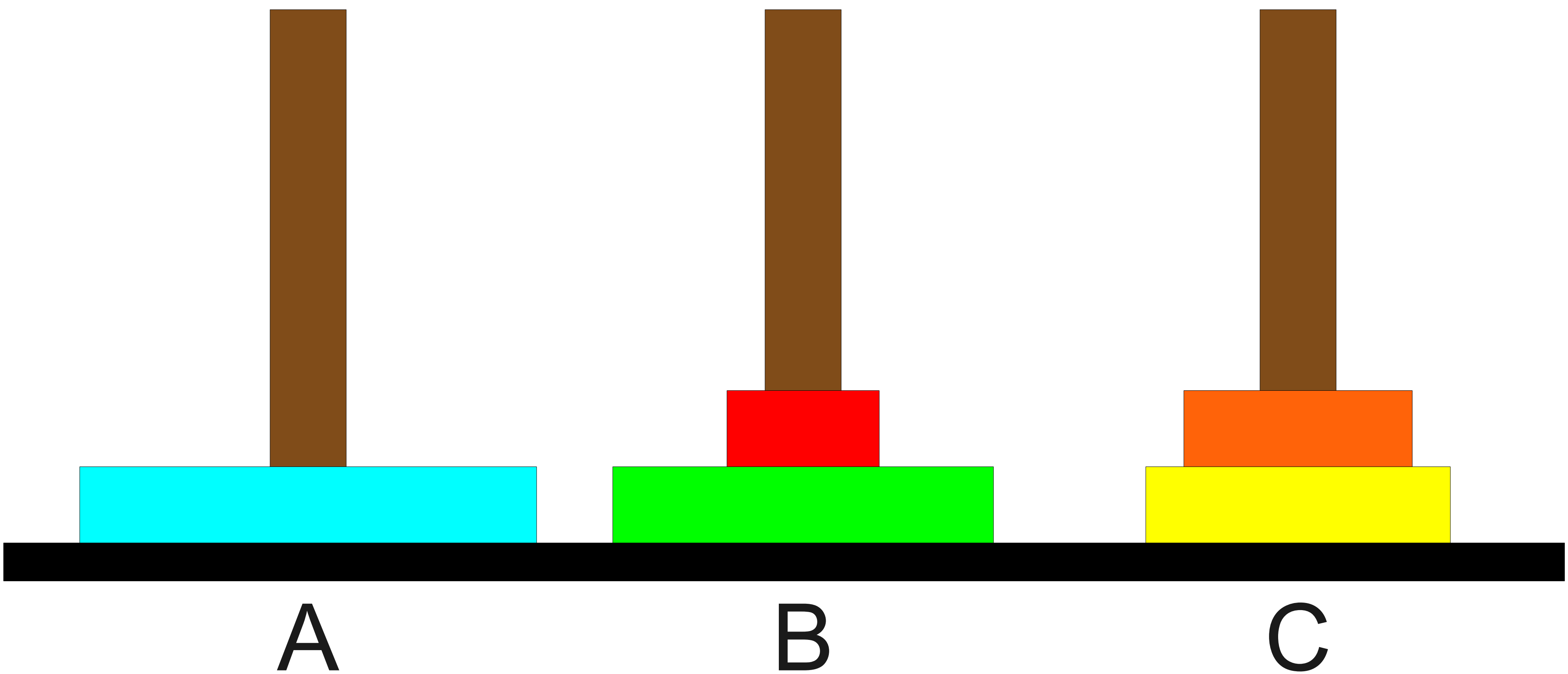
Towers of Hanoi



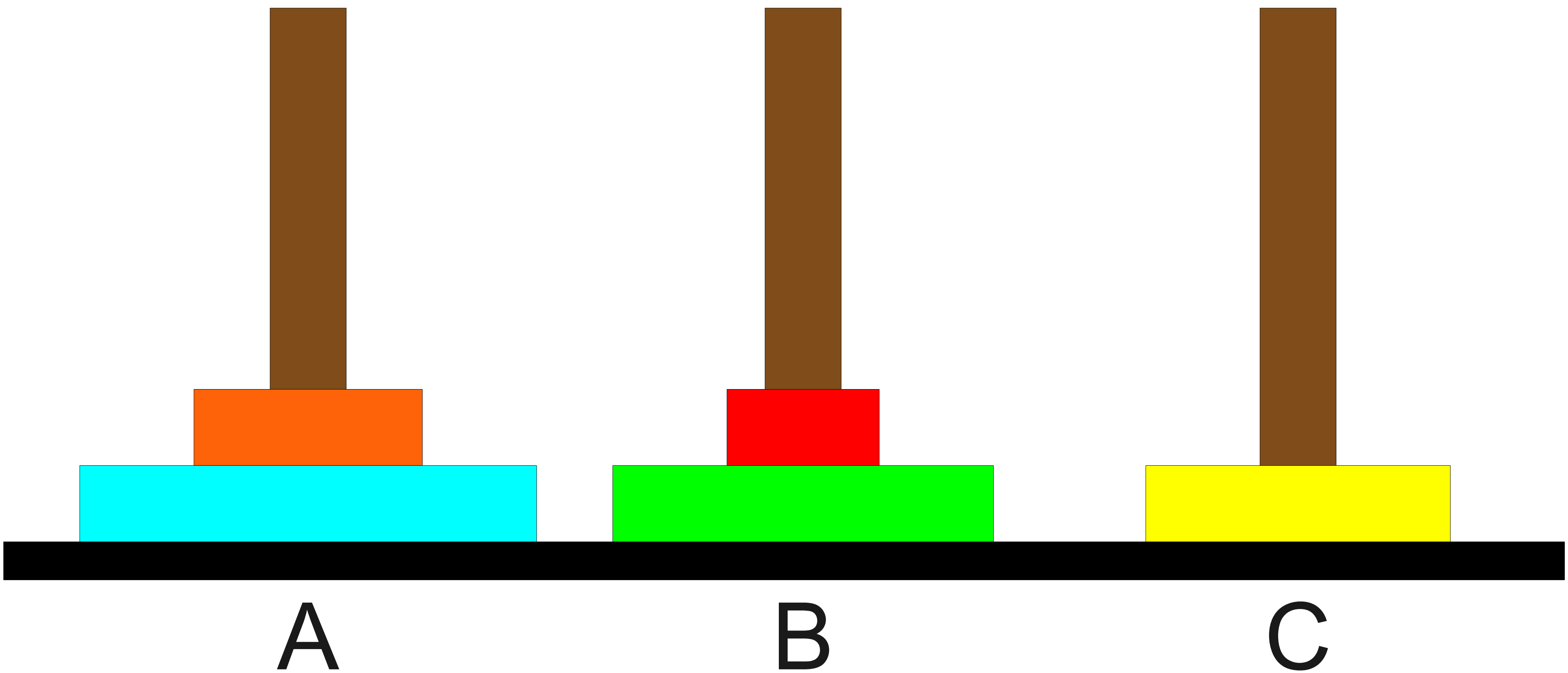
Towers of Hanoi



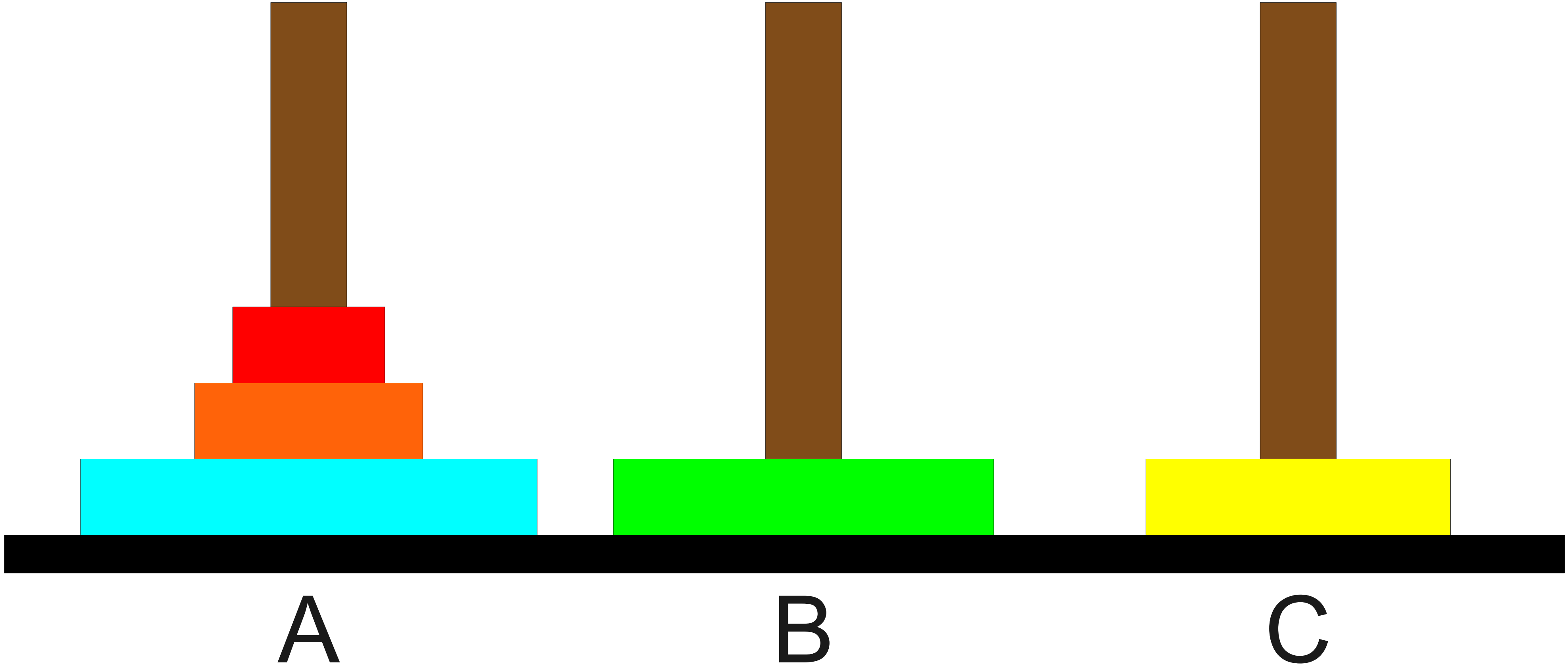
Towers of Hanoi



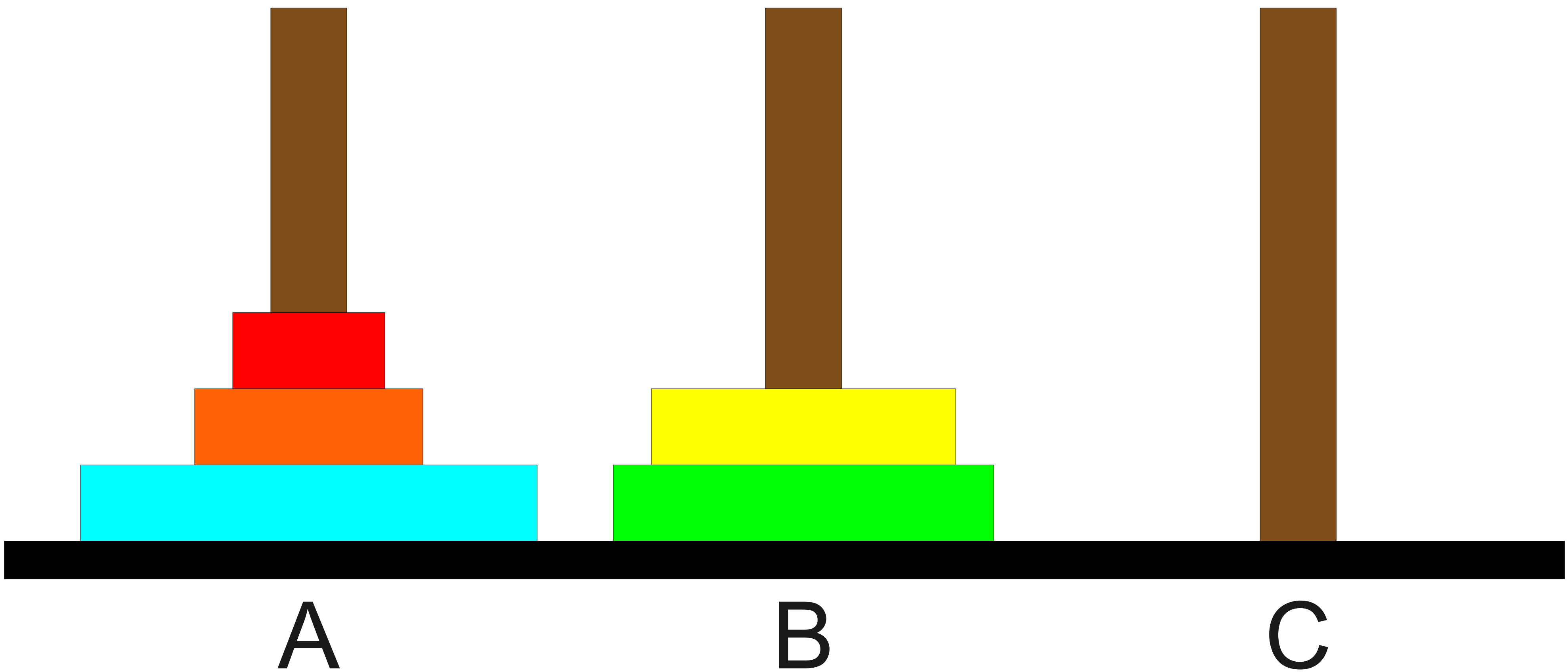
Towers of Hanoi



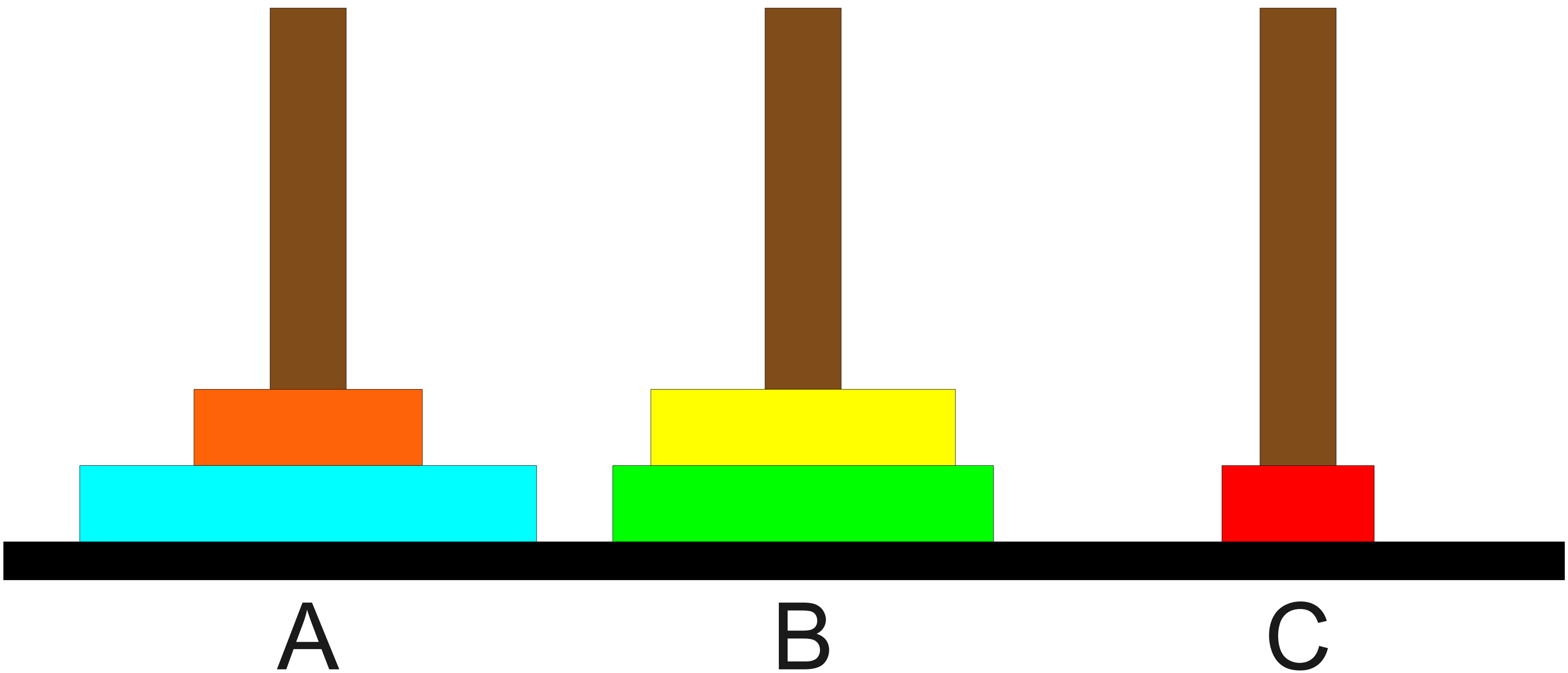
Towers of Hanoi



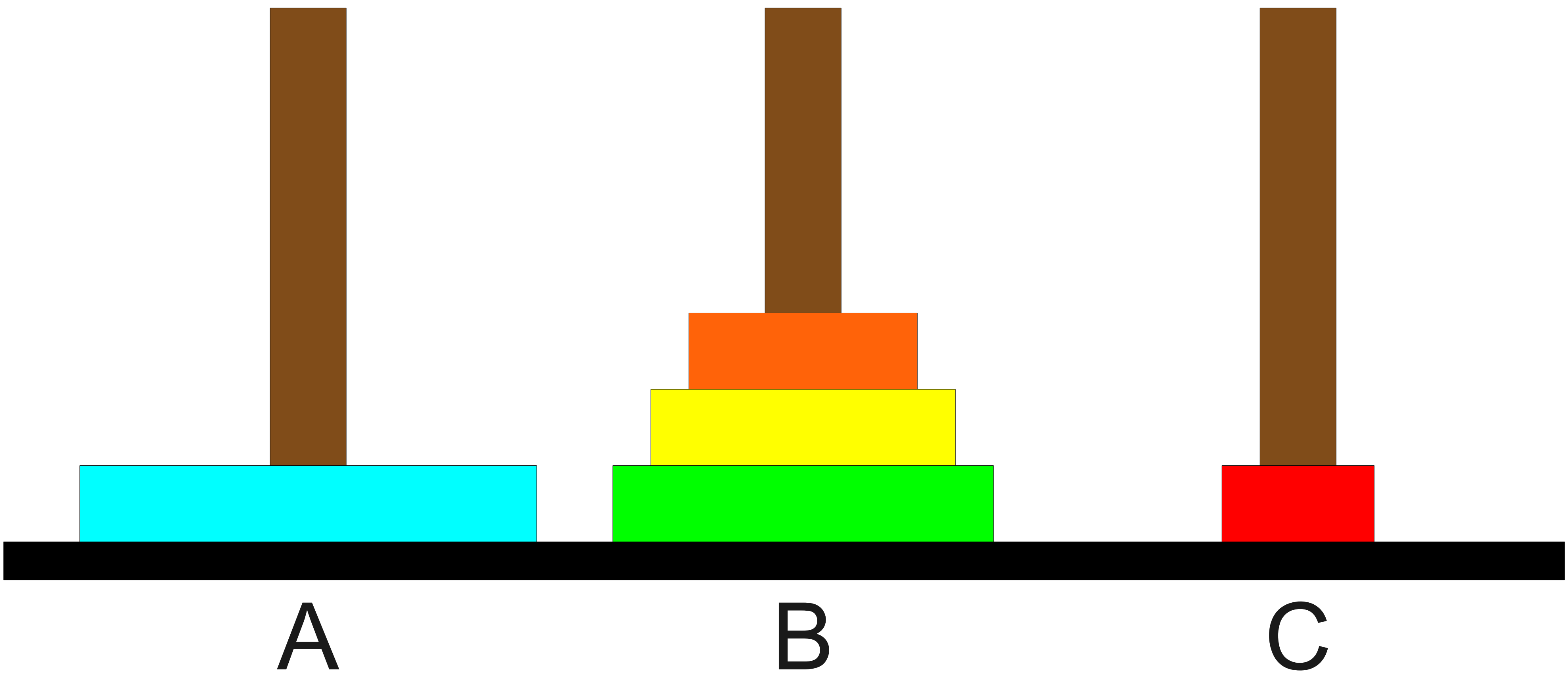
Towers of Hanoi



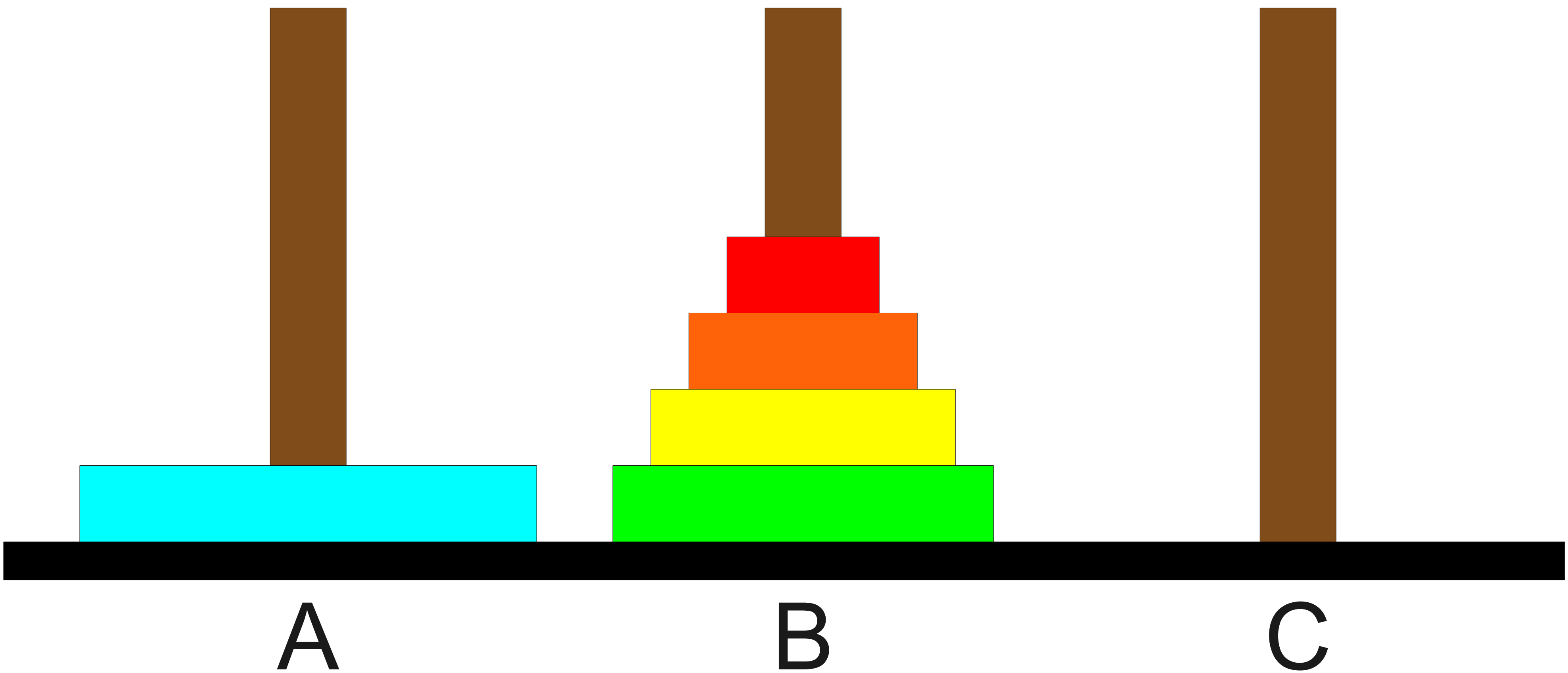
Towers of Hanoi



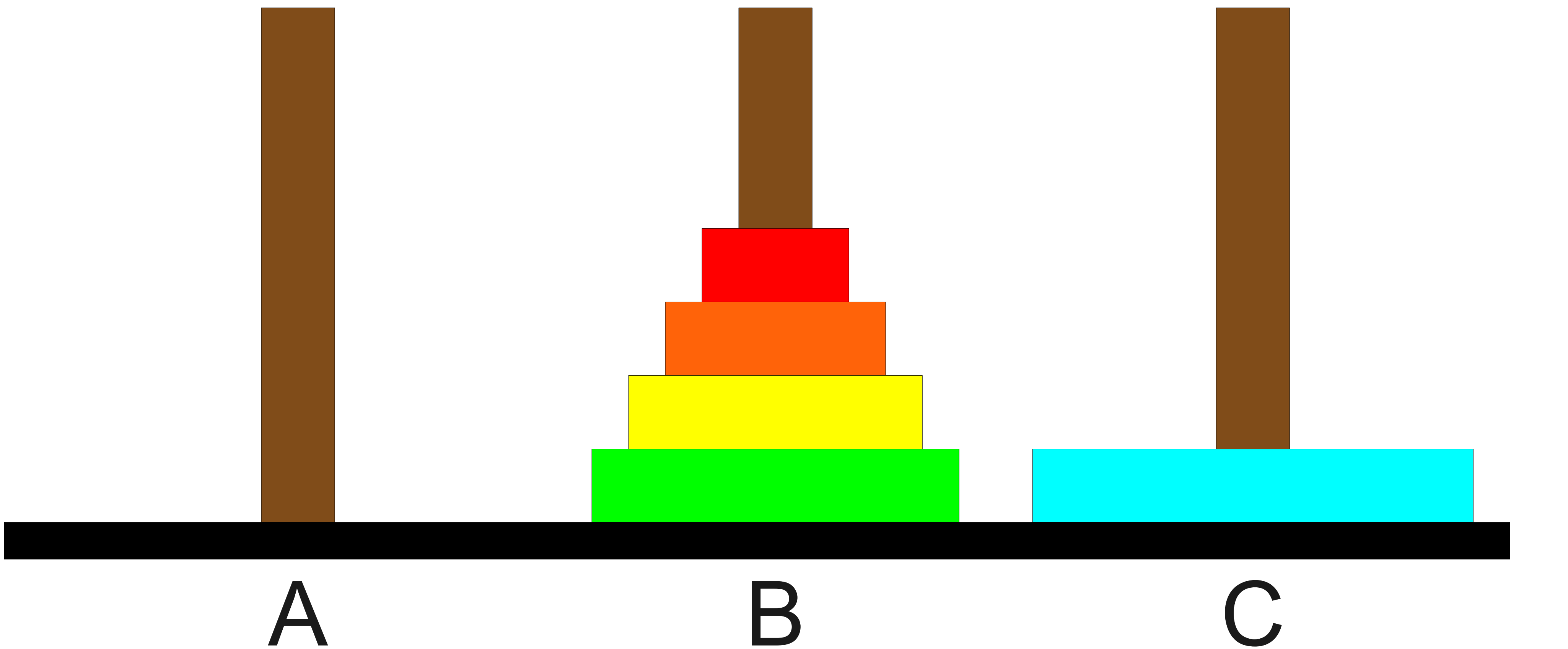
Towers of Hanoi



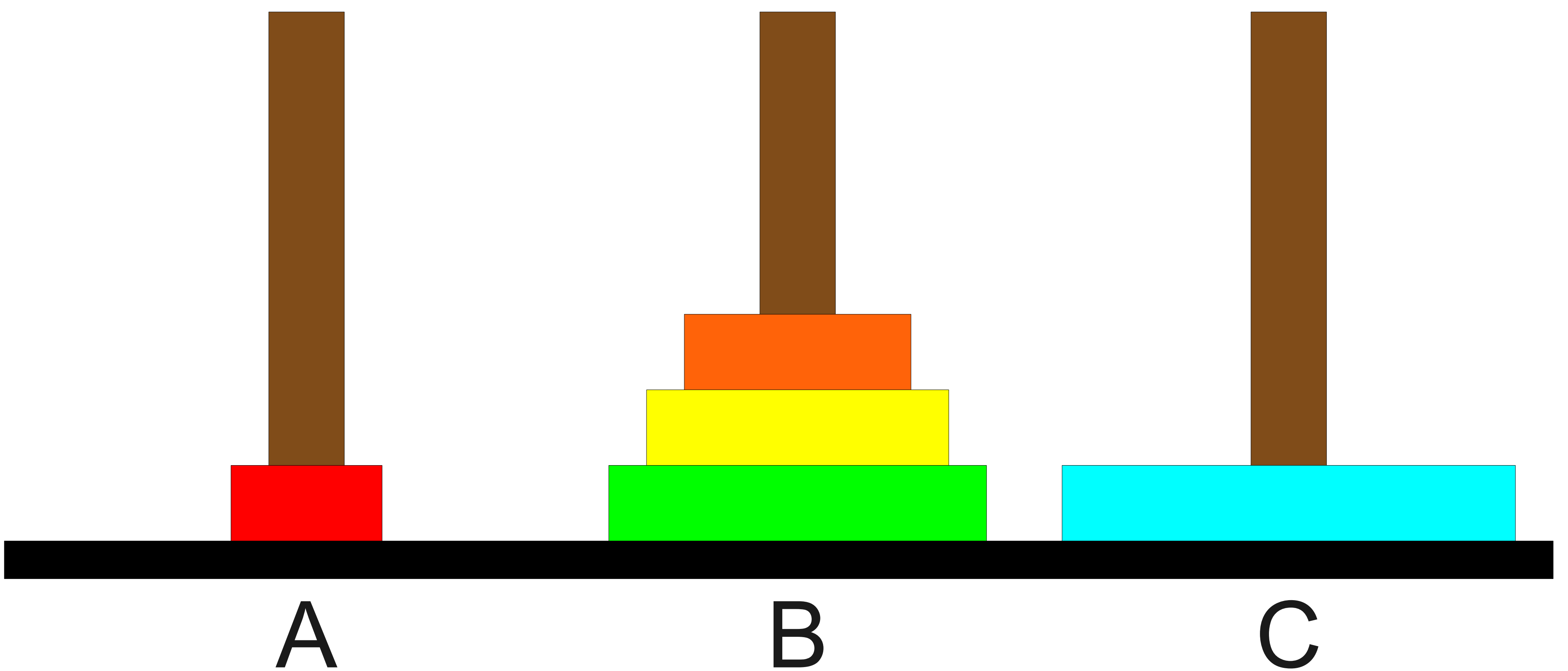
Towers of Hanoi



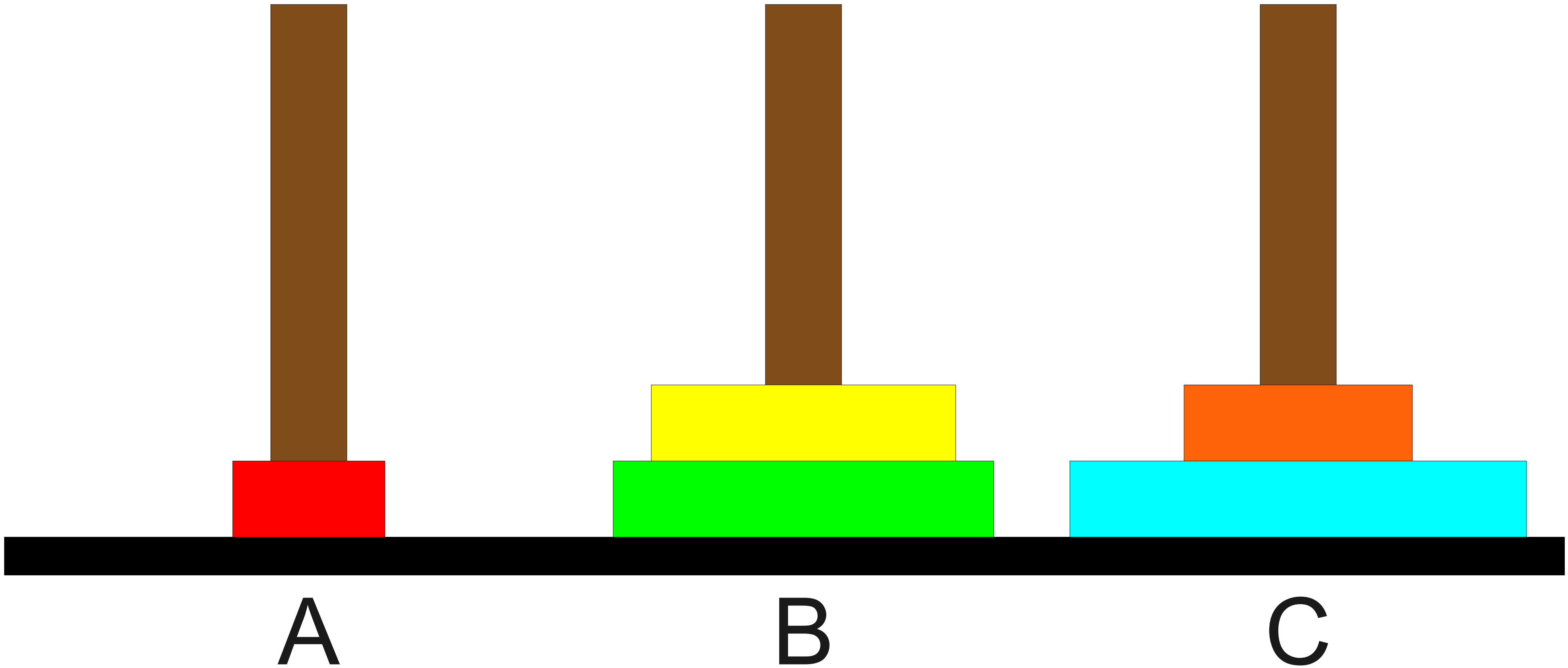
Towers of Hanoi



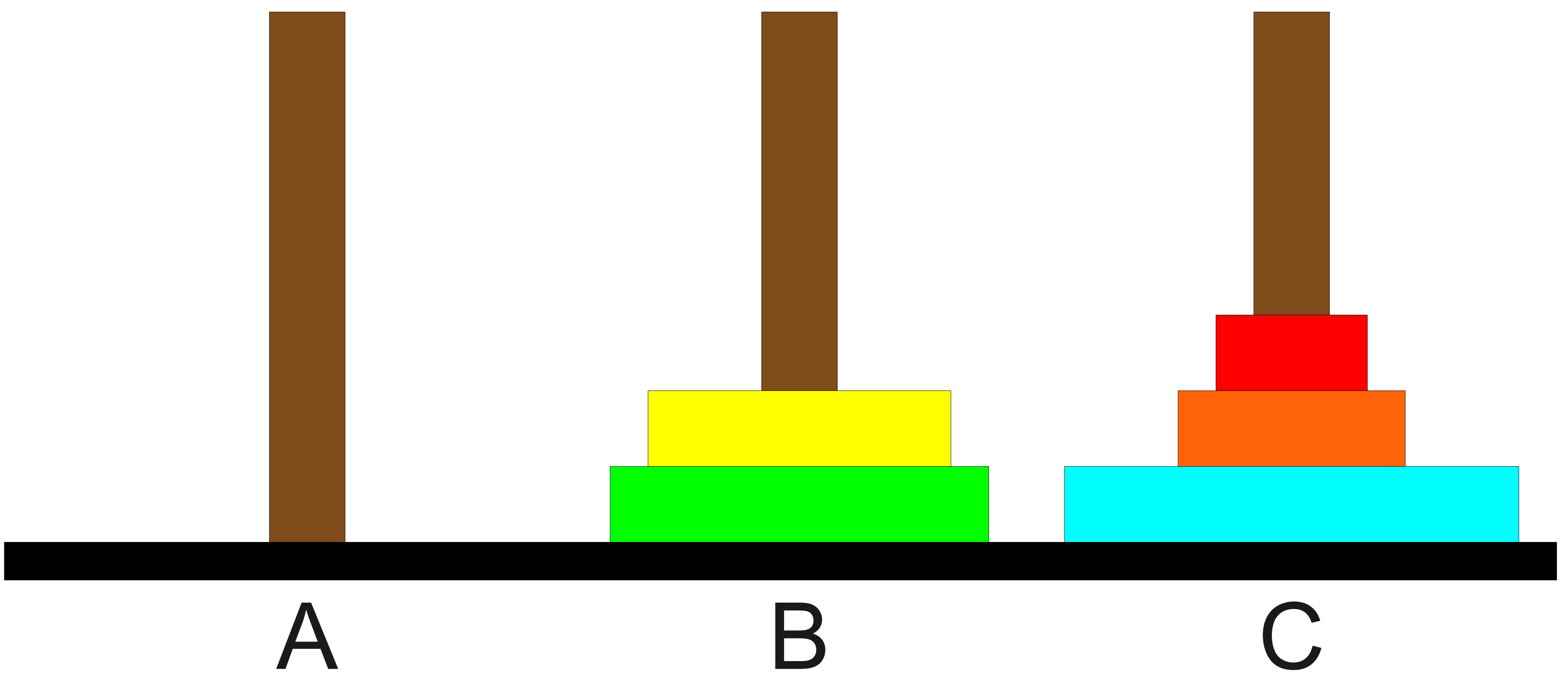
Towers of Hanoi



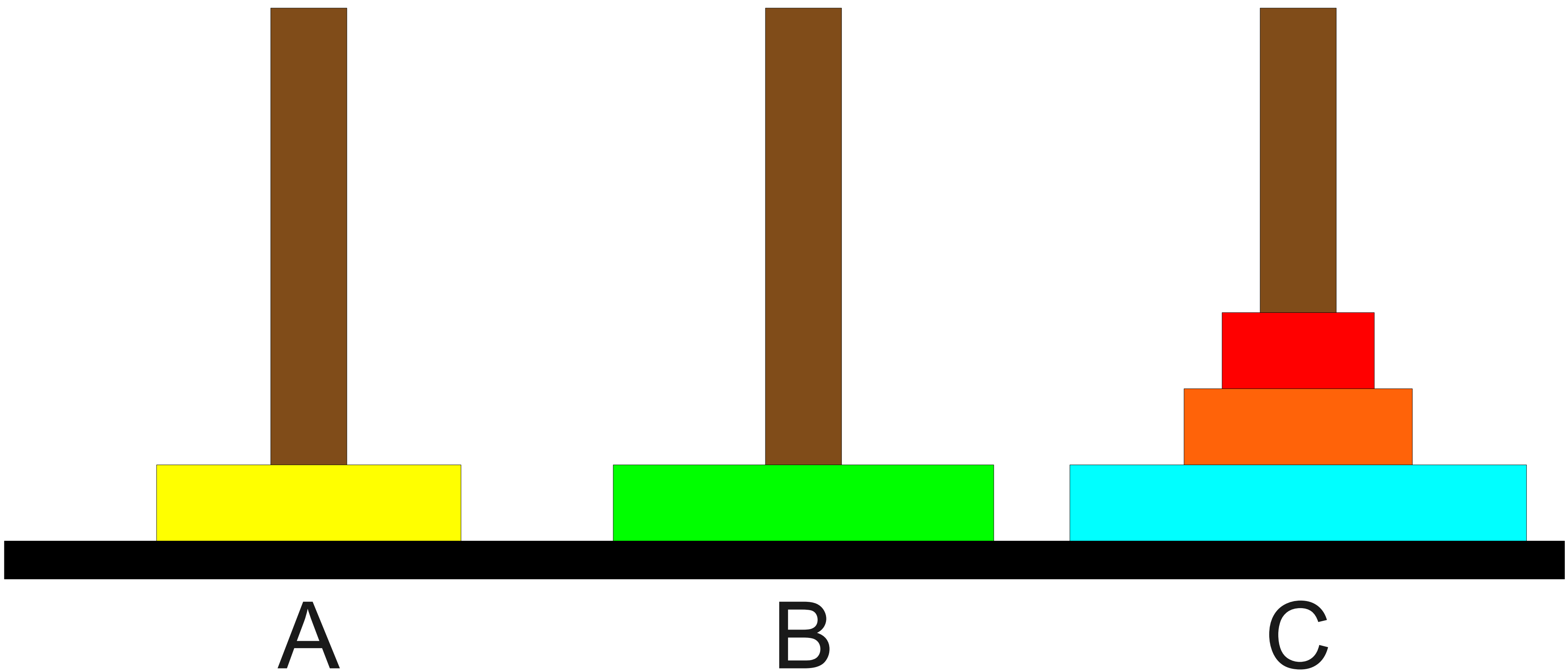
Towers of Hanoi



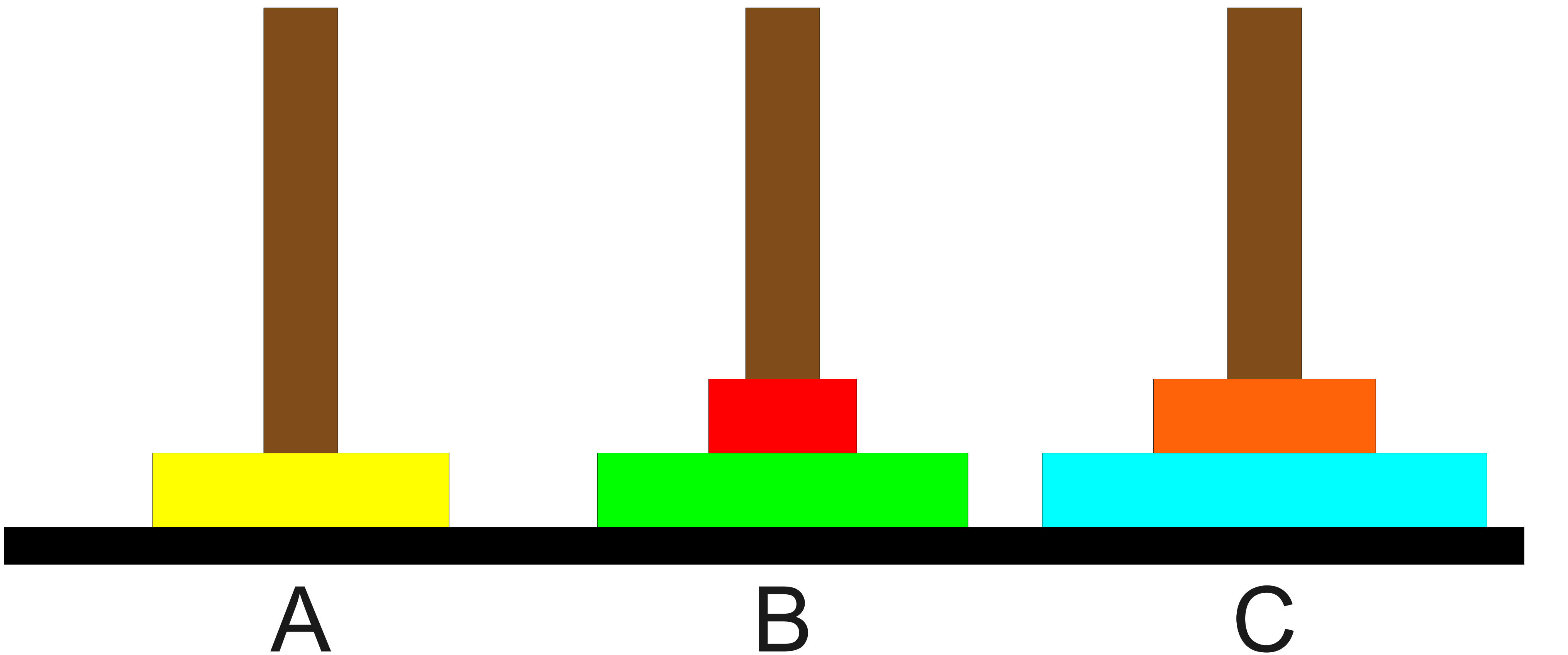
Towers of Hanoi



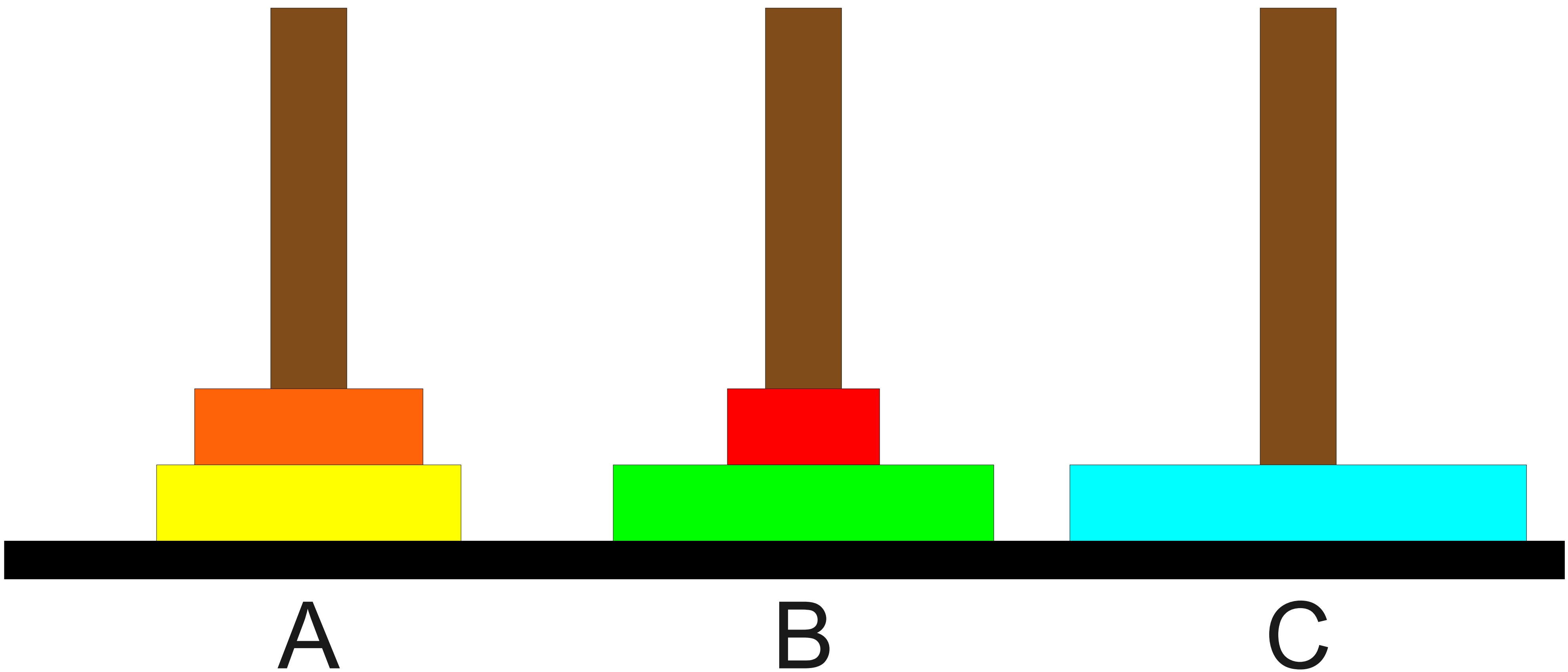
Towers of Hanoi



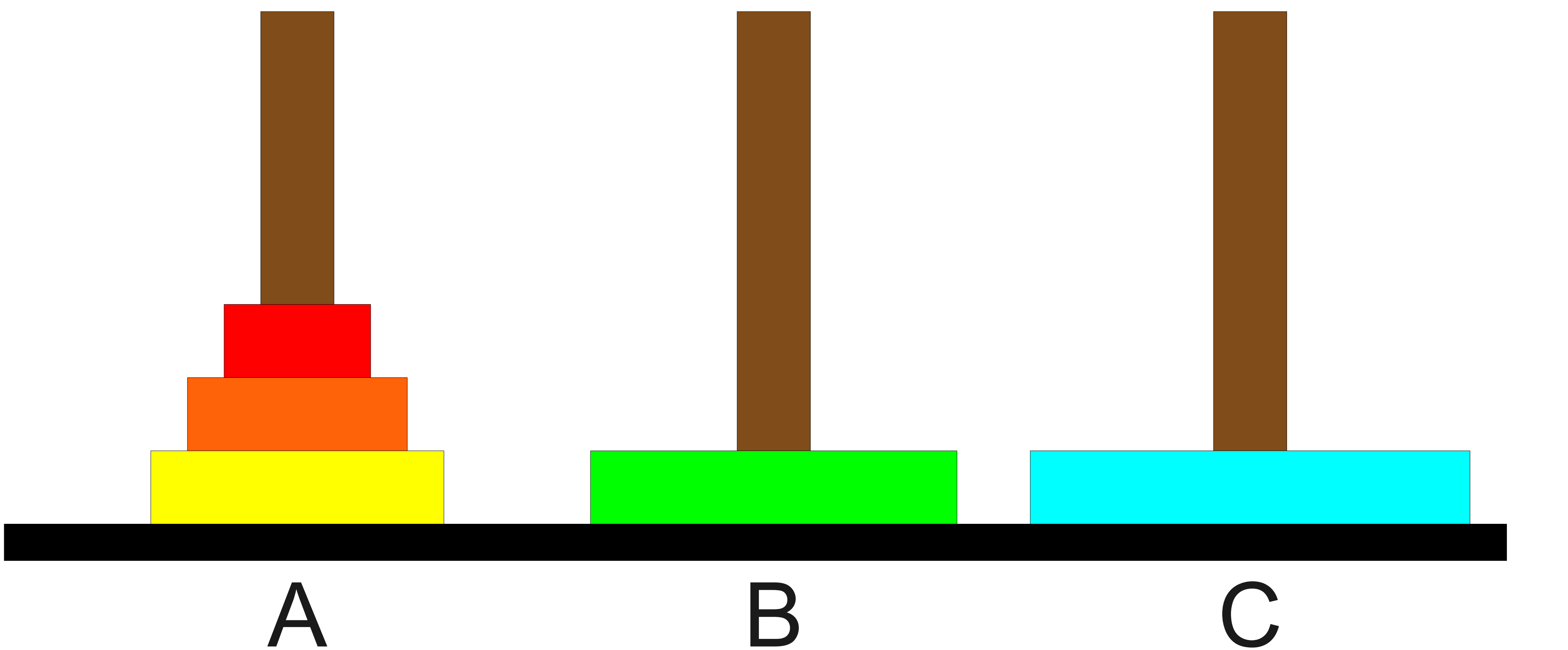
Towers of Hanoi



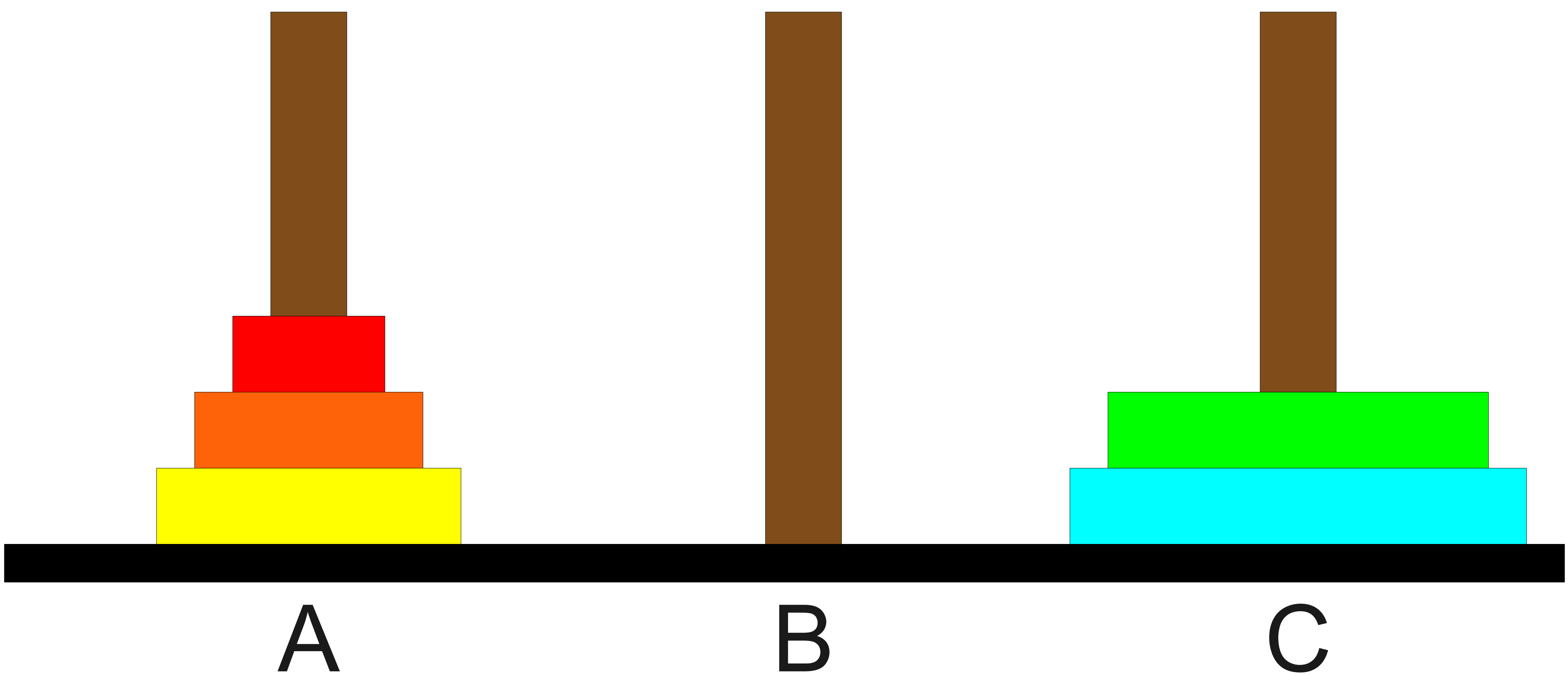
Towers of Hanoi



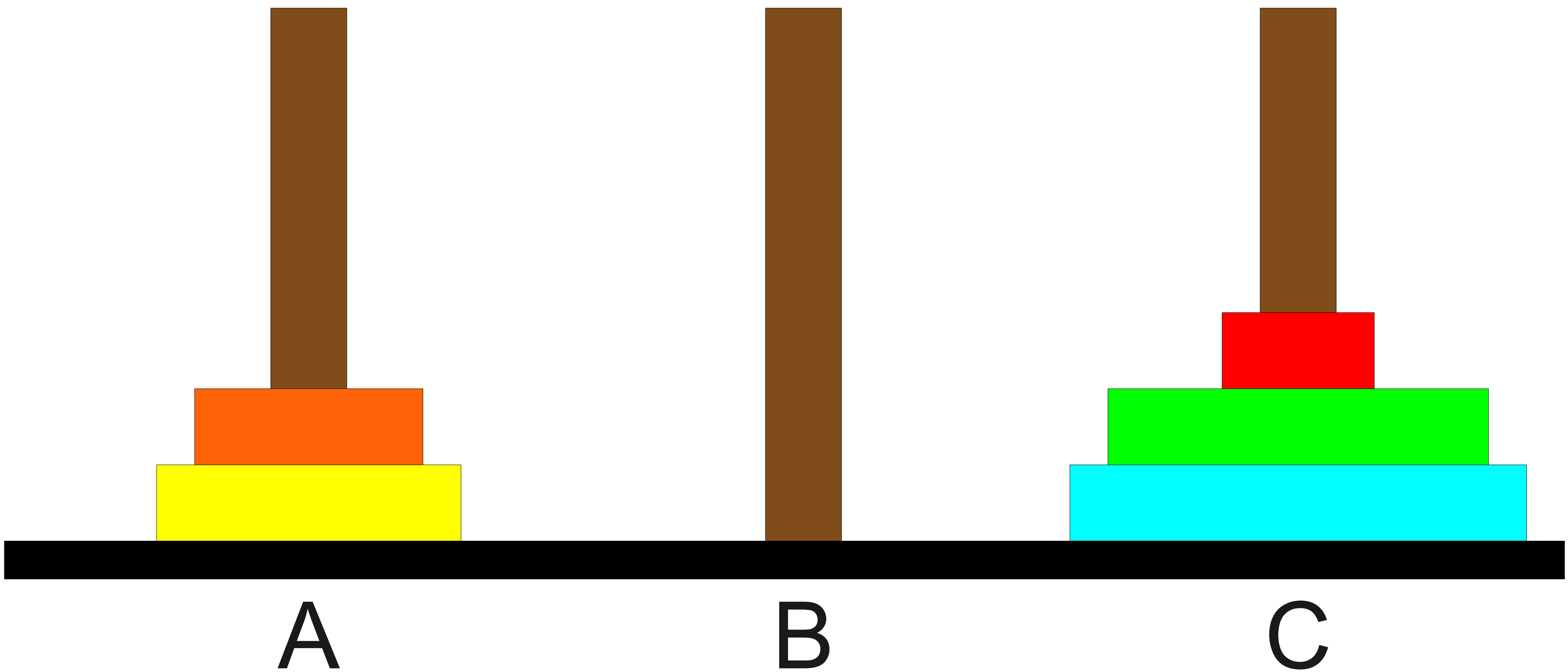
Towers of Hanoi



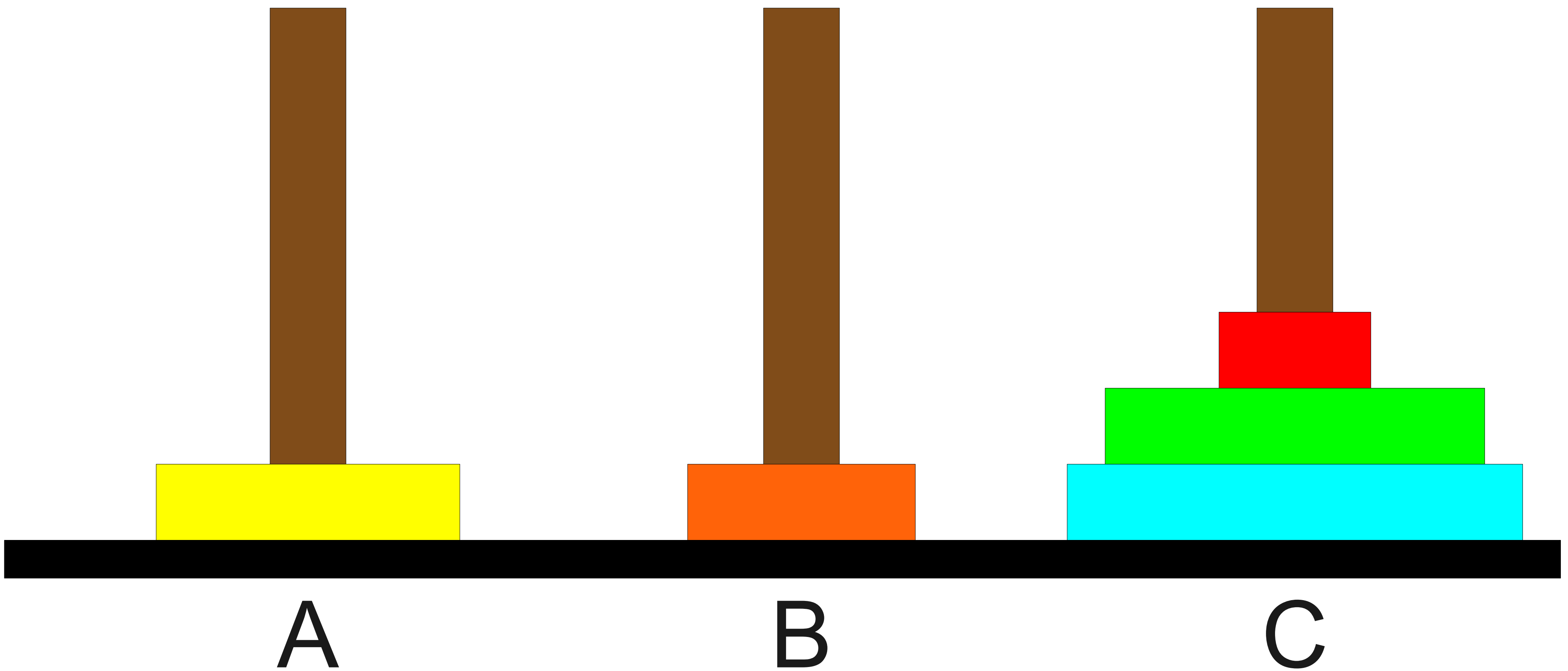
Towers of Hanoi



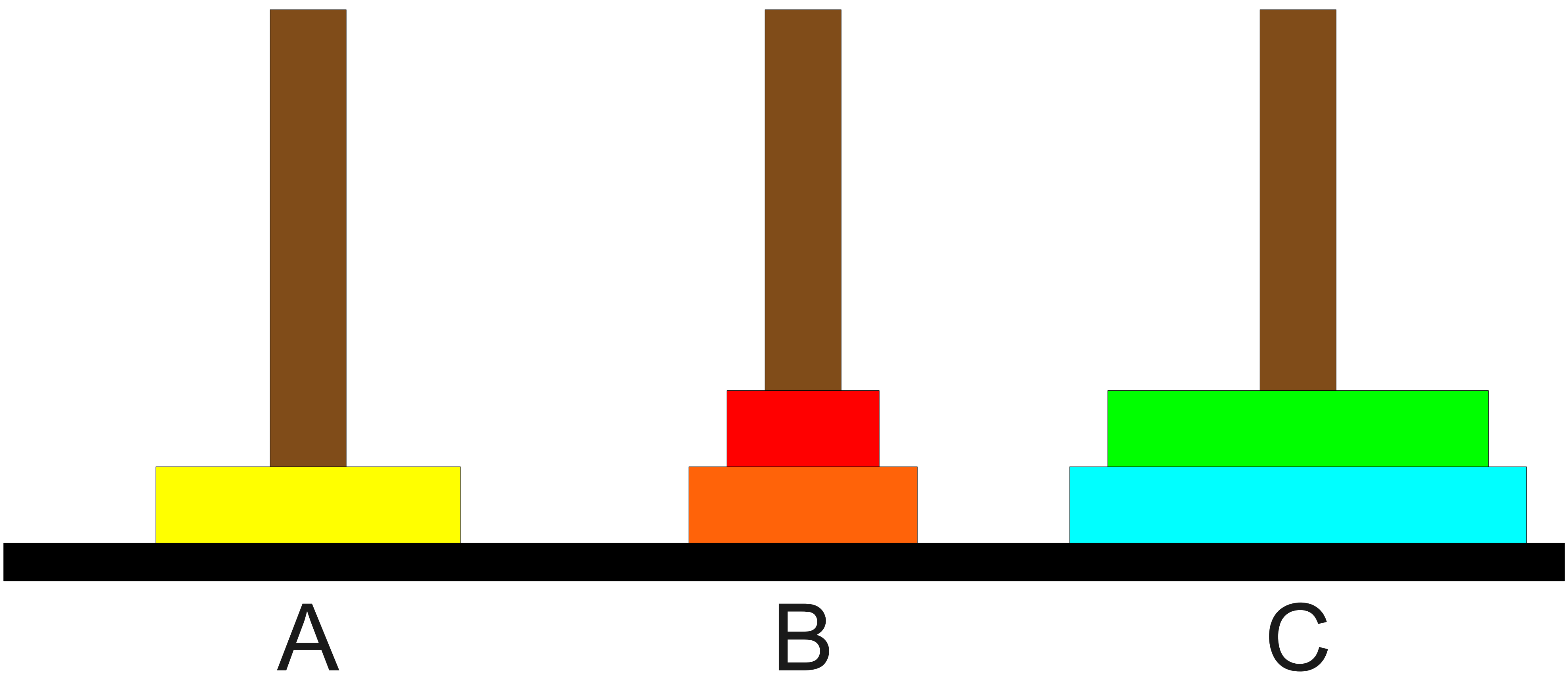
Towers of Hanoi



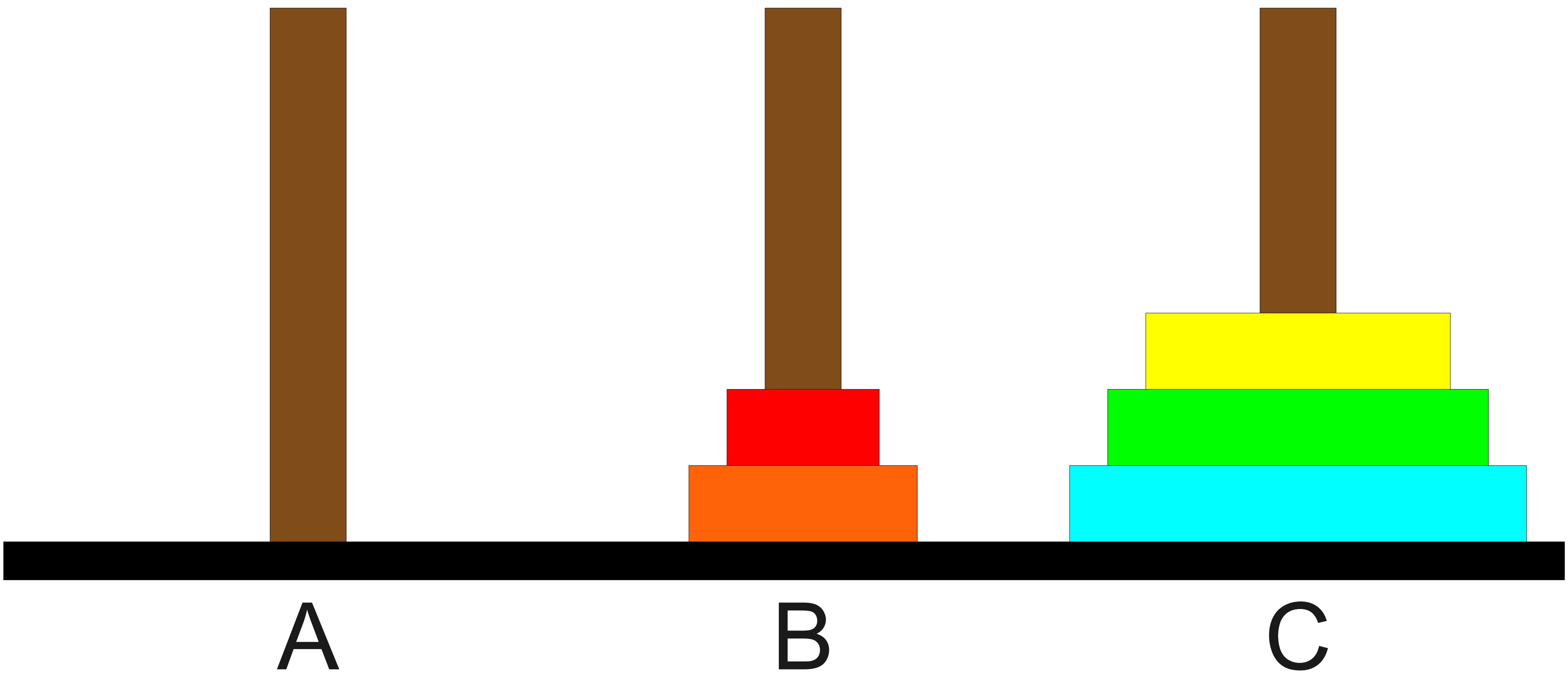
Towers of Hanoi



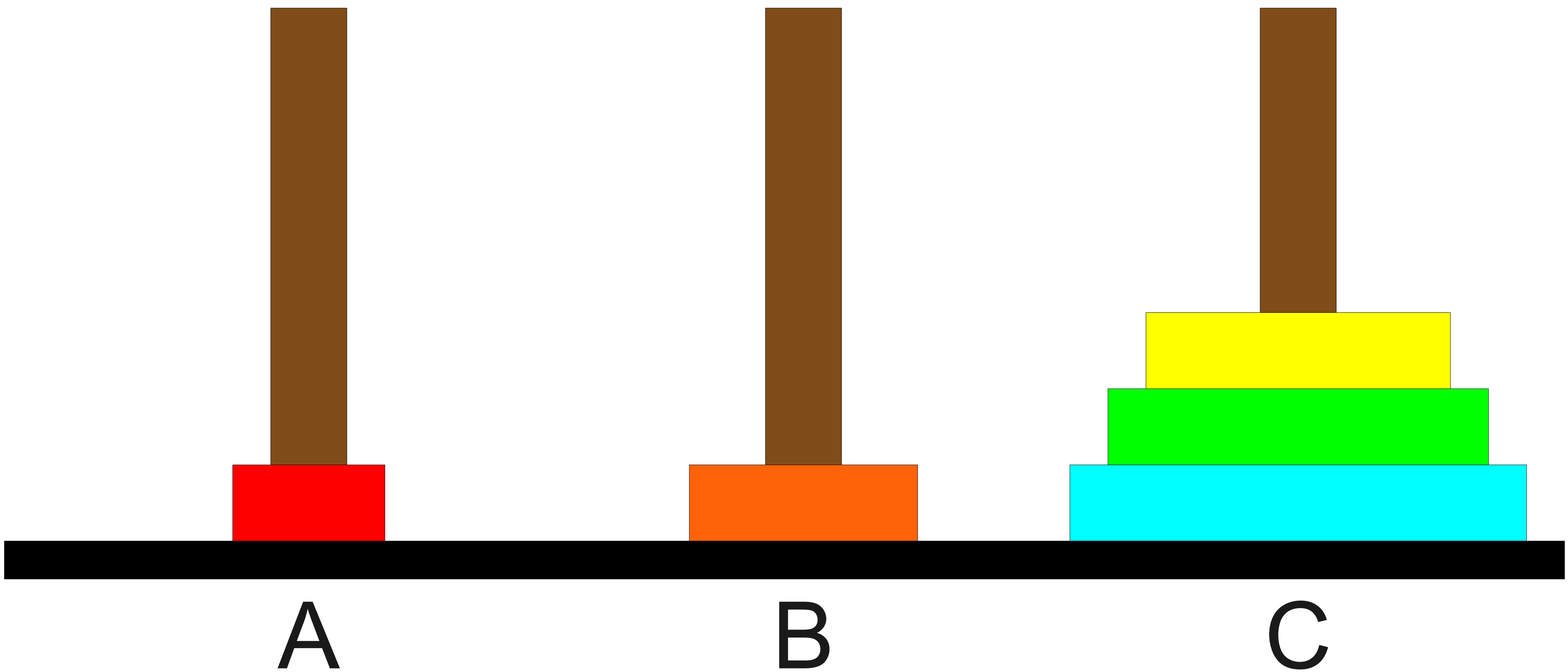
Towers of Hanoi



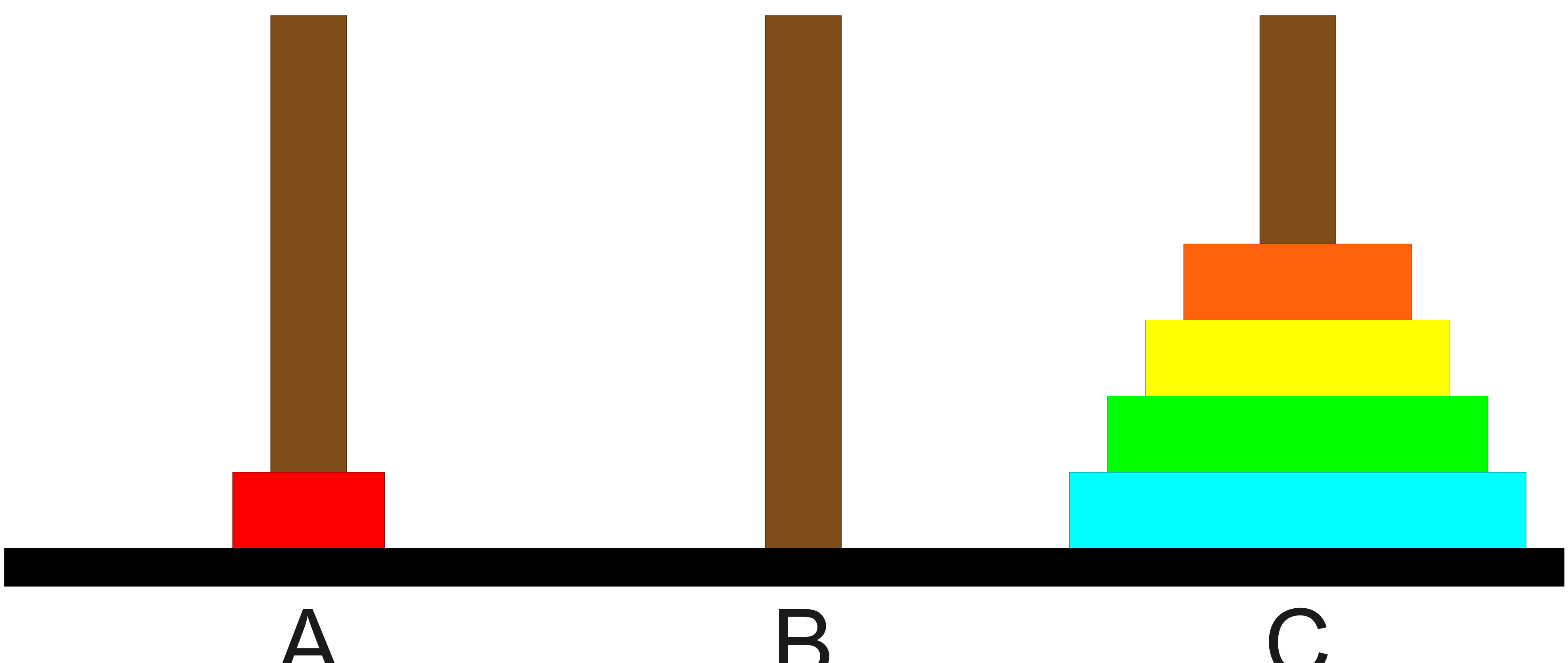
Towers of Hanoi



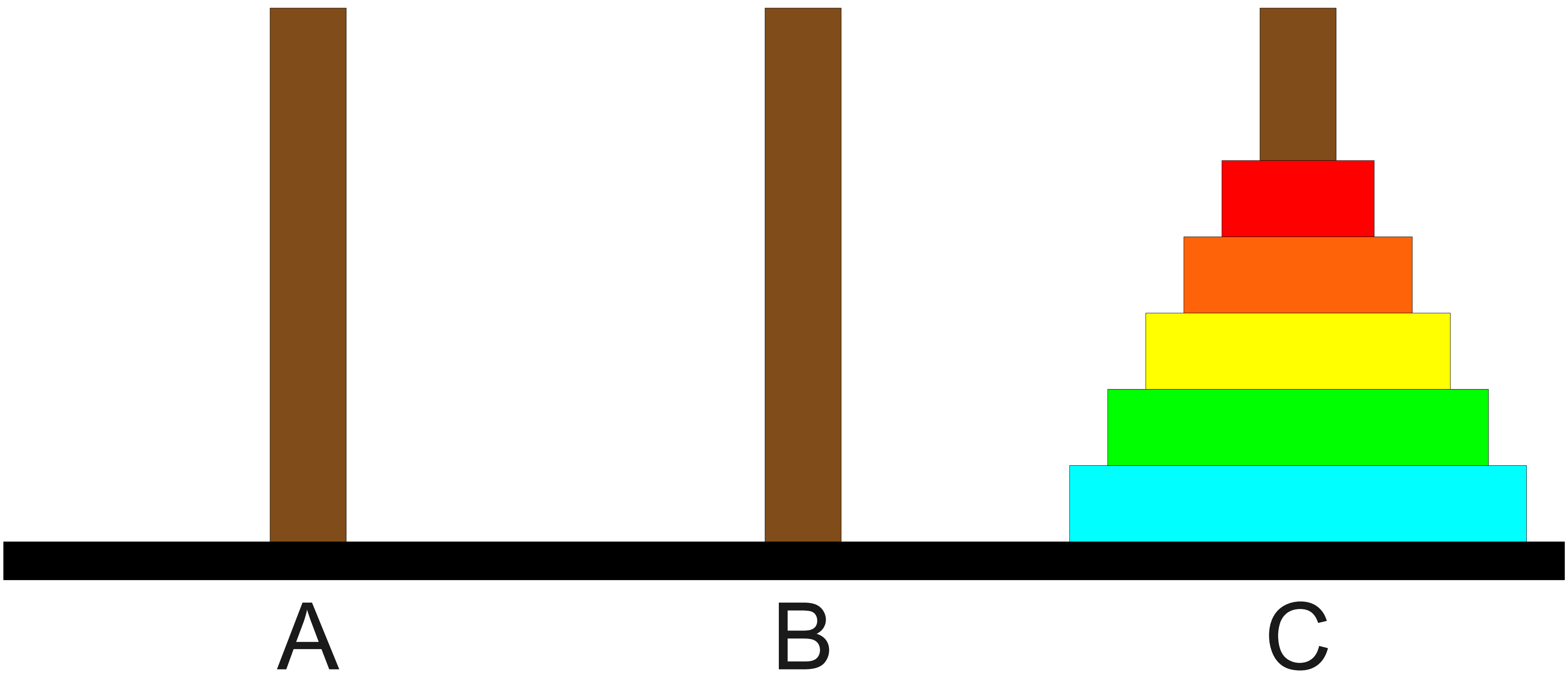
Towers of Hanoi



Towers of Hanoi

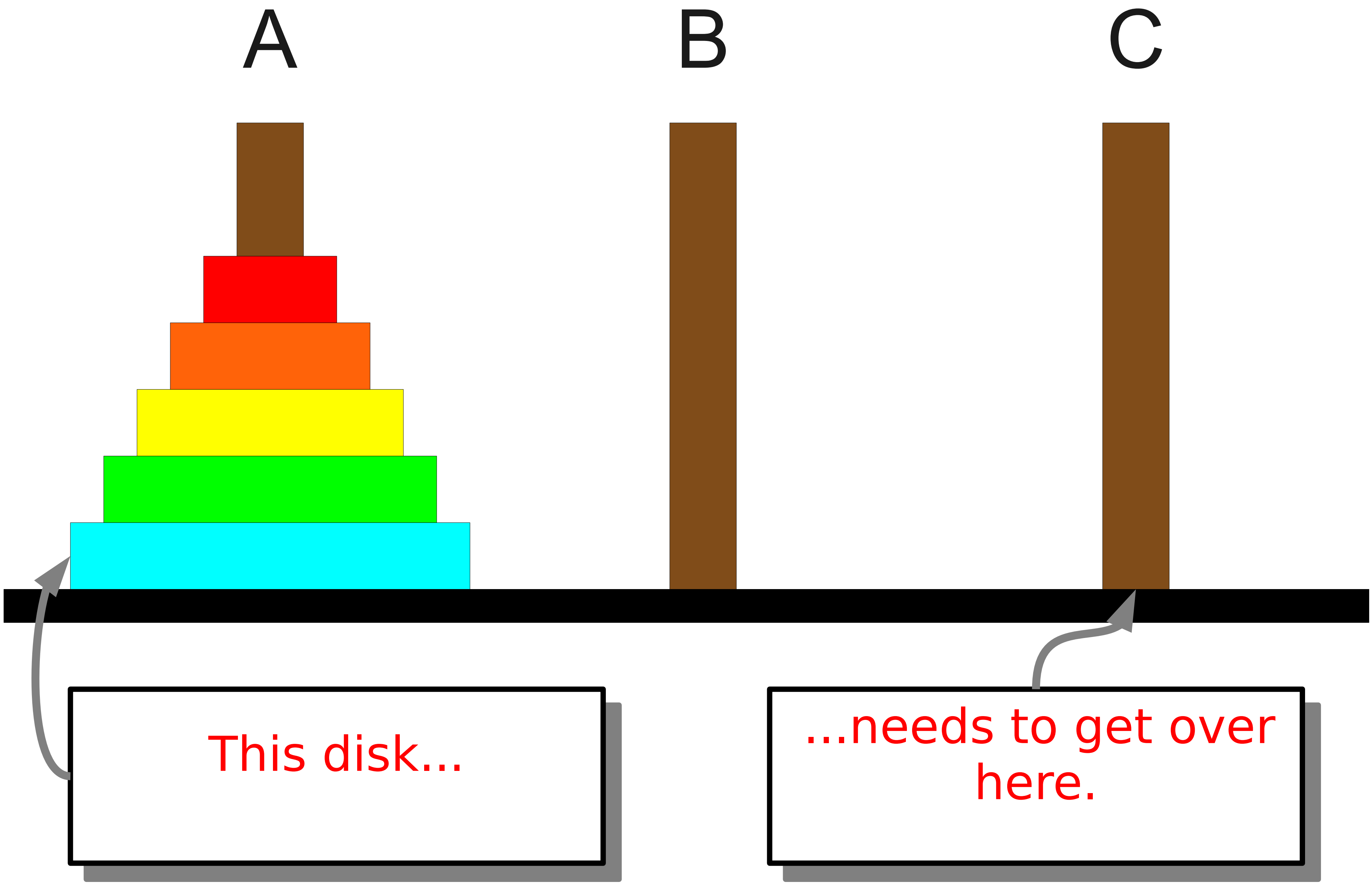


Towers of Hanoi

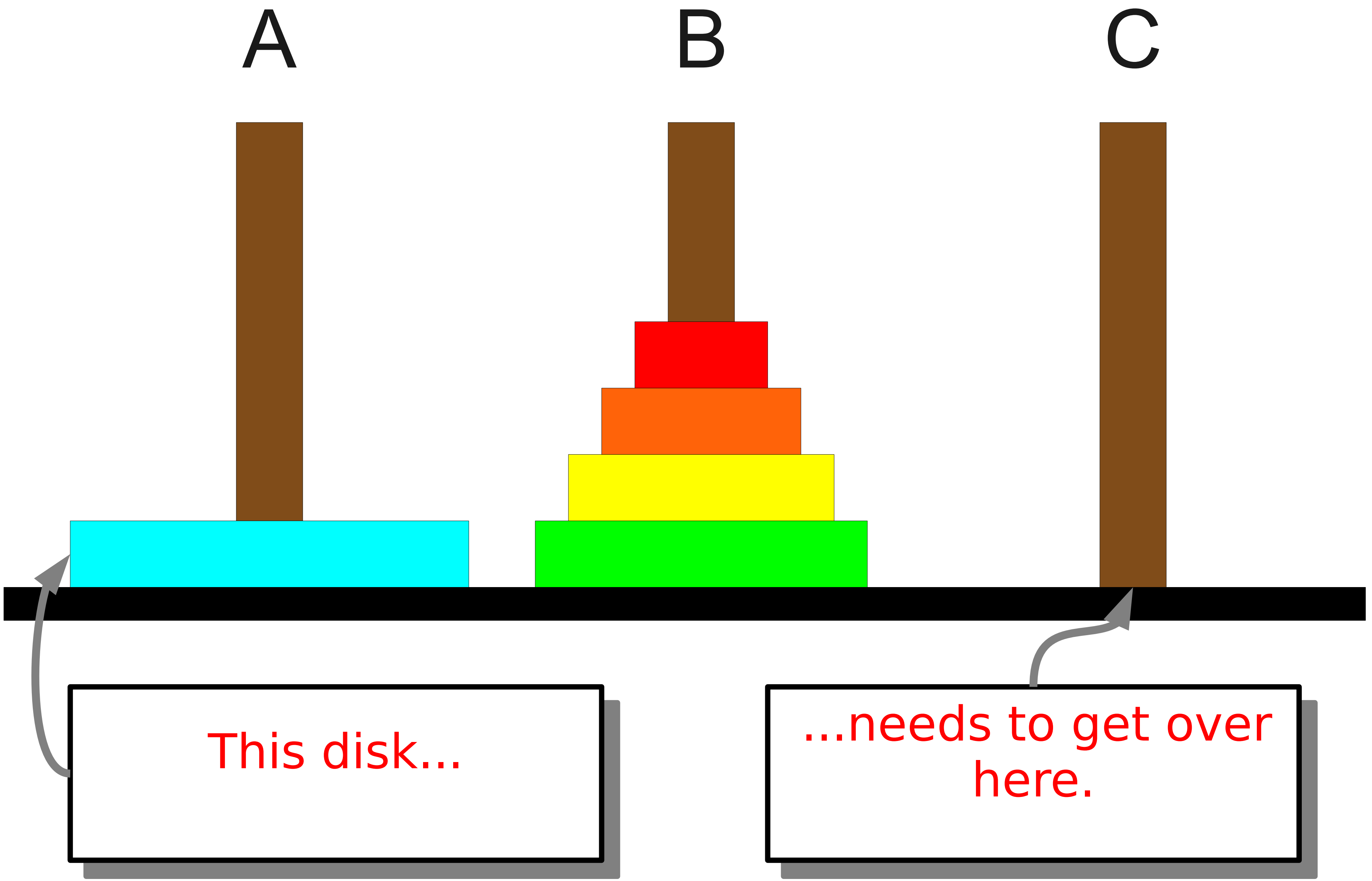


Solving the Towers of Hanoi

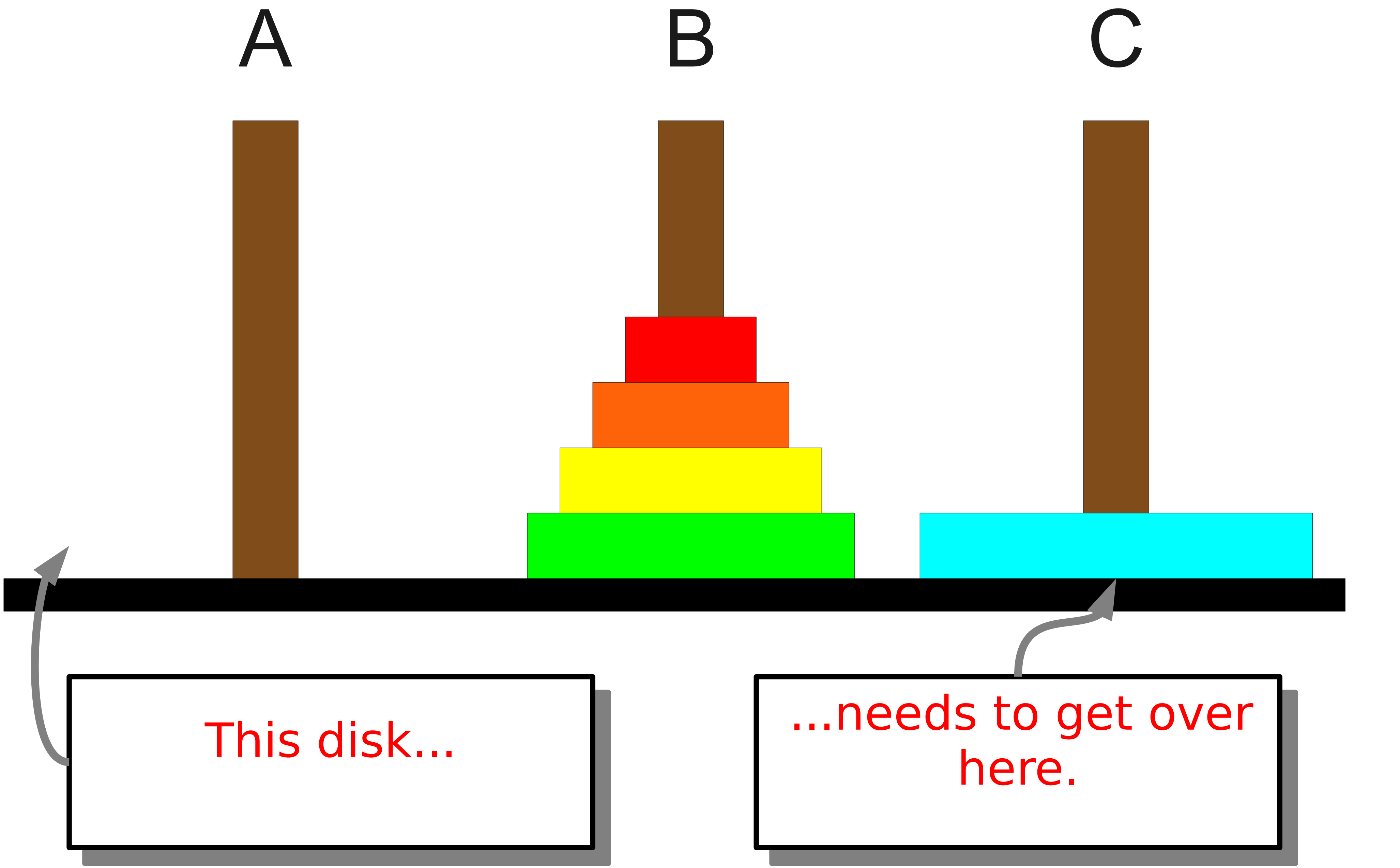
Solving the Towers of Hanoi



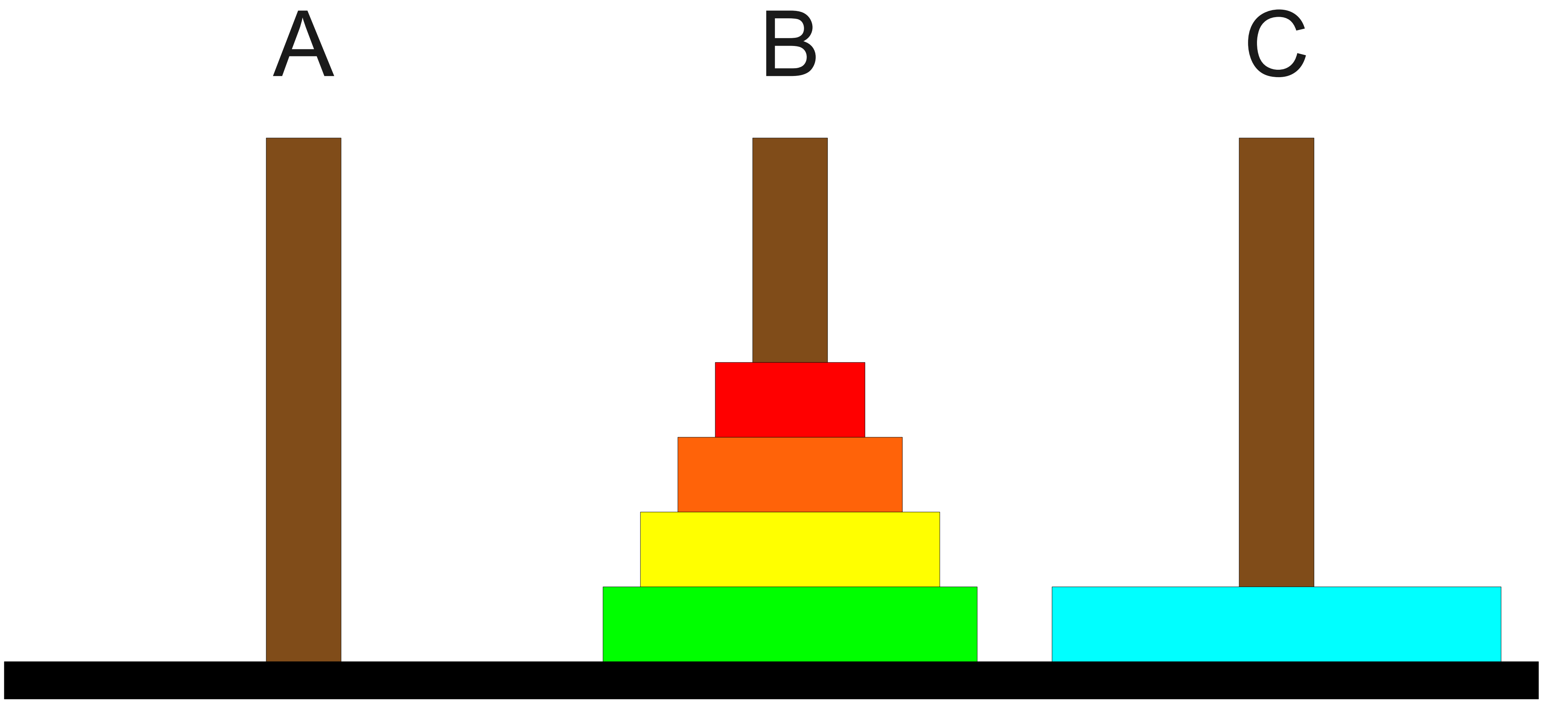
Solving the Towers of Hanoi



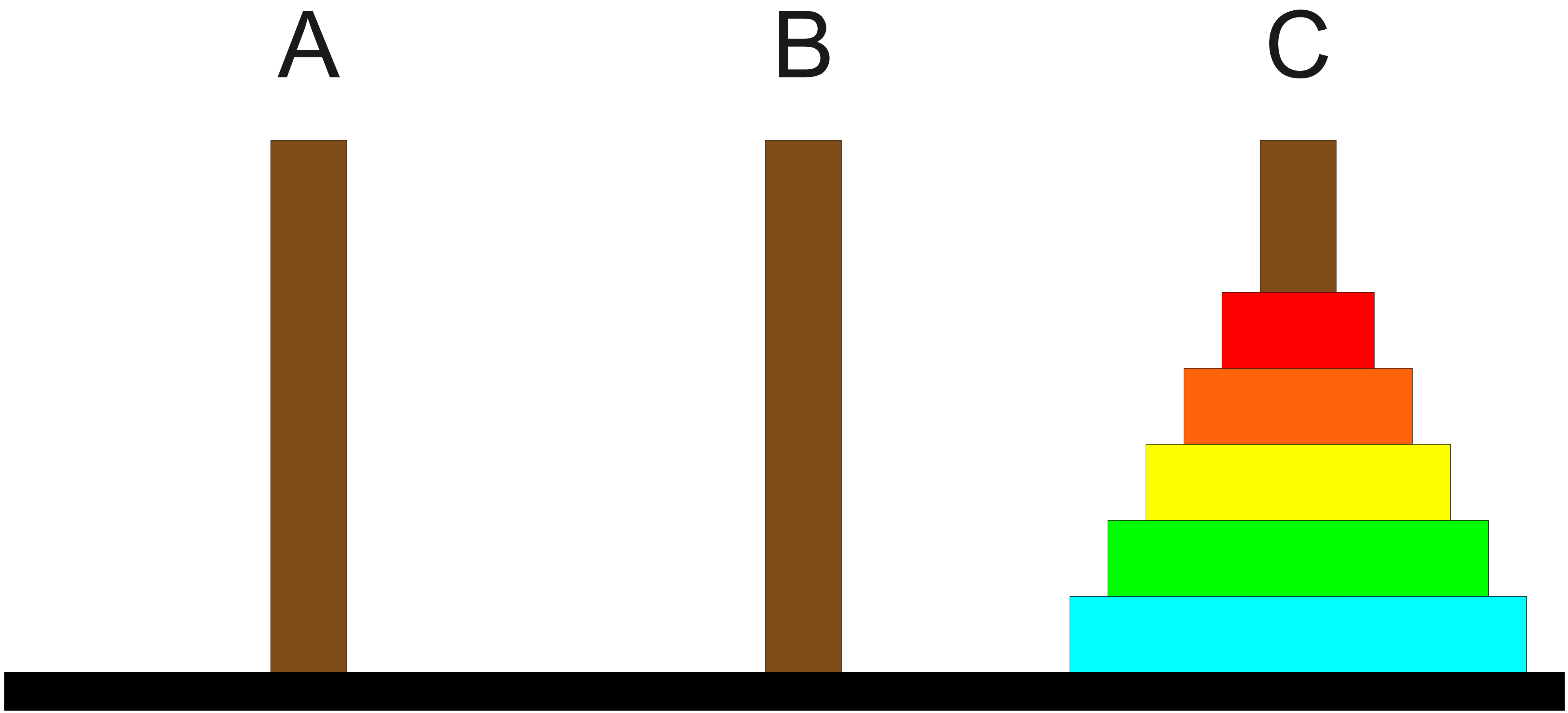
Solving the Towers of Hanoi



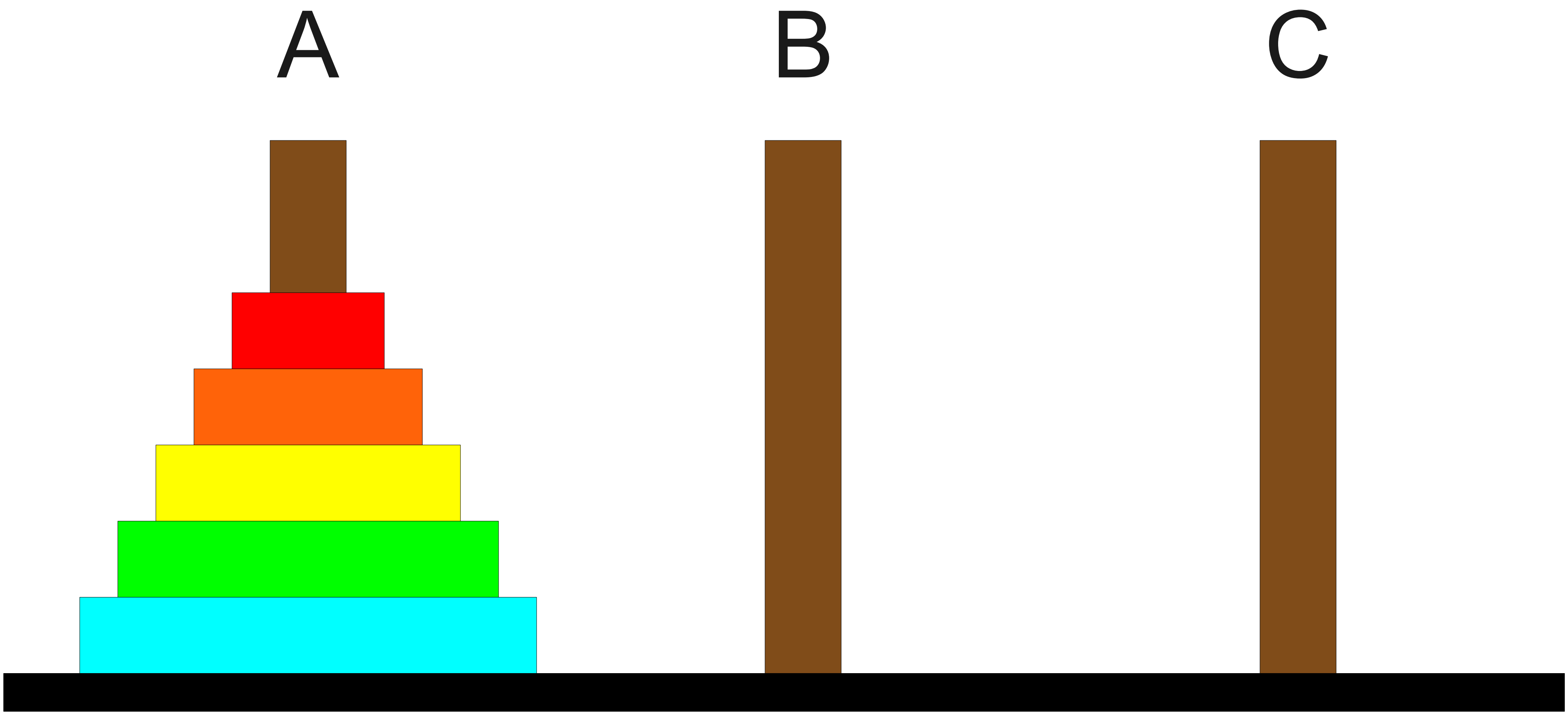
Solving the Towers of Hanoi



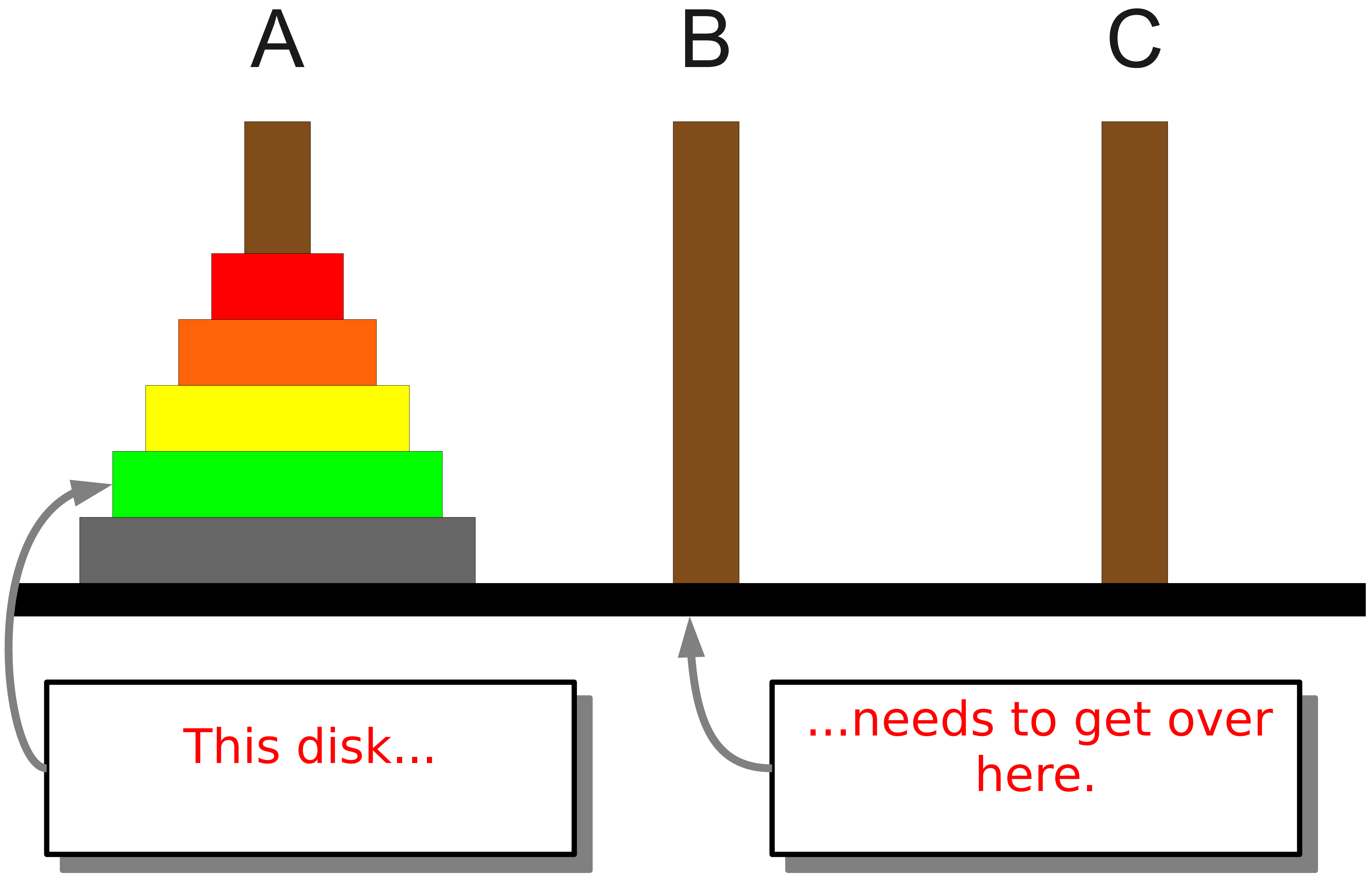
Solving the Towers of Hanoi



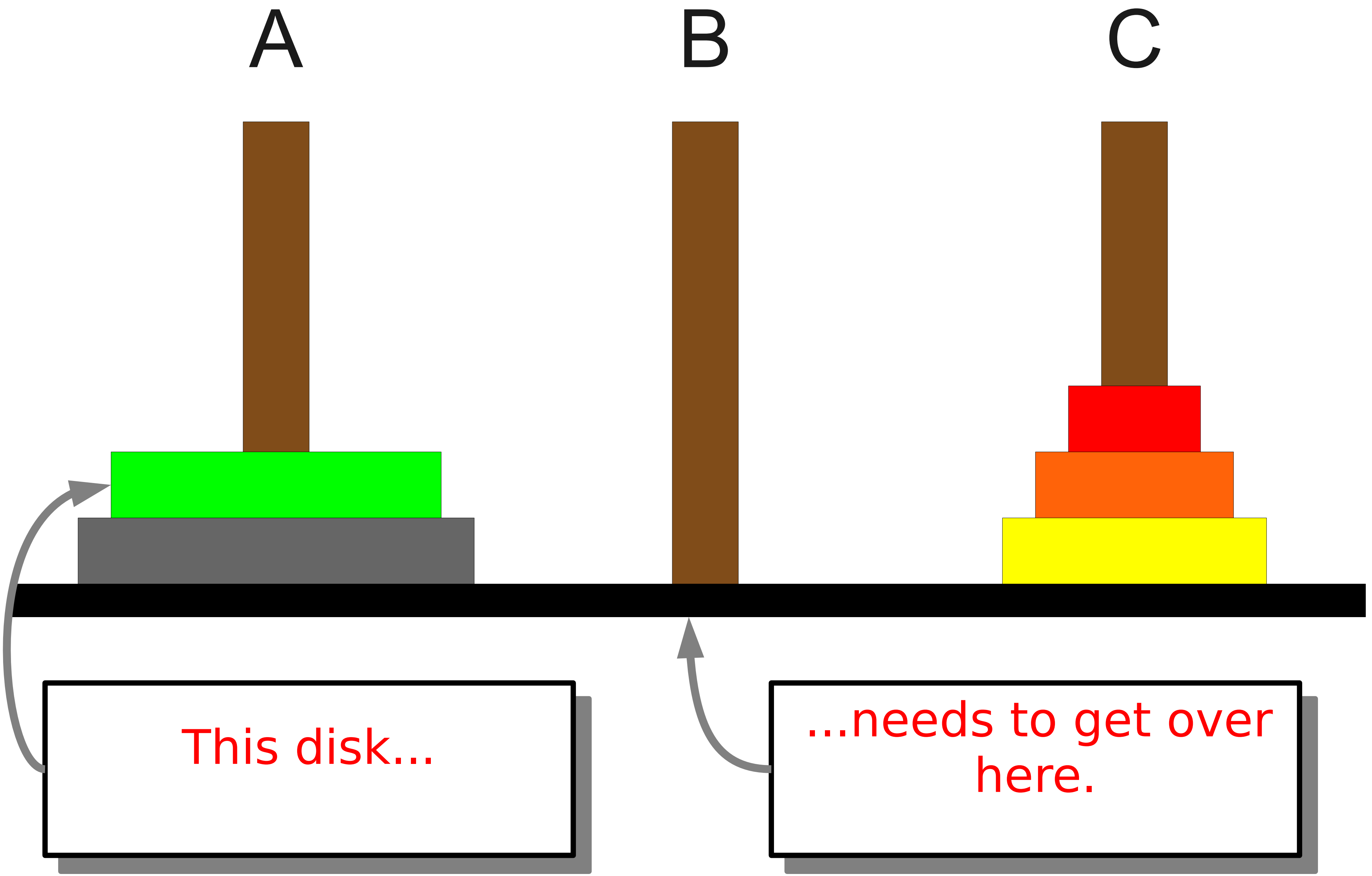
Solving the Towers of Hanoi



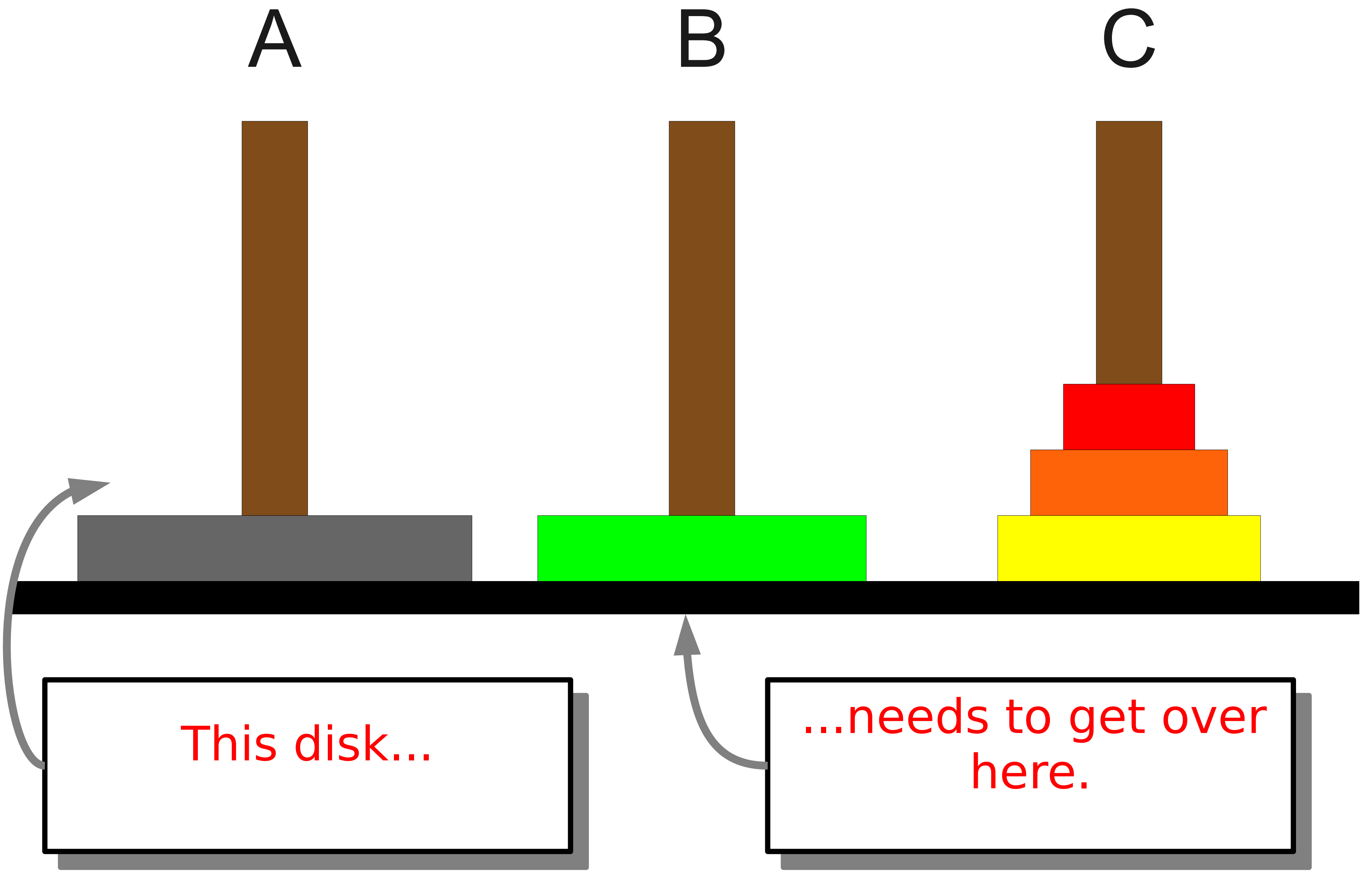
Solving the Towers of Hanoi



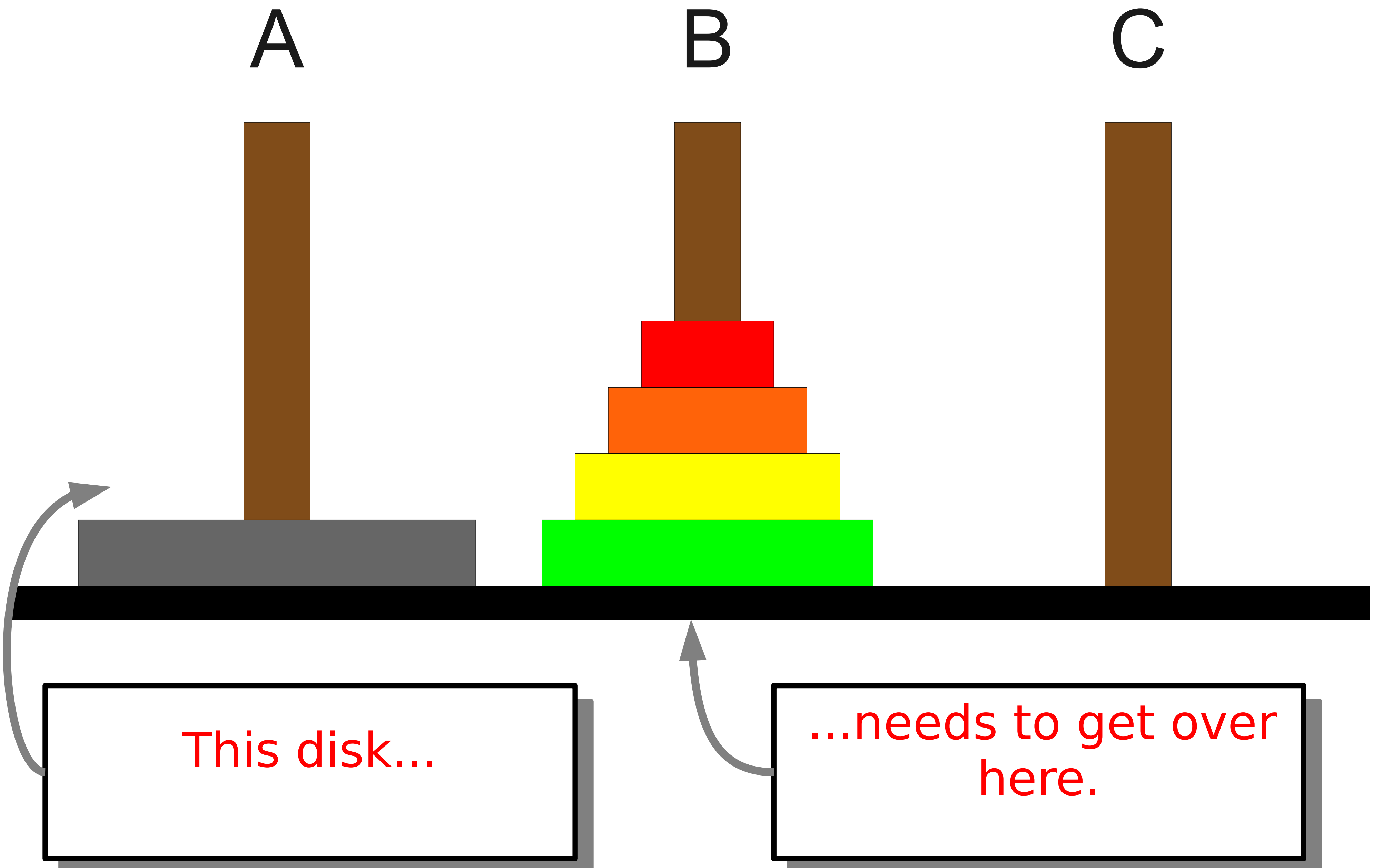
Solving the Towers of Hanoi



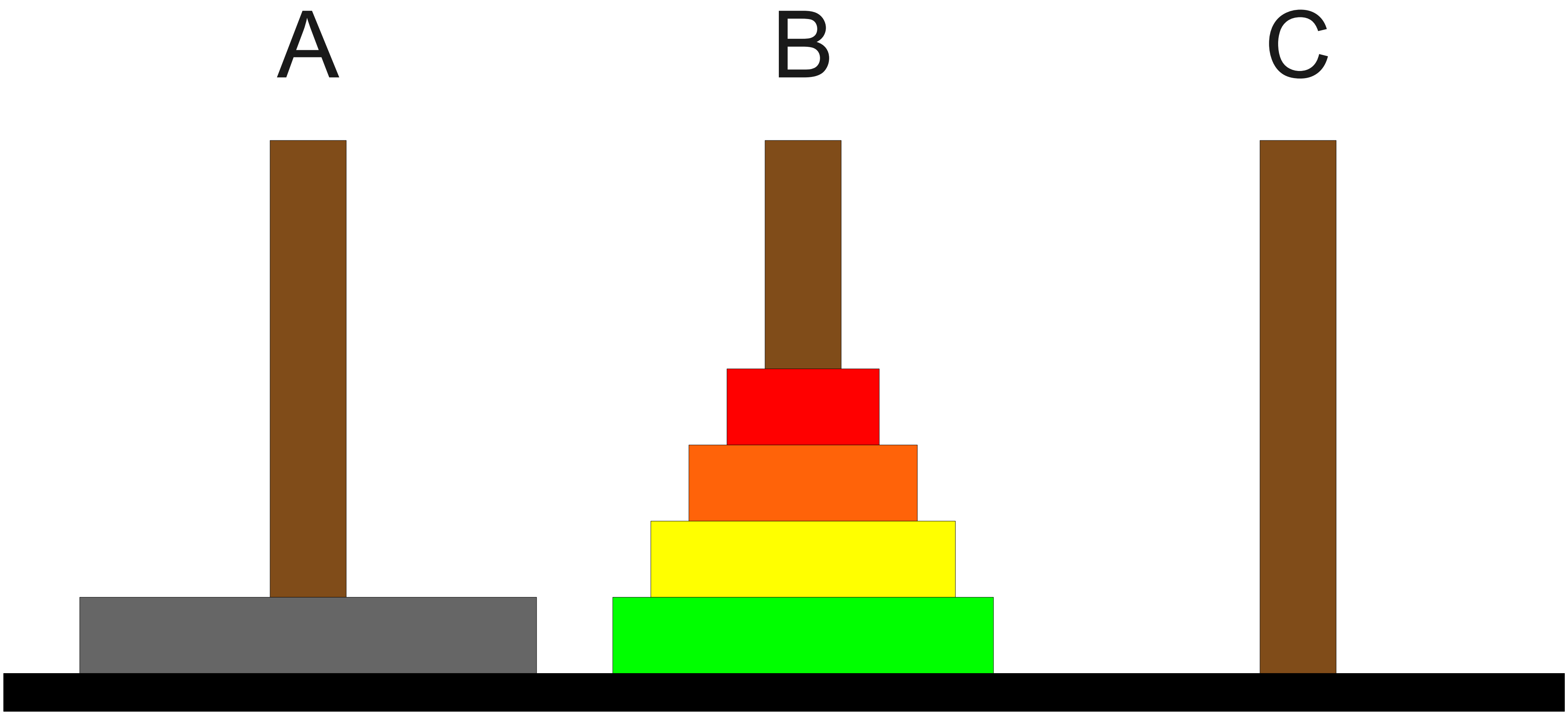
Solving the Towers of Hanoi



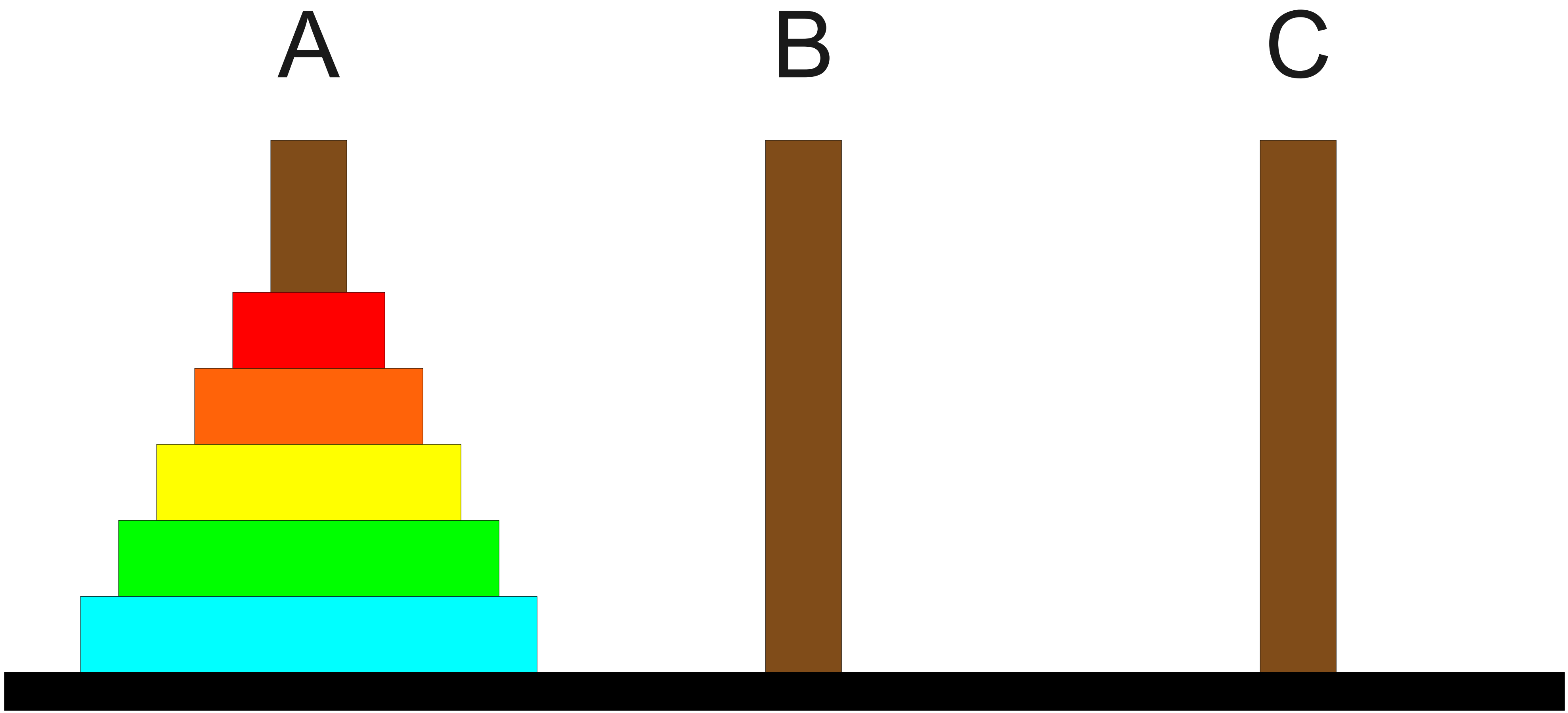
Solving the Towers of Hanoi



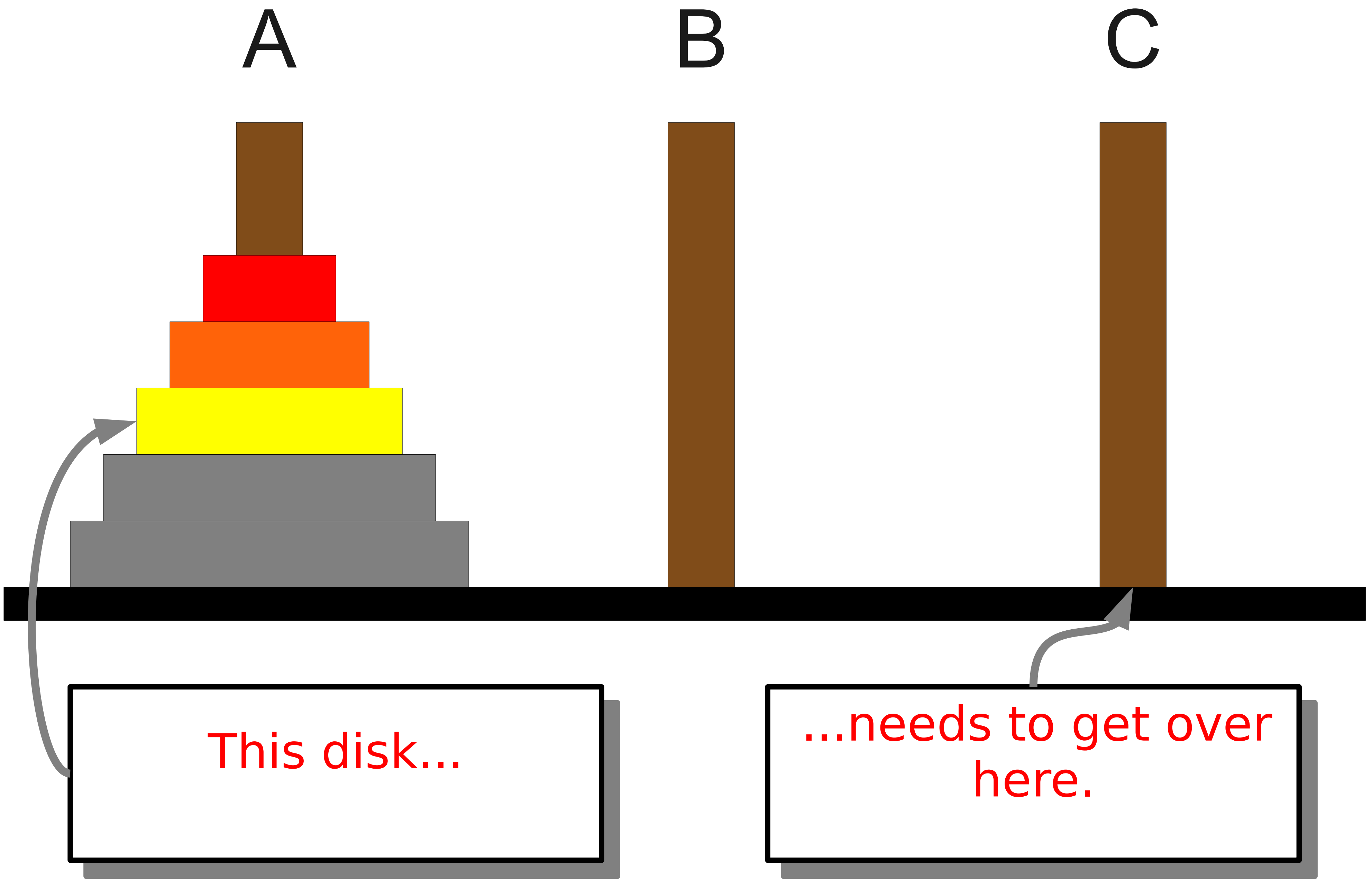
Solving the Towers of Hanoi



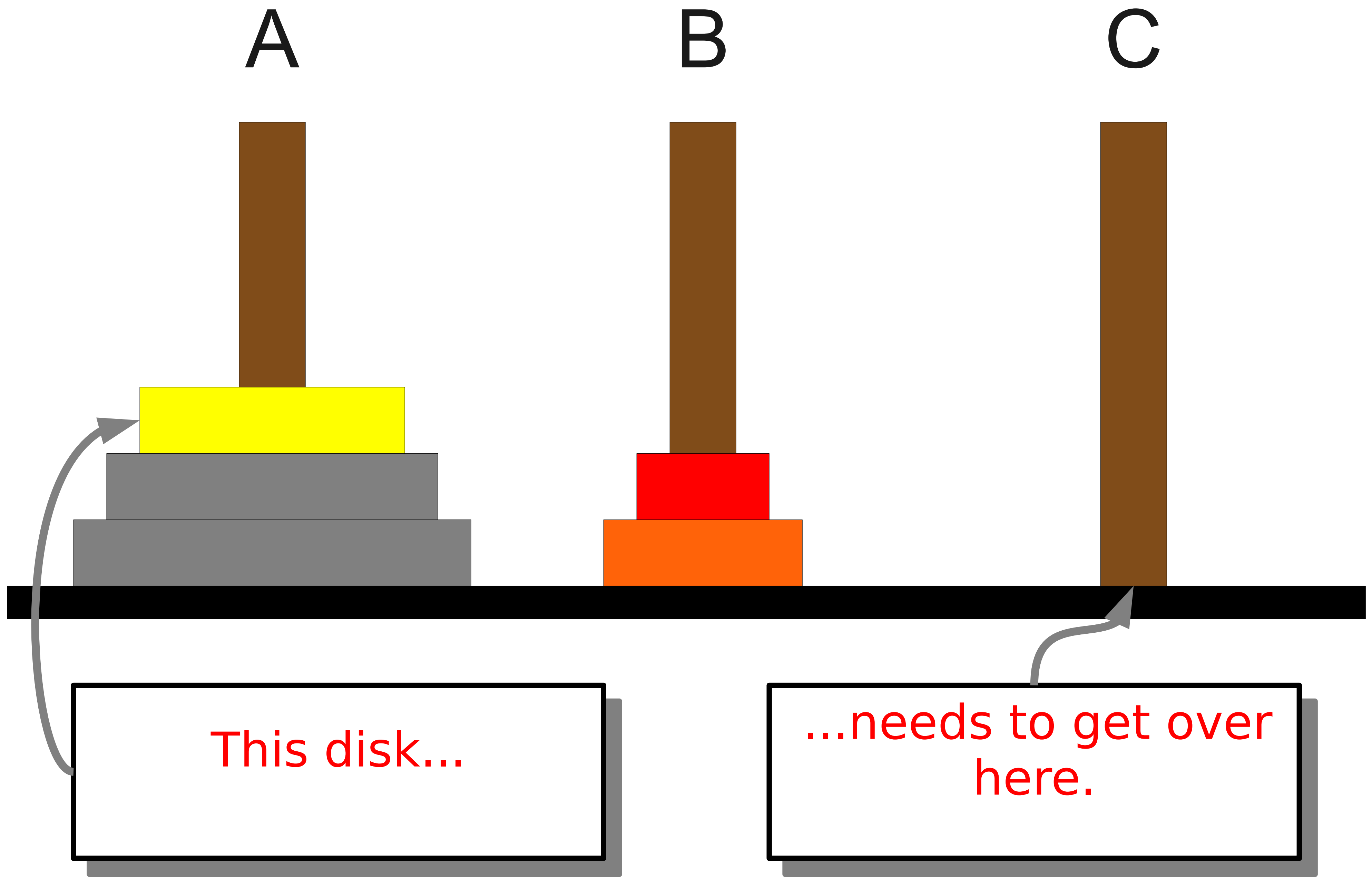
Solving the Towers of Hanoi



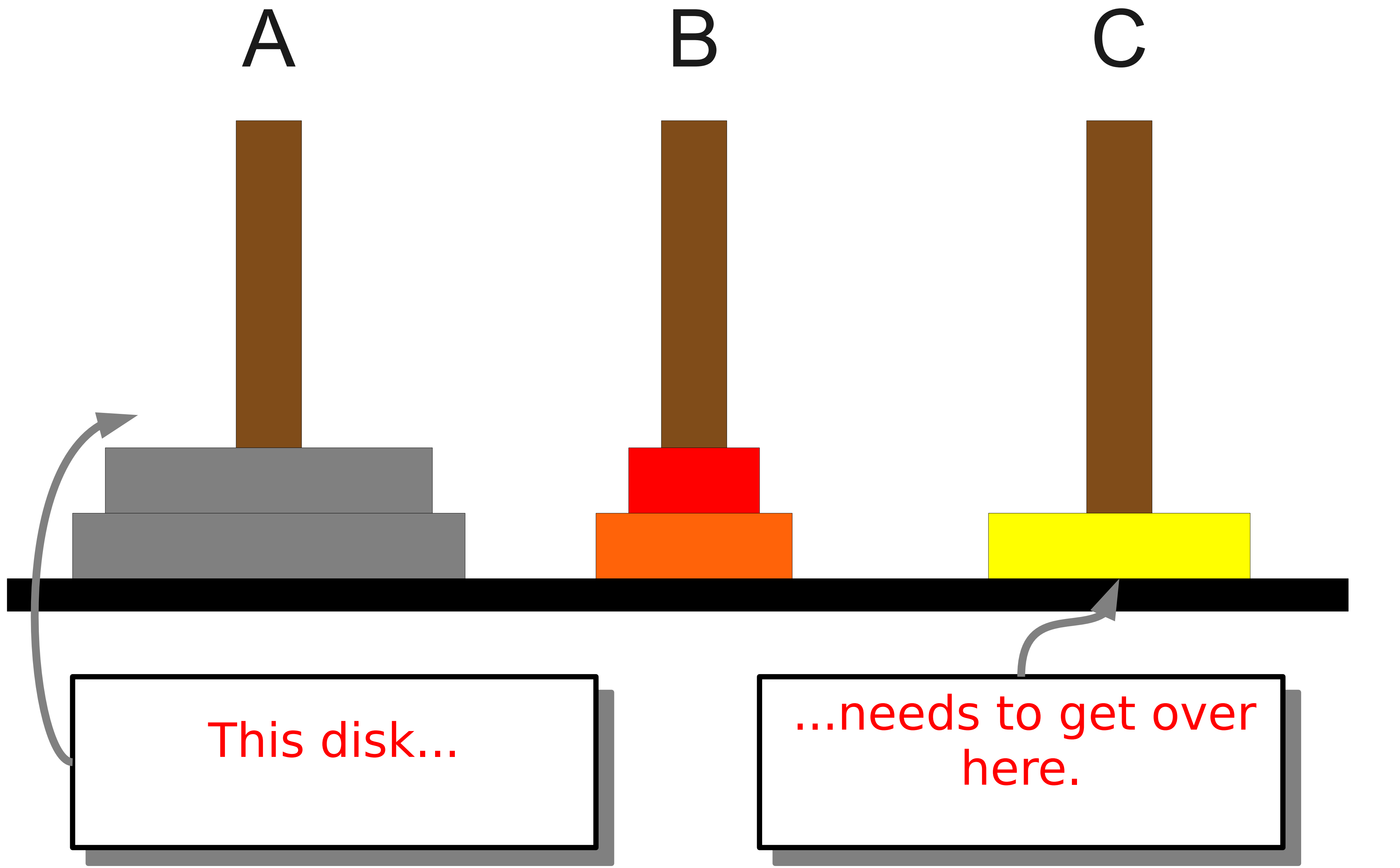
Solving the Towers of Hanoi



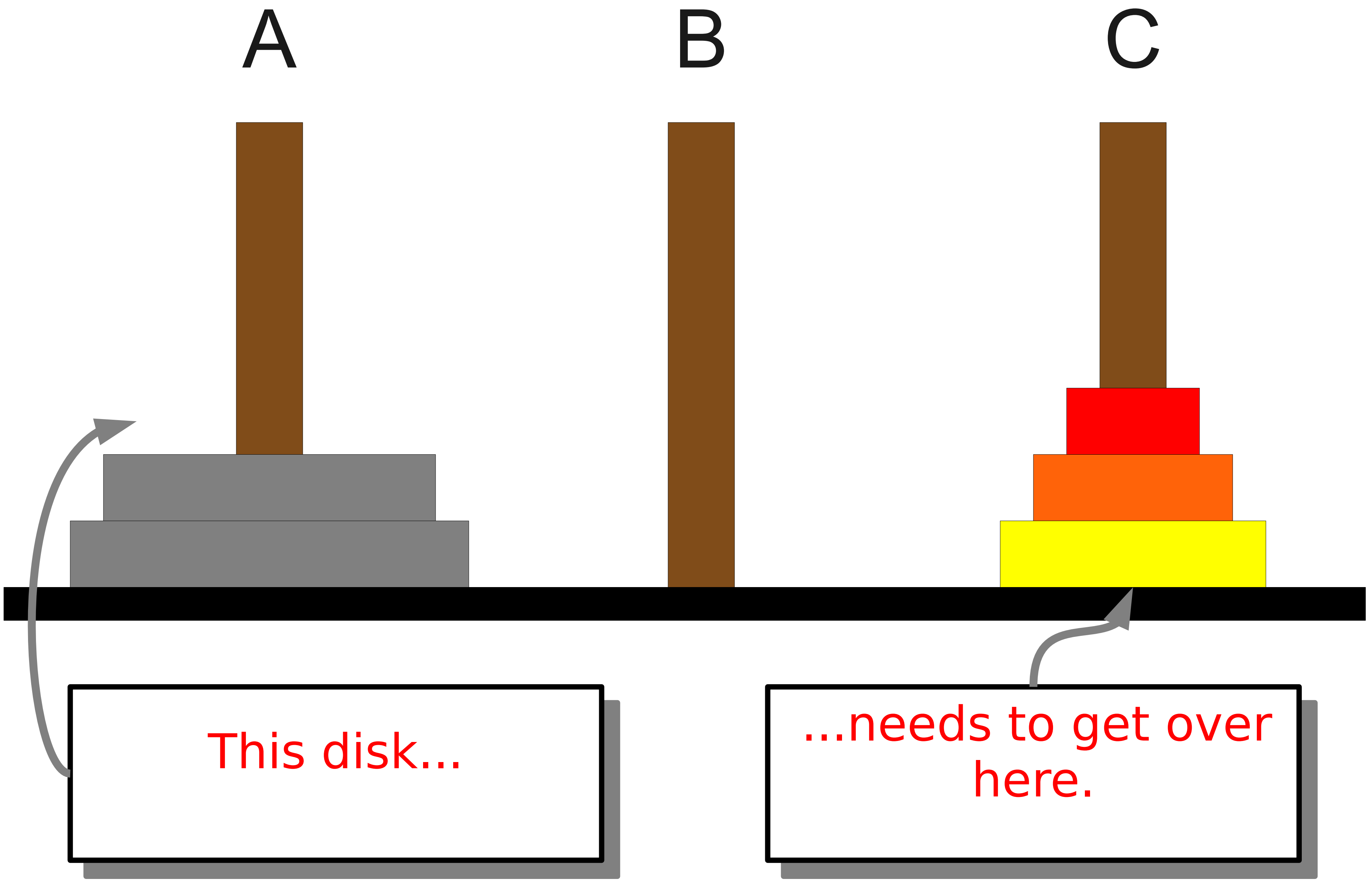
Solving the Towers of Hanoi



Solving the Towers of Hanoi



Solving the Towers of Hanoi



The Recursive Decomposition

- To get a tower of N+1 disks from spindle X to spindle Y, using Z as a temporary:
  - **Recursively** move the top N disks from spindle X to spindle Z, using Y as a temporary.
  - Move the N+1st disk from X to Y.
  - **Recursively** move the N disks from spindle Z to spindle Y, using X as a temporary.

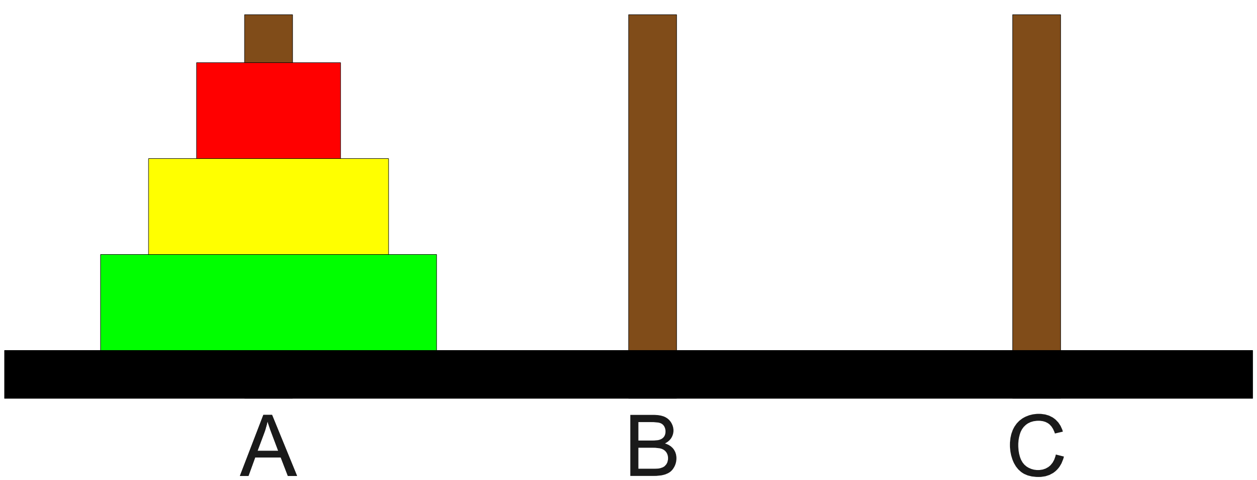
The Base Case

- 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.

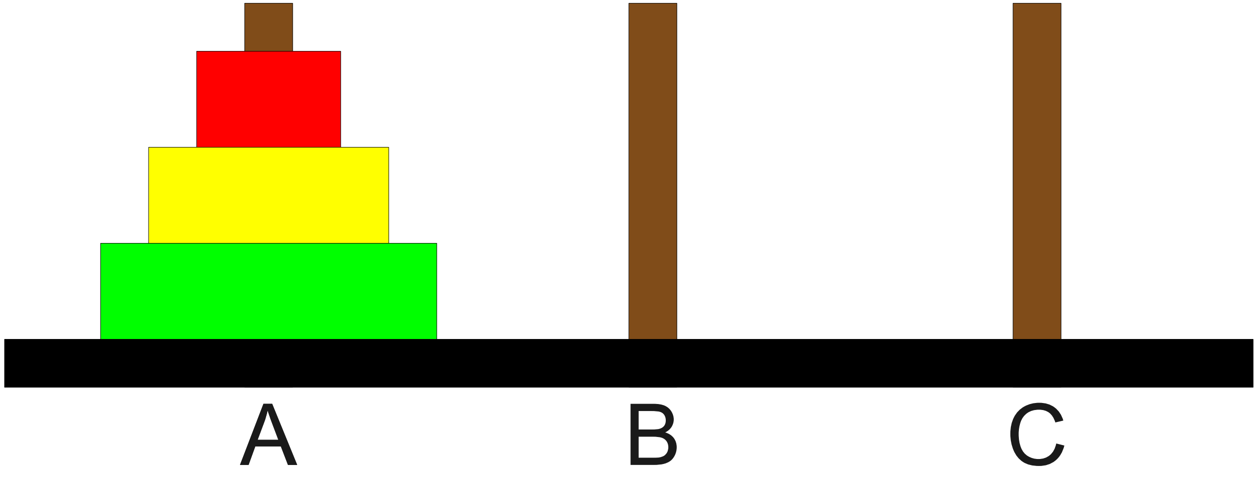
And now, the solution...

```
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');
}
```



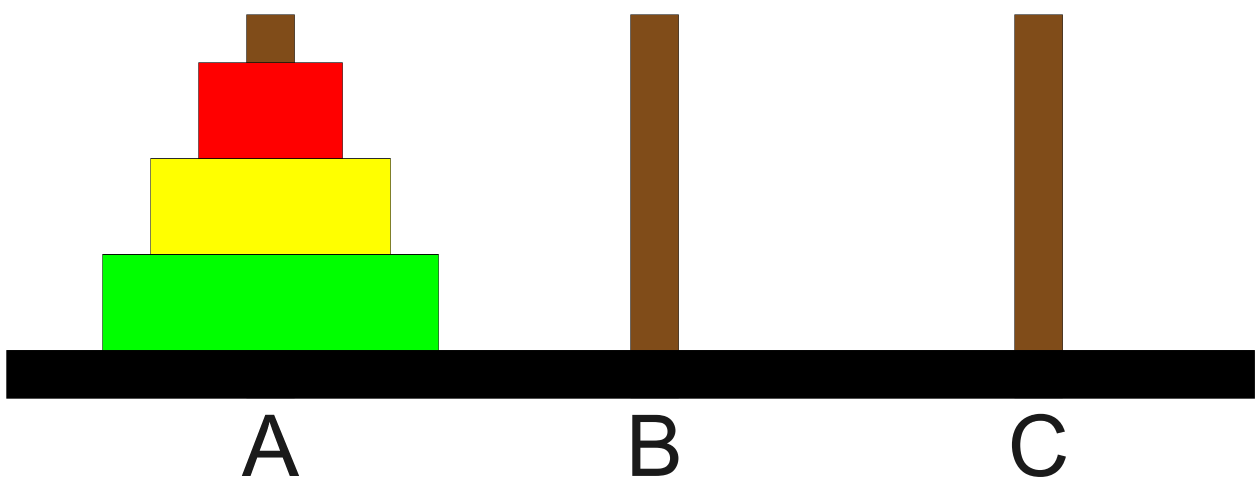
```
int main() {
    moveTower(3, 'a', 'c', 'b');
}
```



```
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);
    }
}
```

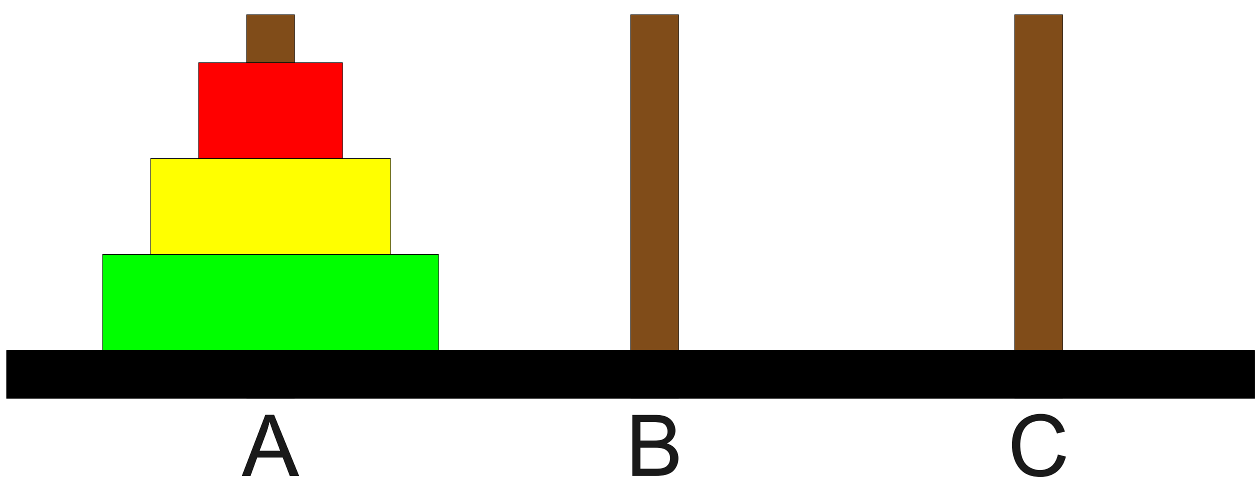
n	from	to	temp
3	a	c	b



```
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);
    }
}
```

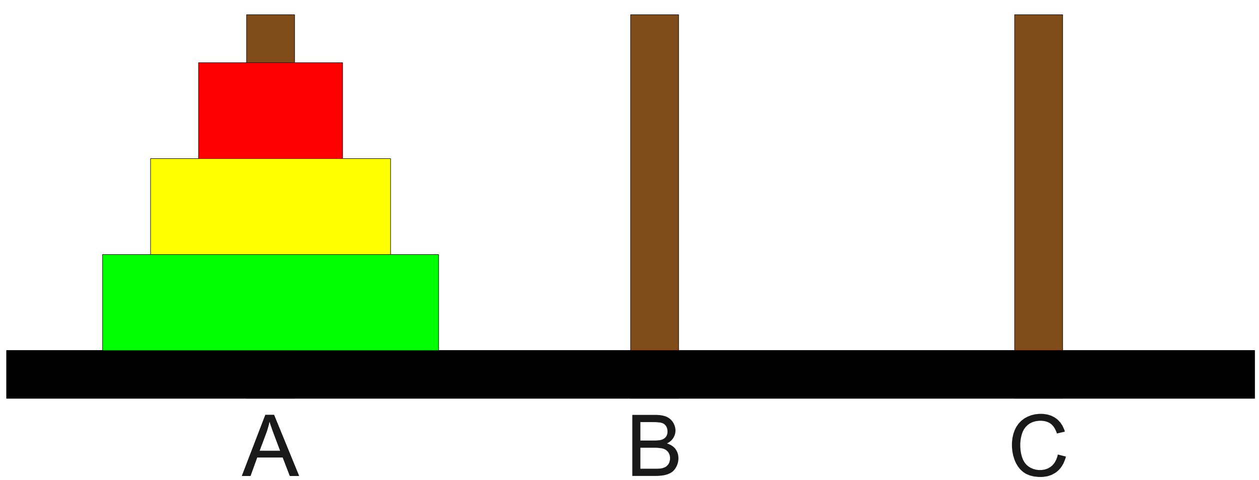
n	from	to	temp
3	a	c	b



```
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);
    }
}
```

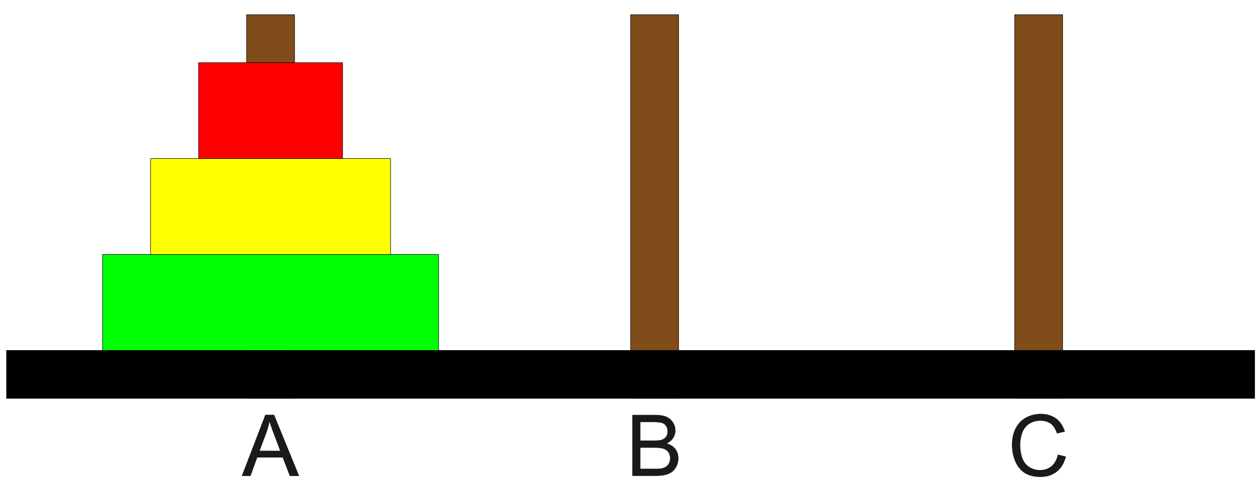
n	from	to	temp
3	a	c	b



```
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

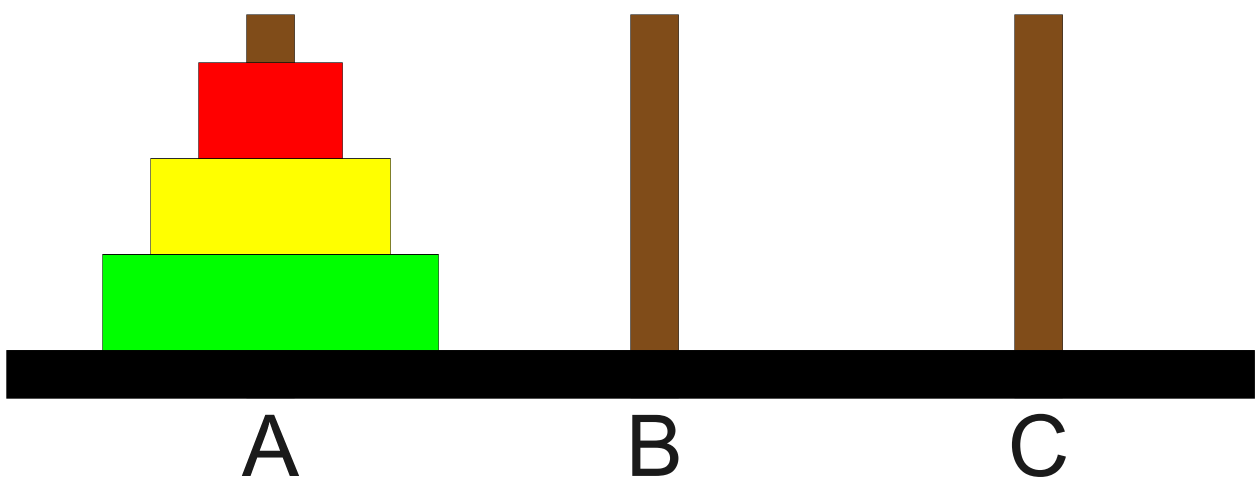


```
int main() {

}
```

```
void moveTower(int n, char from, char to, char temp) {

}
```

```
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

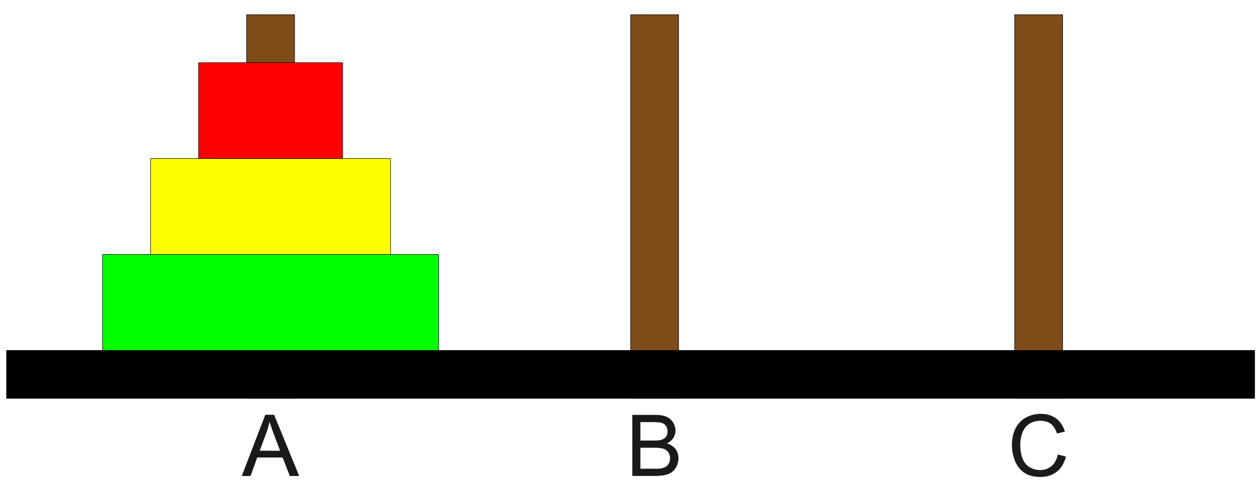


```
int main() {

}
```

```
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	from	to	temp
2	a	b	c

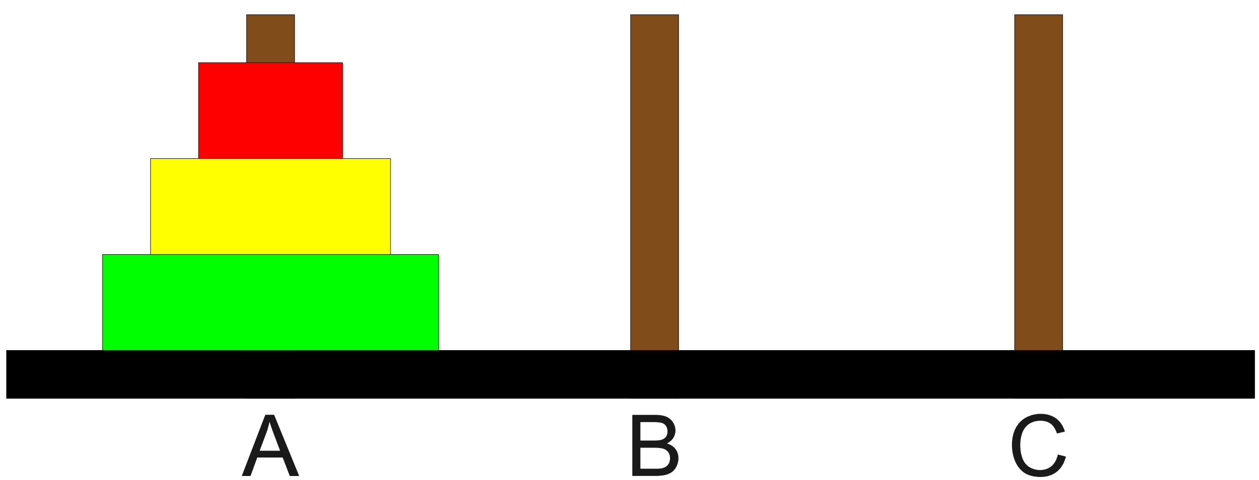


```
int main() {

}
```

```
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

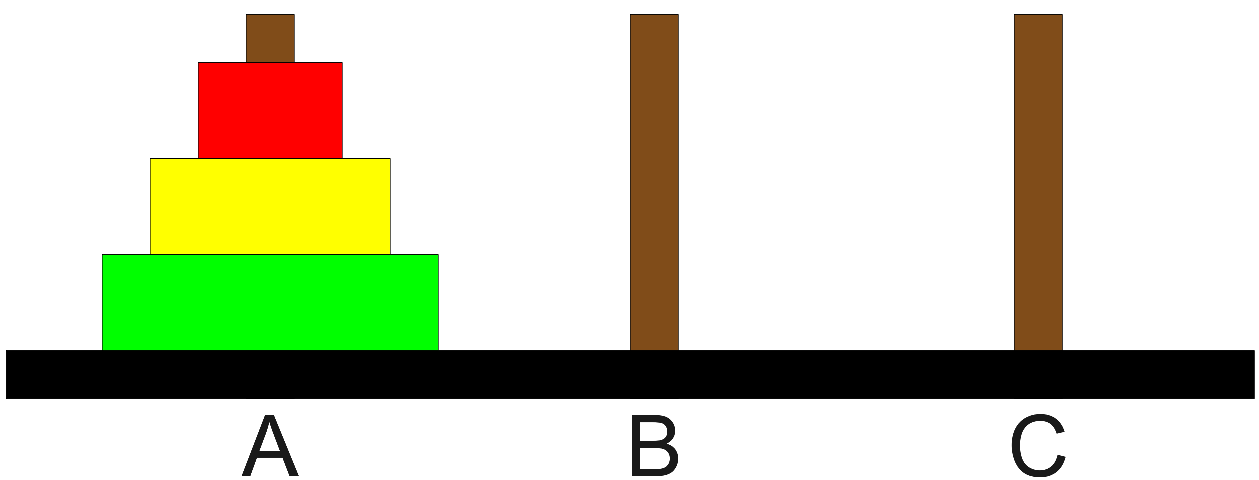


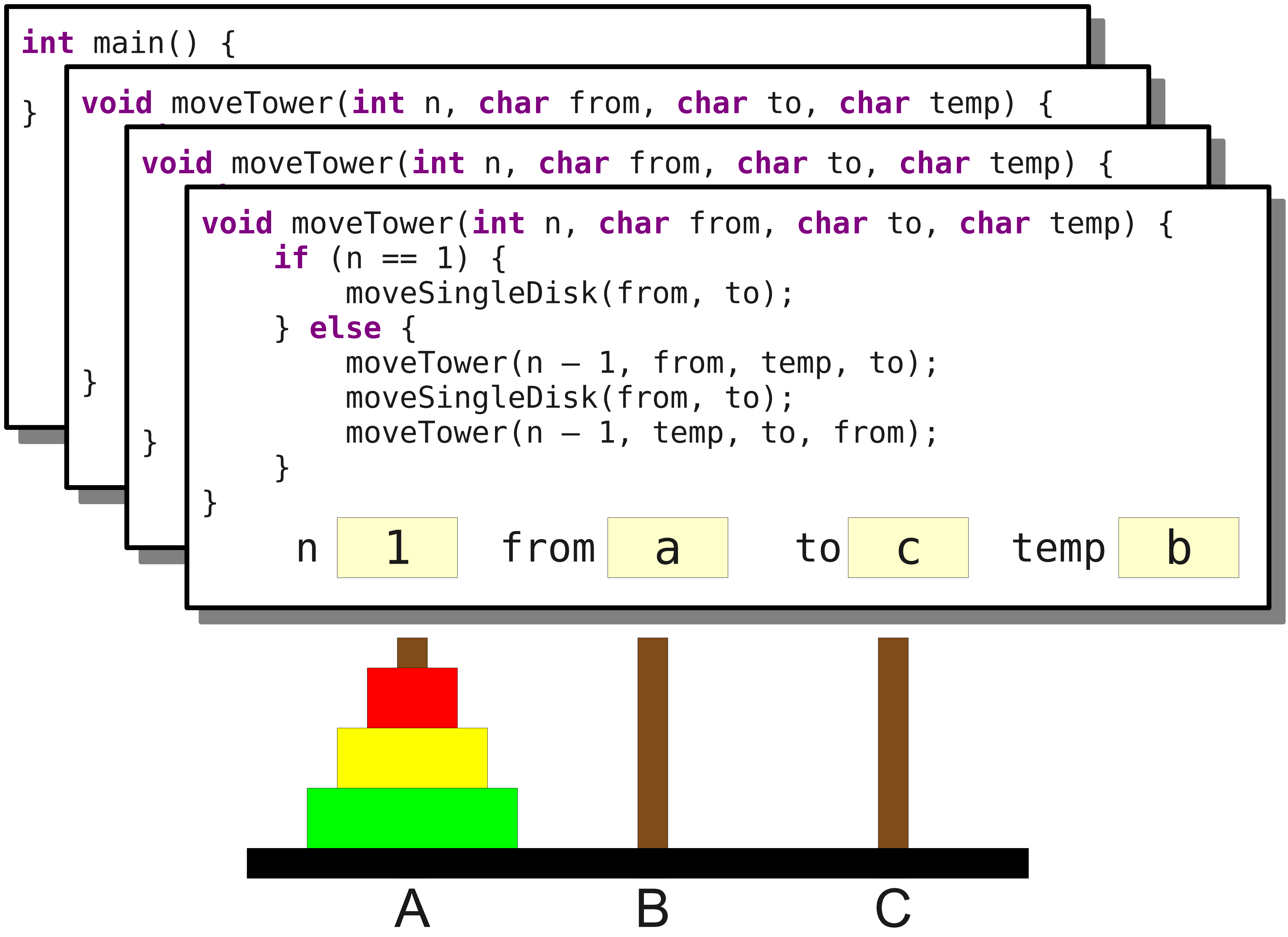
```
int main() {

}
```

```
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	from	to	temp
2	a	b	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
A
B
C

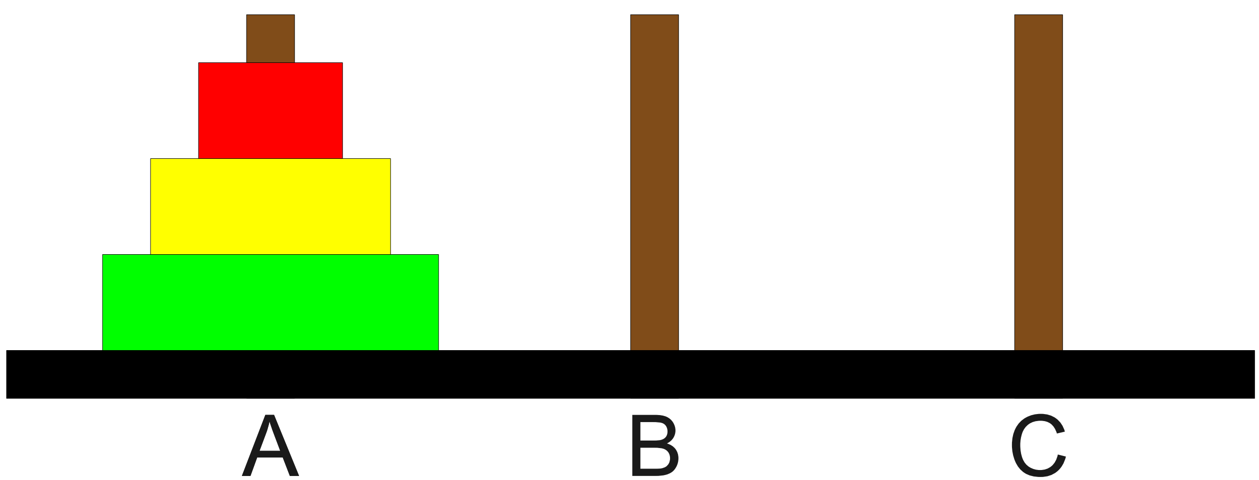
```
int main() {
}
```

```
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	from	to	temp
1	a	c	b



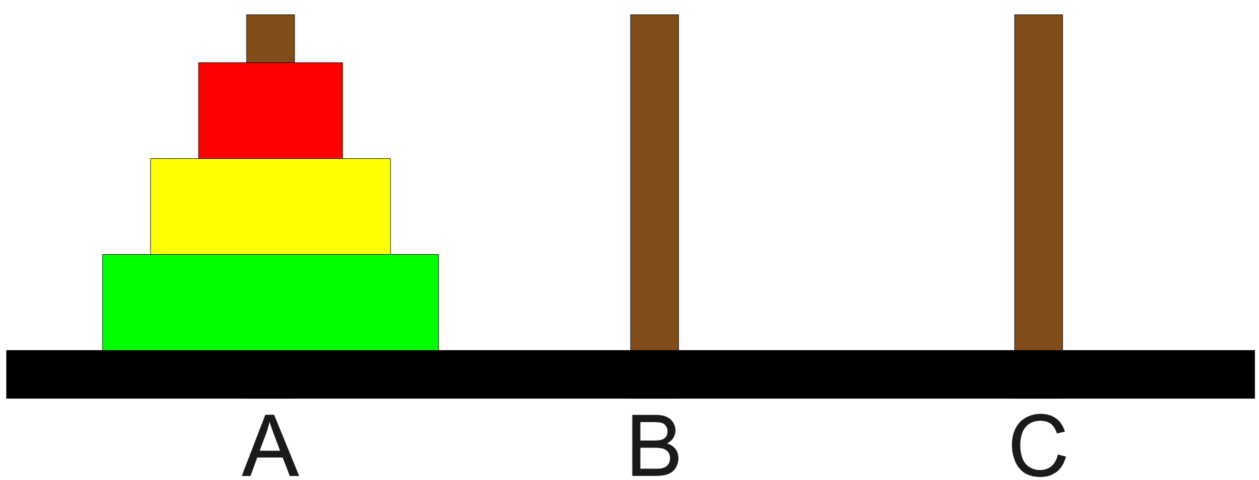
```
int main() {

}
```

```
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	from	to	temp
1	a	c	b



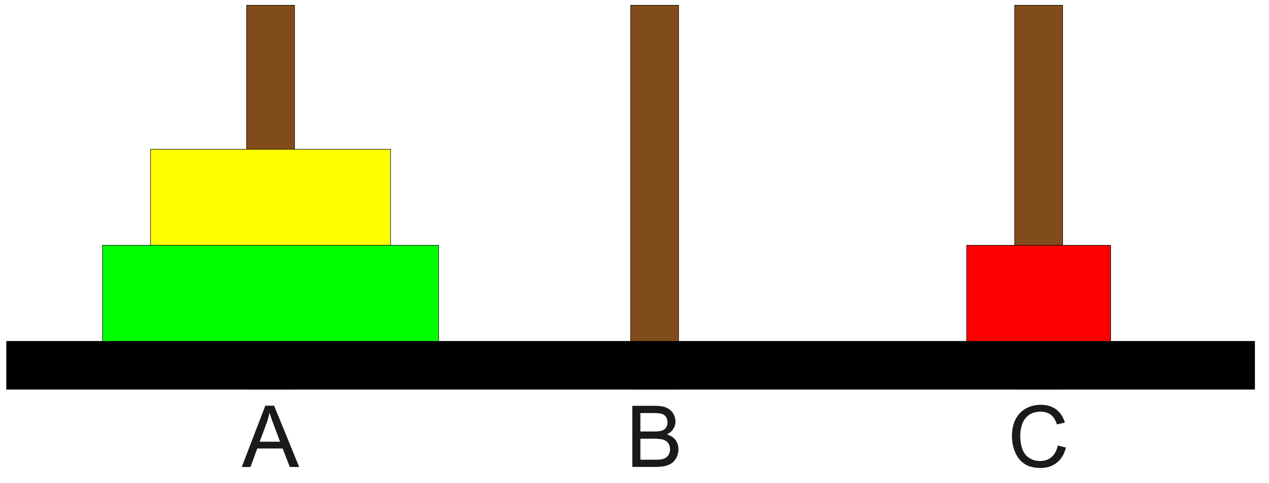
```
int main() {

}
```

```
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	from	to	temp
1	a	c	b

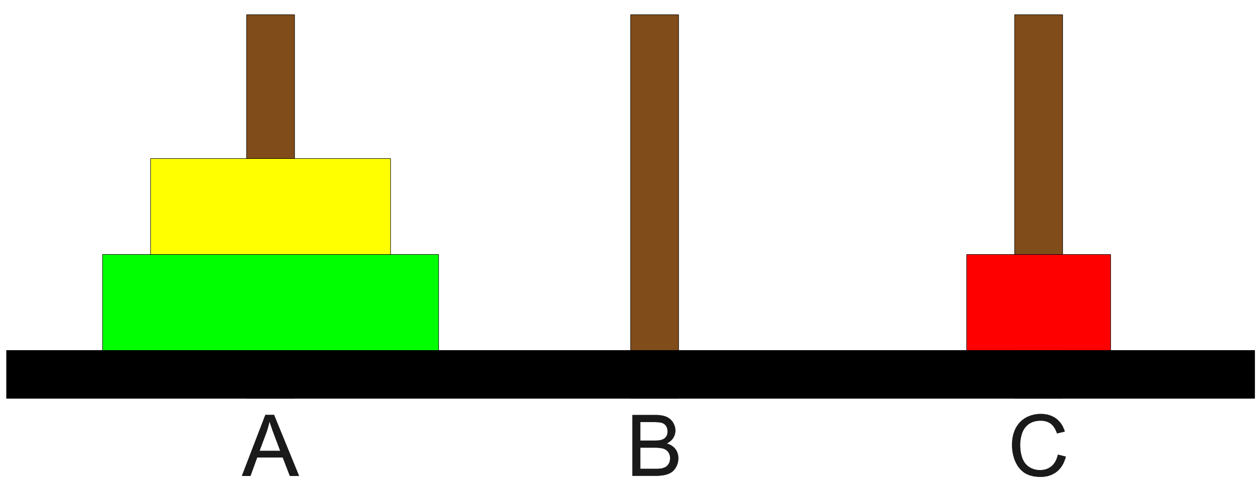


```
int main() {

}
```

```
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	from	to	temp
2	a	b	c

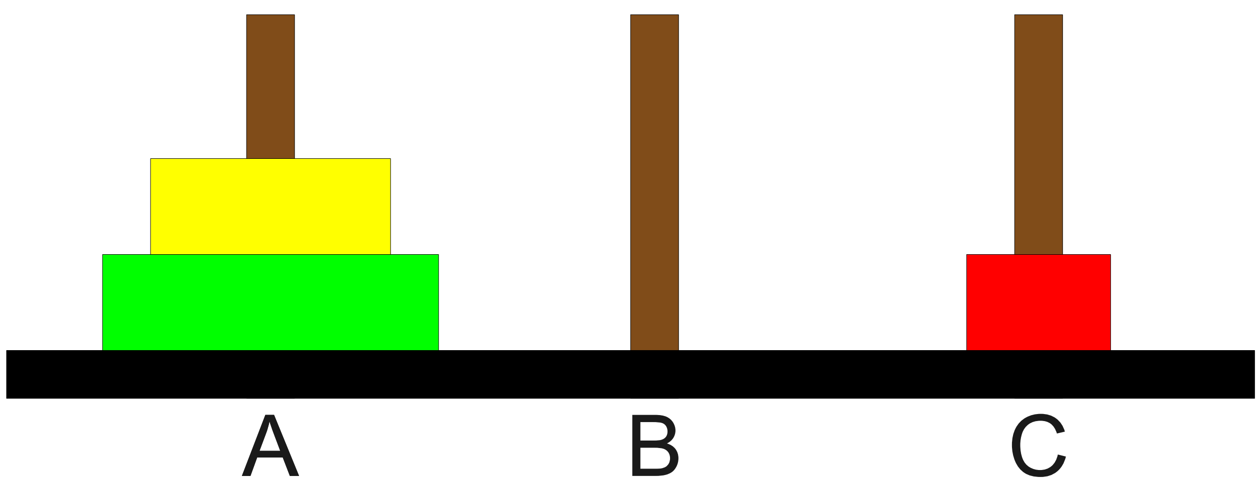


```
int main() {

}
```

```
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

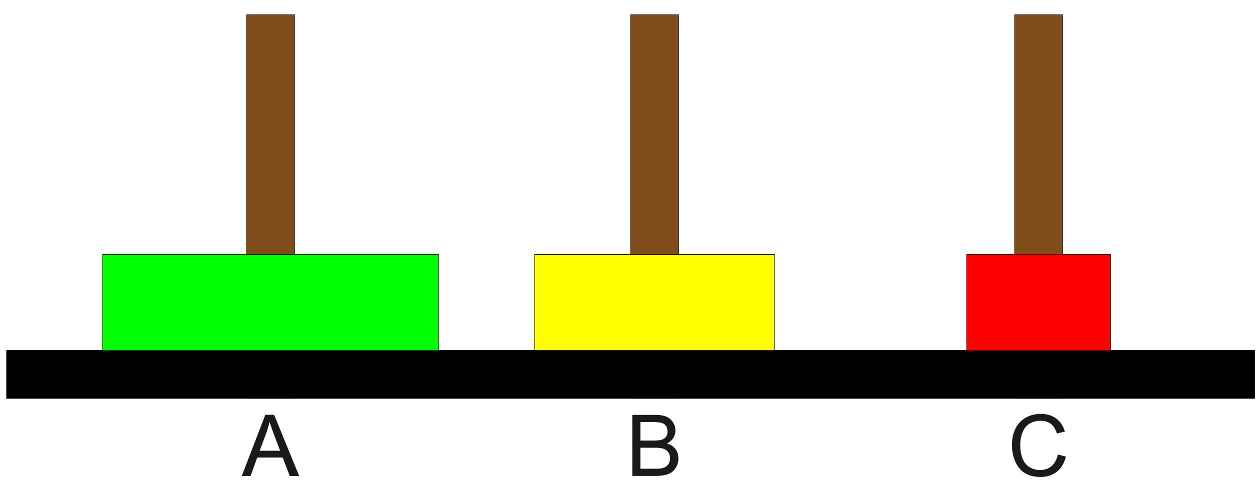


```
int main() {

}
```

```
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**

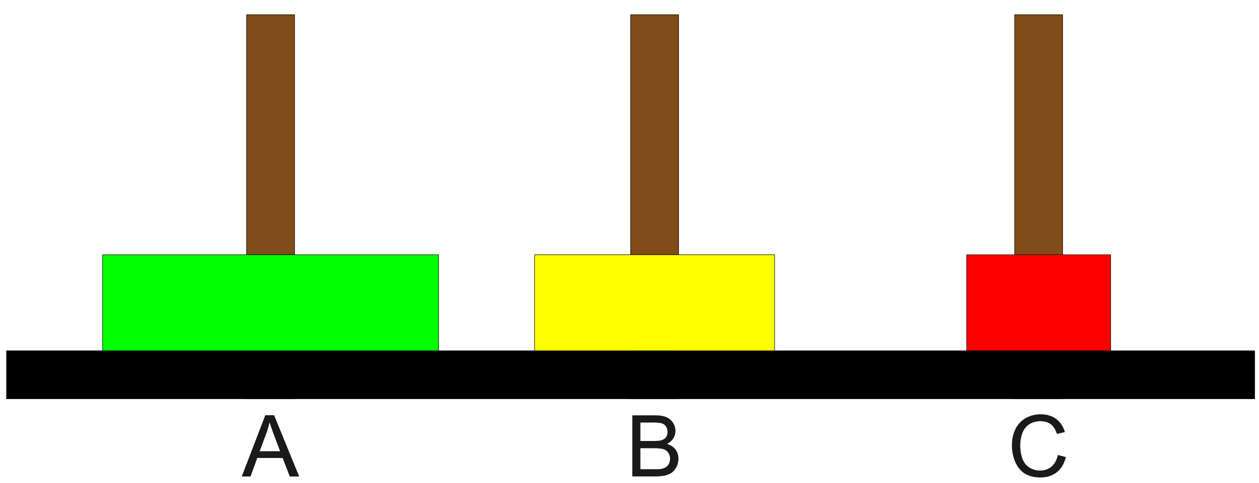


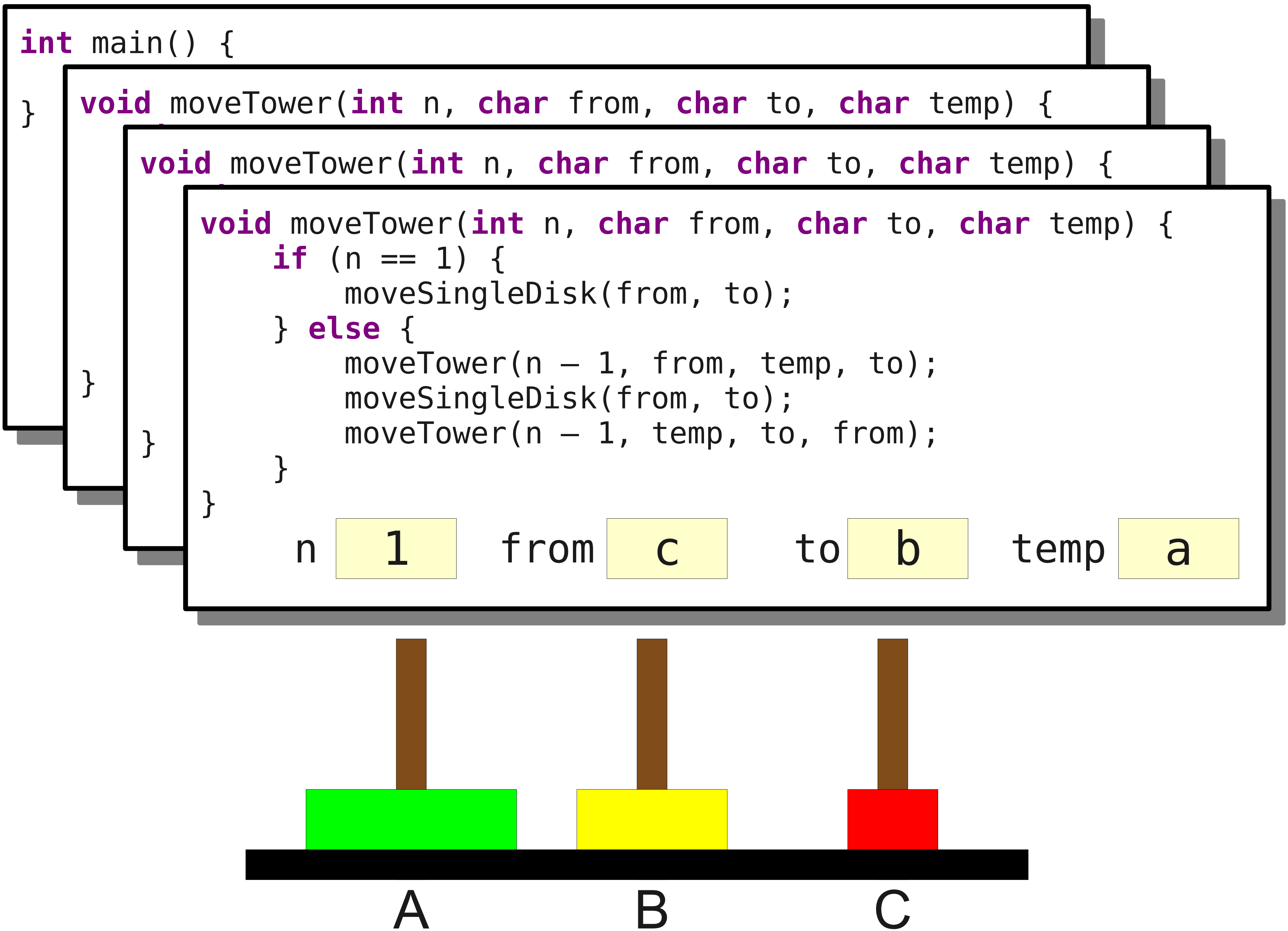
```
int main() {

}
```

```
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	from	to	temp
2	a	b	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
A
B
C

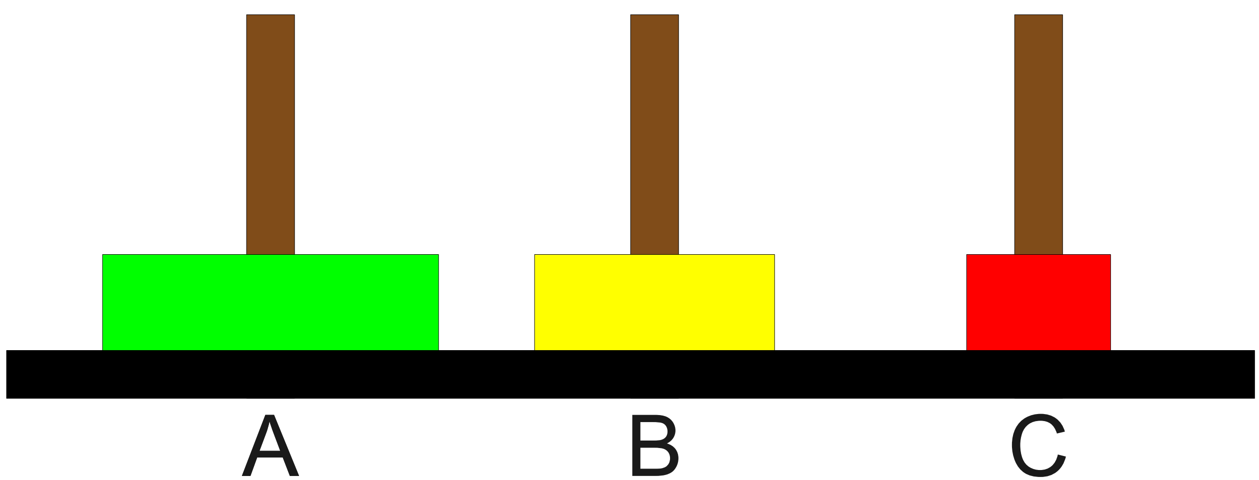
```
int main() {

}
```

```
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



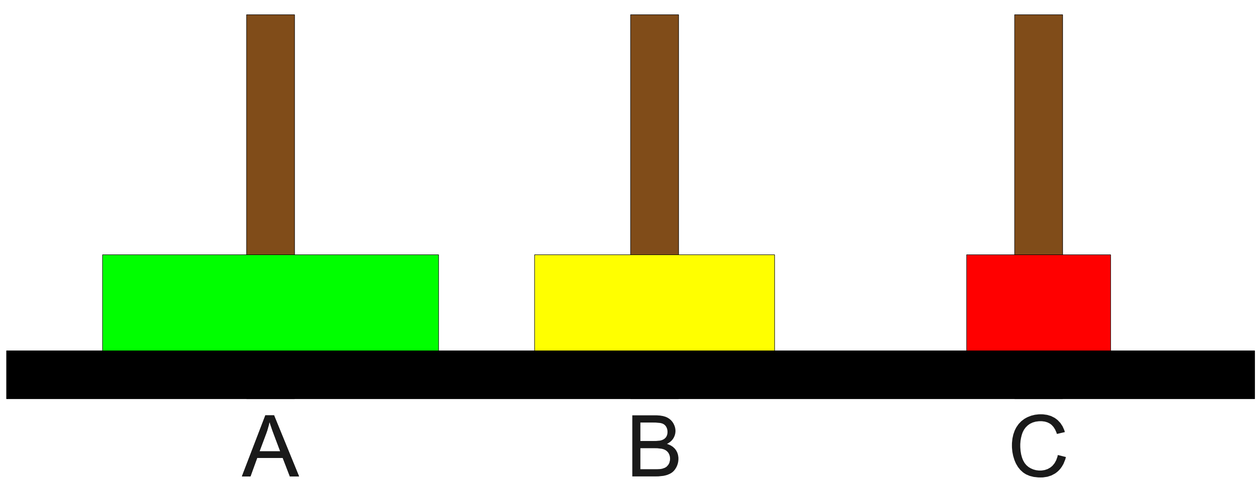
```
int main() {

}
```

```
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



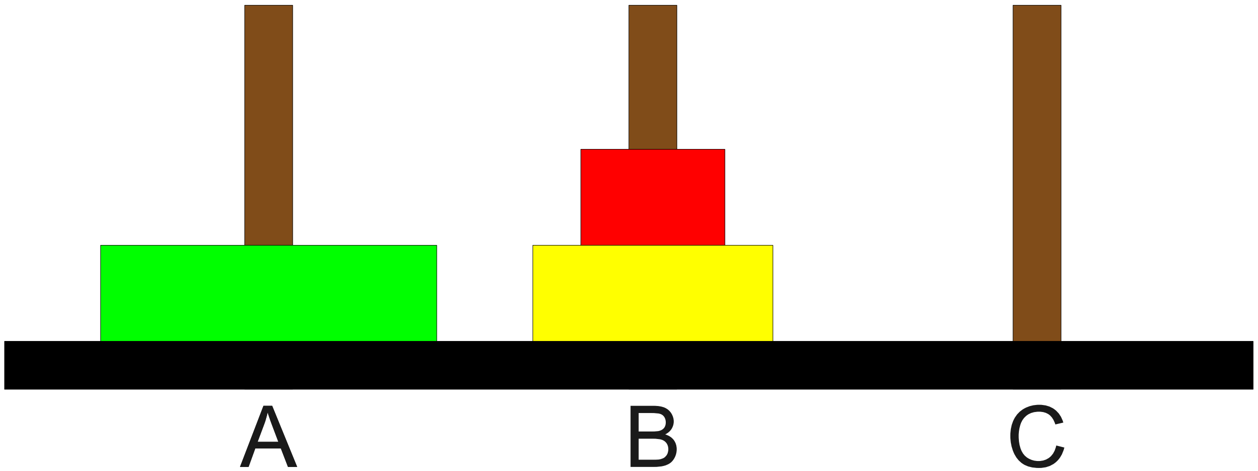
```
int main() {

}
```

```
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

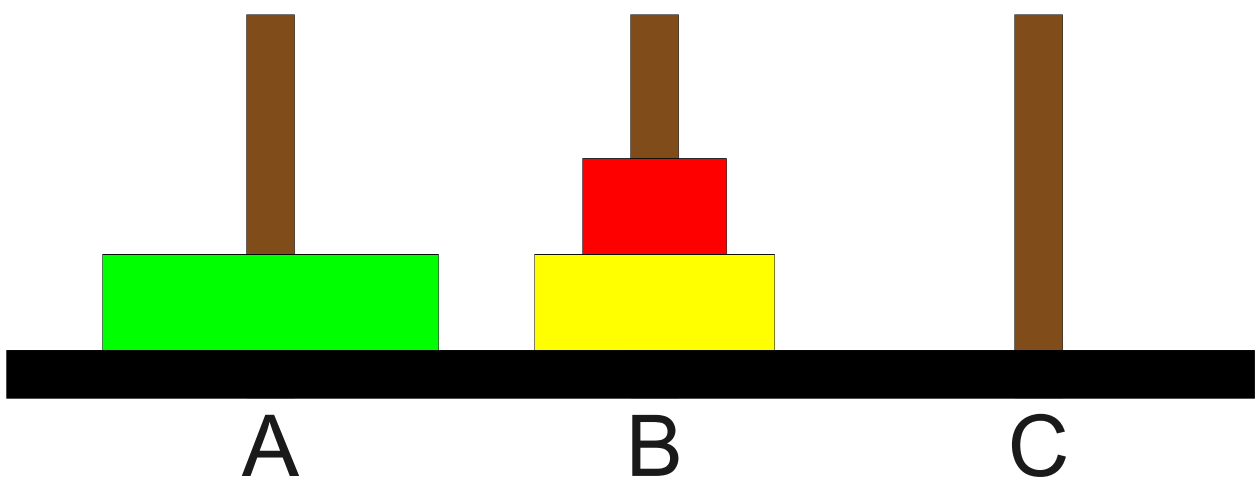


```
int main() {

}
```

```
void moveTower(int n, char from, char to, char temp) {

}
```

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

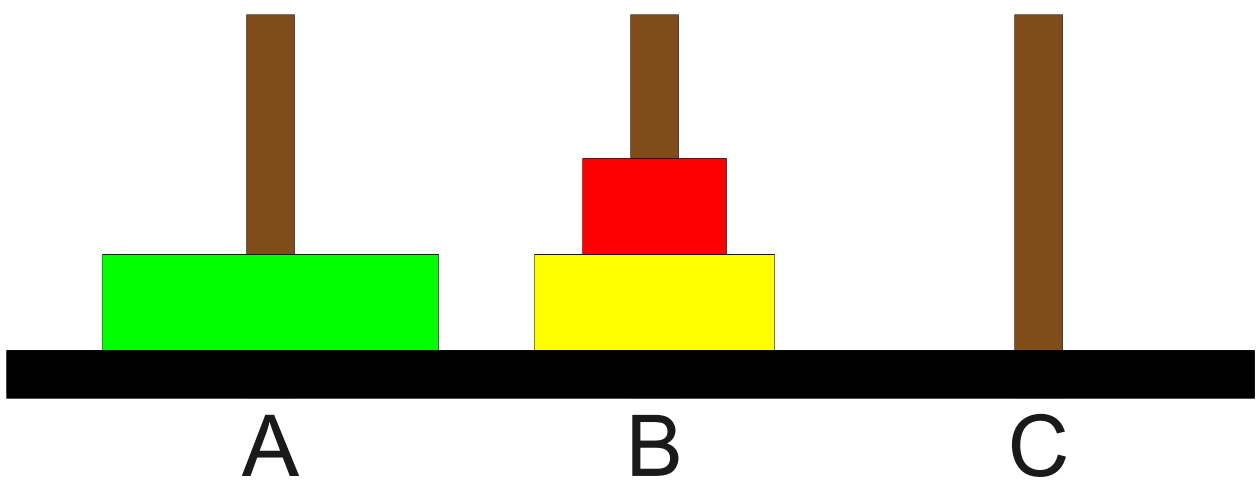
n: 2 | from: a | to: b | temp: c



```
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);
    }
}
```

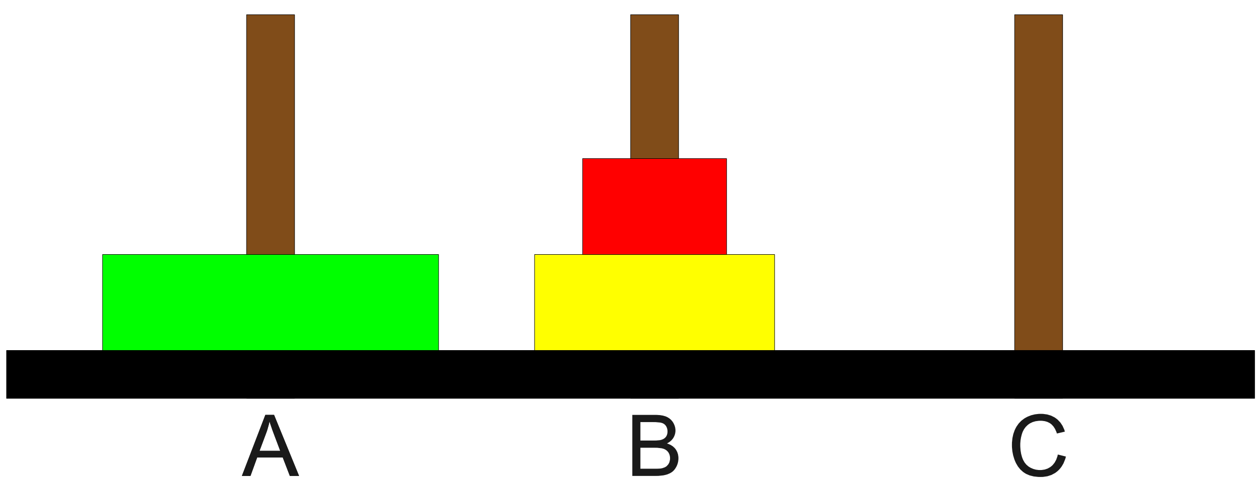
n 3 from a to c temp b



```
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);
    }
}
```

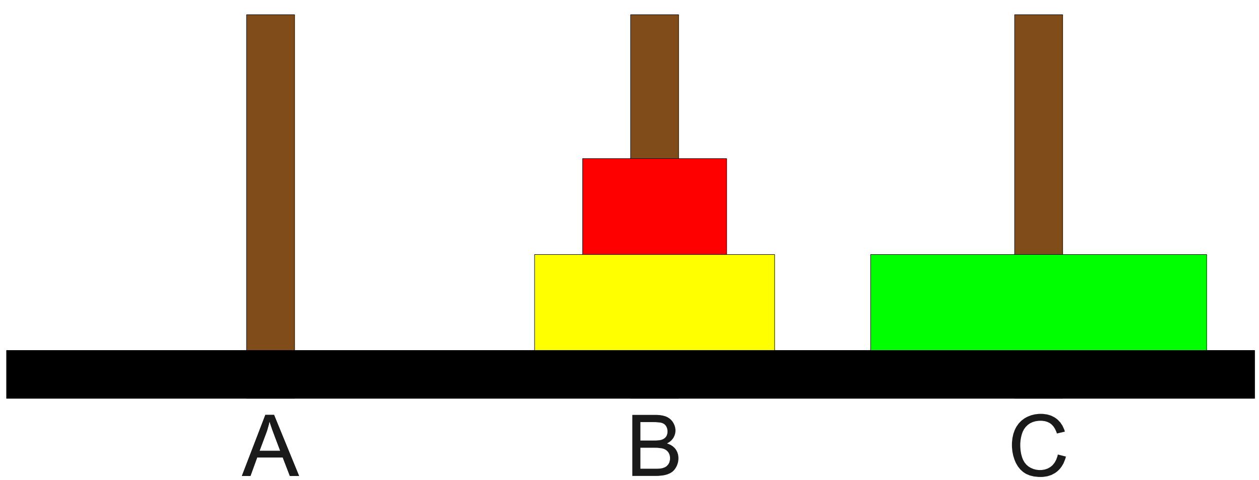
n	from	to	temp
3	a	c	b



```
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);
    }
}
```

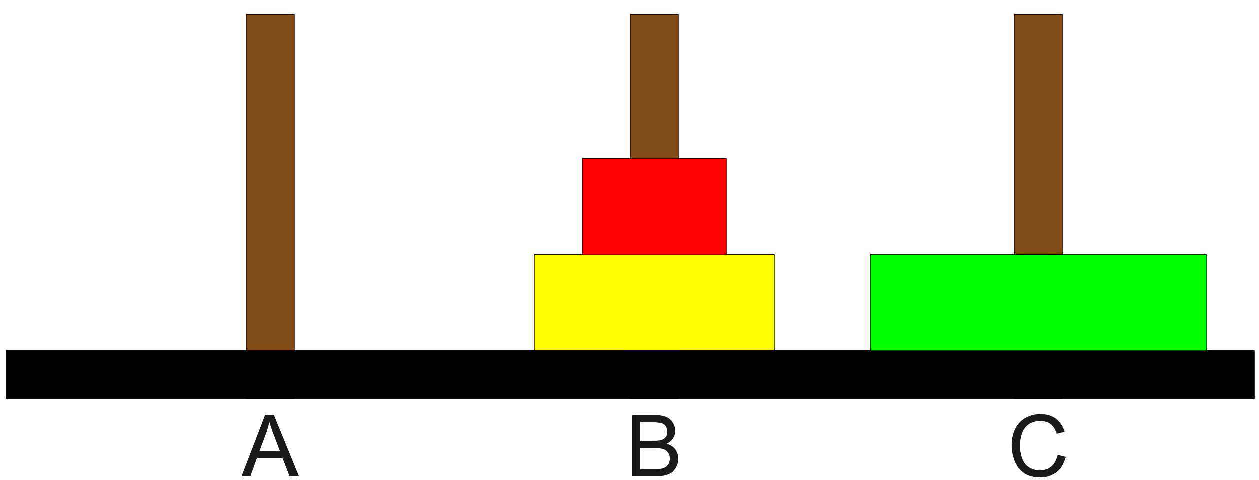
n 3 from a to c temp b



```
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	from	to	temp
3	a	c	b

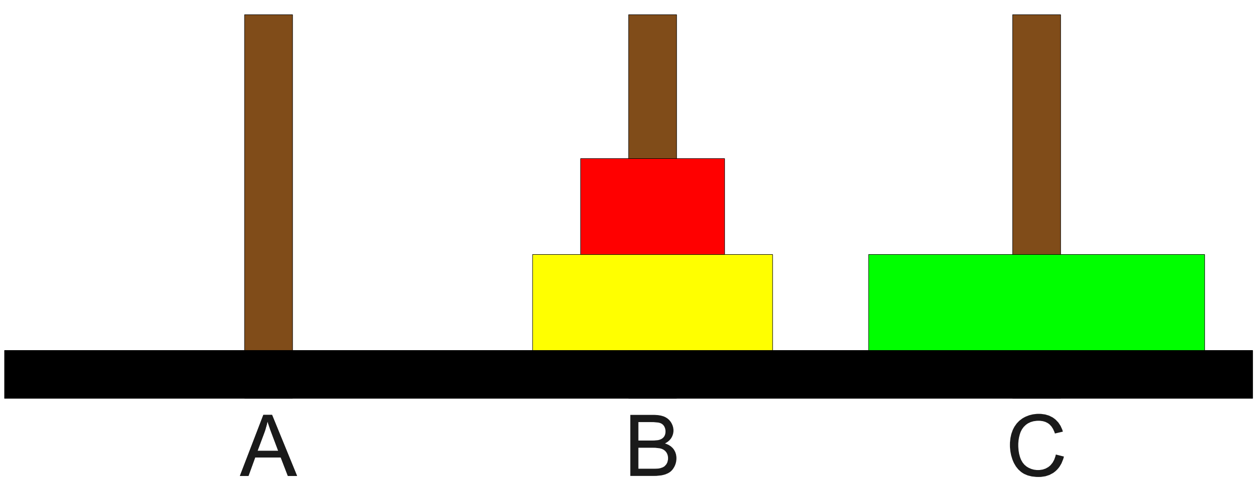


```
int main() {

}
```

```
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	from	to	temp
2	b	c	a

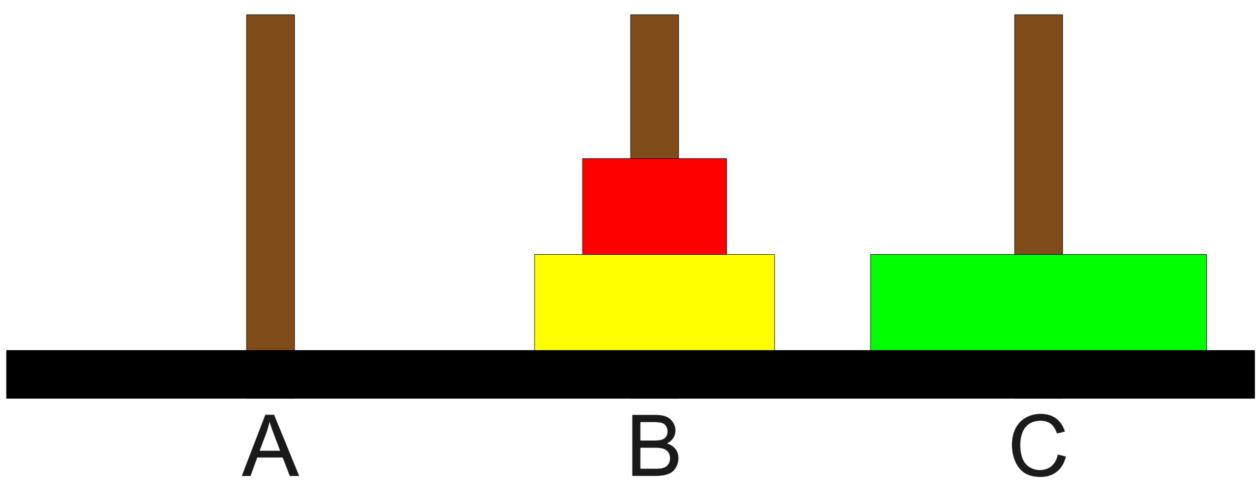


```
int main() {

}
```

```
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

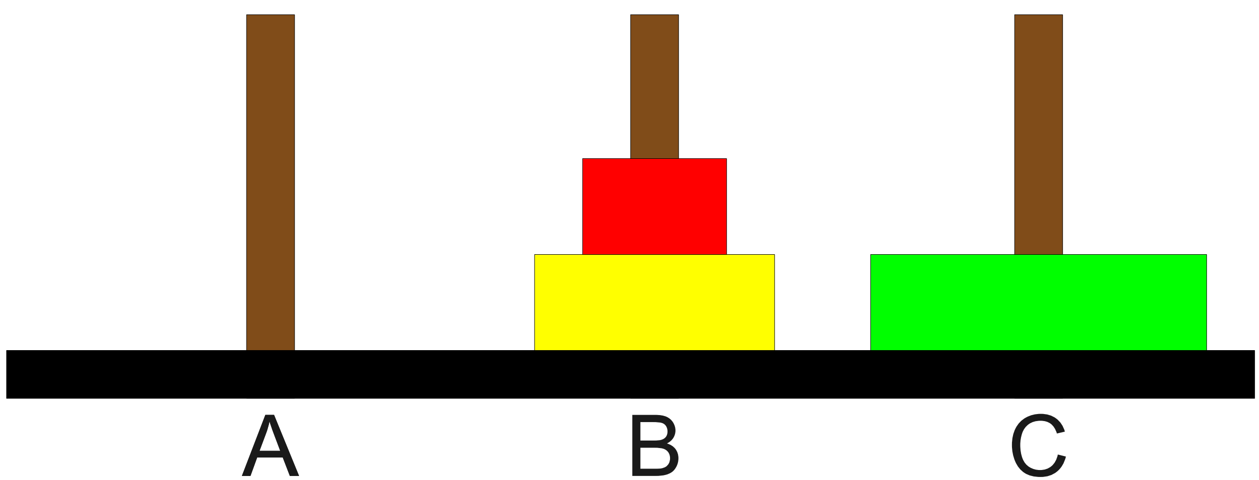


```
int main() {

}
```

```
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	from	to	temp
2	b	c	a

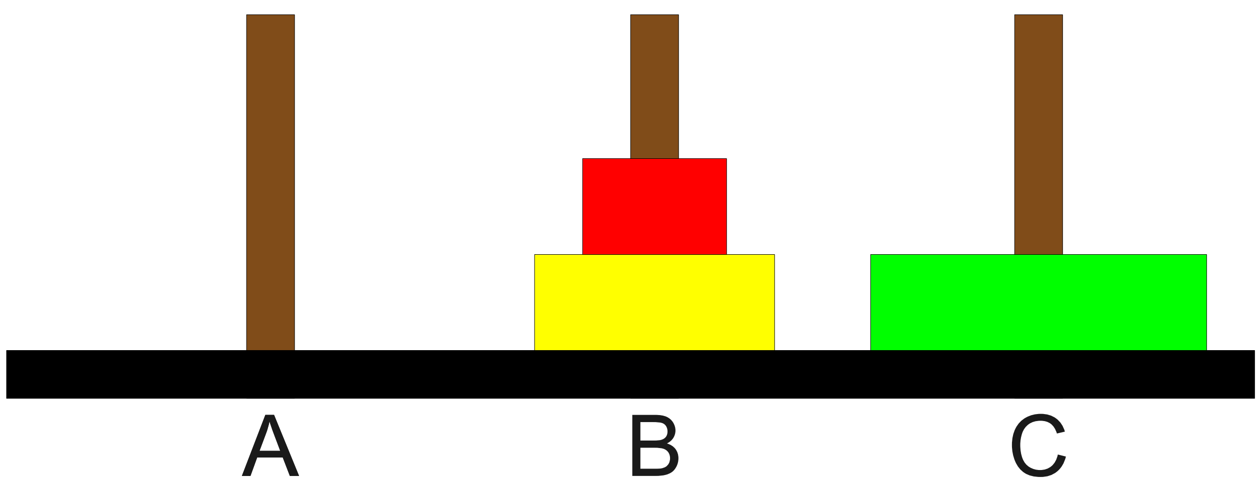


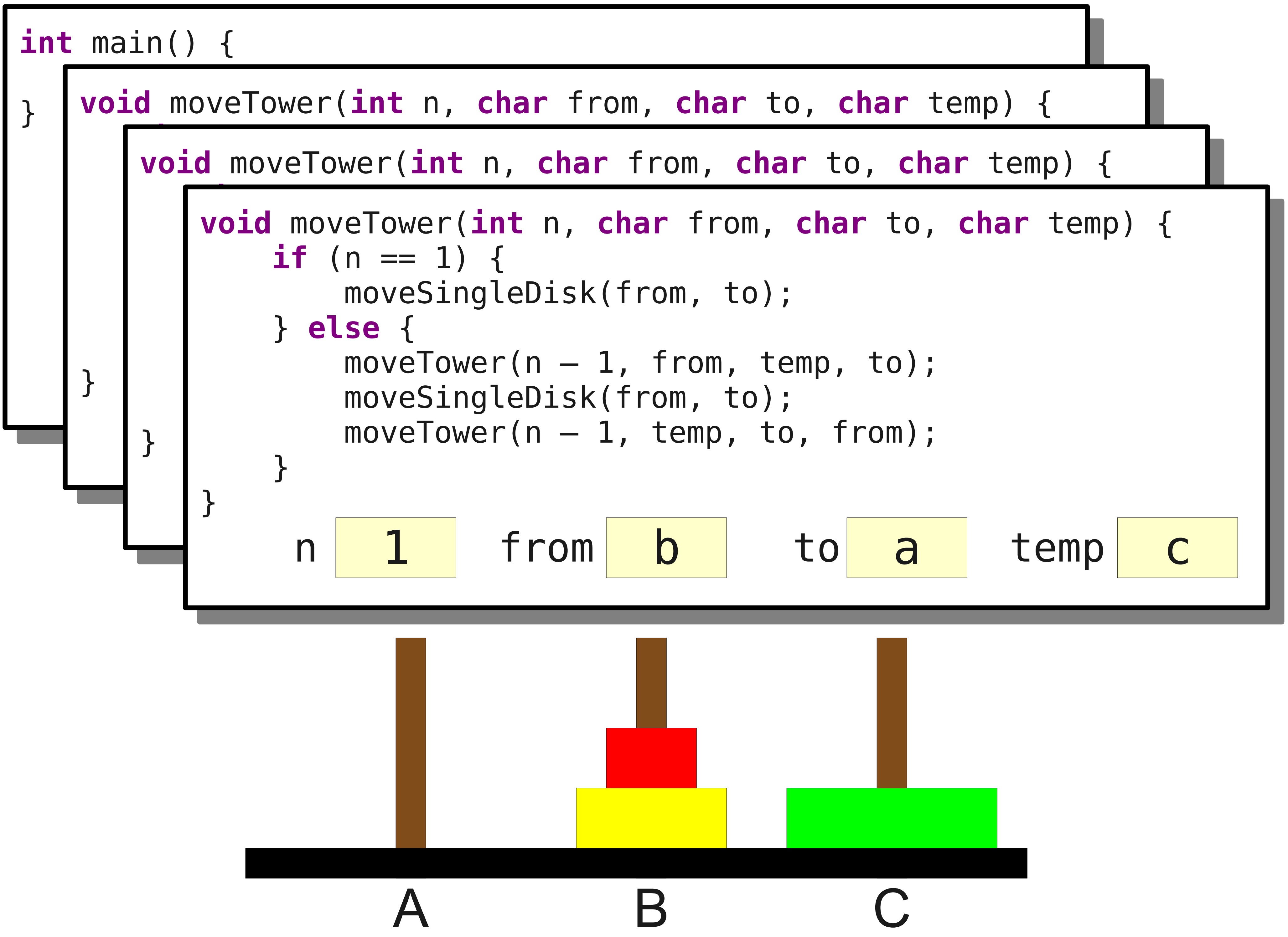
```
int main() {

}
```

```
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
A
B
C

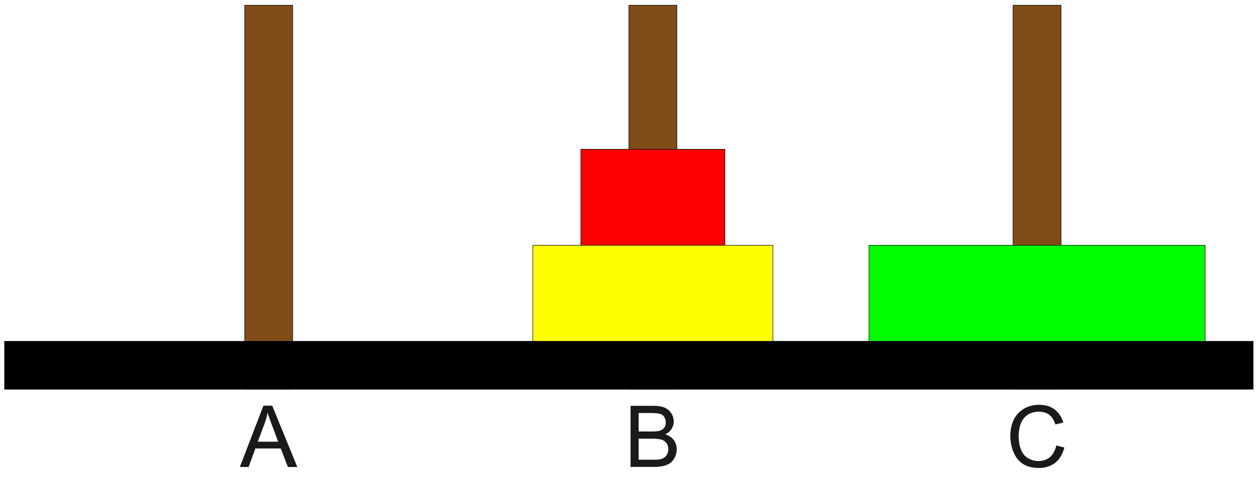
```
int main() {

}
```

```
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	from	to	temp
1	b	a	c



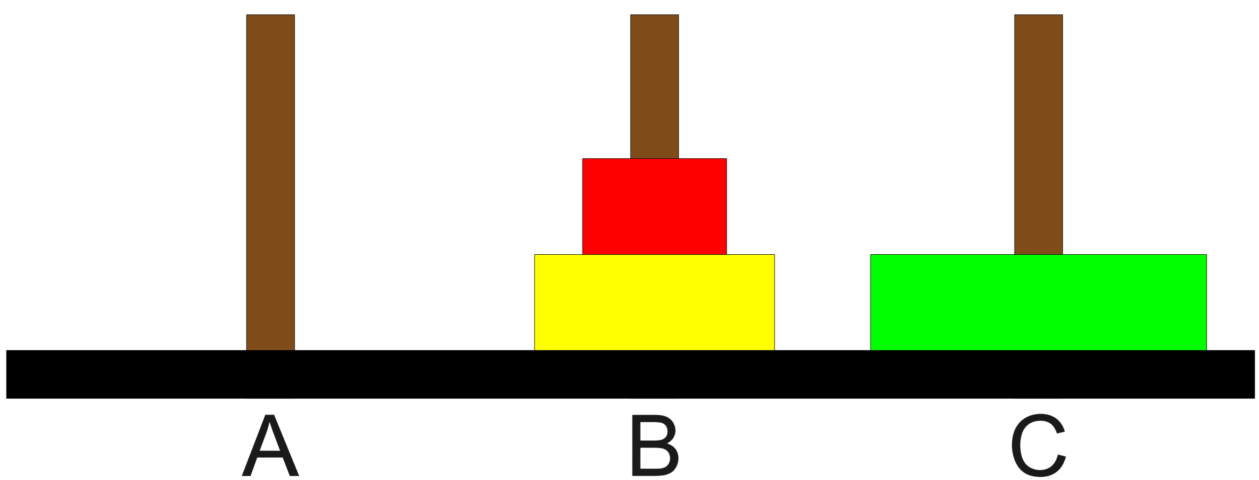
```
int main() {

}
```

```
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



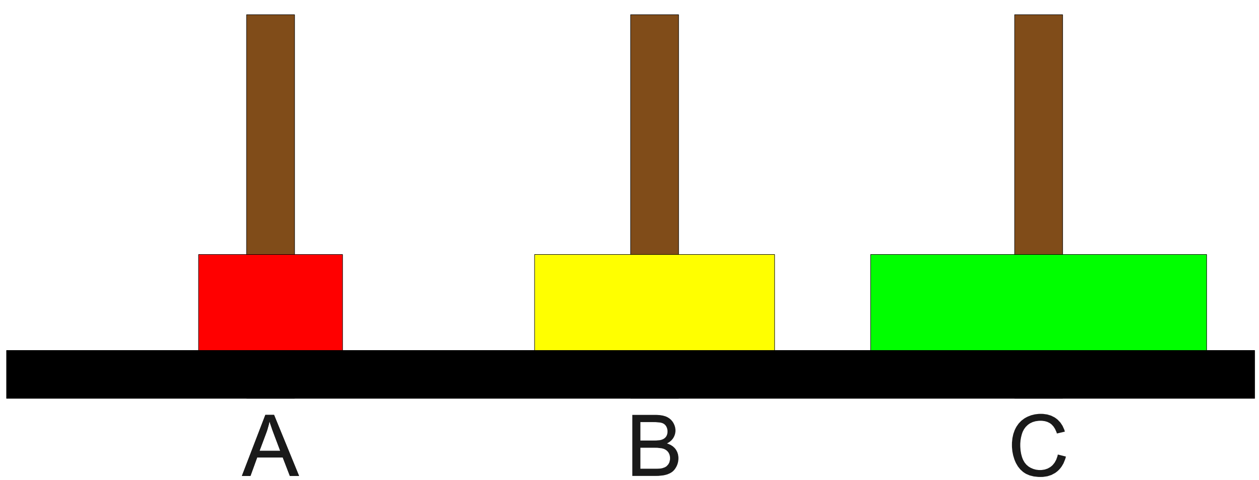
```
int main() {

}
```

```
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	from	to	temp
1	b	a	c

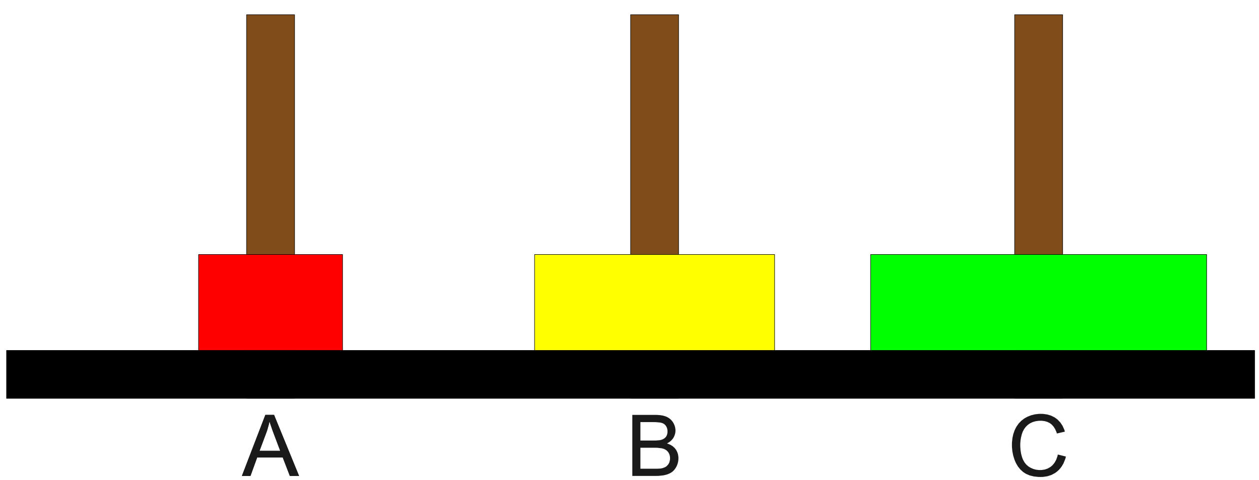


```
int main() {

}
```

```
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

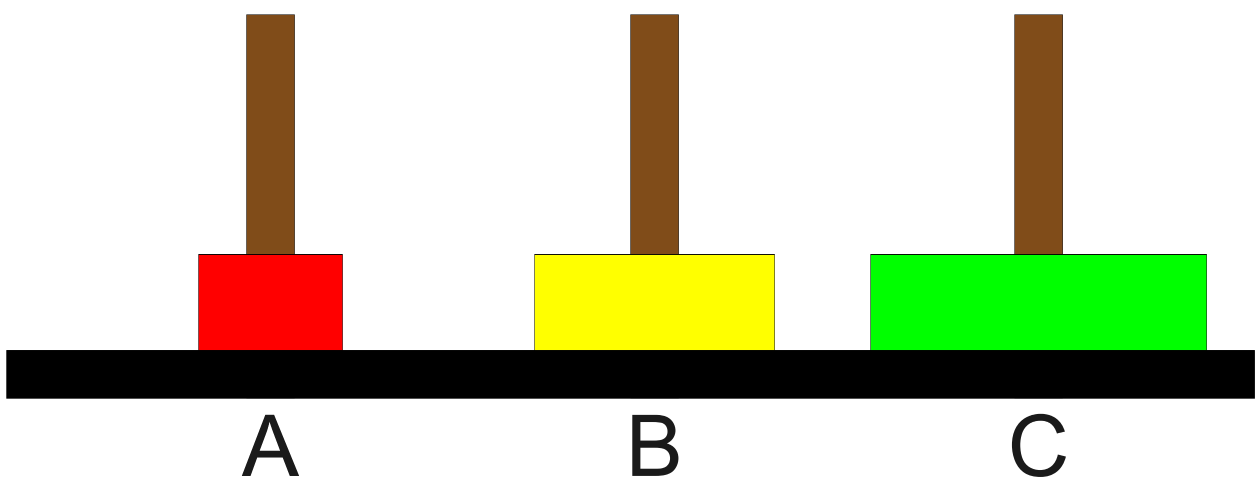


```
int main() {

}
```

```
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**

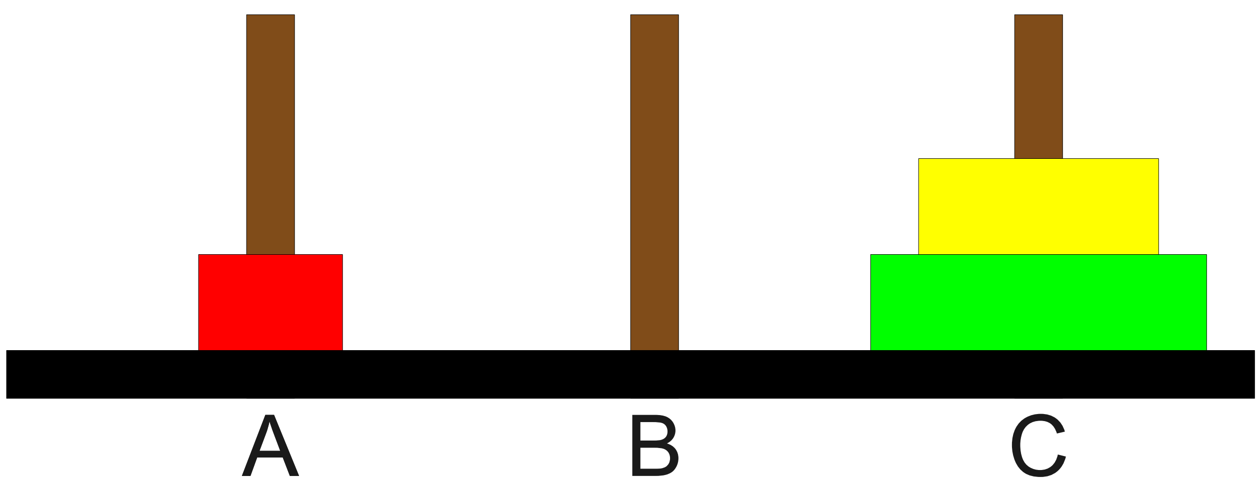


```
int main() {

}
```

```
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

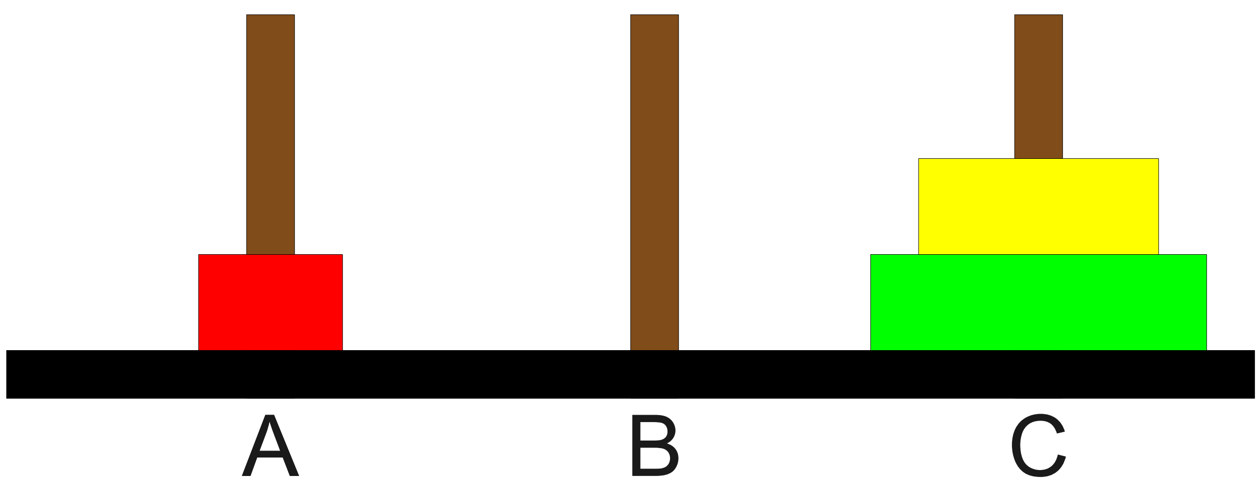


```
int main() {

}
```

```
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	from	to	temp
2	b	c	a



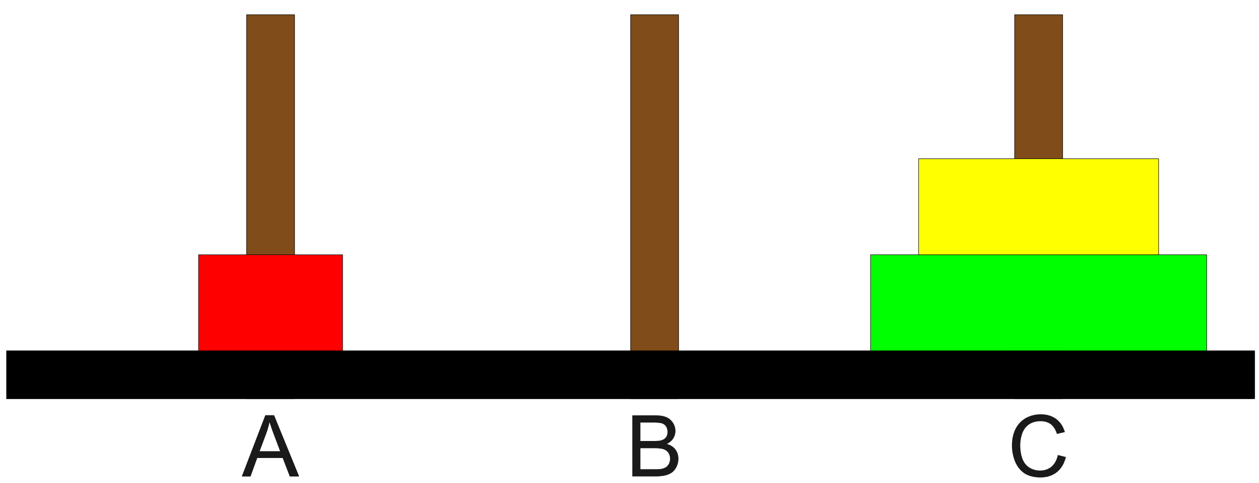
```
int main() {

}
```

```
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	from	to	temp
1	a	c	b



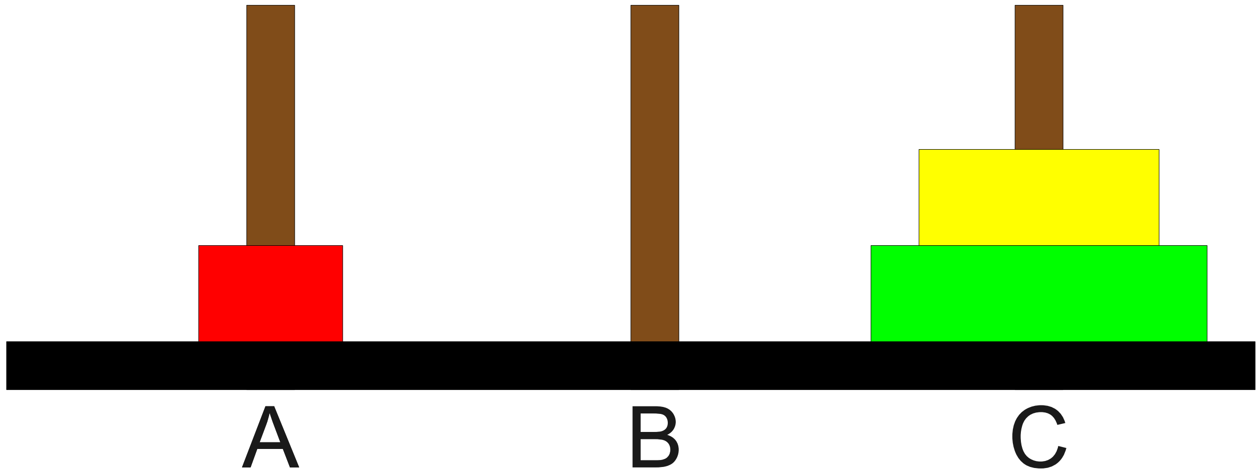
```
int main() {

}
```

```
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



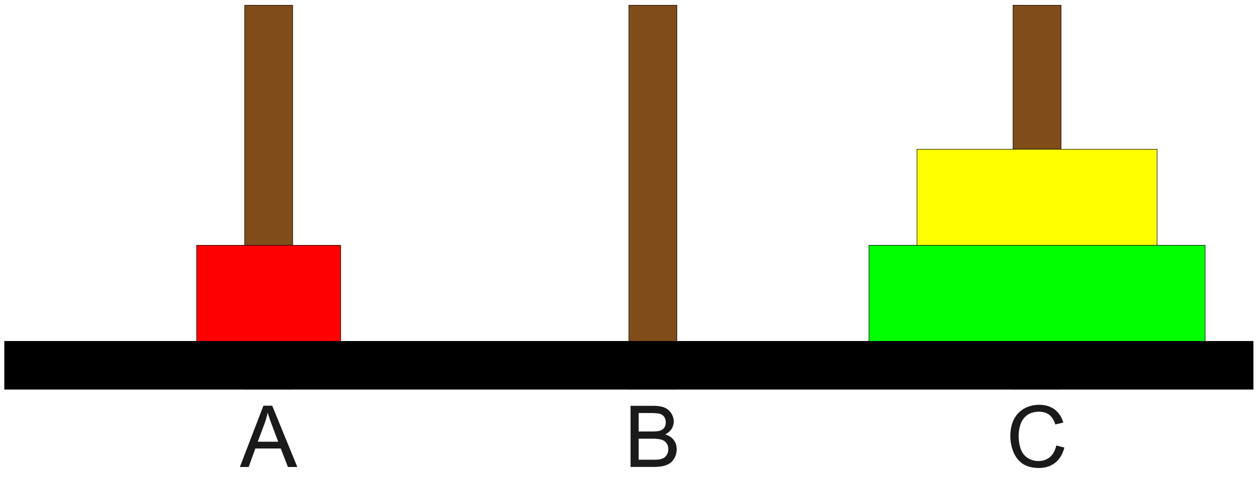
```
int main() {

}
```

```
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**



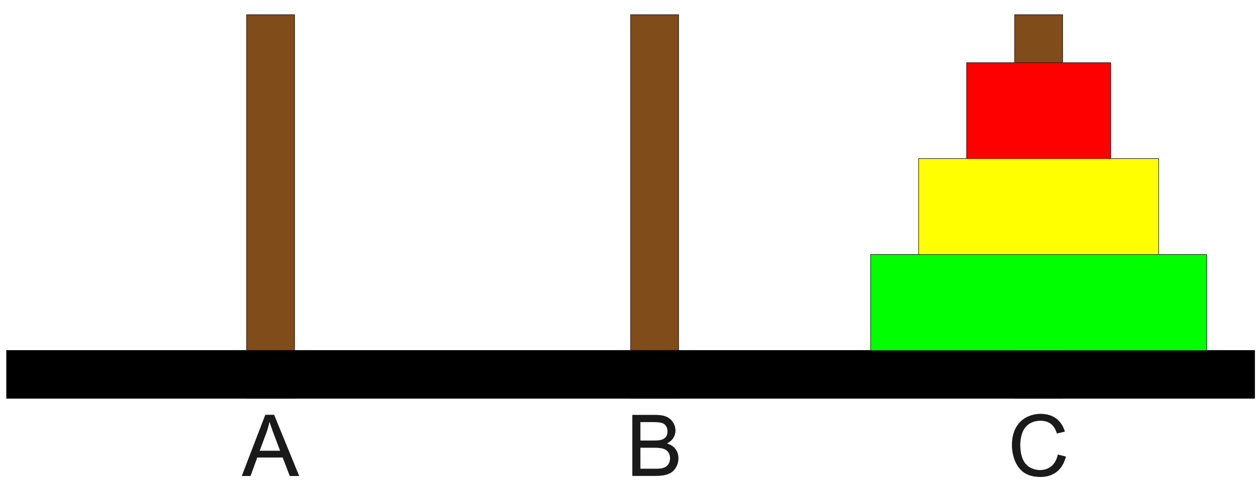
```
int main() {

}
```

```
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	from	to	temp
1	a	c	b

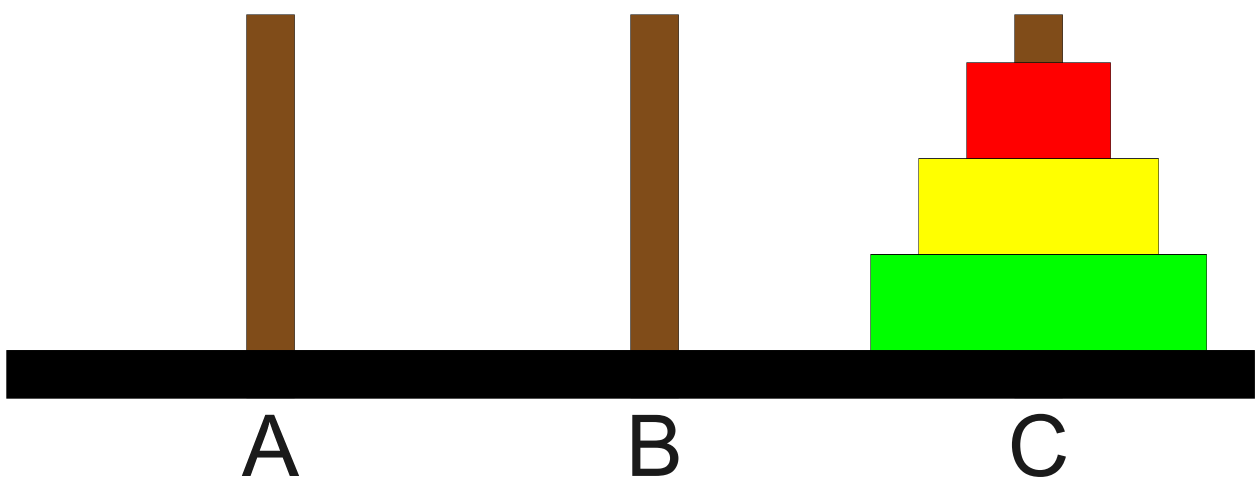


```
int main() {

}
```

```
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);
    }
}
```

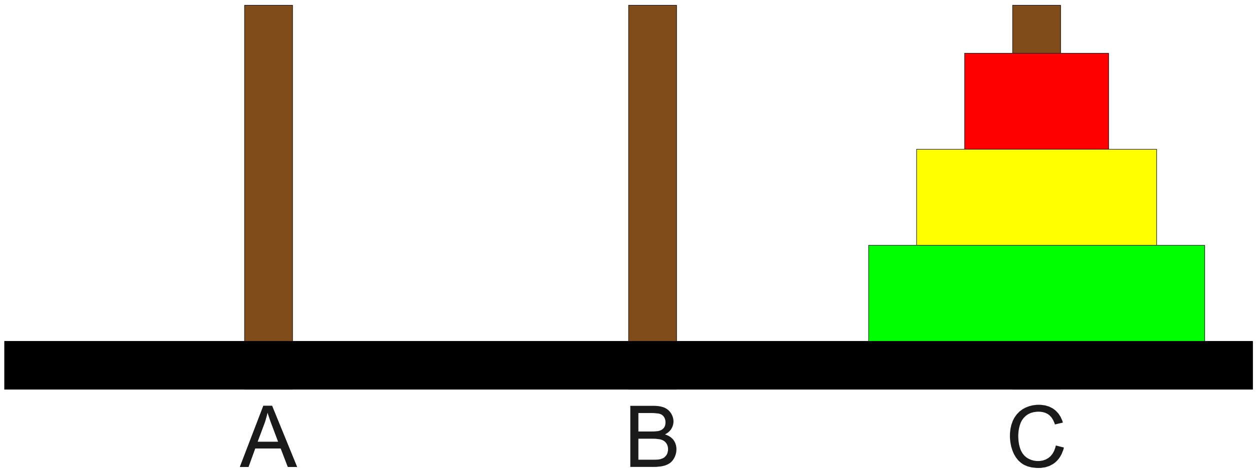
n	from	to	temp
2	b	c	a



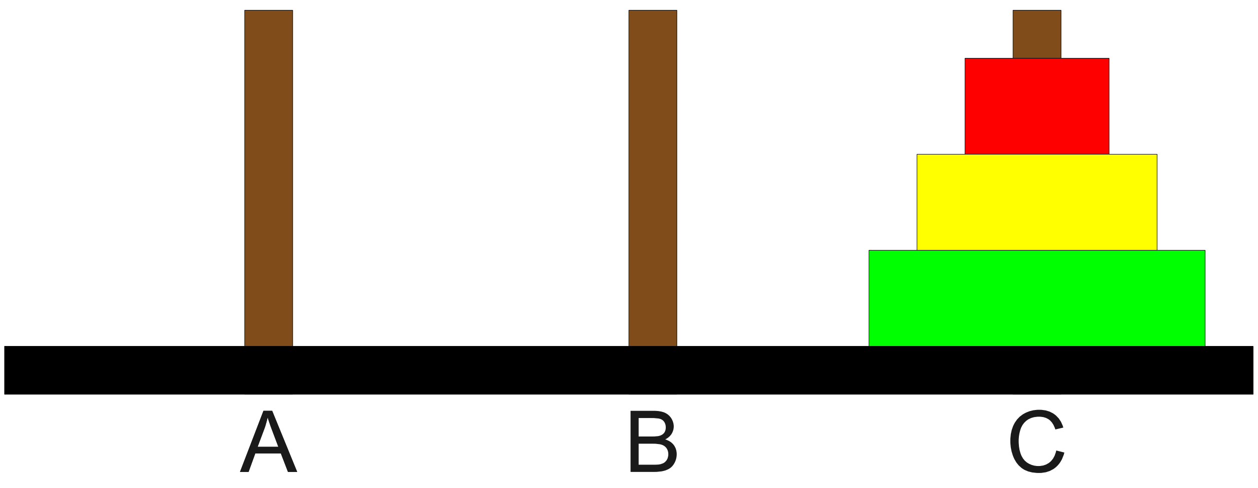
```
int main() {

}
```

```
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.

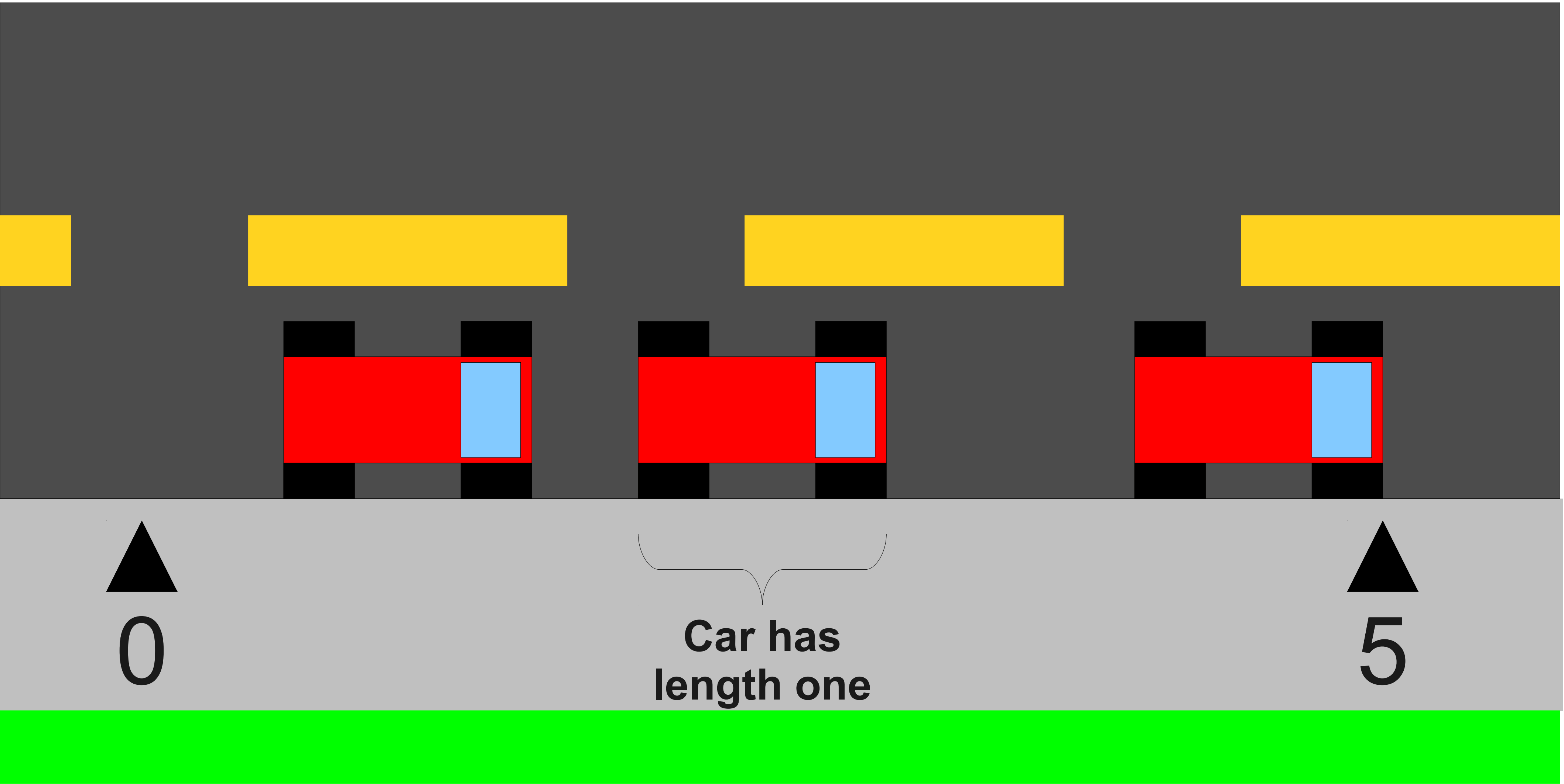
```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

```
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);
    }
}
```

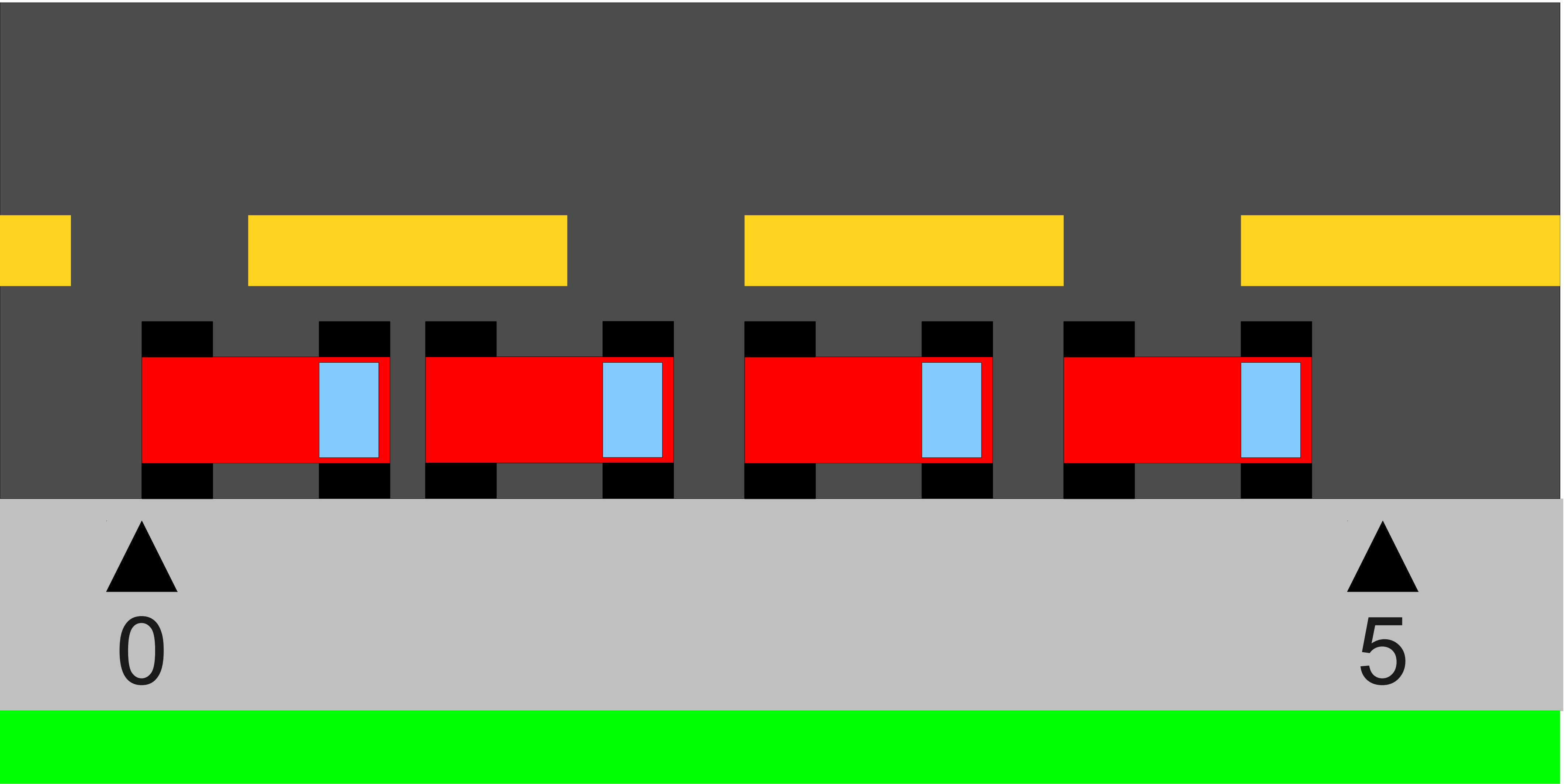
```
void moveTower(int n, char from, char to, char temp) {
    if (n == 0) {

    } 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);
    }
}
```

Parking Randomly



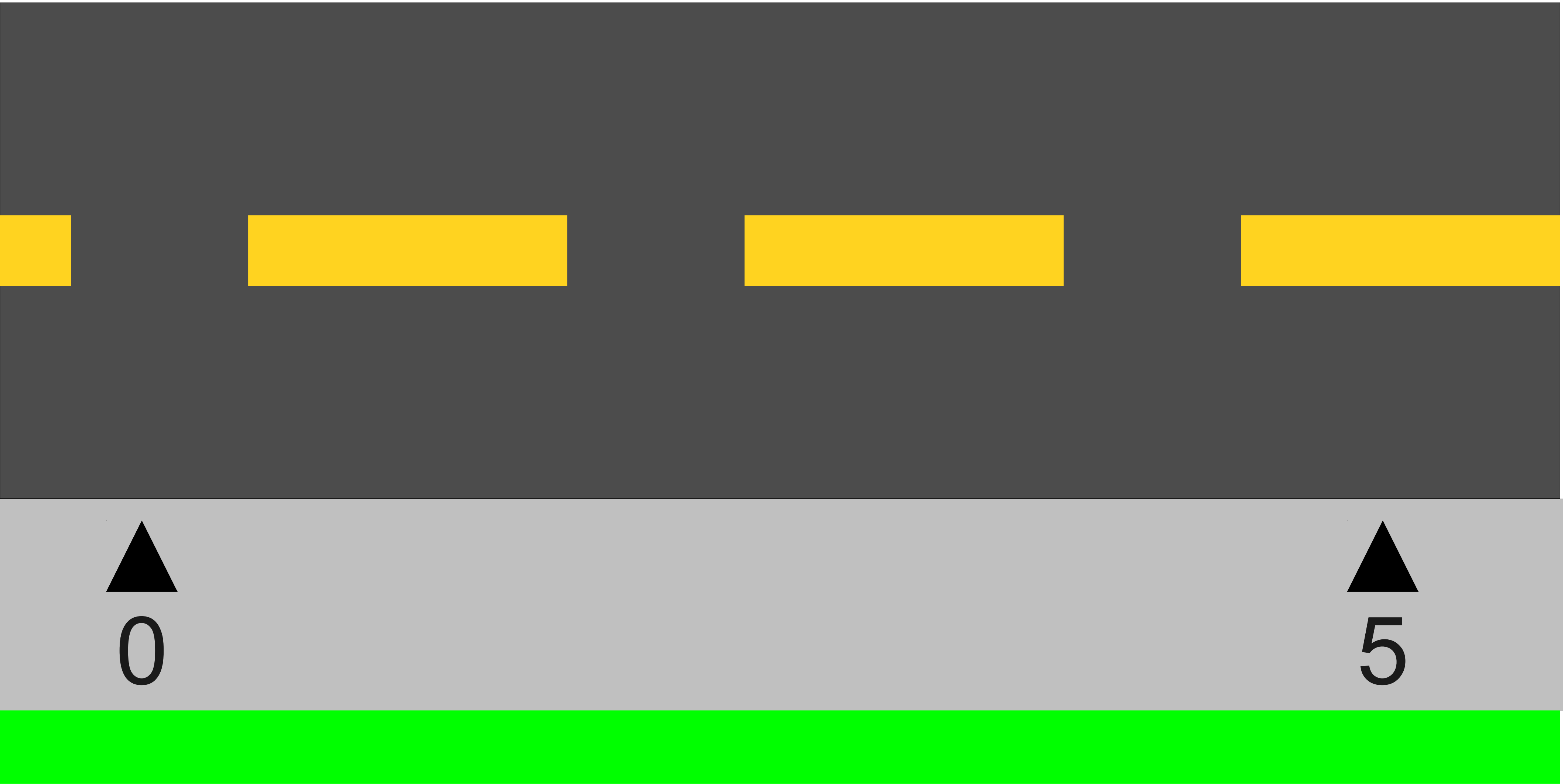
Parking Randomly



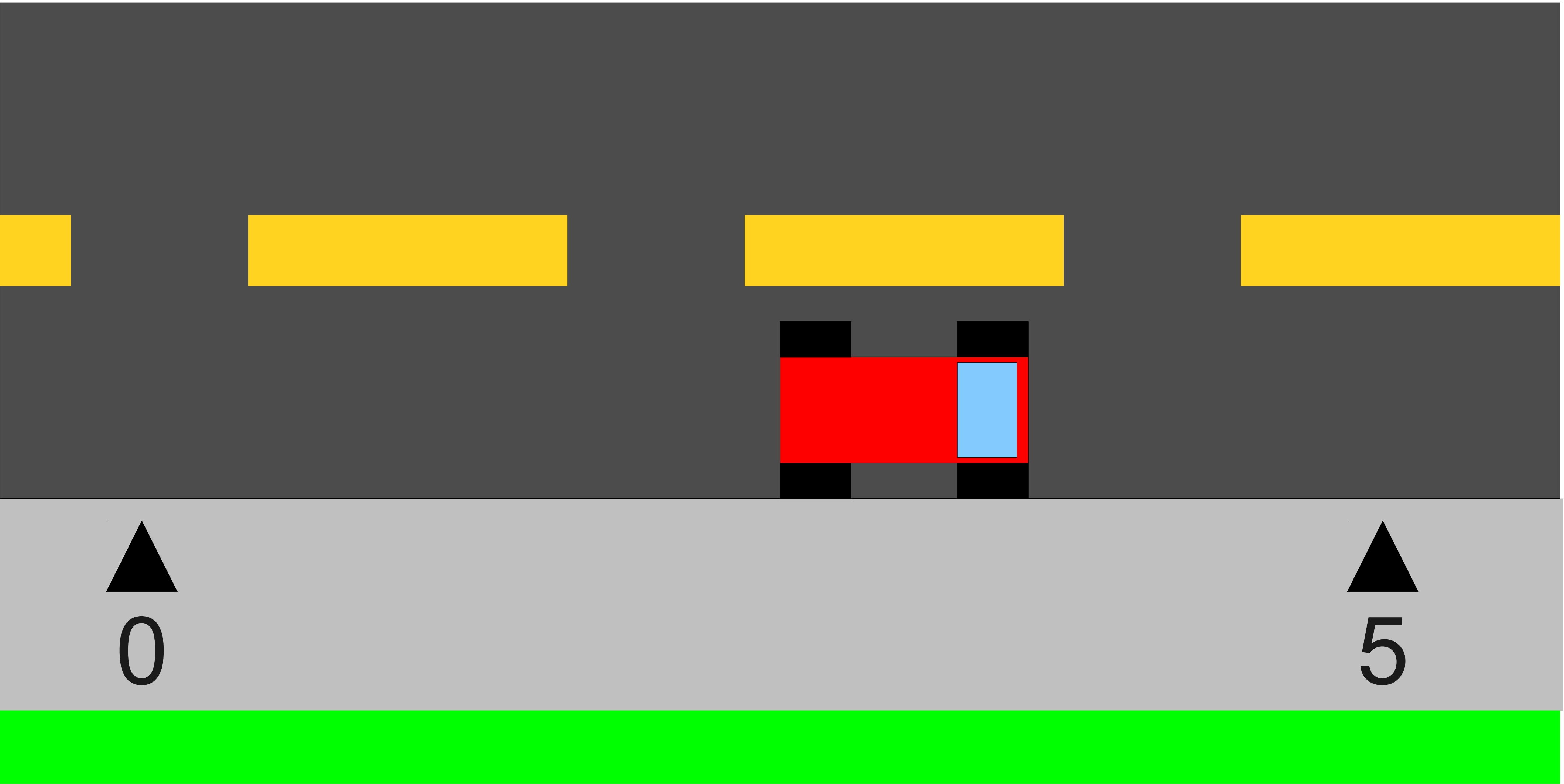
Parking Randomly

- 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?

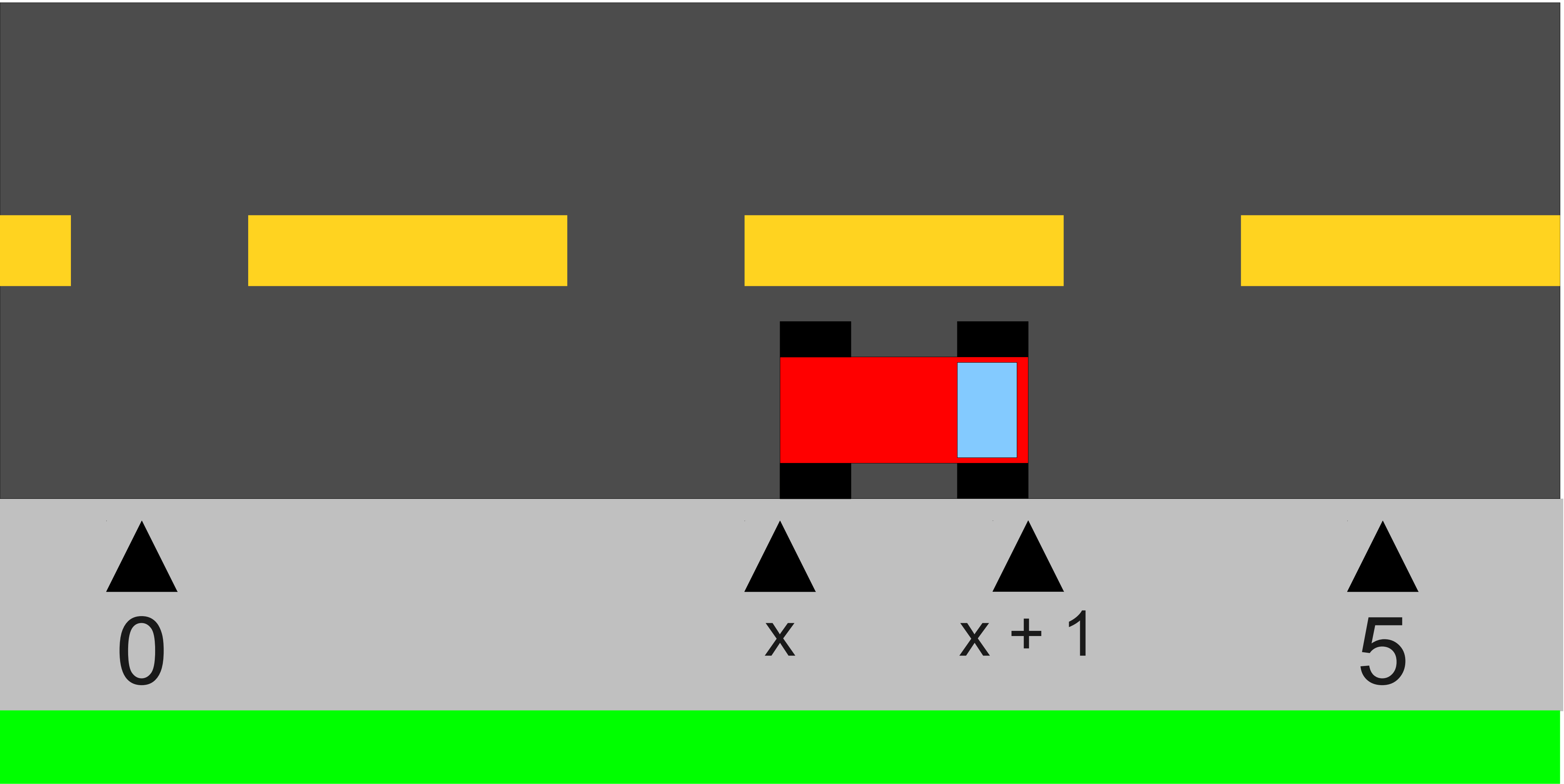
Parking Randomly



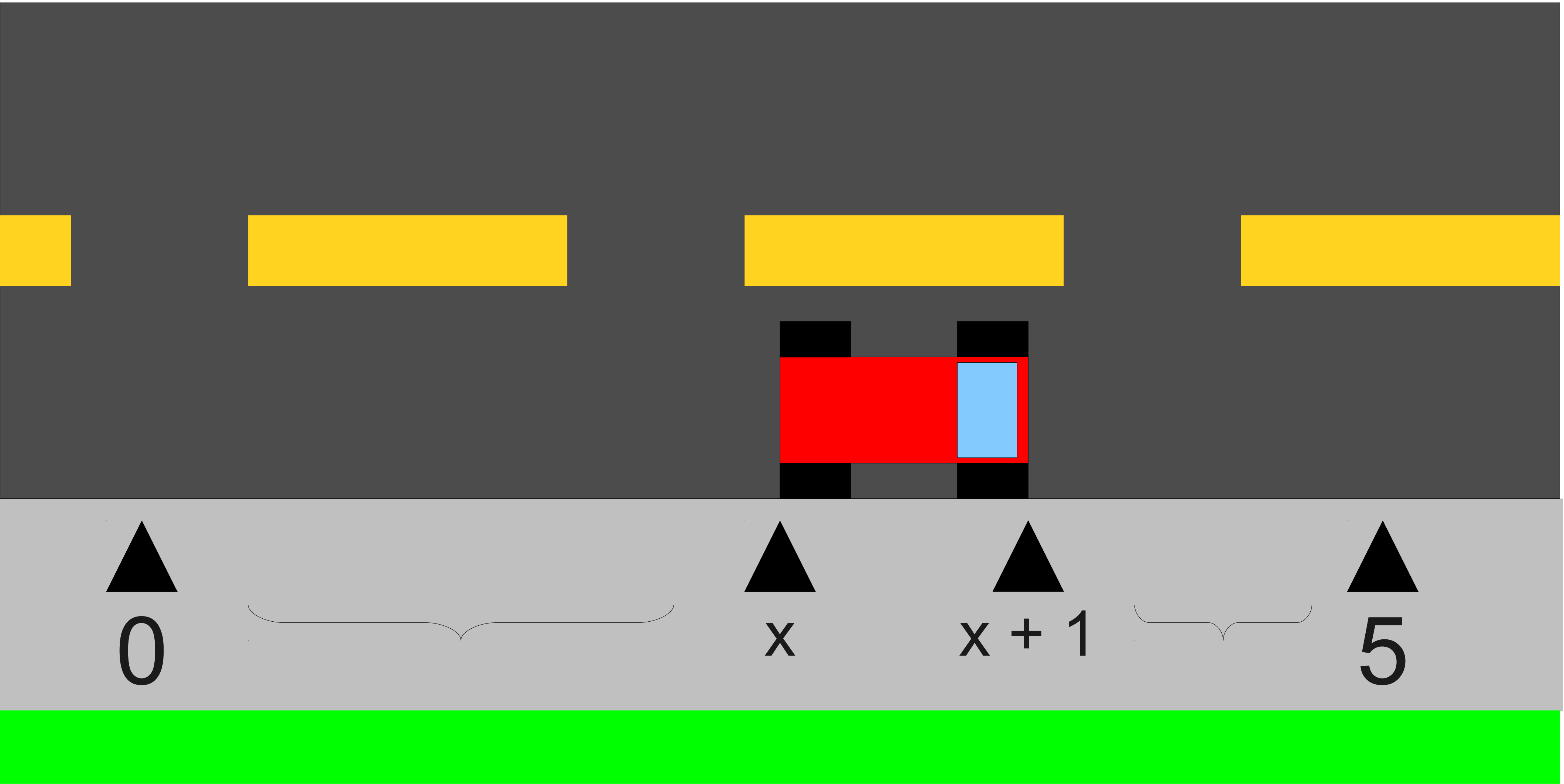
Parking Randomly



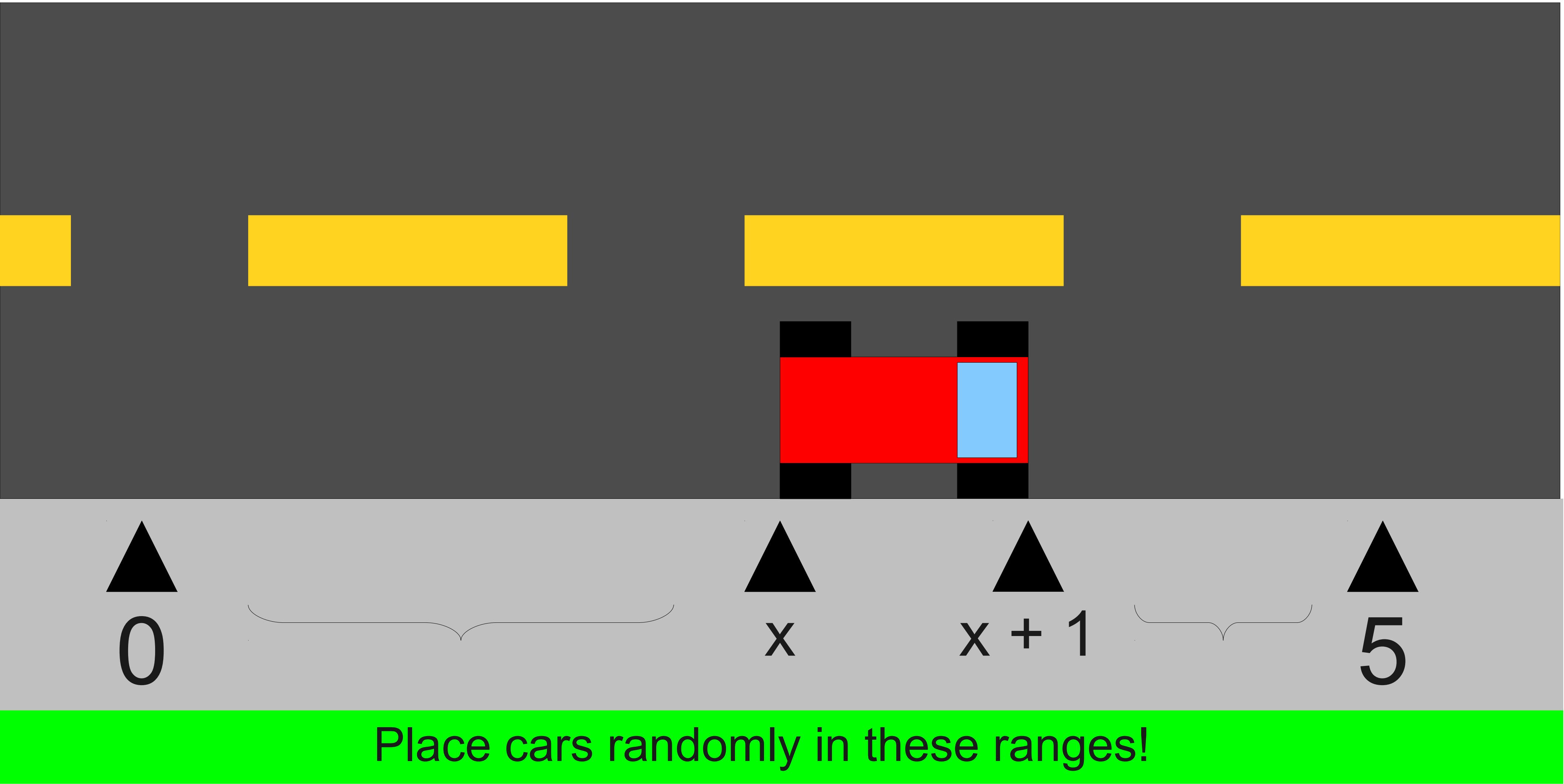
Parking Randomly



Parking Randomly



Parking Randomly



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, 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 the range into smaller, independent pieces and solve each piece separately.

- Many problems can be solved this way.