The Pigeonhole Principle

The **pigeonhole principle** is the following:

If *m* objects are placed into *n* bins, where $m > n$, then some bin contains at least two objects.

(We proved this in Lecture #02)

Why This Matters

- The pigeonhole principle can be used to show a surprising number of results must be true because they are “too big to fail.”
- Given a large enough number of objects with a bounded number of properties, eventually at least two of them will share a property.
- The applications are extremely deep and thought-provoking.

Using the Pigeonhole Principle

- To use the pigeonhole principle:
  - Find the *m* objects to distribute.
  - Find the $n < m$ buckets into which to distribute them.
  - Conclude by the pigeonhole principle that there must be two objects in some bucket.
- The details of how to proceeds from there are specific to the particular proof you're doing.

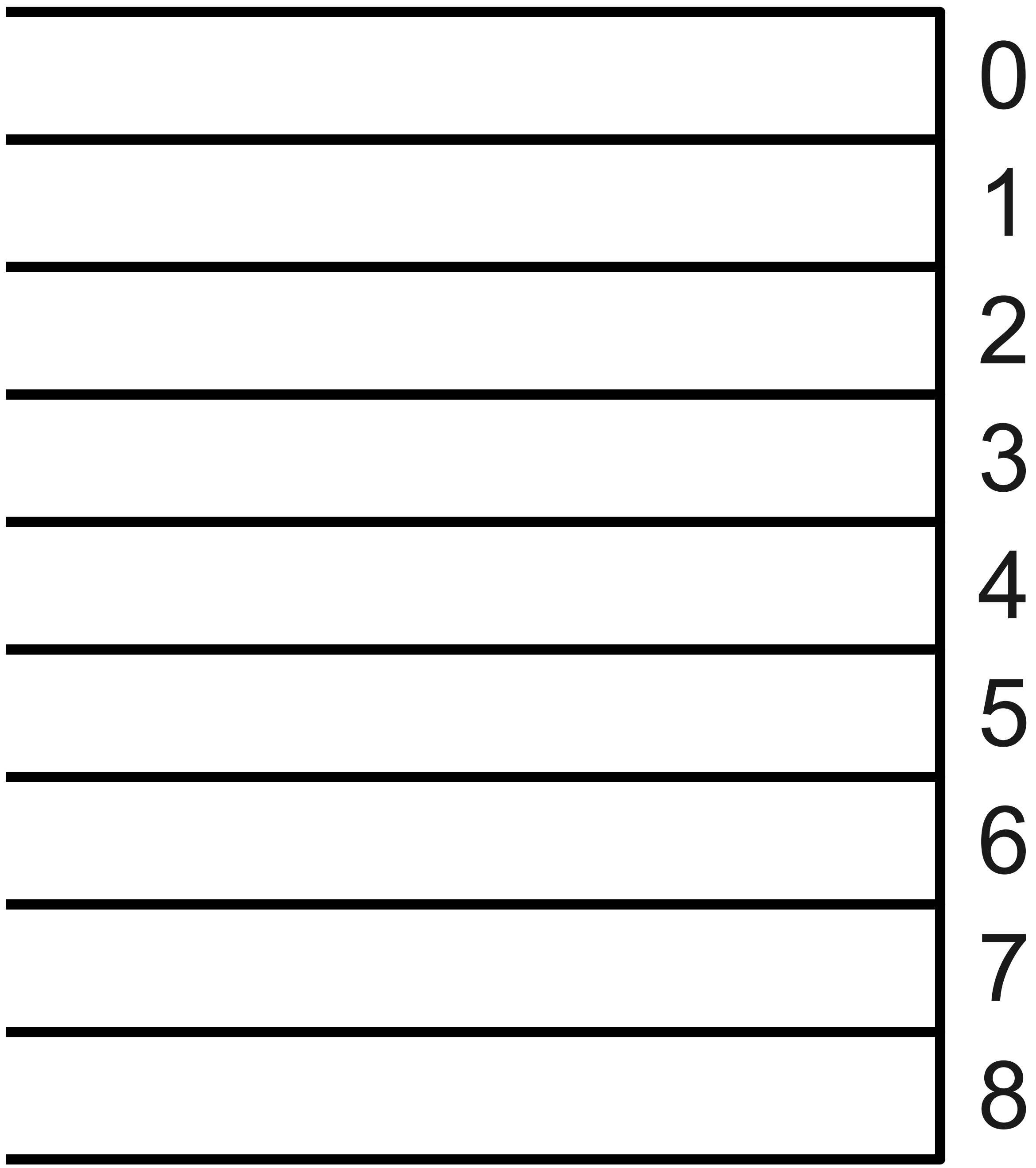
Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.

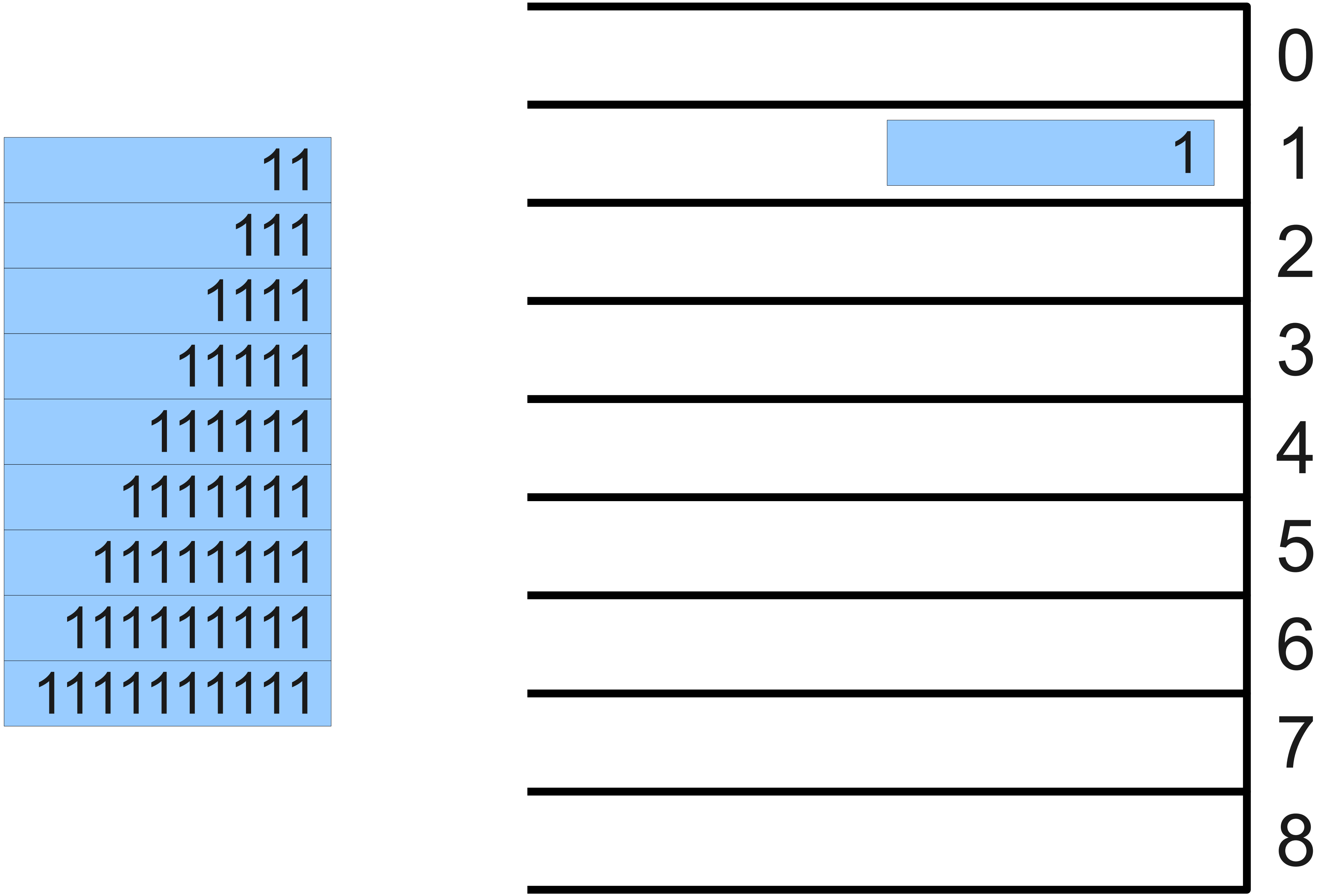
$$\left.\begin{array}{|r|}\hline 1\\\hline 11\\\hline 111\\\hline 1111\\\hline 11111\\\hline 111111\\\hline 1111111\\\hline 11111111\\\hline 111111111\\\hline 1111111111\\\hline\end{array}\right\}\text{There are 10 objects here.}$$

Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.

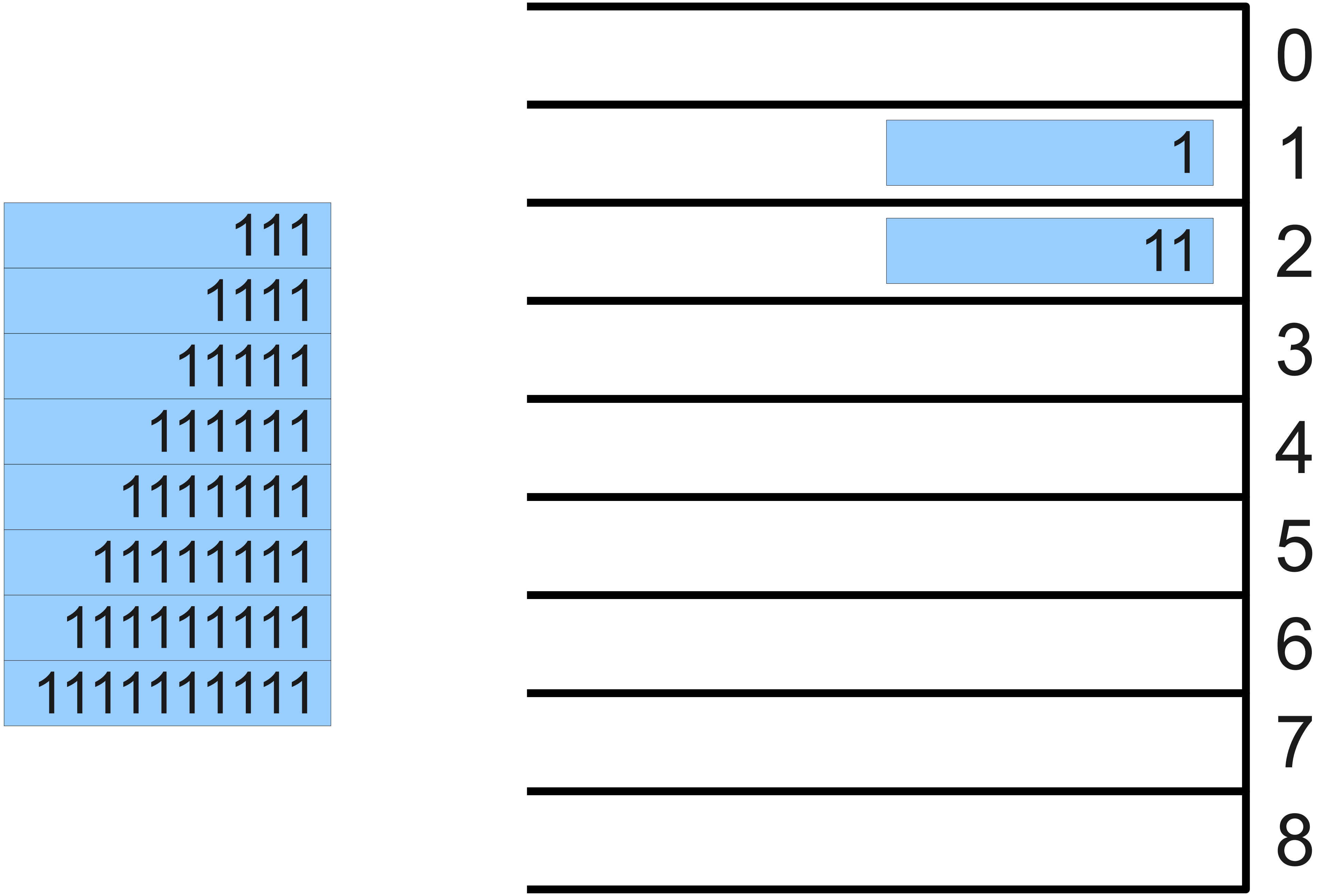
1
11
111
1111
11111
111111
1111111
11111111
111111111
1111111111



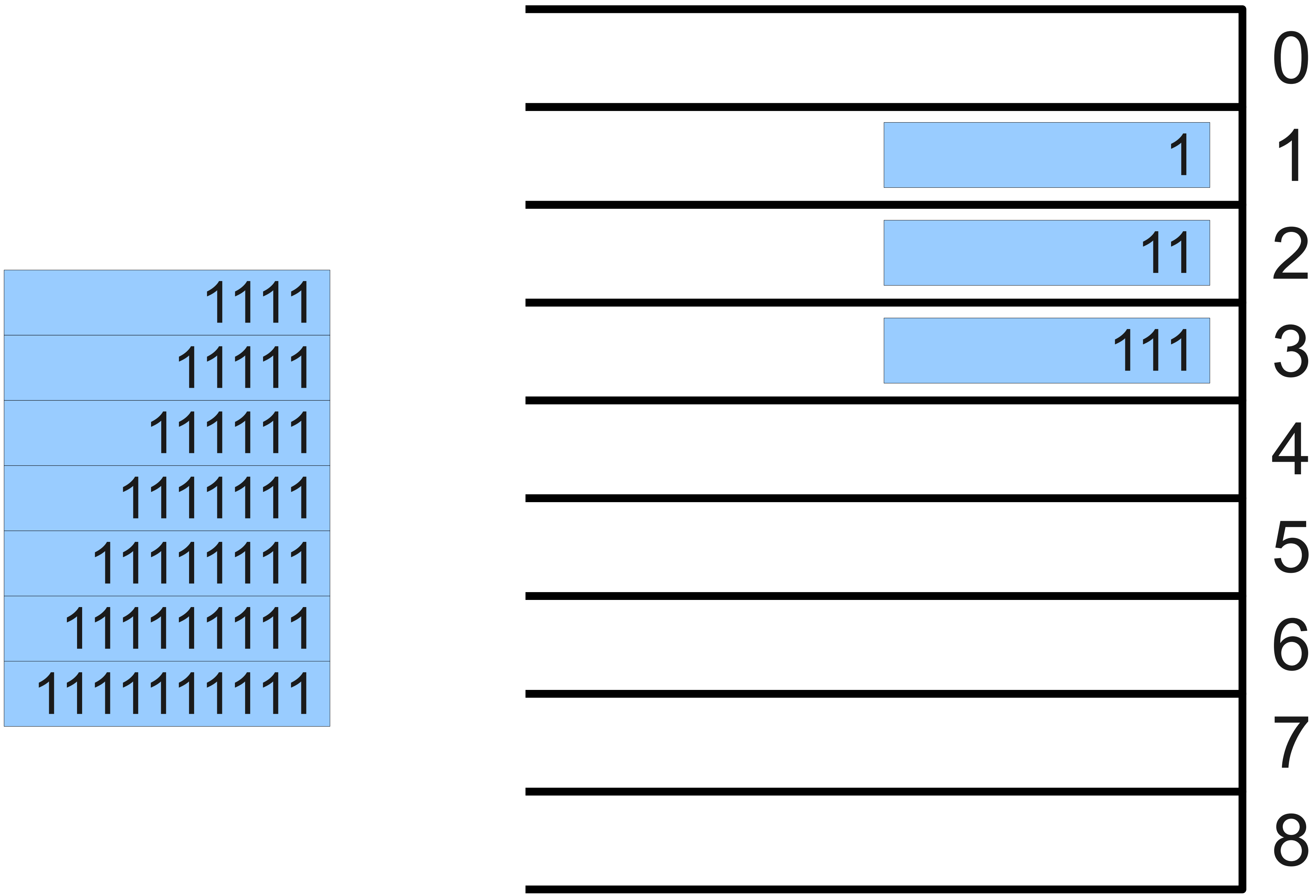
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



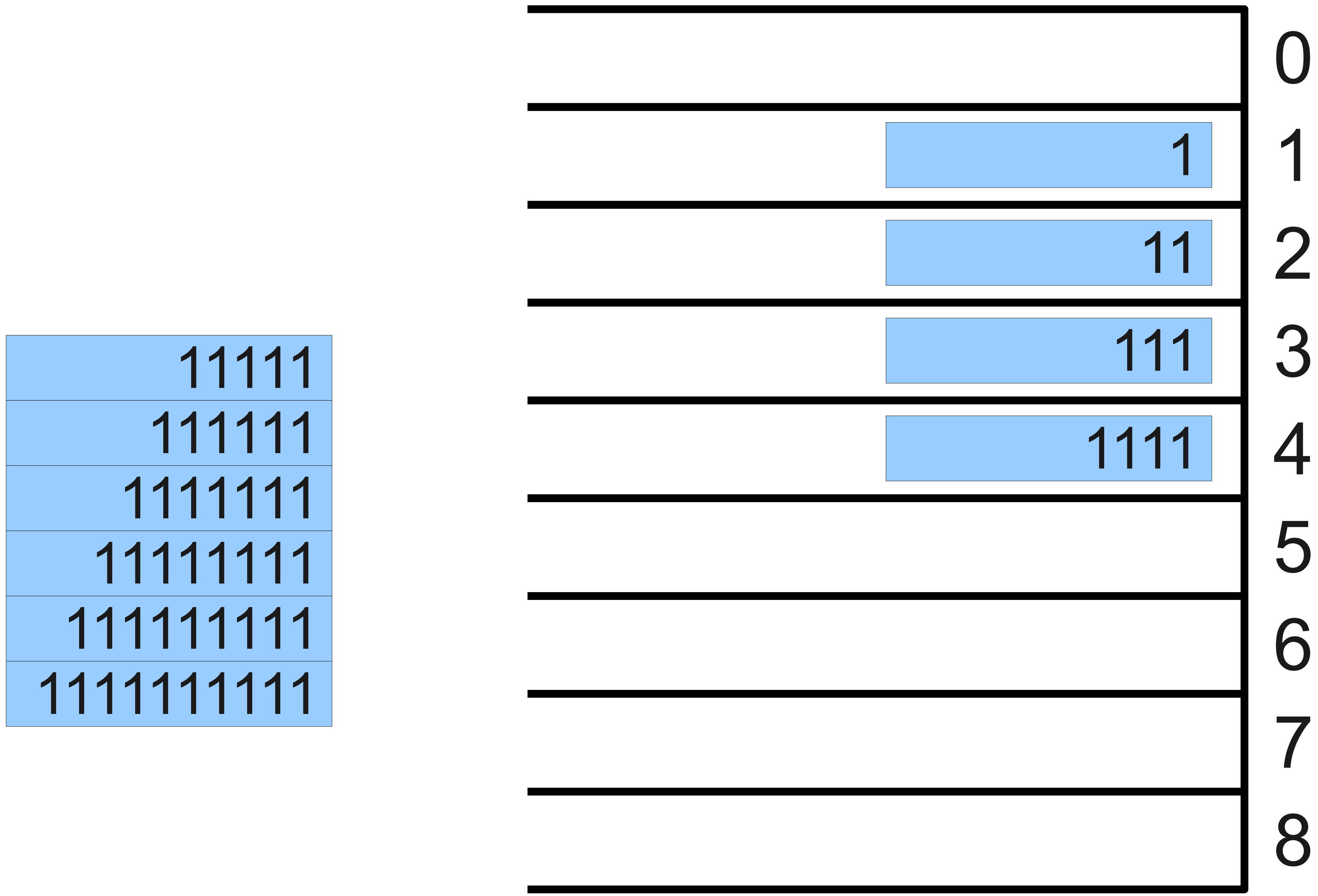
Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.



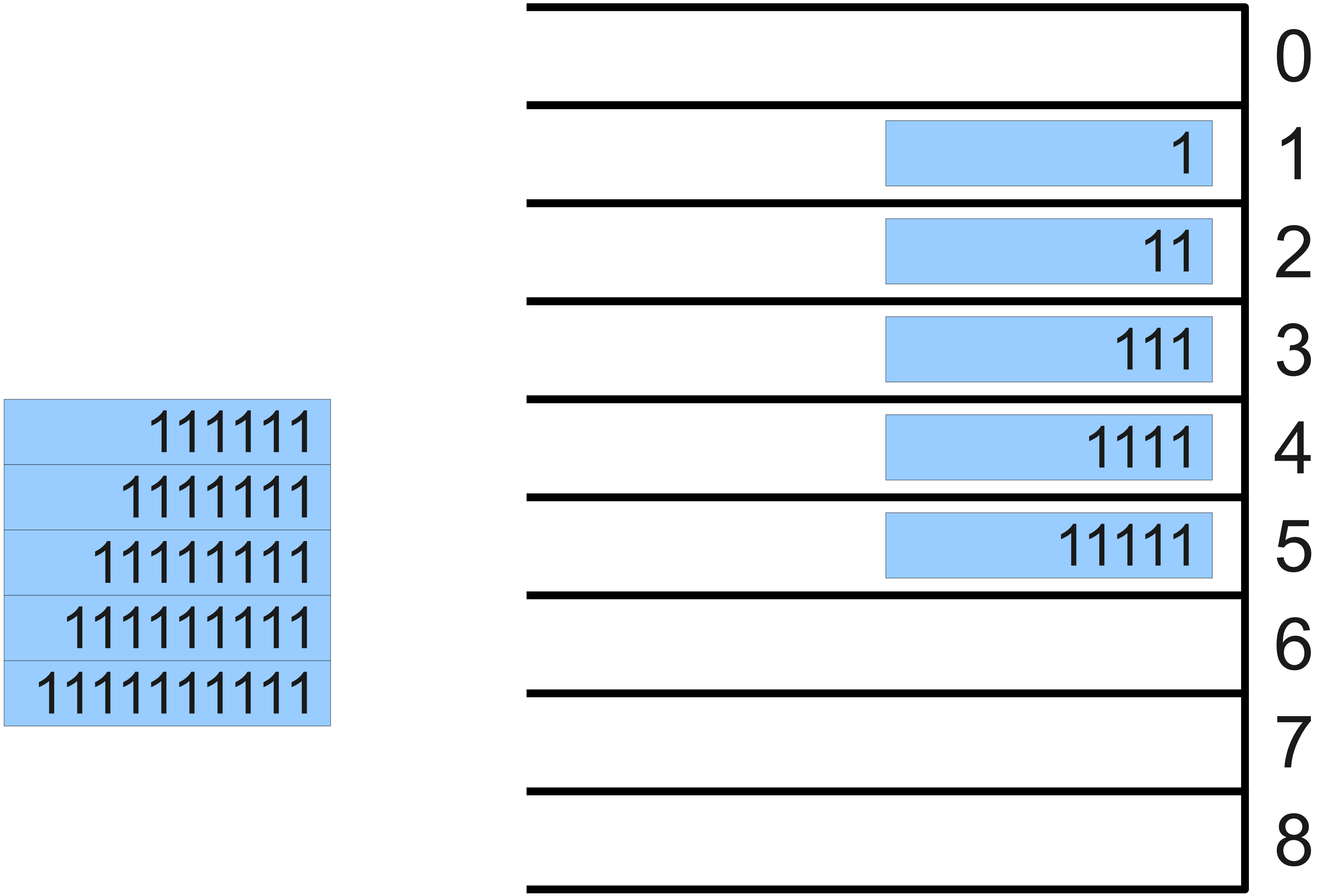
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



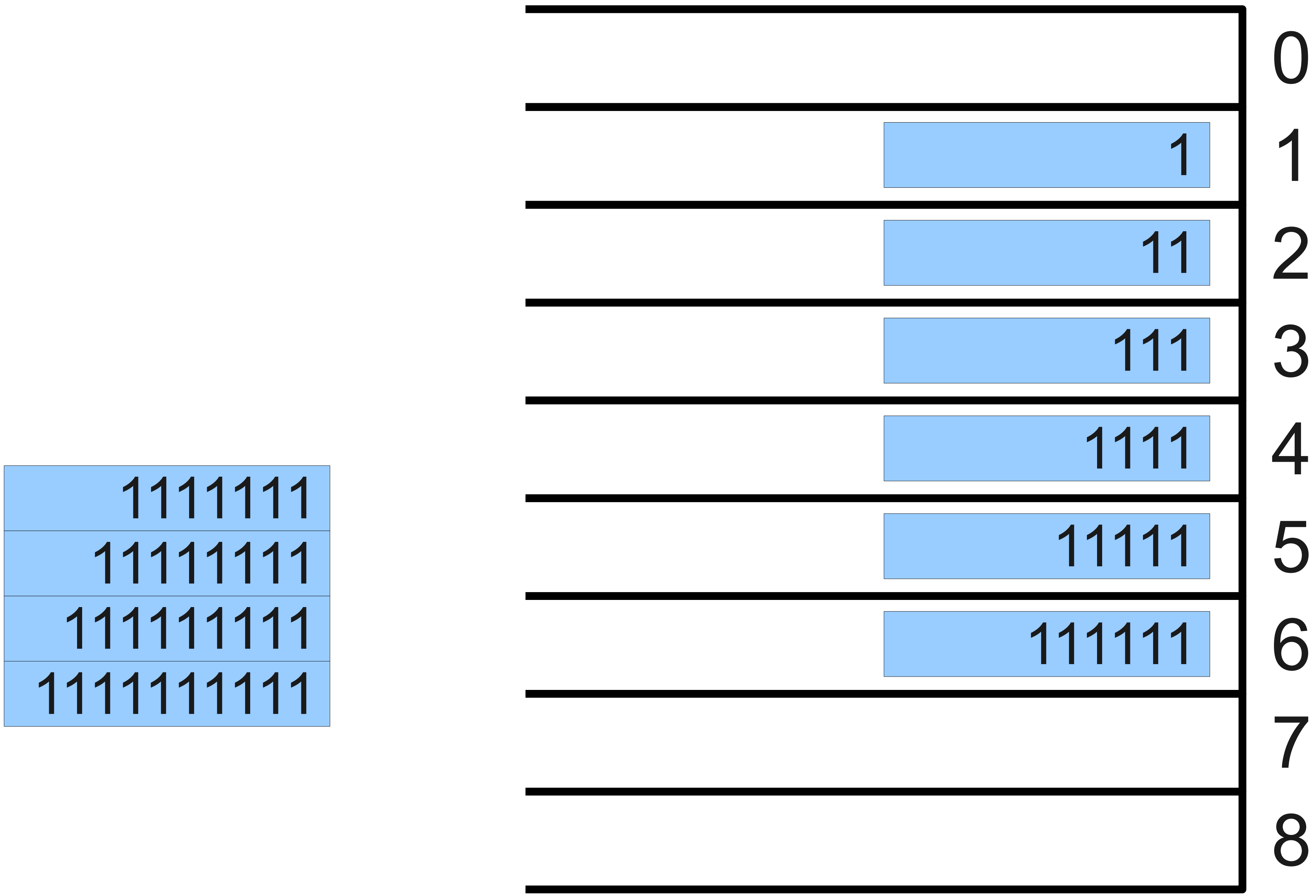
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



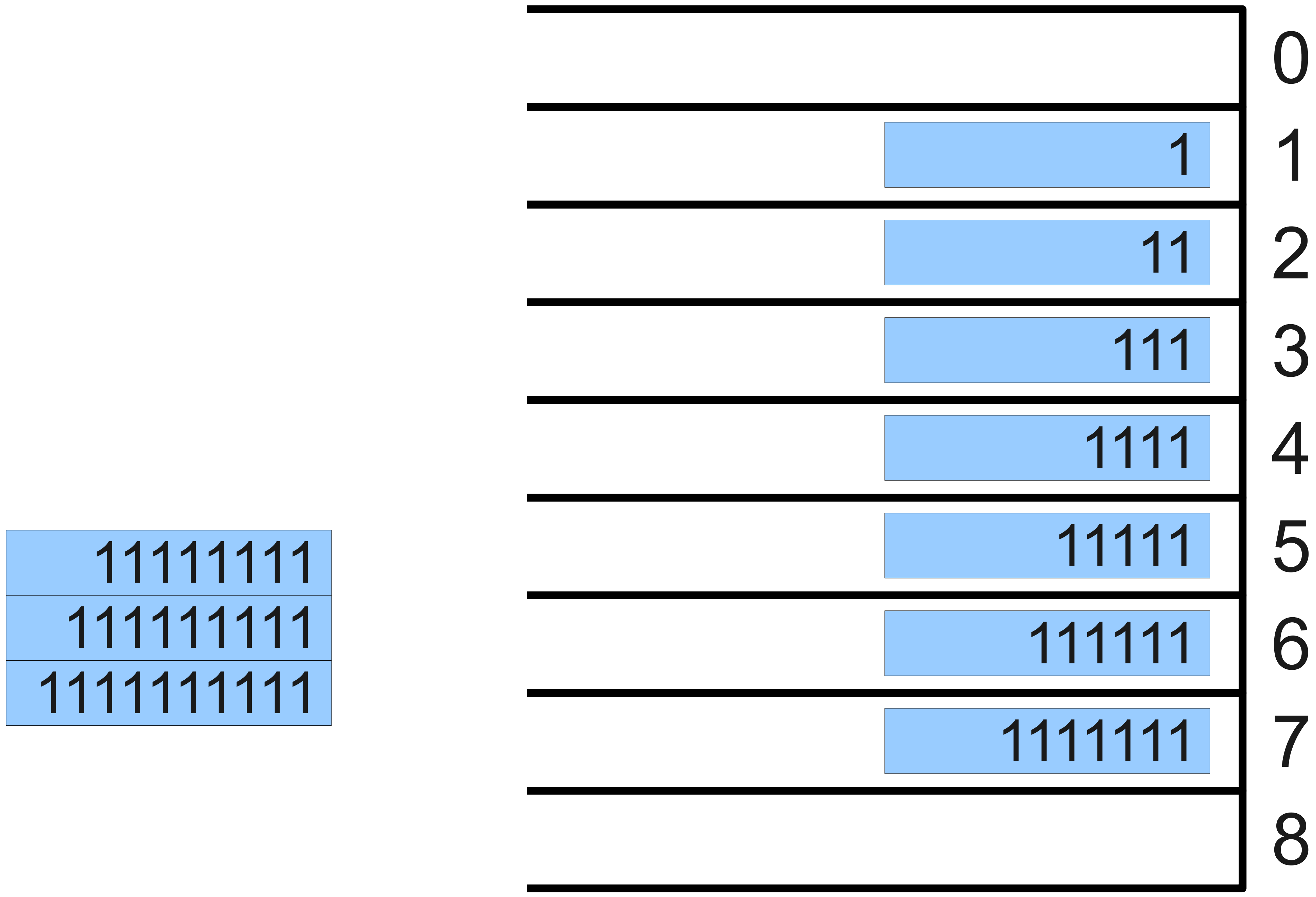
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



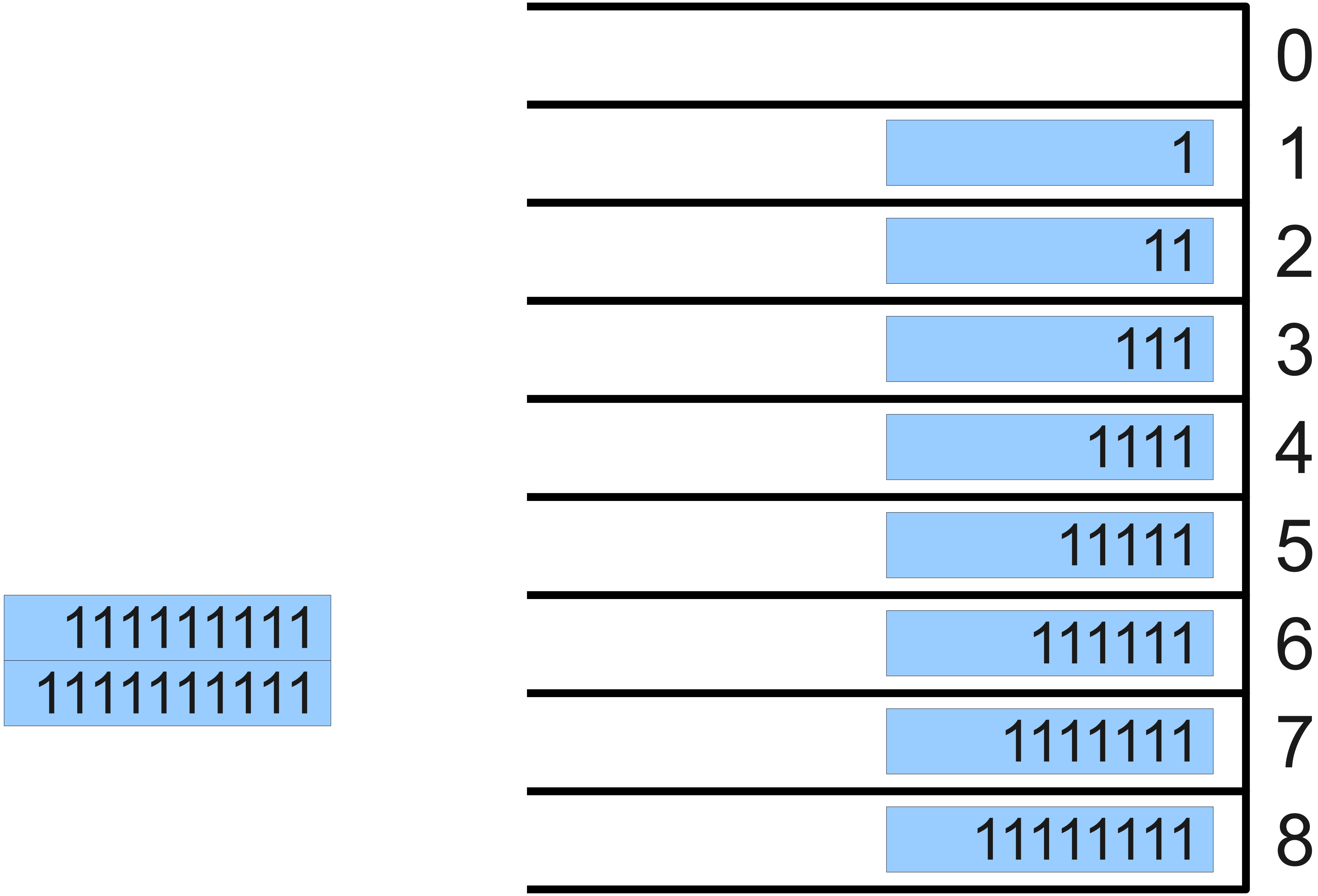
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



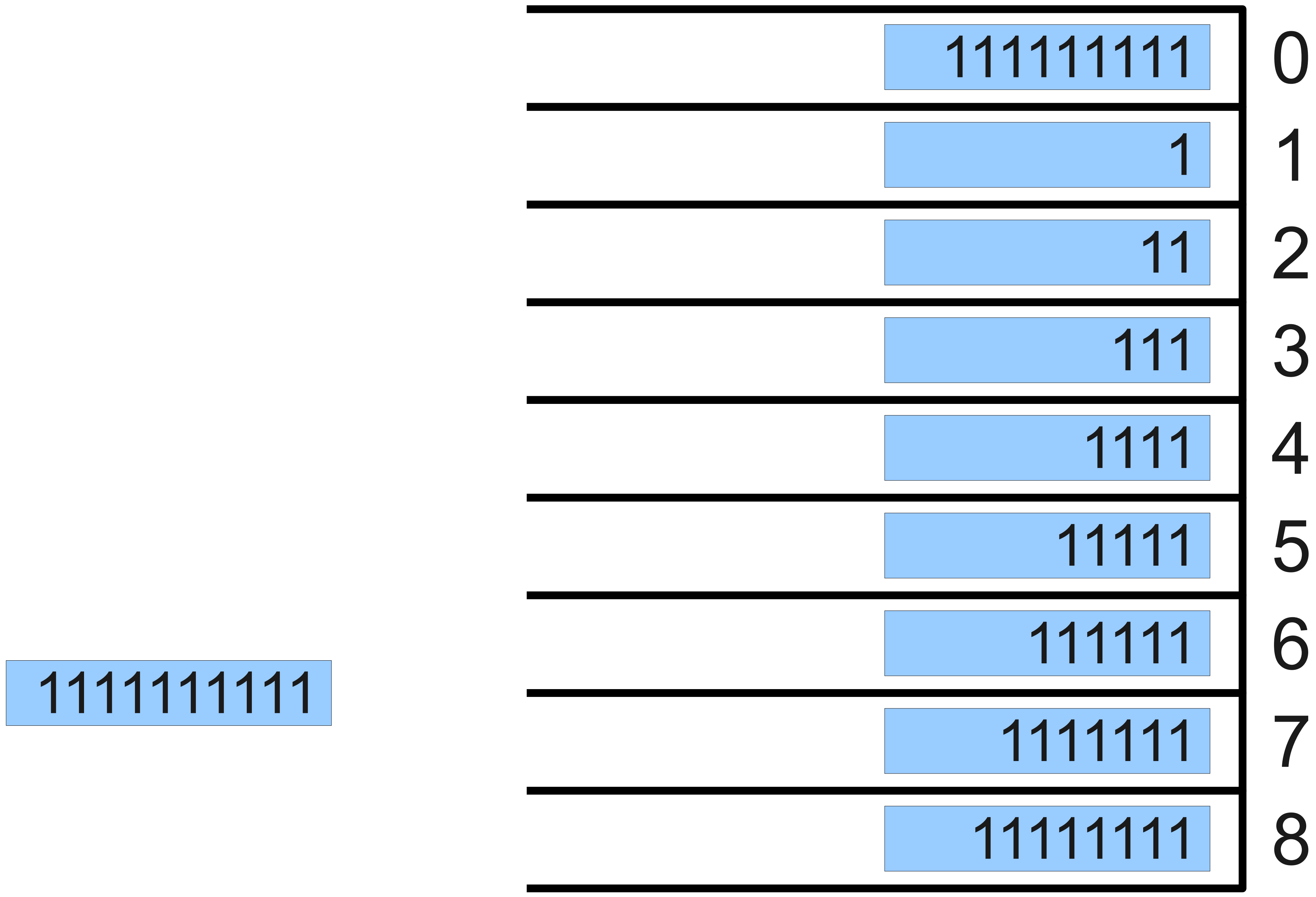
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



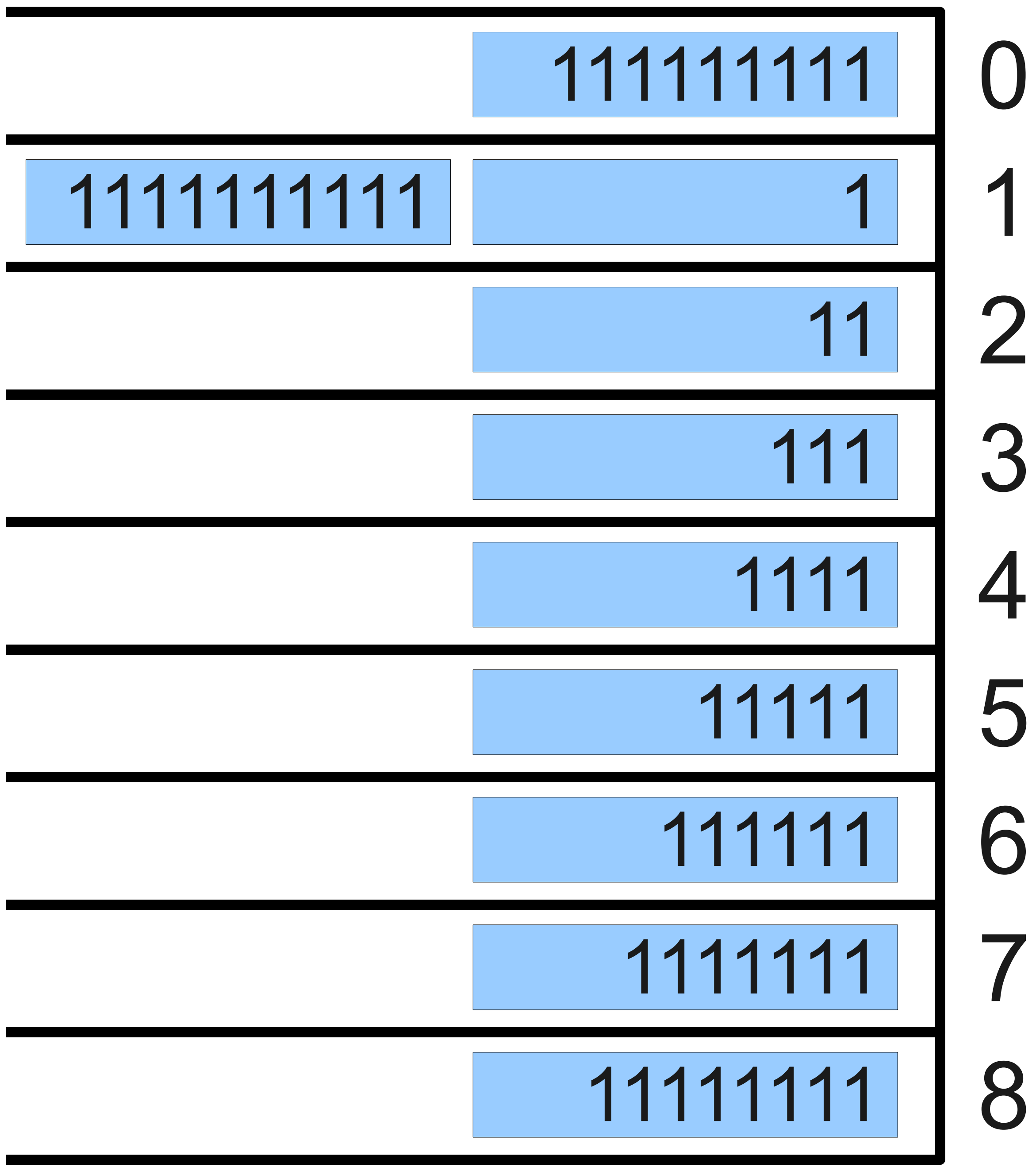
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



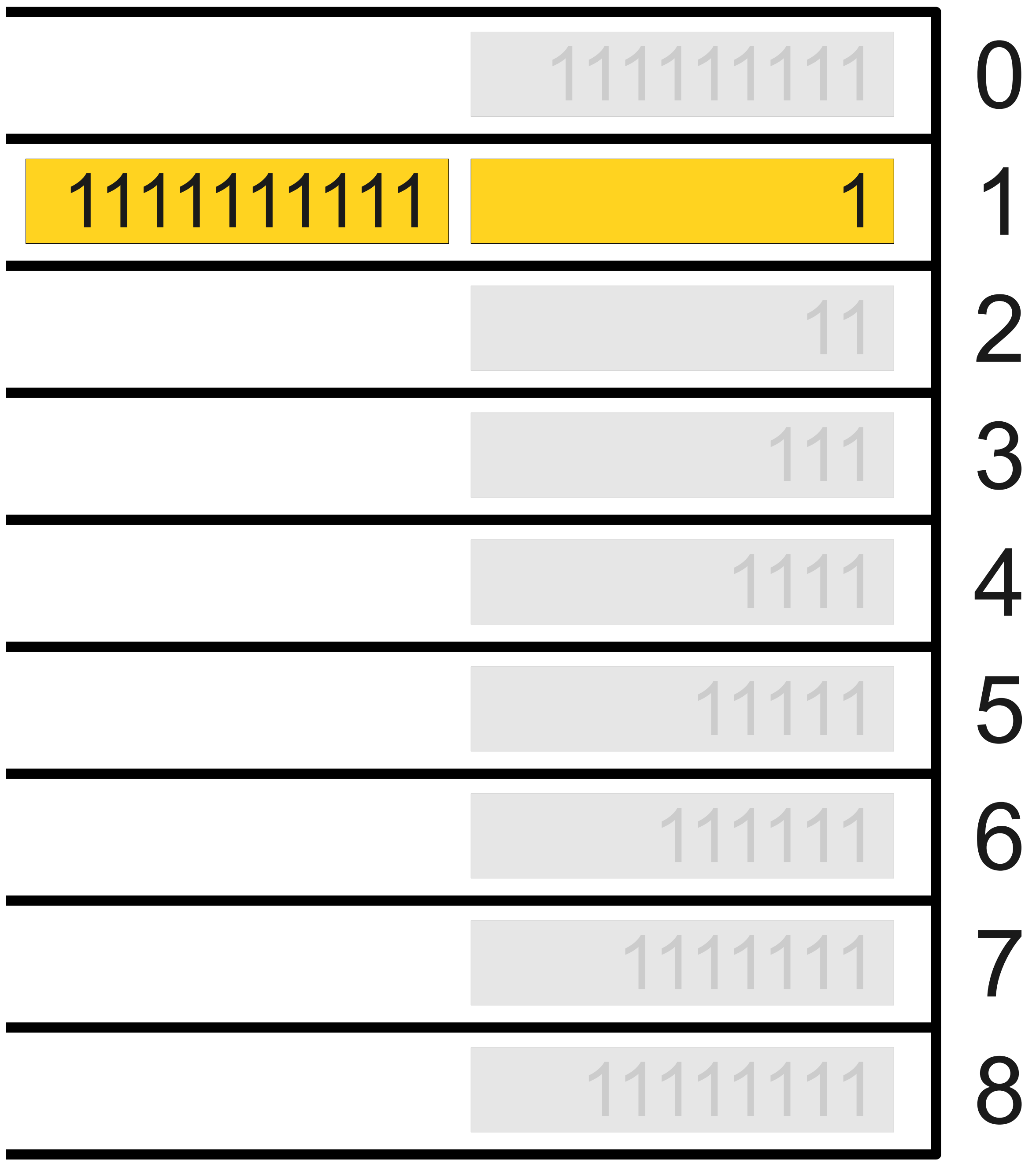
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



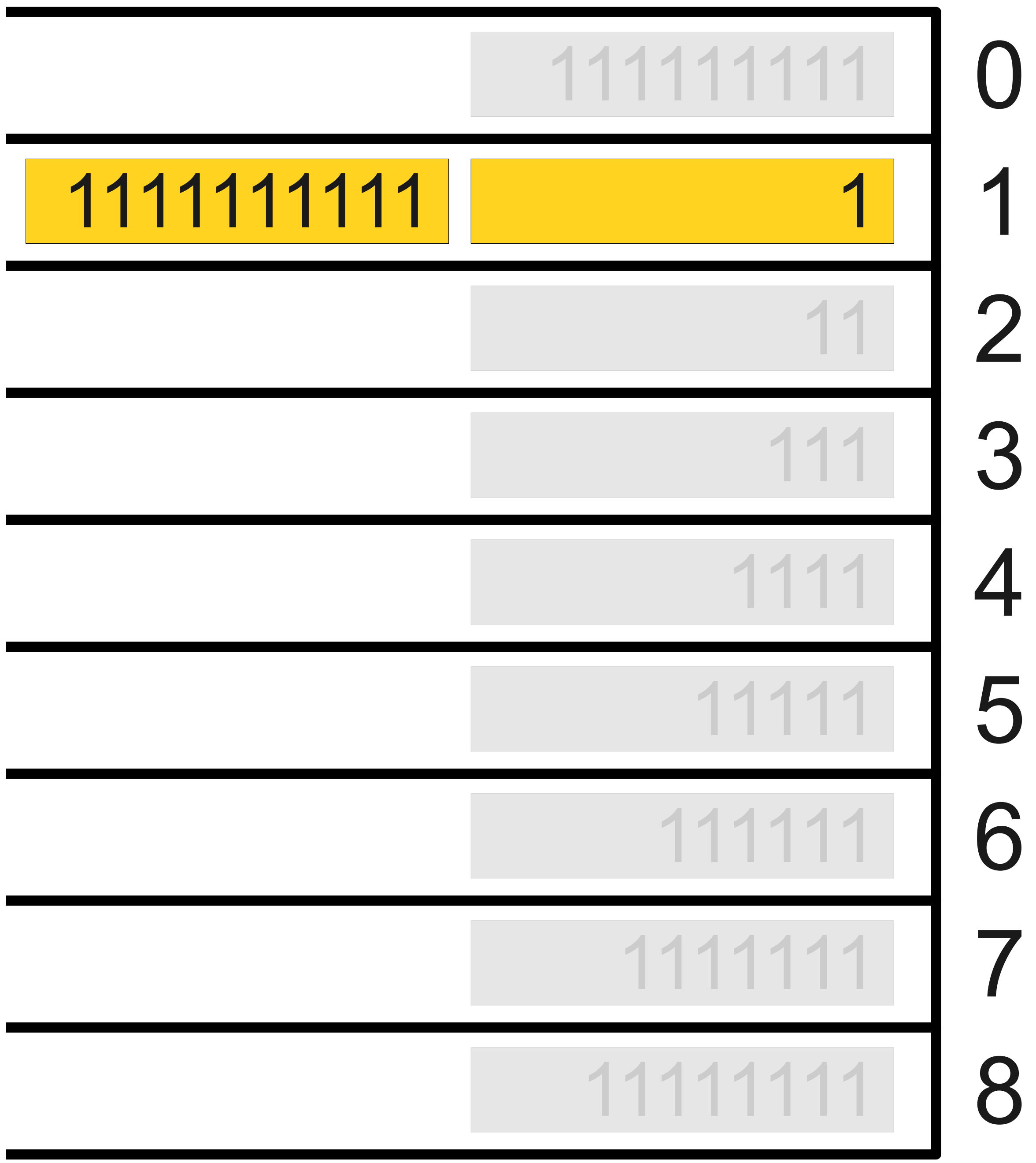
Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.



```
  1111111111
-          1
------------
  1111111110
```

Proof Idea

- For any natural number $n \geq 2$ generate the numbers 1, 11, 111, … until $n + 1$ numbers are generated.
- There are n possible remainders modulo n, so two of these numbers have the same remainder.
- Their difference is a multiple of n.
- Their difference consists of just 1s and 0s.

Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \le k \le n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo *n*.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \le k \le n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then
$S_x - S_y = (nq_x + r) - (nq_y + r)$

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then
$S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y$

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x – S_y = (nq_x + r) – (nq_y + r) = nq_x – nq_y = n(q_x – q_y)$. Thus $S_x – S_y$ is a multiple of n.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of n. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y$$

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \le k \le n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of n. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i$$

Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo *n*. Since there are $n + 1$ S_k's and *n* remainders modulo *n*, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo *n*. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be *r*.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of *n*. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i = \sum_{i=y+1}^{x} 10^i$$

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of n. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i = \sum_{i=y+1}^{x} 10^i$$

So $S_x - S_y$ is a sum of distinct powers of ten, so its digits are zeros and ones.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of n. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i = \sum_{i=y+1}^{x} 10^i$$

So $S_x - S_y$ is a sum of distinct powers of ten, so its digits are zeros and ones. Since $x > y$, we know that $x \geq y + 1$ and so the sum is nonzero.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \le k \le n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of n. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i = \sum_{i=y+1}^{x} 10^i$$

So $S_x - S_y$ is a sum of distinct powers of ten, so its digits are zeros and ones. Since $x > y$, we know that $x \ge y + 1$ and so the sum is nonzero. Therefore $S_x - S_y$ is a nonzero multiple of n consisting of 0s and 1s.

Theorem: For any natural number *n*, there is a nonzero multiple of *n* whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo *n*. Since there are $n + 1$ S_k's and *n* remainders modulo *n*, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo *n*. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be *r*.

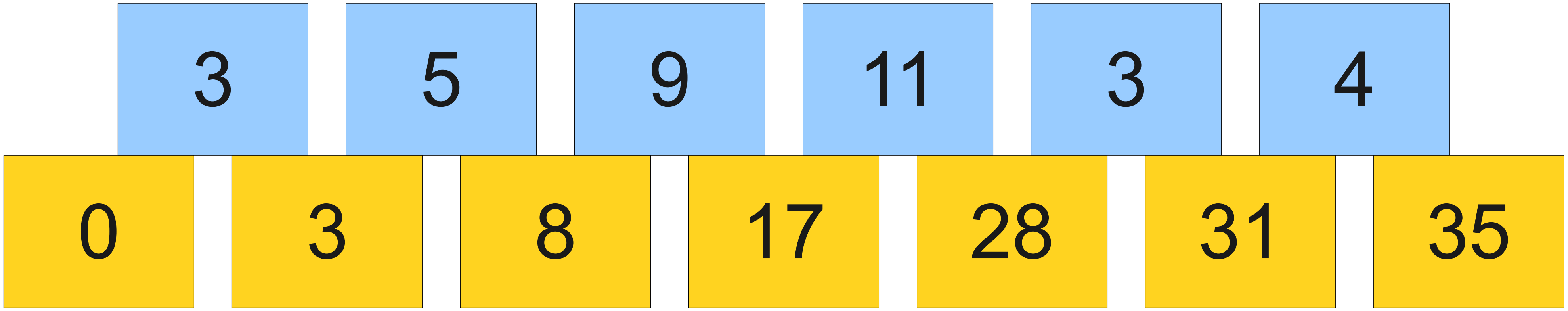
Since $S_x \equiv_n r$, there exists $q_x \in \mathbb{Z}$ such that $S_x = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_x - q_y)$. Thus $S_x - S_y$ is a multiple of *n*. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i = \sum_{i=y+1}^{x} 10^i$$

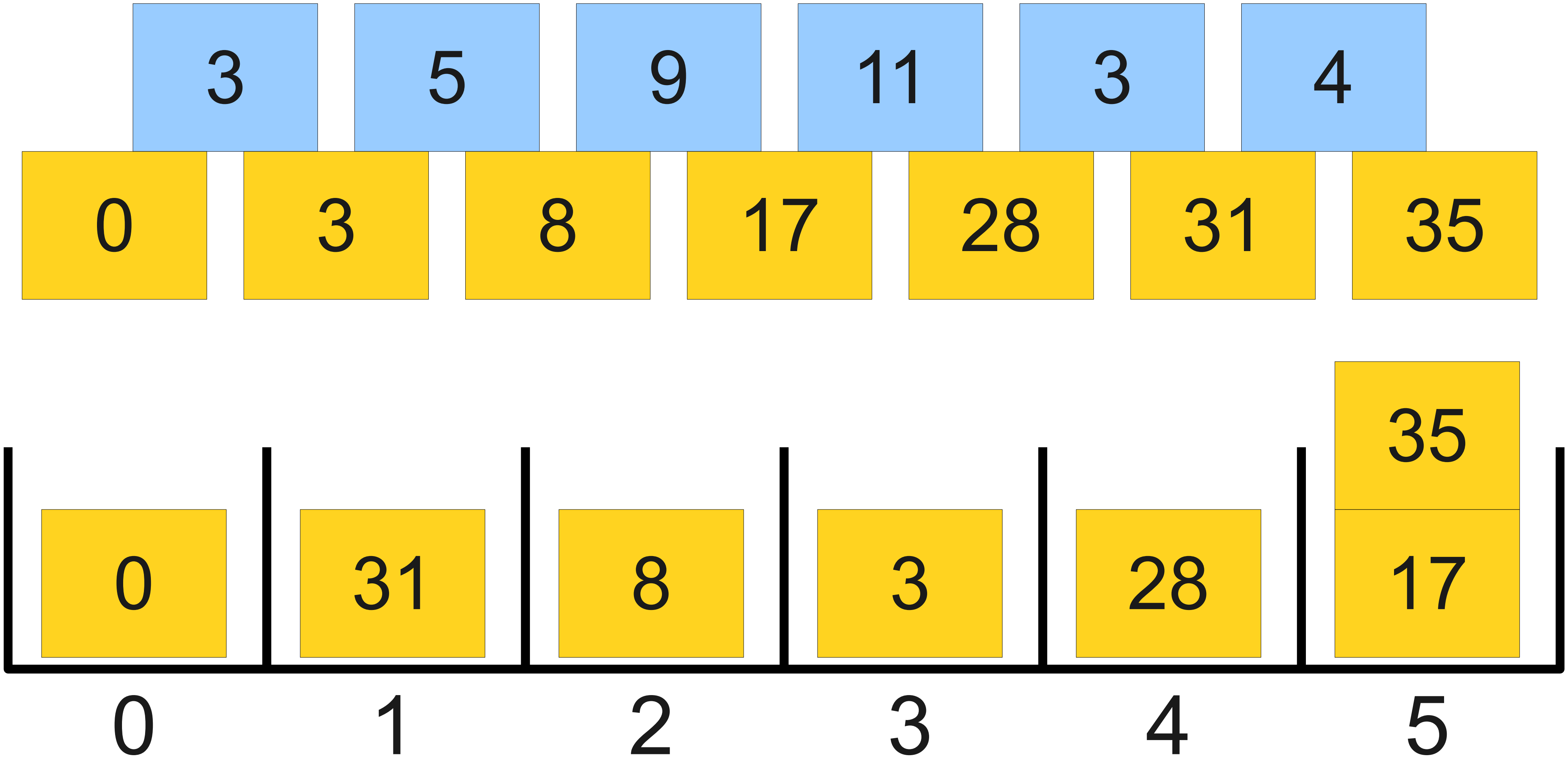
So $S_x - S_y$ is a sum of distinct powers of ten, so its digits are zeros and ones. Since $x > y$, we know that $x \geq y + 1$ and so the sum is nonzero. Therefore $S_x - S_y$ is a nonzero multiple of *n* consisting of 0s and 1s. ■

Theorem: In any set A of $n \geq 2$ integers, there is nonempty subset of A whose sum is a multiple of n.

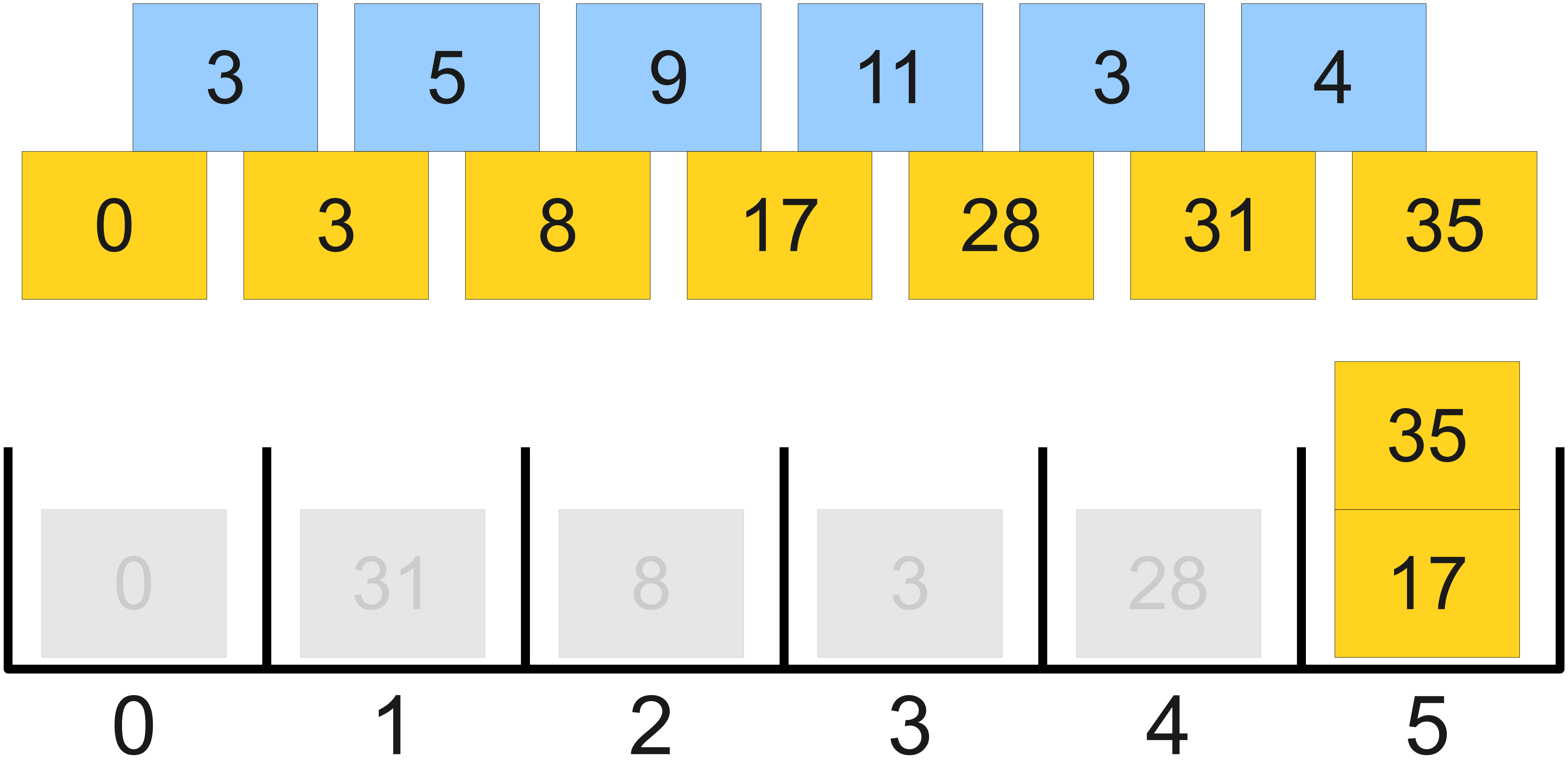
Theorem: In any set A of $n \geq 2$ integers, there is nonempty subset of A whose sum is a multiple of n.



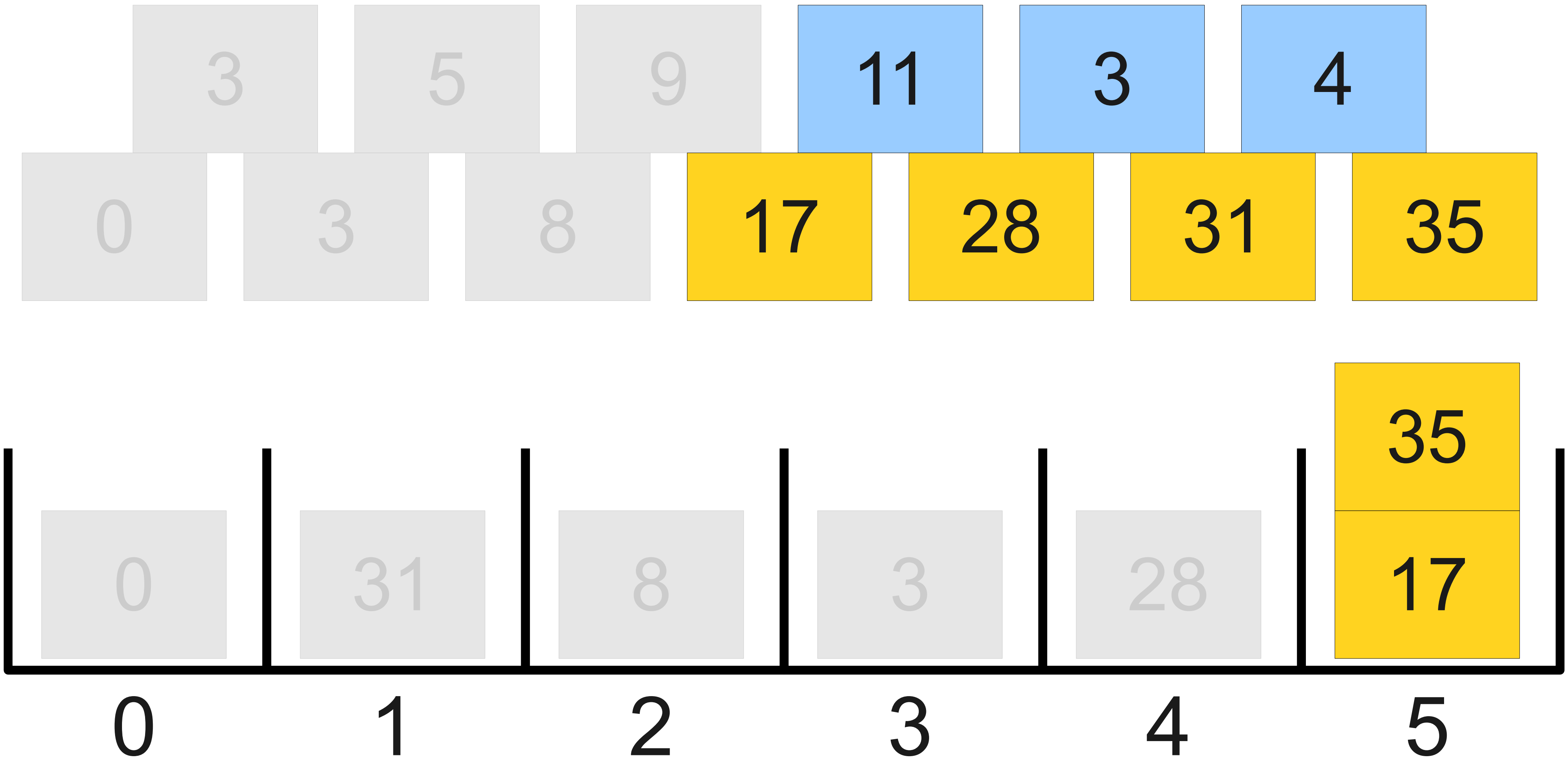
Theorem: In any set A of $n \geq 2$ integers, there is nonempty subset of A whose sum is a multiple of n.



Theorem: In any set A of $n \geq 2$ integers, there is nonempty subset of A whose sum is a multiple of n.



Theorem: In any set A of $n \geq 2$ integers, there is nonempty subset of A whose sum is a multiple of n.



The Formalization

- Compute the sum of the first 0, first 1, first 2, …, first *n* elements.
  - This is a set of *n* + 1 integers.
- Consider the numbers of remainders modulo *n*.
  - There are *n* of them.
- Therefore, two partial sums must have the same remainder modulo *n*.
- Their difference is therefore 0 modulo *n*.
- The sum of the elements between these two points therefore is a multiple of *n*.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y)$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y$$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=1}^{x} a_i - \sum_{i=1}^{y} a_i$$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=1}^{x} a_i - \sum_{i=1}^{y} a_i = \sum_{i=y+1}^{x} a_i$$

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=1}^{x} a_i - \sum_{i=1}^{y} a_i = \sum_{i=y+1}^{x} a_i$$

Since $x > y$, we know $x \geq y + 1$.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=1}^{x} a_i - \sum_{i=1}^{y} a_i = \sum_{i=y+1}^{x} a_i$$

Since $x > y$, we know $x \geq y + 1$. Consequently, this sum is nonempty.

Theorem: In any set A of $n \geq 2$ integers, there is a nonempty subset of A whose sum is a multiple of n.

Proof: Consider any set A of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each k in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo n. There are $n + 1$ different S_k's and n different remainders modulo n, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder r. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of n. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=1}^{x} a_i - \sum_{i=1}^{y} a_i = \sum_{i=y+1}^{x} a_i$$

Since $x > y$, we know $x \geq y + 1$. Consequently, this sum is nonempty. Thus the set $R = \{ a_i \mid y + 1 \leq i \leq x \}$ is a nonempty set whose sum is a multiple of n, as required.

Theorem: In any set *A* of $n \geq 2$ integers, there is a nonempty subset of *A* whose sum is a multiple of *n*.

Proof: Consider any set *A* of $n \geq 2$ integers. Number those integers $a_1, a_2, \ldots, a_n$. Now, for each *k* in the range $0 \leq k \leq n$, define

$$S_k = \sum_{i=1}^{k} a_i$$

Now consider the remainders of the S_k's modulo *n*. There are $n + 1$ different S_k's and *n* different remainders modulo *n*, so by the pigeonhole principle, at least two of the S_k's have the same remainder. Choose two of these terms with the same remainder and label them S_x and S_y, with $x > y$, and call the remainder *r*. Since $S_x \equiv_n r$, there exists some $q_y \in \mathbb{Z}$ such that $S_x = r + nq_0$, and since $S_y \equiv_n r$, we know that $S_y = r + nq_1$ for some $q_1 \in \mathbb{Z}$. Thus $S_x - S_y = (r + nq_x) - (r + nq_y) = nq_x - nq_y = n(q_x - q_y)$, so $S_x - S_y$ is a multiple of *n*. Moreover, we know that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=1}^{x} a_i - \sum_{i=1}^{y} a_i = \sum_{i=y+1}^{x} a_i$$

Since $x > y$, we know $x \geq y + 1$. Consequently, this sum is nonempty. Thus the set $R = \{ a_i \mid y + 1 \leq i \leq x \}$ is a nonempty set whose sum is a multiple of *n*, as required. ■

The Limits of Data Compression

The Pigeonhole Principle and Functions

- Consider a function $f : A \to B$ for finite sets A and B.
- If $|A| > |B|$, then by the pigeonhole principle some element of B has at least two elements of A that map to it.
- Thus f cannot be injective.

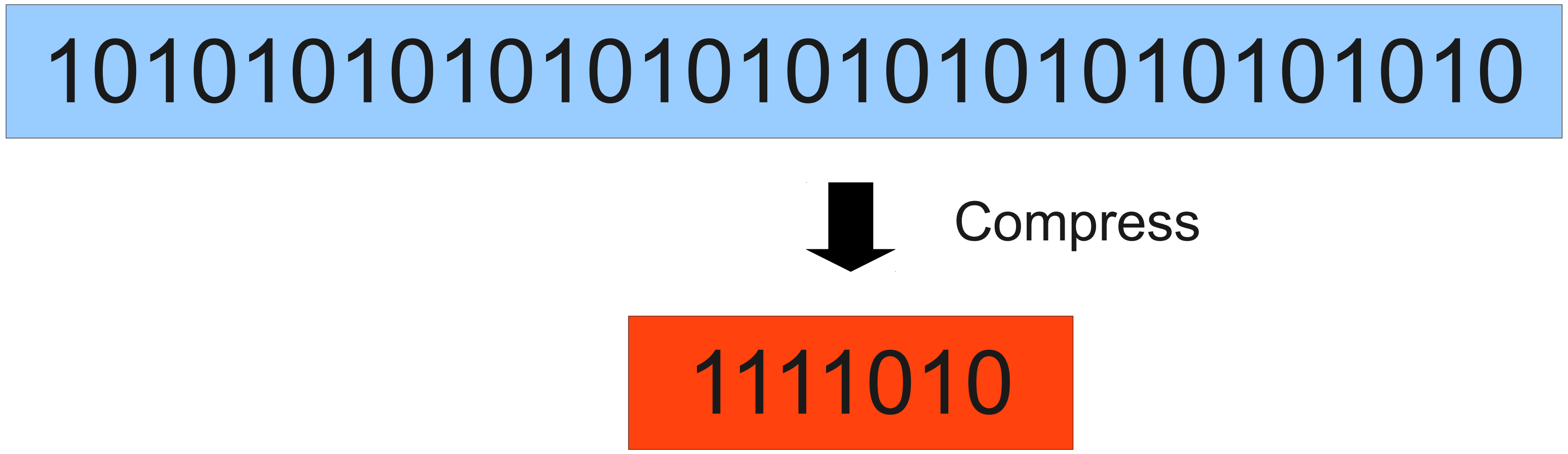
Bitstrings

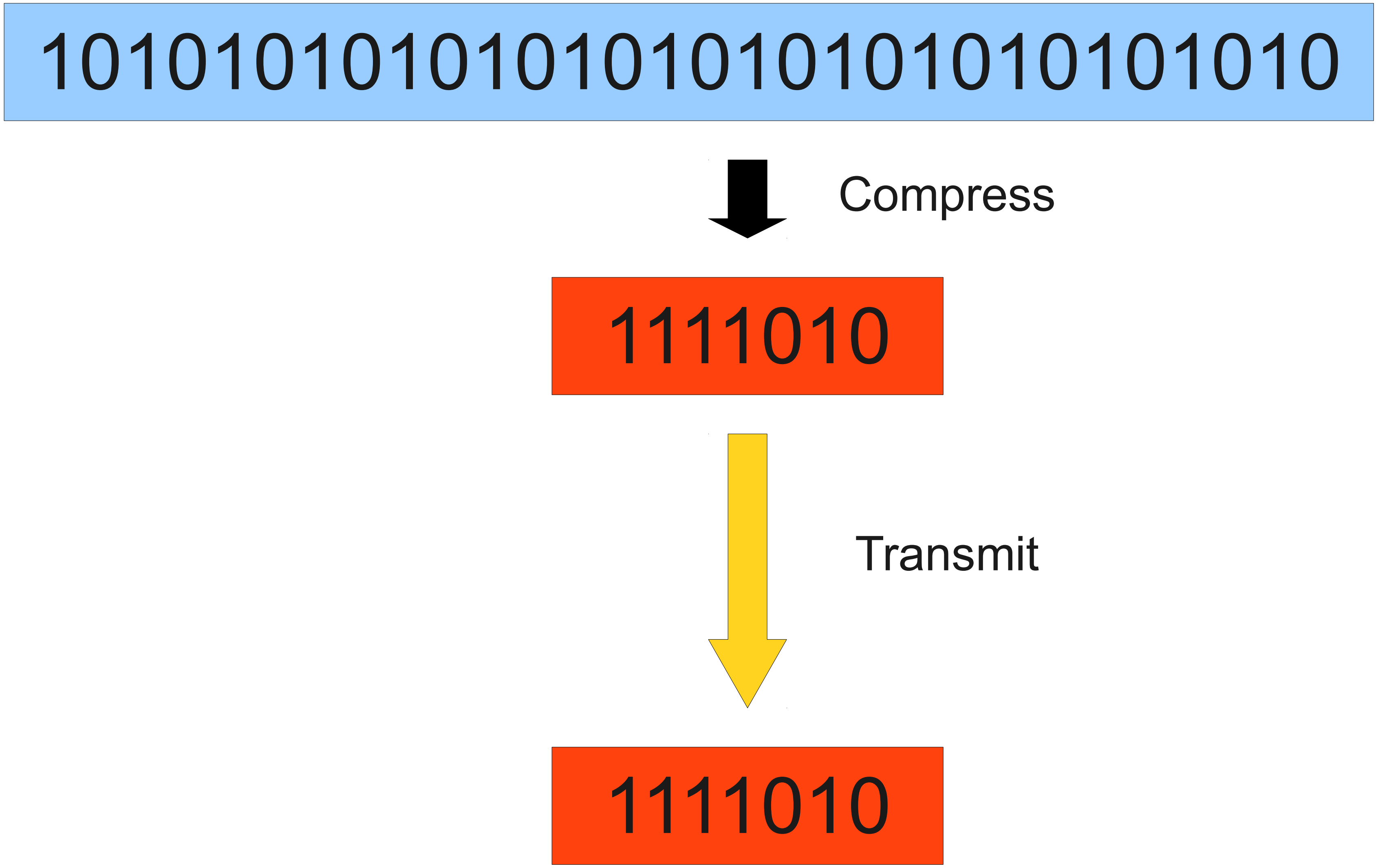
- A **bitstring** is a finite sequence of 0s and 1s.
- Examples:
  - 11011100
  - 010101010101
  - 0000
  - ε (the **empty string**)
- There are 2^n bitstrings of length exactly n.
- There are $2^0 + 2^1 + \dots + 2^{n-1} = 2^n - 1$ bitstrings of length less than n.
  - We proved this by induction earlier in the quarter.

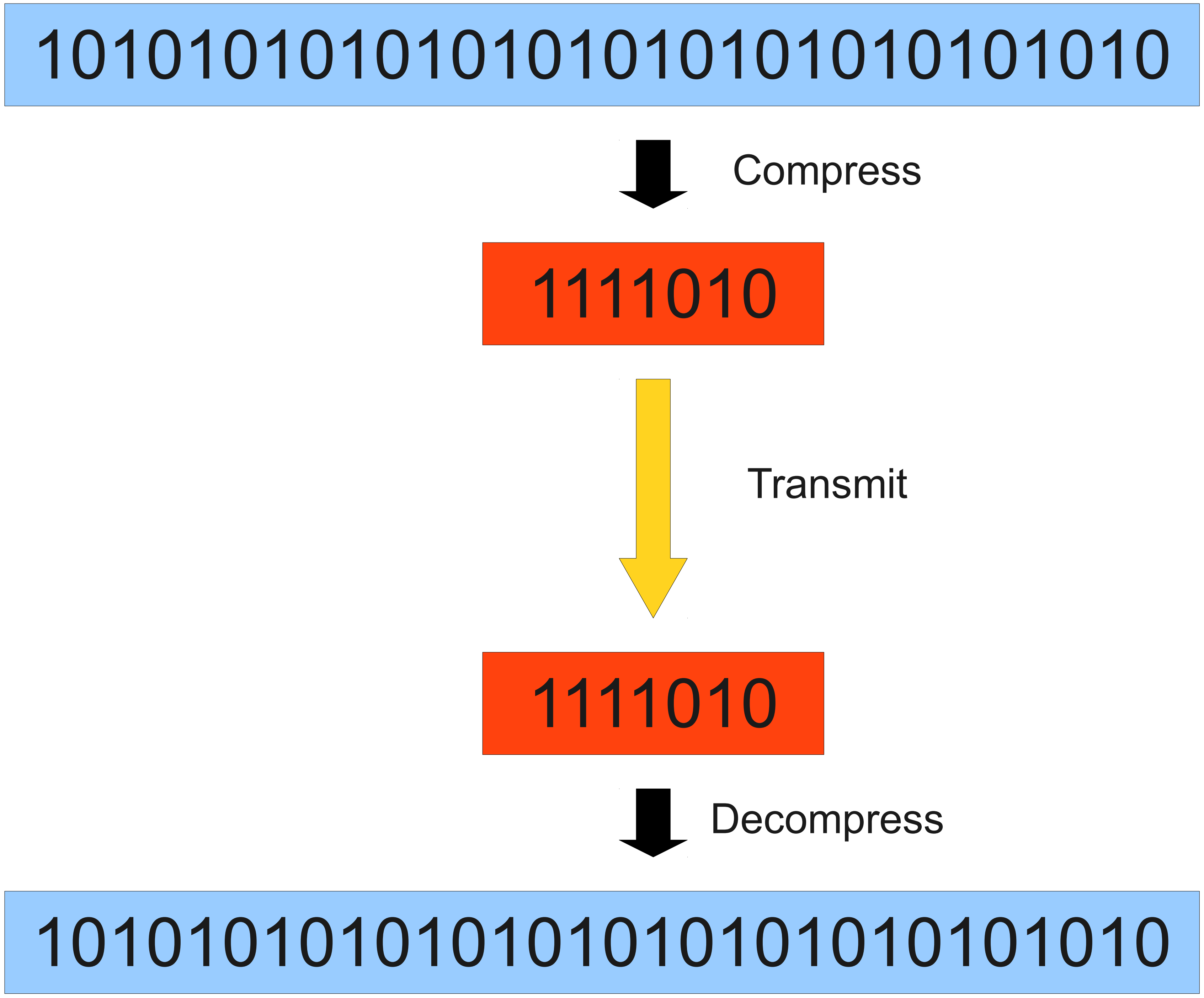
Data Compression

- Inside a computer, all data are represented as sequences of 0s and 1s (bitstrings)
- To transfer data (across a network, on DVDs, on a flash drive, etc.), it is advantageous to try to reduce the number of 0s and 1s before transferring it.
- Most real-world data can be compressed by exploiting redundancies.
  - Text repeats common patterns ("the", "and", etc.)
  - Bitmap images use similar colors throughout the image.
- **Idea**: Replace each bitstring with a *shorter* bitstring that contains all the original information.
  - This is called **lossless data compression**.

101010101010101010101010101010


101010101010101010101010101010
Compress
1111010


101010101010101010101010101010
Compress
1111010
Transmit
1111010


101010101010101010101010101010
Compress
1111010
Transmit
1111010
Decompress
101010101010101010101010101010

Lossless Data Compression

- In order to losslessly compress data, we need two functions:
  - A **compression function** C, and
  - A **decompression function** D.
- These functions must be inverses of one another: $D = C^{-1}$.
  - Otherwise, we can't uniquely encode or decode some bitstring.
- Since C is invertible, C must be a bijection.

A Perfect Compression Function

- Ideally, the compressed version of a bitstring would always be shorter than the original bitstring.
- **Question**: Can we find a lossless compression algorithm that always compresses a string into a shorter string?
- To handle the issue of the empty string (which can't get any shorter), let's assume we only care about strings of length at least 10.

A Counting Argument

- Let $\mathbb{B}^n$ be the set of bitstrings of length n, and $\mathbb{B}^{<n}$ be the set of bitstrings of length less than n.
- How many bitstrings of length n are there?
  - **Answer**: 2^n
- How many bitstrings of length *less than n* are there?
  - **Answer:** $2^0 + 2^1 + \ldots + 2^{n-1} = 2^n - 1$
- Using our earlier result, by the pigeonhole principle, there cannot be an injective function from $\mathbb{B}^n$ to $\mathbb{B}^{<n}$.
- Since every bijection is an injection, there is no bijection between $\mathbb{B}^n$ and $\mathbb{B}^{<n}$.
- **There is no perfect compression function!**

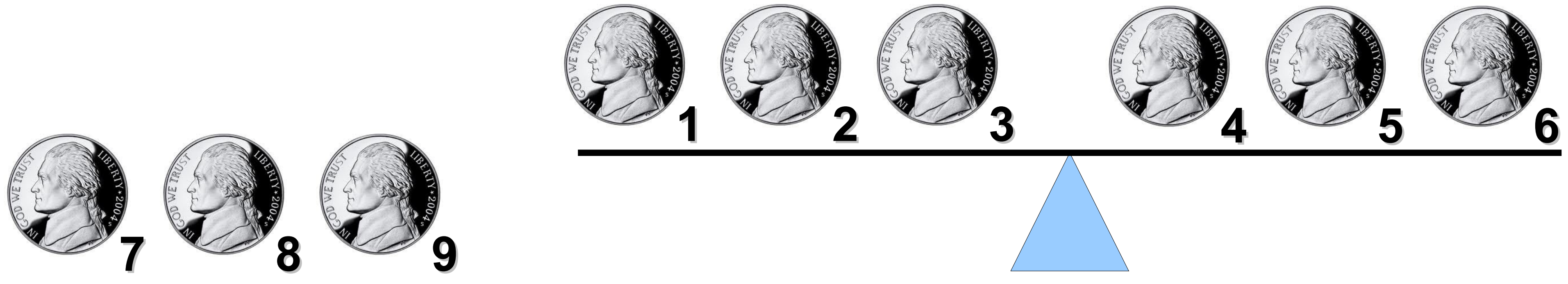
Why this Result is Interesting

- Our result says that no matter how hard we try, it is **impossible** to compress every string into a shorter string.
- No matter how clever you are, you cannot write a lossless compression algorithm that always makes strings shorter.
- In practice, only highly redundant data can be compressed.
- The fields of **information theory** and **Kolmogorov complexity** explore the limits of compression; if you're interested, go explore!

The Limits of Counterfeit Detection

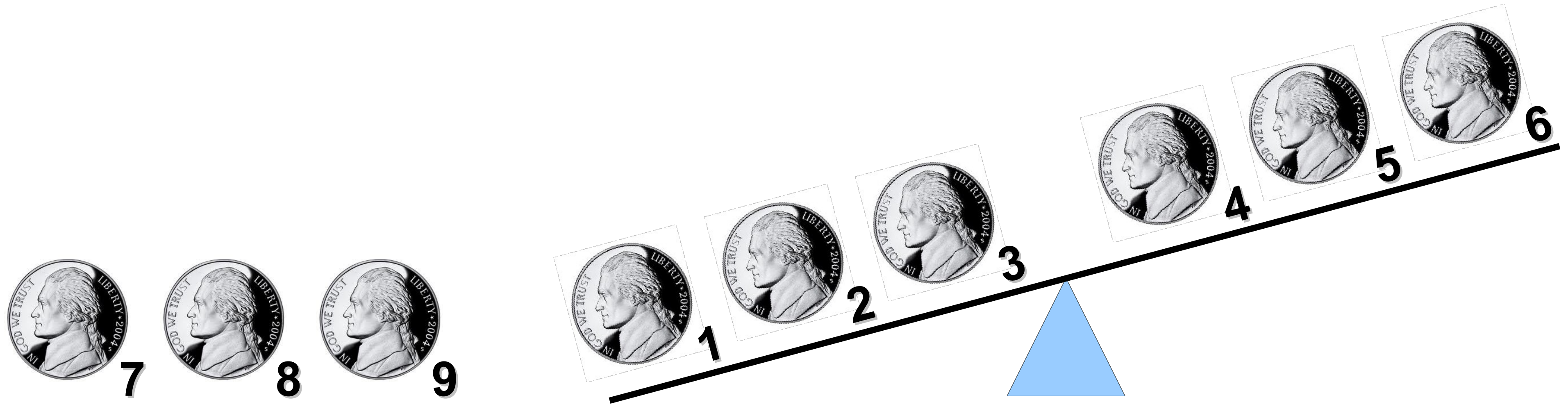
The Counterfeit Coin Problem

- Given 3^n coins, one of which weighs more than the rest, find that coin with at most *n* weighings on a balance.



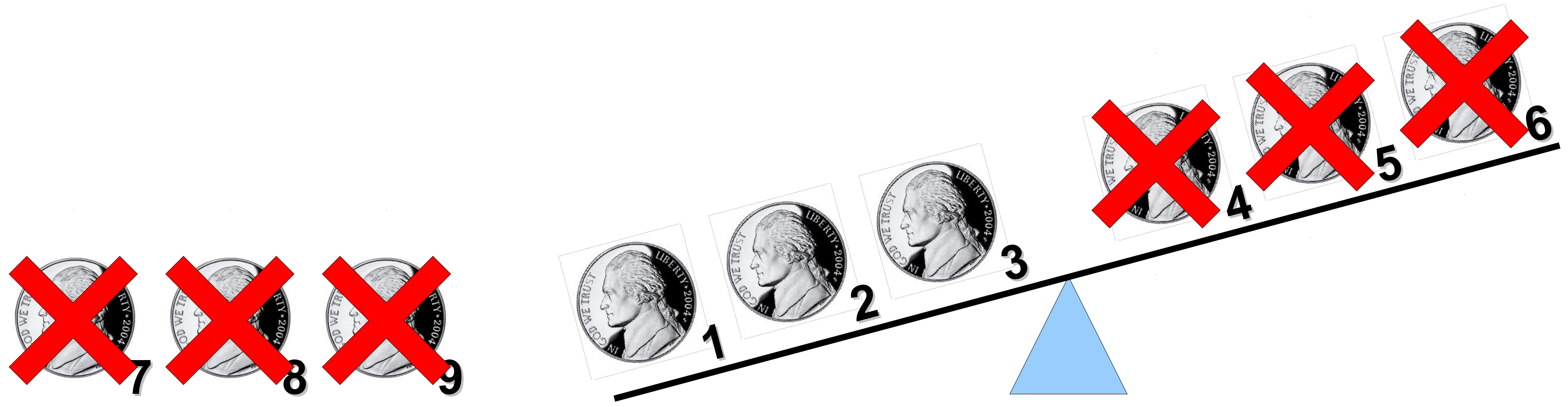
The Counterfeit Coin Problem

- Given 3^n coins, one of which weighs more than the rest, find that coin with at most *n* weighings on a balance.



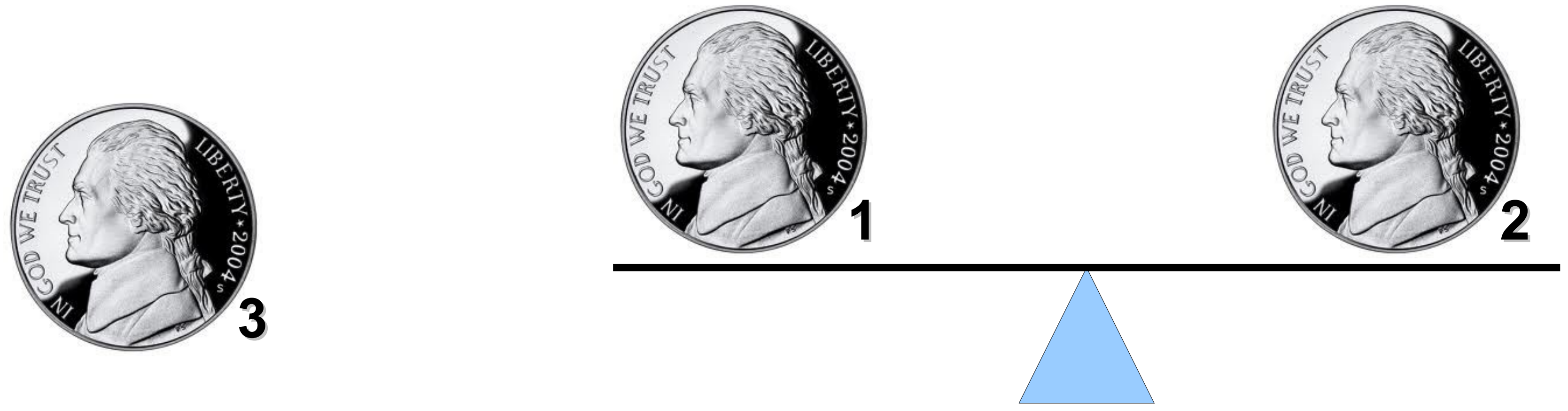
The Counterfeit Coin Problem

- Given 3^n coins, one of which weighs more than the rest, find that coin with at most *n* weighings on a balance.



The Counterfeit Coin Problem

- Given 3^n coins, one of which weighs more than the rest, find that coin with at most *n* weighings on a balance.

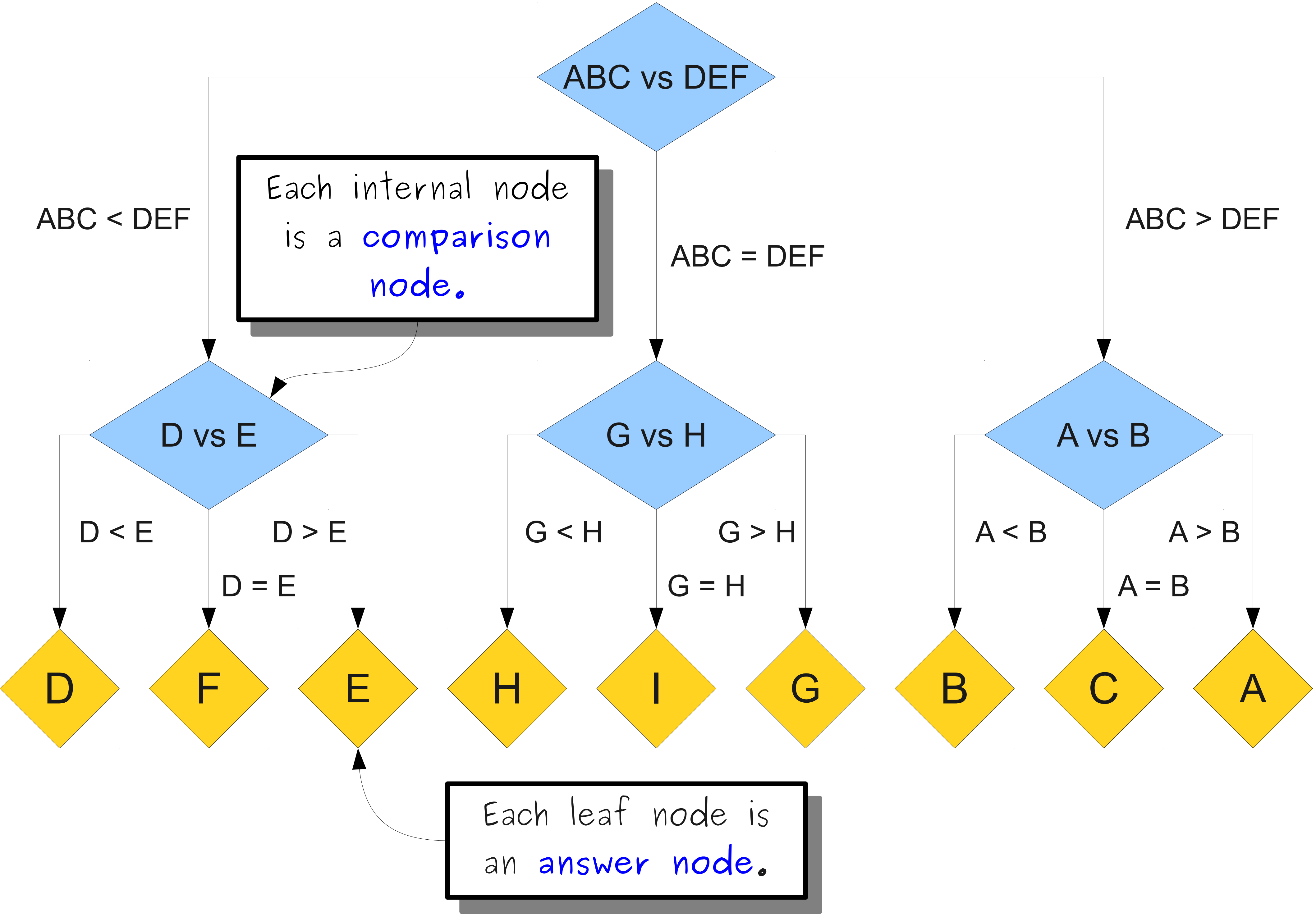


The Counterfeit Coin Problem

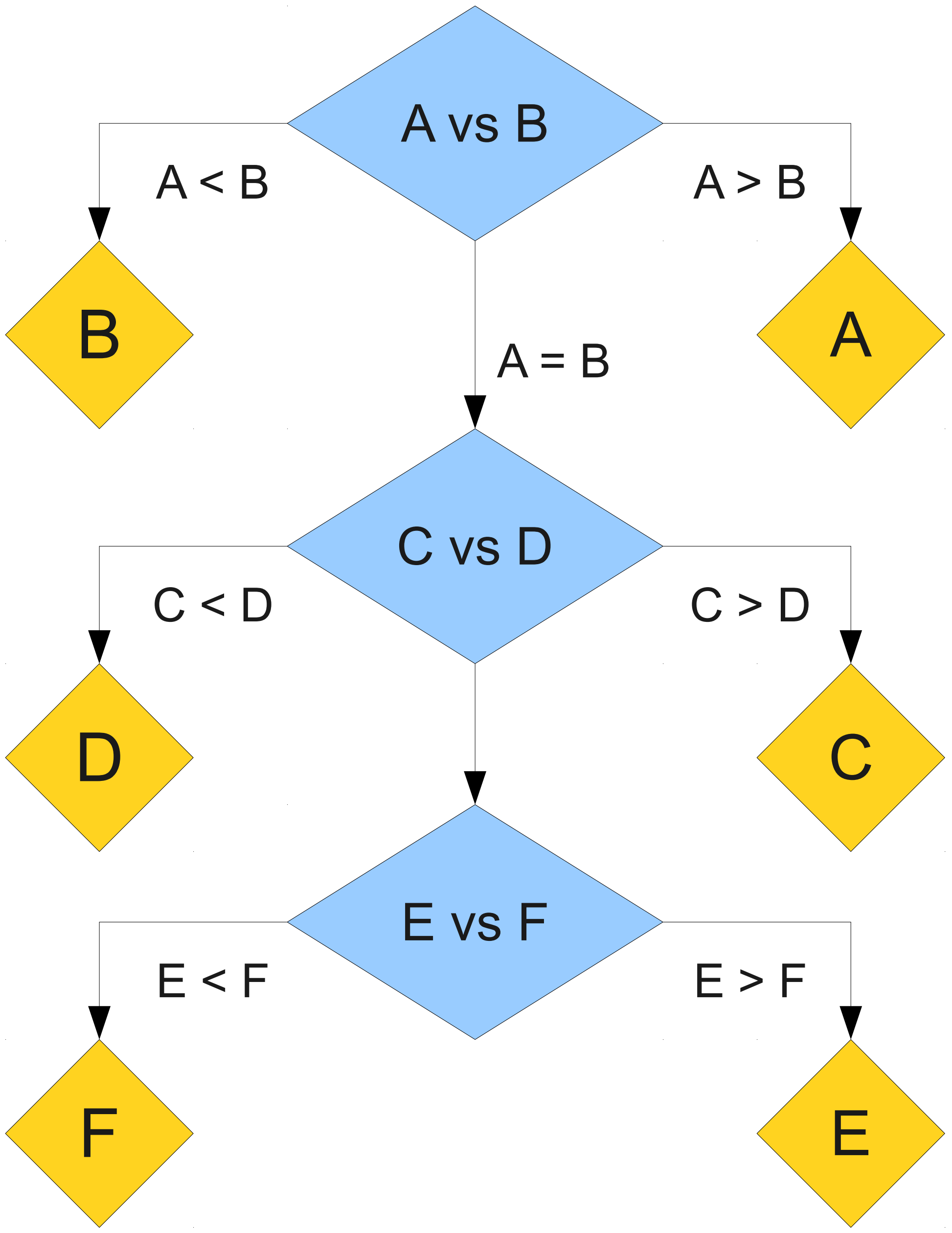
- Given 3^n coins, one of which weighs more than the rest, find that coin with at most *n* weighings on a balance.
- **Question**: Can we find the counterfeit coin out of *more* than 3^n coins using just *n* weighings?

Modeling an Algorithm

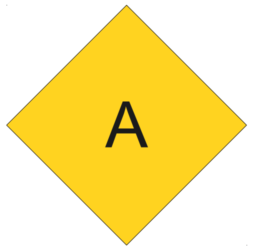
- In order to reason about the maximum number of coins, we need to find some way to reason about all possible algorithms for finding the coin.
- Main assumption: The only operation we can perform on the coins is weighing them on the scale.
    - We can't test their density, give them to the Secret Service, etc.
- We'll call such an algorithm a **comparison-based algorithm**, since the only way of distinguishing coins is through comparisons.


ABC vs DEF
ABC < DEF
ABC = DEF
ABC > DEF
Each internal node is a comparison node.
D vs E
G vs H
A vs B
D < E
D = E
D > E
G < H
G = H
G > H
A < B
A = B
A > B
D
F
E
H
I
G
B
C
A
Each leaf node is an answer node.

An Algorithm for Six Coins



An Algorithm for One Coin



Reasoning about Algorithms

- In this setup, every algorithm corresponds to a tree structure consisting of **comparisons** and **answers**.
- Each comparison node produces one of three outputs.
- Each answer node immediately ends the algorithm with the answer.
- Reasoning about these structures will tell us about the counterfeit coin problem.

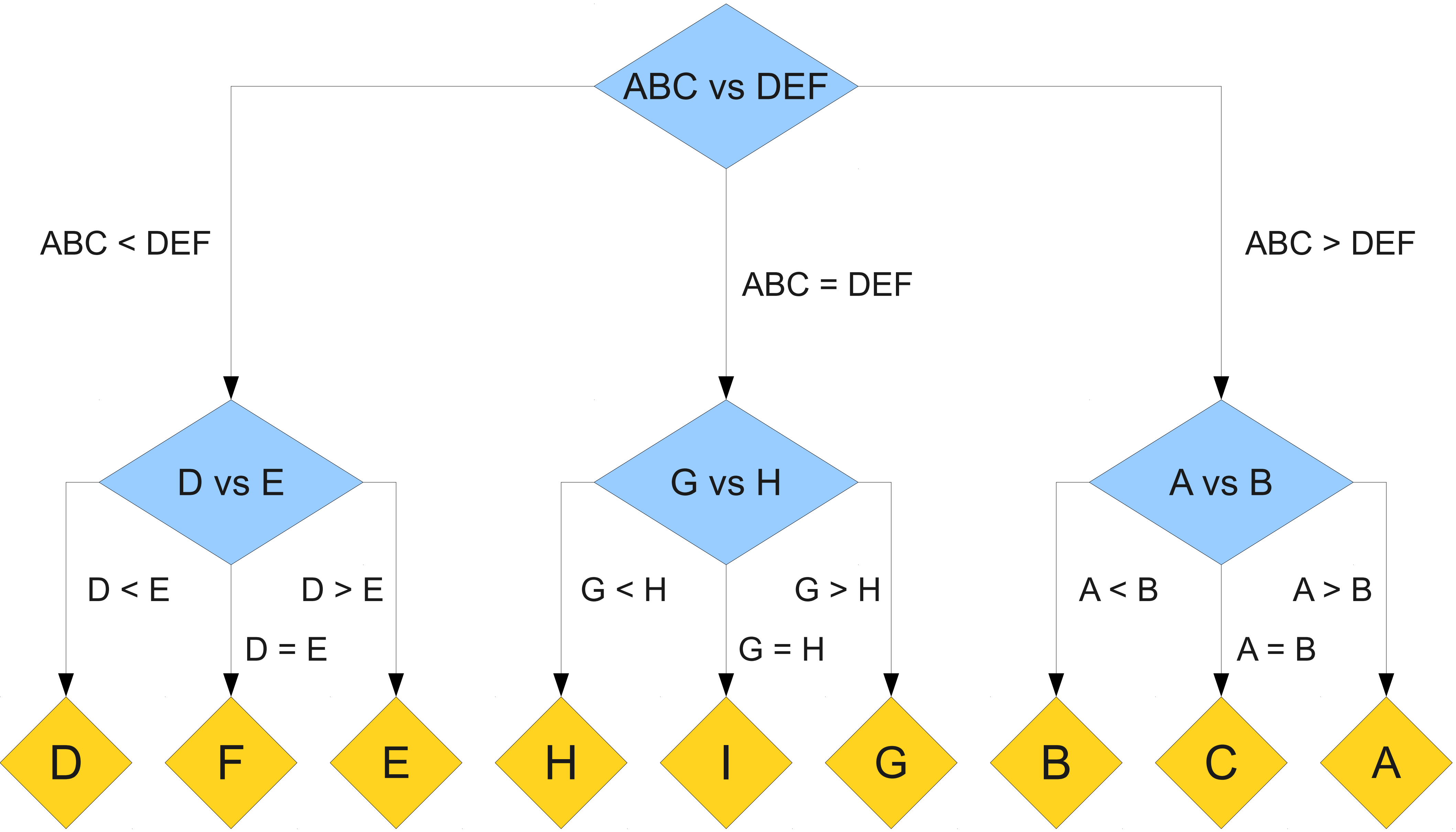
Reasoning about Inputs

- To be precise, we need to reason about the inputs to our algorithm.
- An input is a collection of *k* coins, exactly one of which is heavier than the rest.
  - It doesn't matter how much heavier it is; just that it weighs more than the rest.
- This means that there are exactly *k* possible inputs to the algorithm – one in which the first coin is counterfeit, one in which the second coin is counterfeit, etc.

A Critical Observation

- Suppose that we have an algorithm for finding which of *k* coins is counterfeit.
- There must be at least *k* answer nodes in the tree.
- Reasoning from the pigeonhole principle:
  - Run the algorithm on all *k* possible inputs.
  - Consider the set of counterfeit coins that arrive at each answer node in the tree.
  - If there are more coins than answer nodes, there must be two different coins that arrive at the same answer node.
  - The algorithm has to be wrong at least one of the two inputs.

How many answer nodes can there be in the tree?


ABC vs DEF
ABC < DEF
ABC = DEF
ABC > DEF
D vs E
G vs H
A vs B
D < E
D = E
D > E
G < H
G = H
G > H
A < B
A = B
A > B
D
F
E
H
I
G
B
C
A

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ decision nodes, so the claim holds.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ decision nodes, so the claim holds.

Otherwise, the first step of the algorithm is a comparison.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ decision nodes, so the claim holds.

Otherwise, the first step of the algorithm is a comparison. For each of the three outcomes, the algorithm can then make up to n more comparisons.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ decision nodes, so the claim holds.

Otherwise, the first step of the algorithm is a comparison. For each of the three outcomes, the algorithm can then make up to n more comparisons. By the inductive hypothesis, this means that in each of the subtrees corresponding to these outcomes, there can be up to 3^n answer nodes.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ decision nodes, so the claim holds.

Otherwise, the first step of the algorithm is a comparison. For each of the three outcomes, the algorithm can then make up to n more comparisons. By the inductive hypothesis, this means that in each of the subtrees corresponding to these outcomes, there can be up to 3^n answer nodes. Consequently, the overall algorithm can have at most $3(3^n) = 3^{n+1}$ answer nodes, completing the induction.

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n decision nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ decision nodes, so the claim holds.

Otherwise, the first step of the algorithm is a comparison. For each of the three outcomes, the algorithm can then make up to n more comparisons. By the inductive hypothesis, this means that in each of the subtrees corresponding to these outcomes, there can be up to 3^n answer nodes. Consequently, the overall algorithm can have at most $3(3^n) = 3^{n+1}$ answer nodes, completing the induction. ■

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs.

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all k possible inputs.

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all k possible inputs. By the pigeonhole principle, since there are k inputs and fewer than k answer nodes, at least two inputs must arrive in the same answer node.

Theorem: Any comparison-based algorithm for finding which of *k* coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all *k* possible inputs. By the pigeonhole principle, since there are *k* inputs and fewer than *k* answer nodes, at least two inputs must arrive in the same answer node. Since these two inputs have different answers, this means that the algorithm must be incorrect on at least one of these two inputs, contradicting the fact that the algorithm finds which of the coins is heavier.

Theorem: Any comparison-based algorithm for finding which of *k* coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all *k* possible inputs. By the pigeonhole principle, since there are *k* inputs and fewer than *k* answer nodes, at least two inputs must arrive in the same answer node. Since these two inputs have different answers, this means that the algorithm must be incorrect on at least one of these two inputs, contradicting the fact that the algorithm finds which of the coins is heavier.

We have reached a contradiction, so our assumption must have been wrong.

Theorem: Any comparison-based algorithm for finding which of *k* coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all *k* possible inputs. By the pigeonhole principle, since there are *k* inputs and fewer than *k* answer nodes, at least two inputs must arrive in the same answer node. Since these two inputs have different answers, this means that the algorithm must be incorrect on at least one of these two inputs, contradicting the fact that the algorithm finds which of the coins is heavier.

We have reached a contradiction, so our assumption must have been wrong. Thus any comparison-based algorithm for finding the heavier coin out of *k* coins must make at least $\log_3 k$ weighings.

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all k possible inputs. By the pigeonhole principle, since there are k inputs and fewer than k answer nodes, at least two inputs must arrive in the same answer node. Since these two inputs have different answers, this means that the algorithm must be incorrect on at least one of these two inputs, contradicting the fact that the algorithm finds which of the coins is heavier.

We have reached a contradiction, so our assumption must have been wrong. Thus any comparison-based algorithm for finding the heavier coin out of k coins must make at least $\log_3 k$ weighings. ■

Corollary 1: There is no comparison-based algorithm for finding which of $3^n + k$ coins is counterfeit in n weighings for any $k > 0$.

Proof: By our previous result, we need at least $\log_3 (3^n + k)$ comparisons to determine which of $3^n + k$ coins is heaviest. If $k > 0$, then $\log_3 (3^n + k) > \log_3 3^n = n$. Thus we need strictly more than n weighings to find which of the coins is counterfeit. ■

Corollary 1: There is no comparison-based algorithm for finding which of $3^n + k$ coins is counterfeit in n weighings for any $k > 0$.

Proof: By our previous result, we need at least $\log_3 (3^n + k)$ comparisons to determine which of $3^n + k$ coins is heaviest. If $k > 0$, then $\log_3 (3^n + k) > \log_3 3^n = n$. Thus we need strictly more than n weighings to find which of the coins is counterfeit. ■

Corollary 2: The comparison-based algorithm we developed in class is optimal.

What Just Happened?

- **This is our first theorem about the difficulty of a specific problem!**
- Our procedure was as follows:
    - Build a mathematical model of computation for finding the counterfeit coin.
    - Given the model, reason about the behavior of that model on various inputs.
    - Write a proof that formalizes our reasoning about that behavior.
- We will build many more models like this one later in the quarter.

Suppose you have a set of coins. There is a counterfeit coin among them that weighs more than the rest of the coins.

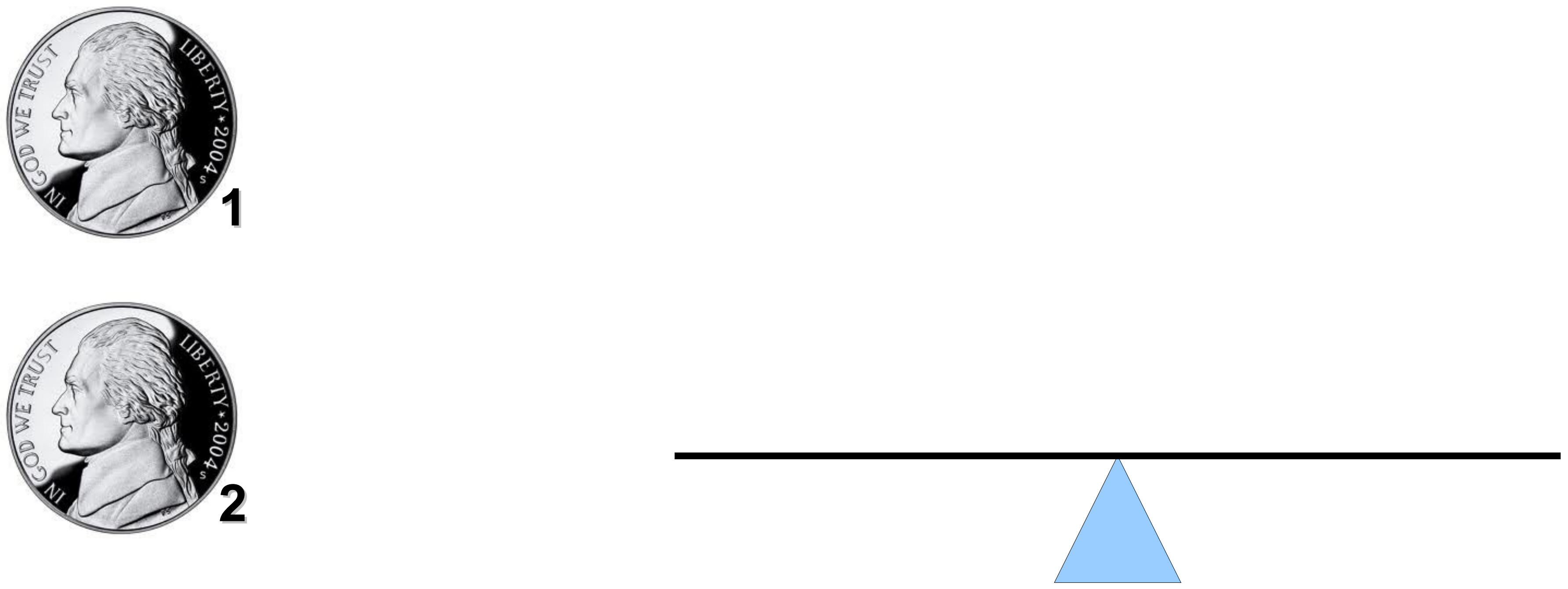
If you have *n* weighings, what is the largest number of coins for which you can solve this problem?

Suppose you have a set of coins. There **may be** a counterfeit coin among them **(though there doesn't have to be)**. If there is a counterfeit, it will weigh more than the rest of the coins.

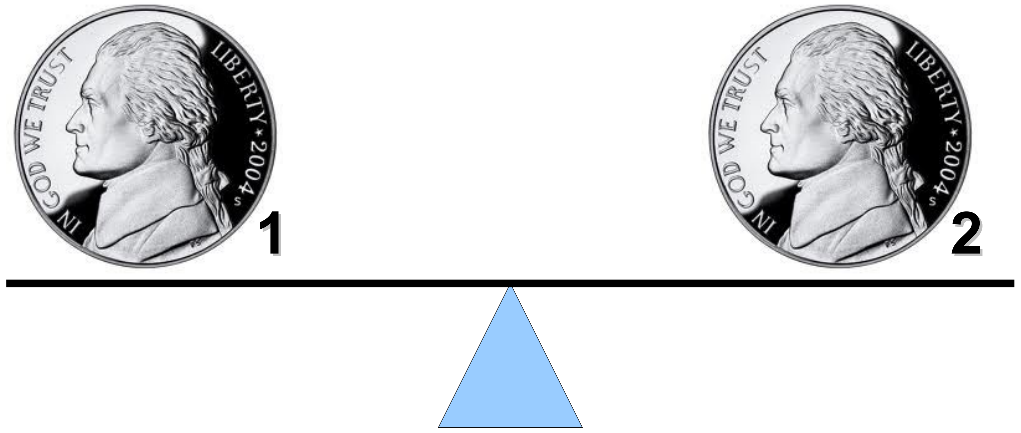
If you have *n* weighings, what is the largest number of coins for which you can solve this problem?

The Possibly Counterfeit Coin Problem

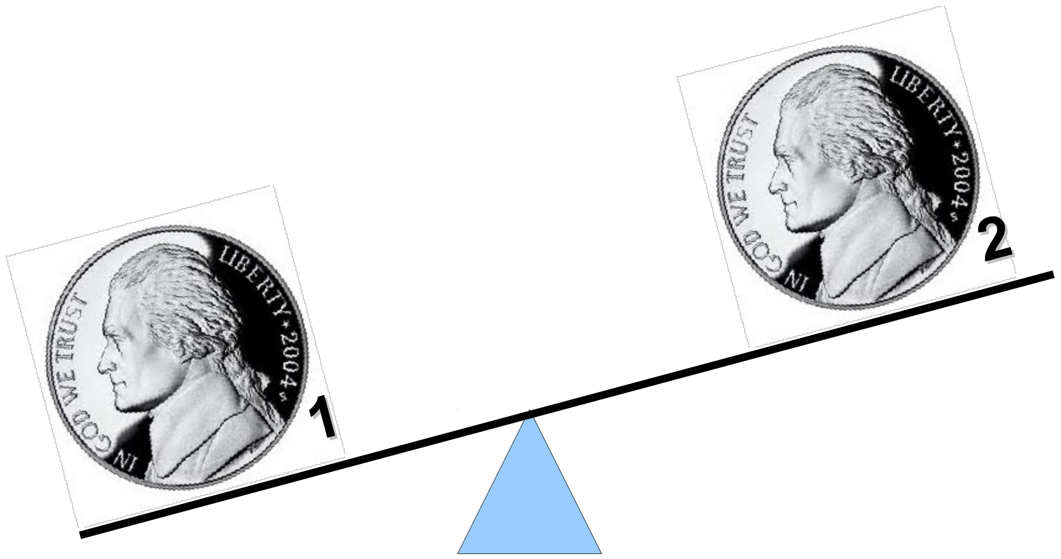
The Possibly Counterfeit Coin Problem



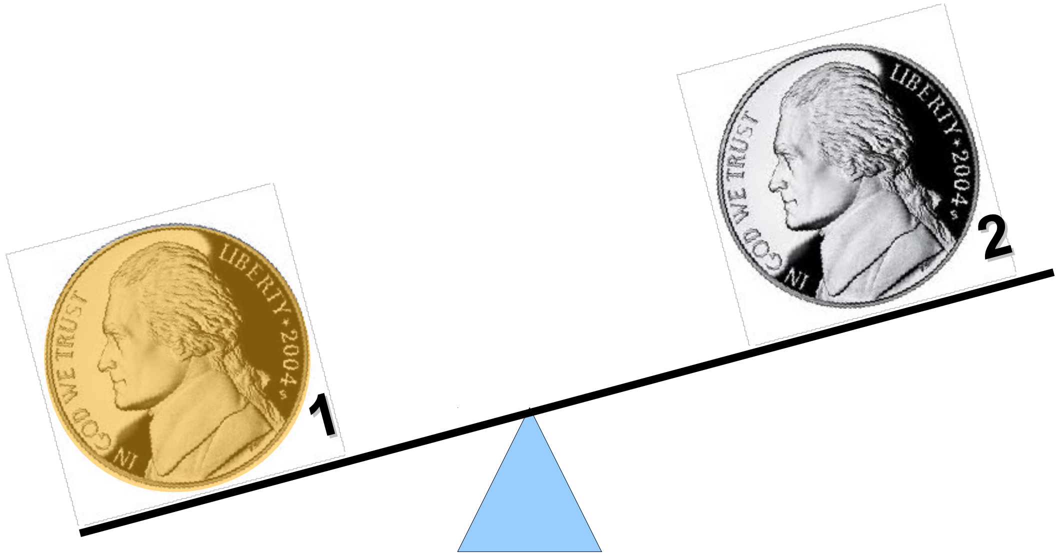
The Possibly Counterfeit Coin Problem



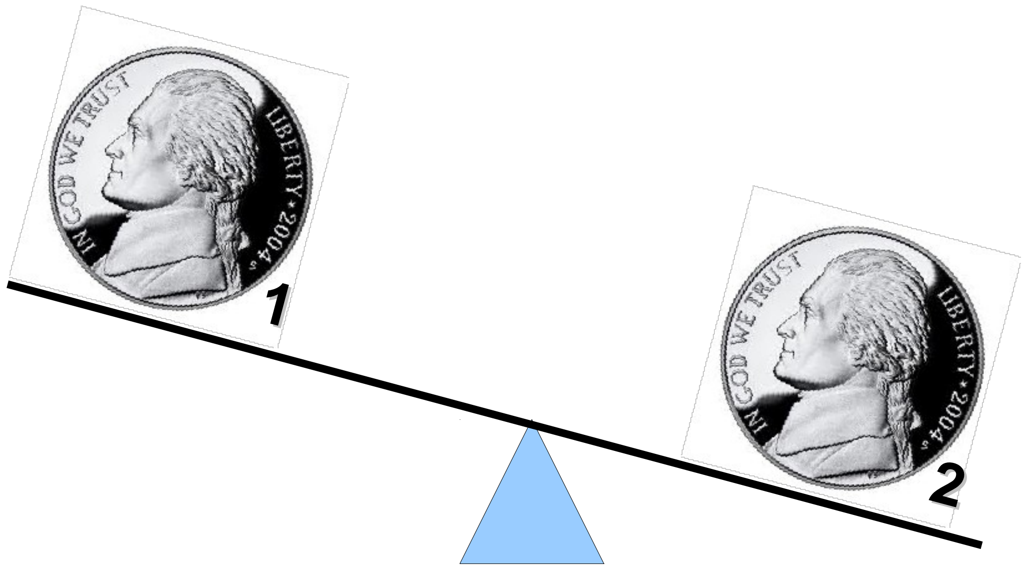
The Possibly Counterfeit Coin Problem



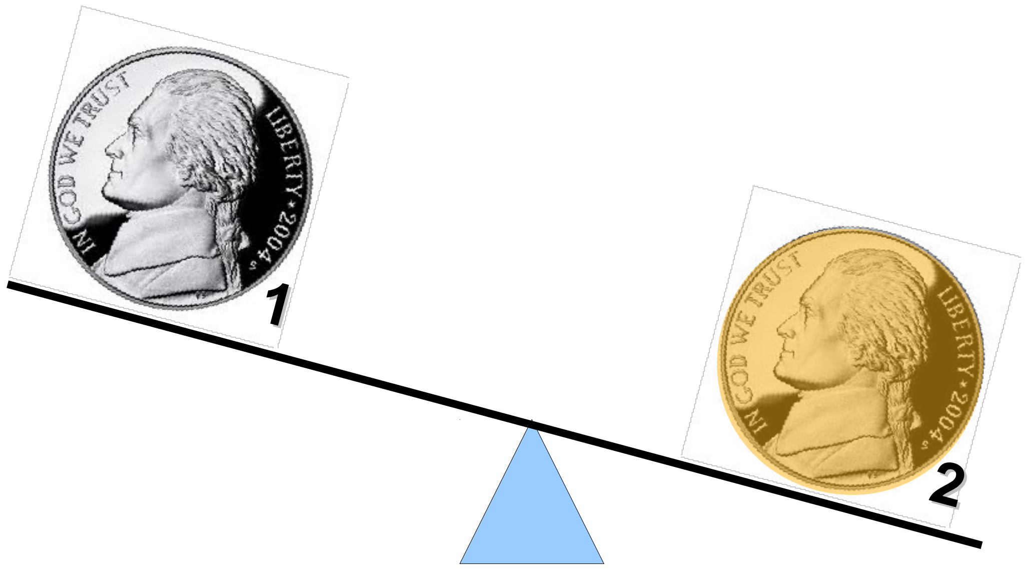
The Possibly Counterfeit Coin Problem



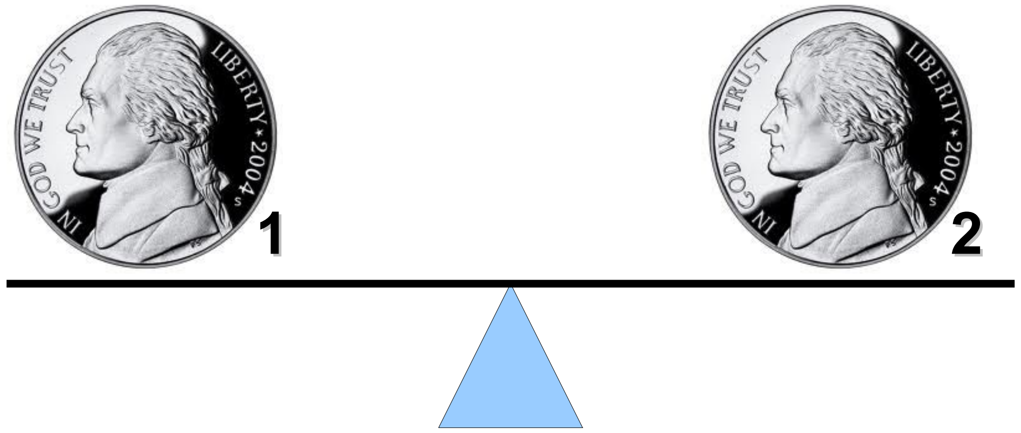
The Possibly Counterfeit Coin Problem



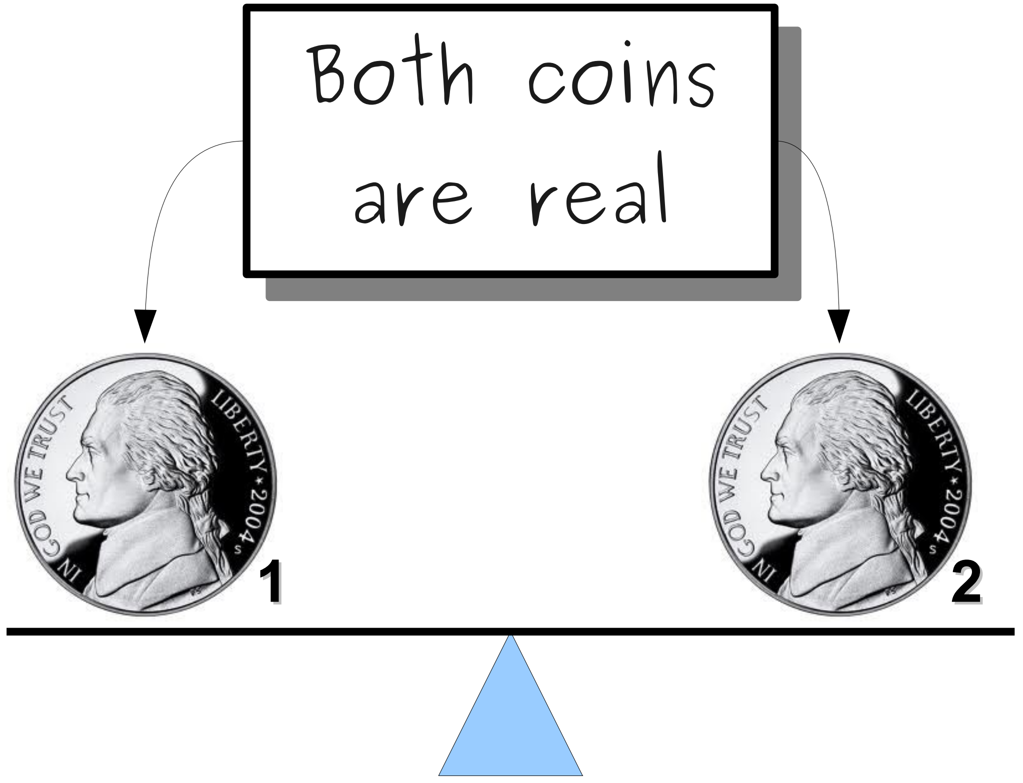
The Possibly Counterfeit Coin Problem



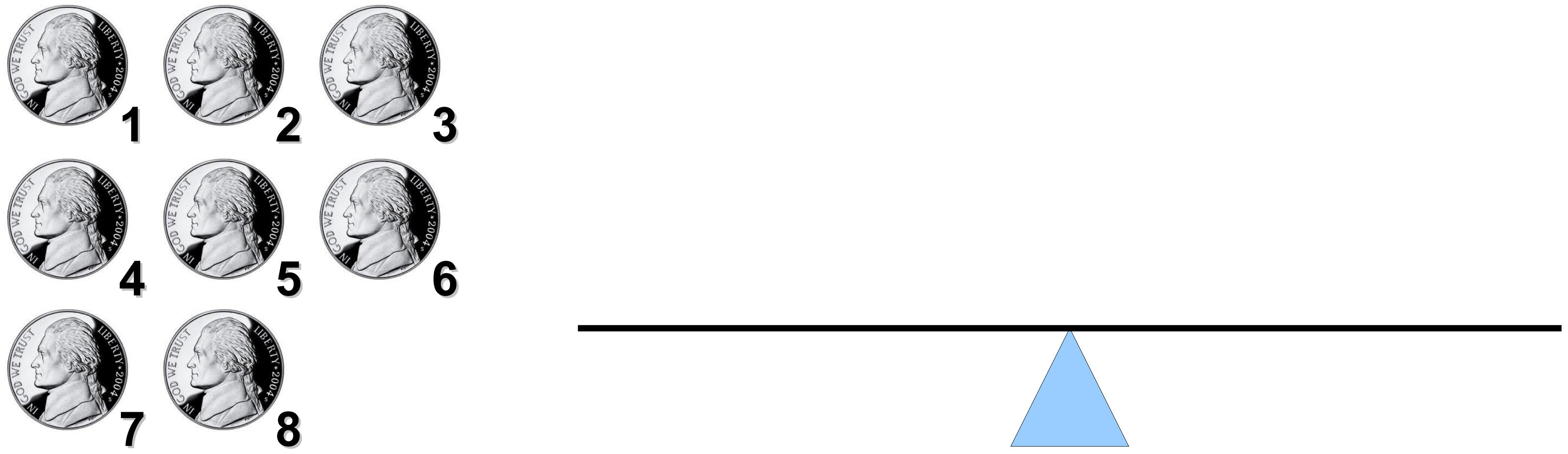
The Possibly Counterfeit Coin Problem



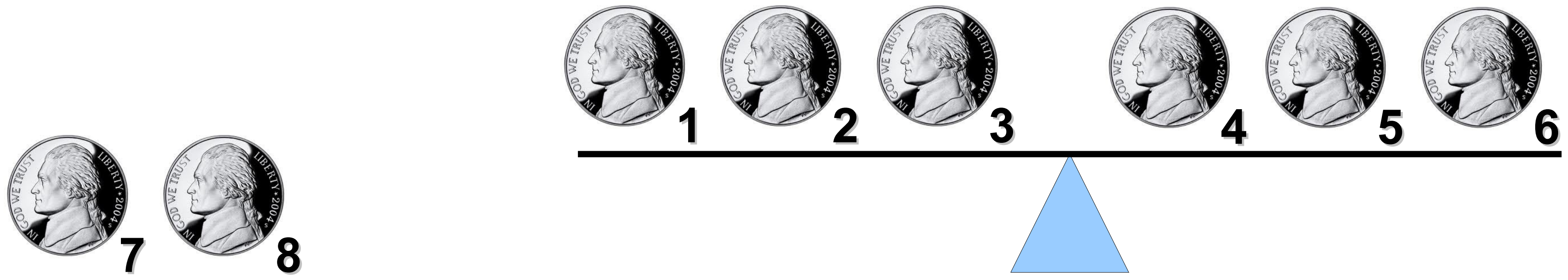
The Possibly Counterfeit Coin Problem



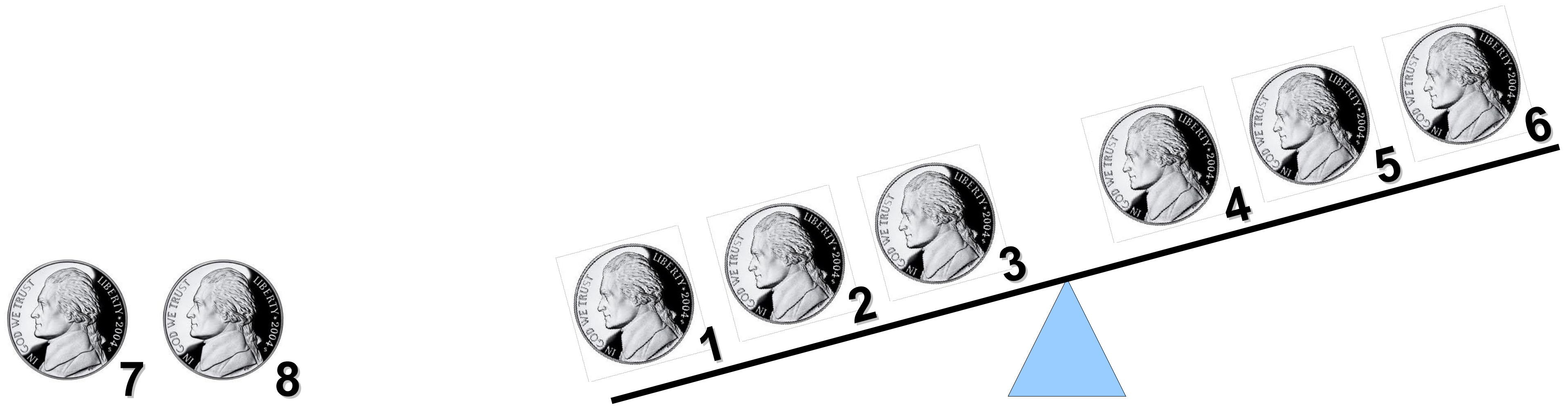
The Possibly Counterfeit Coin Problem



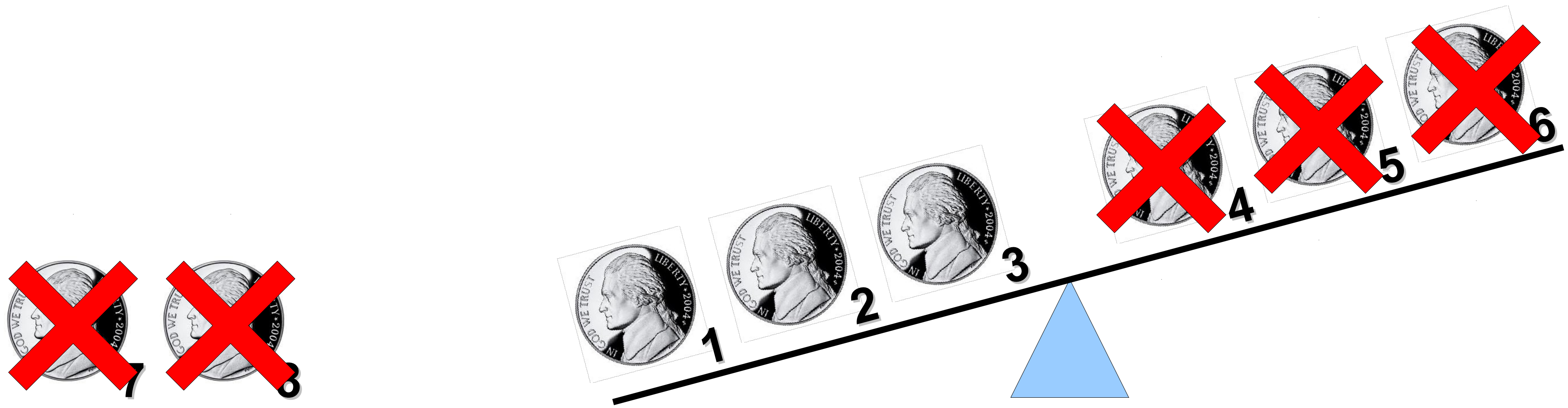
The Possibly Counterfeit Coin Problem



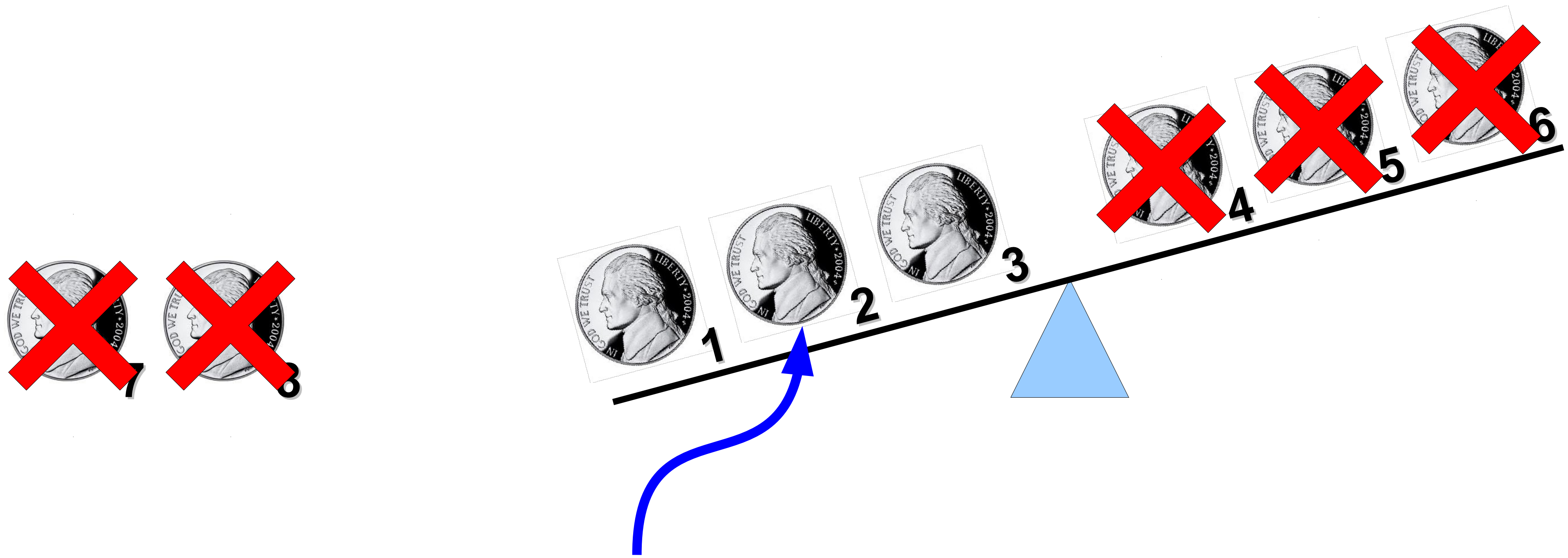
The Possibly Counterfeit Coin Problem



The Possibly Counterfeit Coin Problem

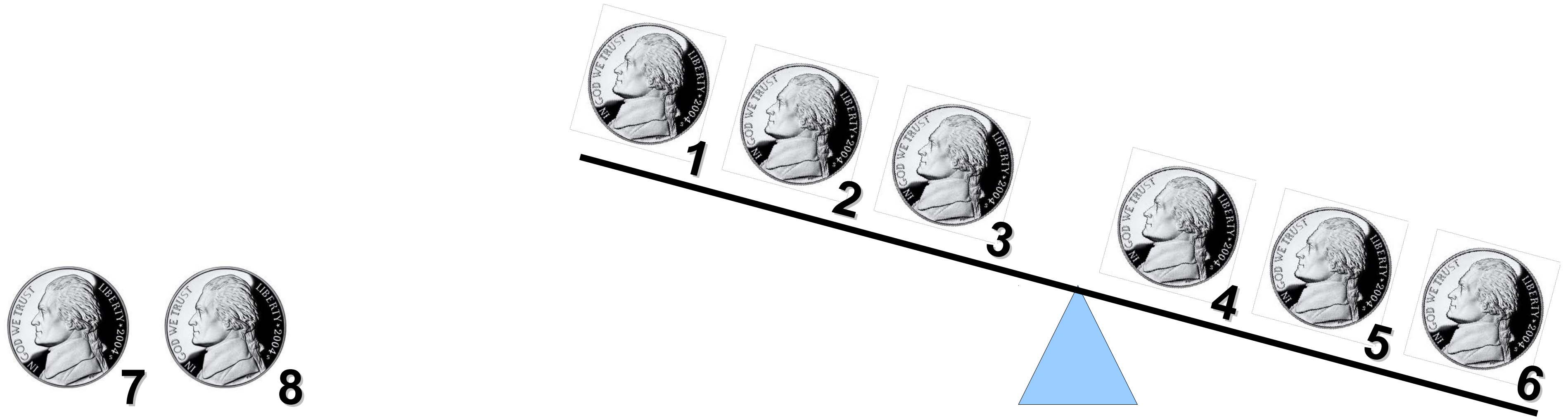


The Possibly Counterfeit Coin Problem

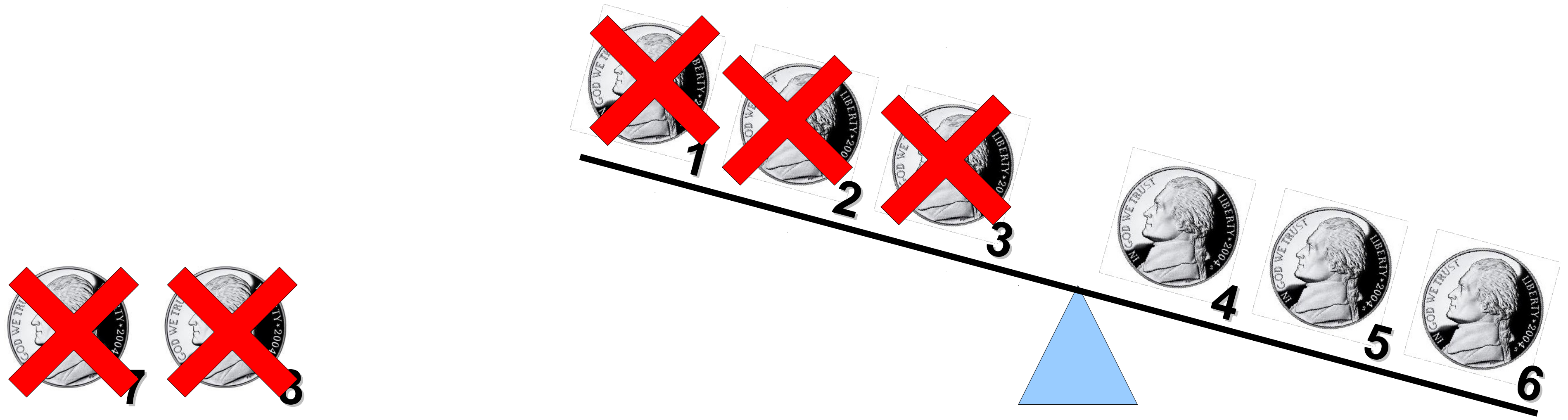


We know for a fact that one of these is heavier than the rest, so we can use our old algorithm!

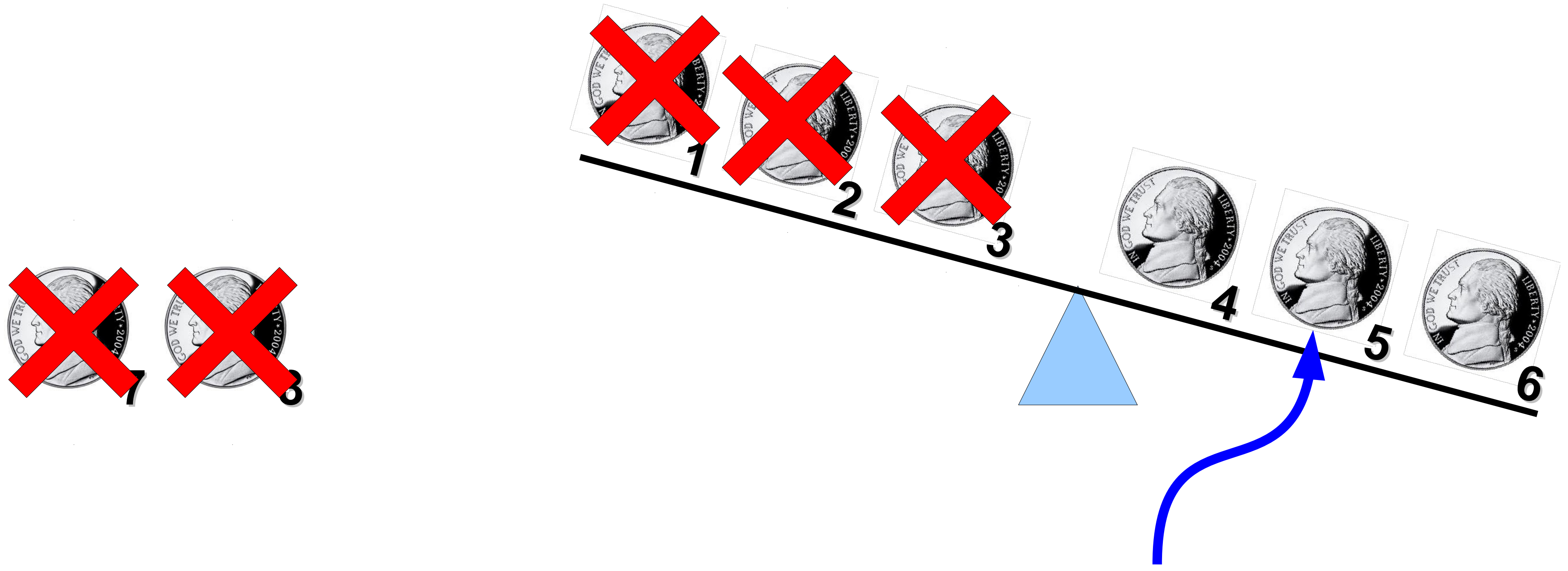
The Possibly Counterfeit Coin Problem



The Possibly Counterfeit Coin Problem

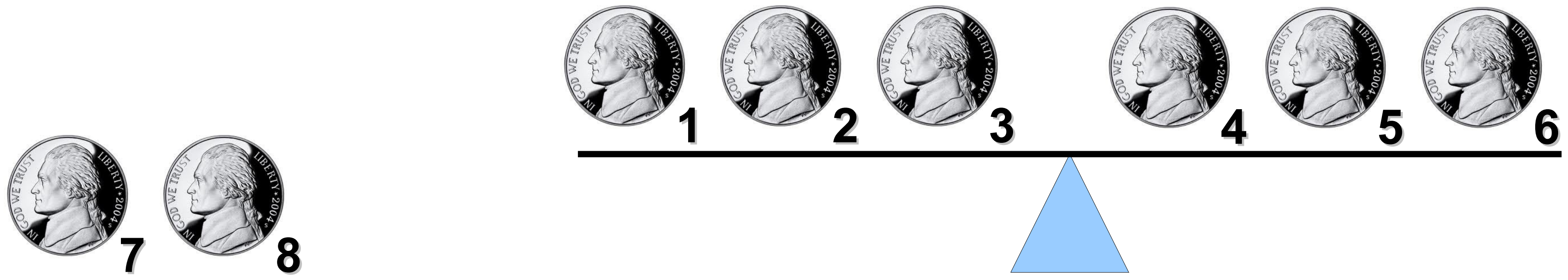


The Possibly Counterfeit Coin Problem

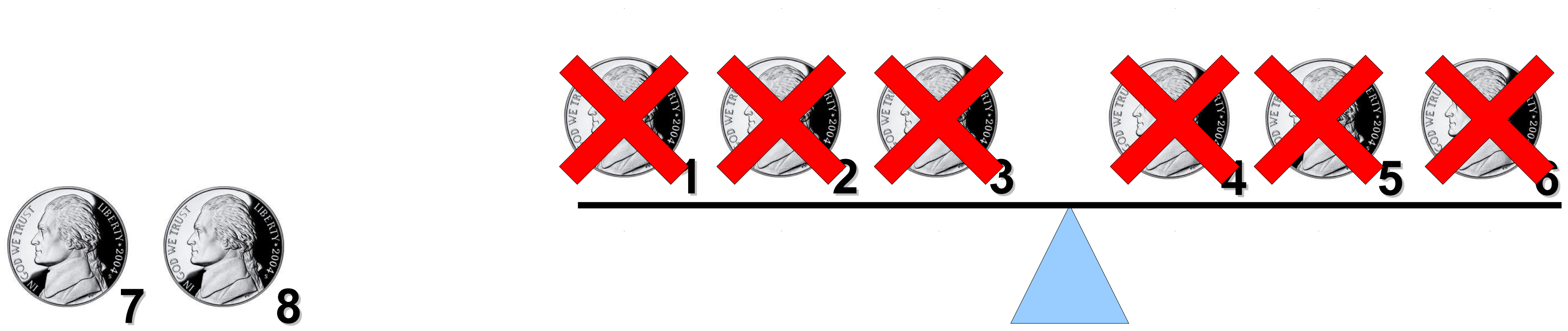


We know for a fact that one of these is heavier than the rest, so we can use our old algorithm!

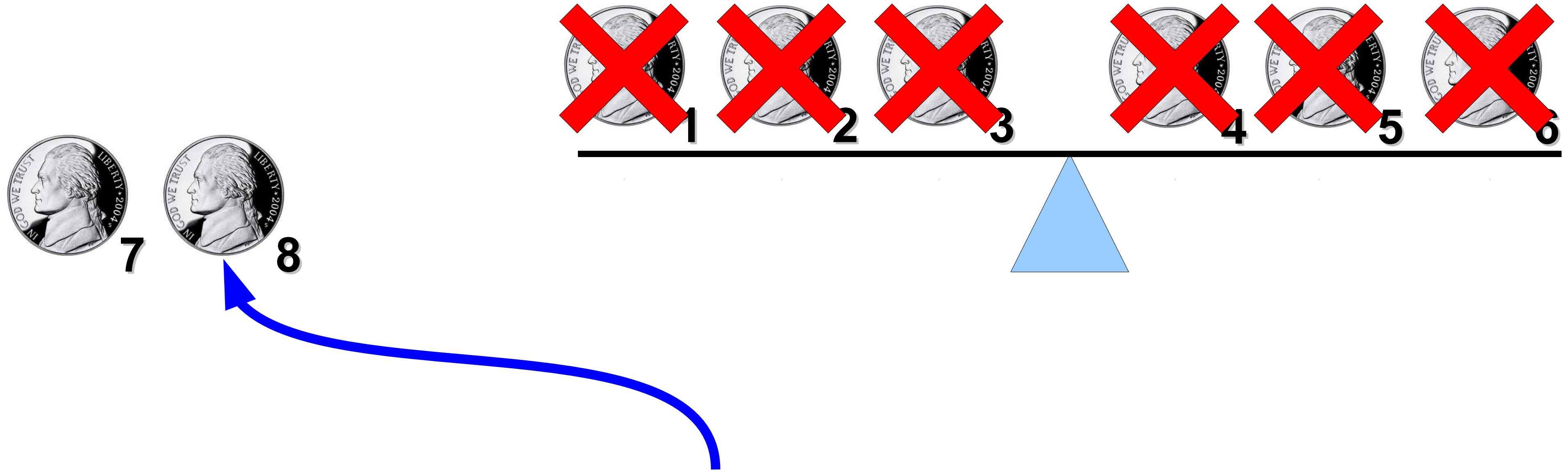
The Possibly Counterfeit Coin Problem



The Possibly Counterfeit Coin Problem



The Possibly Counterfeit Coin Problem



One of these two may be counterfeit, so we can use the case for just one weighing to check.

The Possibly Counterfeit Coin Problem

- With *n* weighings, we have a strategy for finding which of $3^n - 1$ coins, if any, is heavier.
- If *n* = 0, then we can check $3^0 - 1 = 0$ coins and determine they are all real.
- Otherwise:
    - Split the coins into groups of size 3^{n-1}, 3^{n-1}, and $3^{n-1} - 1$. Call them A, B, and C.
    - Weigh A vs. B.
    - If A or B is heavier than the other, one of the 3^{n-1} coins in it is counterfeit. We can find it in *n* – 1 weighings.
    - Otherwise, nothing in A or B is counterfeit. Recursively check the $3^{n-1} - 1$ coins using *n* – 1 weighings.

Is this solution optimal?

Using Our Model

- Let's use the same reasoning as before.
- How many different inputs are there if there are *k* coins?
  - **Answer:** $k + 1$: one for each coin, plus one for “no coin is counterfeit.”
- How many answer nodes are in an algorithm that makes *n* comparisons?
  - **Answer:** 3^n
- Solving $k + 1 = 3^n$, we get $k = 3^n – 1$.
- **Our algorithm has to be optimal!**

Suppose you have a set of coins. There may be a counterfeit coin among them (though there doesn't have to be). If there is a counterfeit, it is heaver than the rest of the coins.

If you have *n* weighings, what is the largest number of coins for which you can solve this problem?

Suppose you have a set of coins. There may be a counterfeit coin among them (though there doesn't have to be). If there is a counterfeit, it **can be either heavier or lighter** than the rest of the coins.

Find the counterfeit coin, if any, and tell whether it is heavier or lighter than the rest.

If you have *n* weighings, what is the largest number of coins for which you can solve this problem?

Decision Tree Reasoning

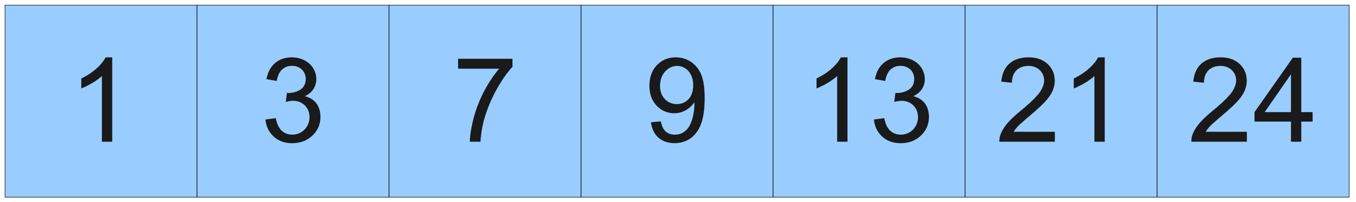
- Before we even try thinking up a solution, let's see if we can figure out a best-case scenario.
- How many different inputs are there with k coins?
  - **Answer**: $2k + 1$: Each of the k coins can be heavier or lighter, or they might all be the same.
- How many answer nodes are there in an algorithm that makes n comparisons?
  - **Answer**: 3^n, as before.
- Solving, we get that $2k + 1 = 3^n$, so $k = (3^n – 1) / 2$.

Complicated Counterfeit Coin Problem

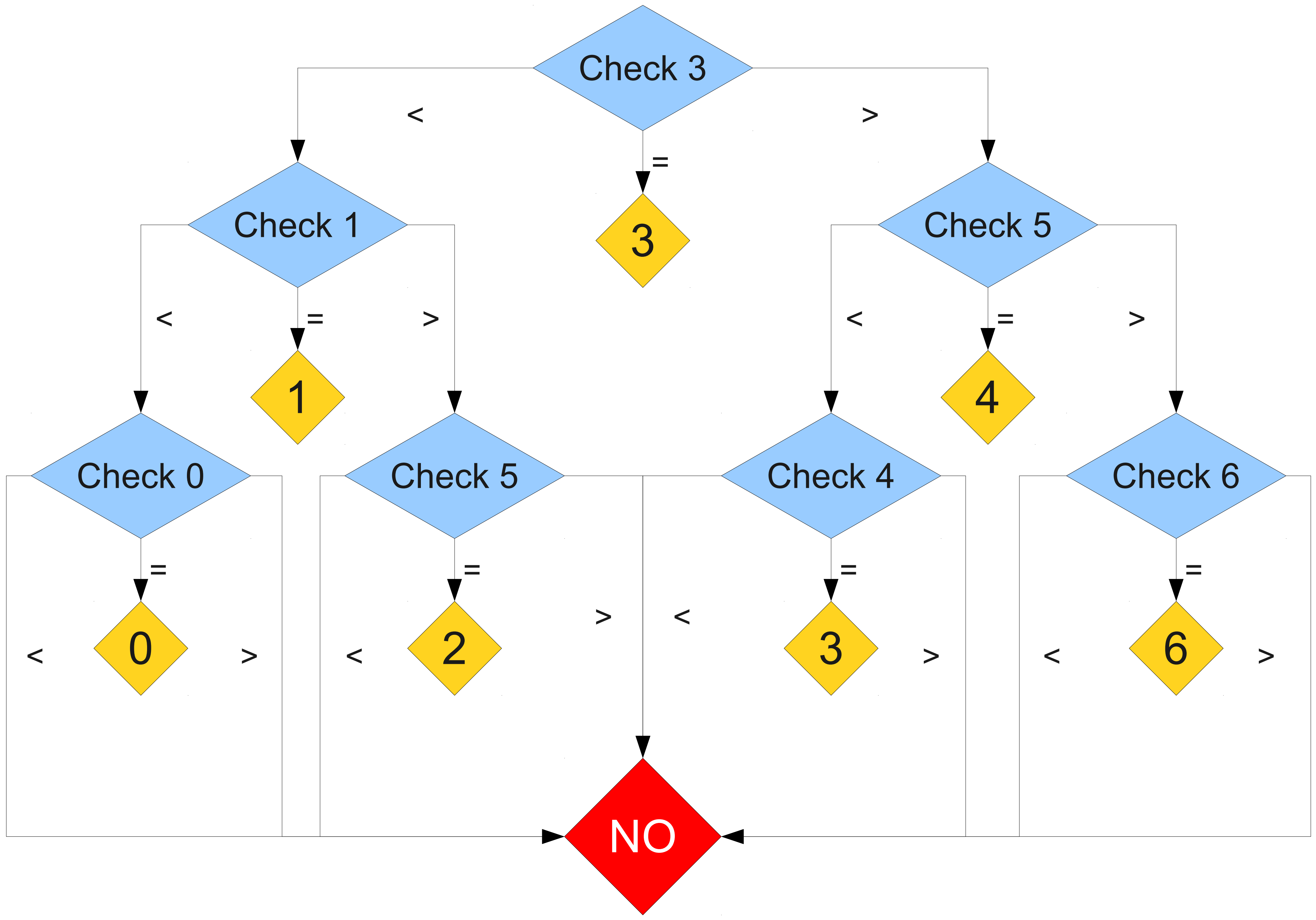
- We haven't even thought up a way to solve this problem yet, but we already have an upper bound on how many coins we can differentiate.
- It turns out that there is an algorithm that can achieve this upper bound, but it's fairly difficult to come up with.
- **Fun puzzle: Given $(3^n – 1) / 2$ coins, determine which (if any) is different, and whether it's heavier or lighter than the rest, in *n* weighings.**

More Advanced Applications

The Complexity of Searching



How efficiently can we search
a sorted array for a single value?


Check 3
<
=
>
3
Check 1
Check 5
<
=
>
1
<
=
>
4
Check 0
Check 5
Check 4
Check 6
=
=
=
=
<
0
>
<
2
>
<
3
>
<
6
>
NO

A Lower Bound on Searching

- Any **comparison-based** searching algorithm on a sorted array must make at most $\log_3 (n + 1)$ comparisons.
- Why?
  - We need there to be $n + 1$ answer nodes, one for each of the n positions in the array, and one to signal "not found."
  - By our previous result, the tree must have depth $\log_3 (n + 1)$ to achieve this result.
- This does **not** mean that it's possible to build an algorithm that makes this many comparisons; instead, it shows that no algorithm can do any better.

A Lower Bound on Sorting

Sorting

21	7	24	1	9	3	13

Sorting

1	3	7	9	13	21	24

Sorting Algorithms

- Suppose *S* is a totally-ordered set.
- Let $x_1, x_2, \ldots, x_n$ be *n* distinct elements drawn from *S*, ordered arbitrarily.
- **Goal**: Reorder the elements to put the sequence into sorted order.
- Perhaps the single most-studied problem in all computer science.

Comparison Sorting

- A **comparison-based sorting algorithm** is a sorting algorithm in which the only way to gain information about the elements being sorted is to compare them.
- Similar to the counterfeit coin problem – we can get the relative weights of the elements, but not their absolute weights.
  - The **relative ordering** of the inputs can be known, not the **absolute values.**
- Question: How many comparisons are required in order to sort *n* values?

Defining Our Inputs

x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2

x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1

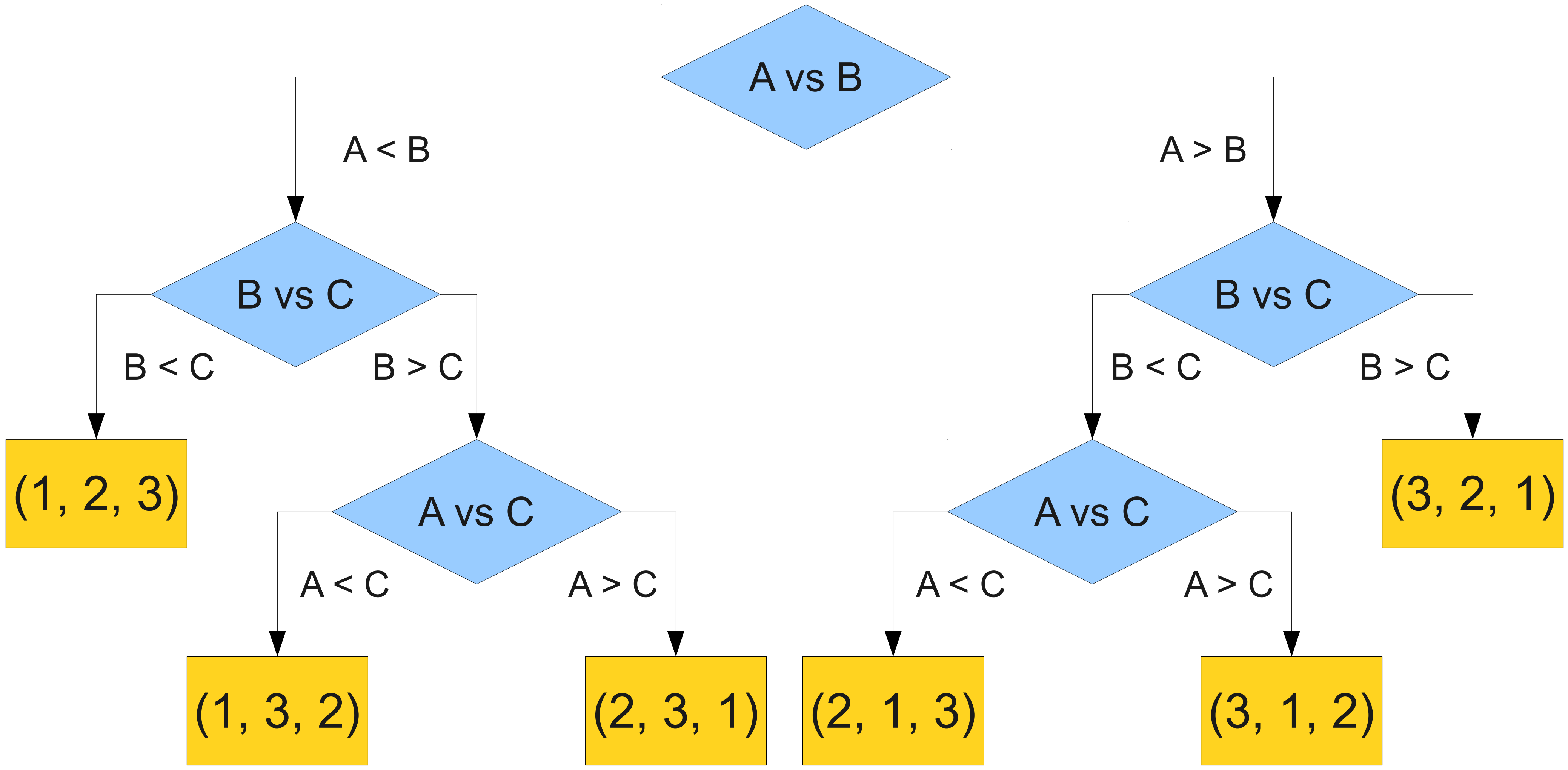
x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_3	x_4	x_2	x_1

x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

Defining Our Inputs

- Suppose we are sorting *n* distinct values.
- There are *n*! different permutations of those values.
- Consequently, there are *n*! different inputs, one for each permutation.

Defining Our Algorithmic Model



Defining Our Algorithmic Model

- Represent as a tree.
  - Internal nodes encode the comparisons being made.
  - Leaf nodes have to say which input permutation we received.
- Why does this work?
  - If we are just counting comparisons, it doesn't matter what the code looks like to do the comparison; it just matters what comparisons are made.
  - We don't need to see what rearrangements the algorithm actually does. If the algorithm can't identify what the input permutation is, then it can't be guaranteed that it it's correct.

The Analysis

- Using a similar proof to before, we can show that in a tree with k comparisons, there are at most 2^k answer nodes.
- There are $n!$ different inputs of length n, each of which must have a unique answer.
- Combined, we must have $2^k \geq n!$, or otherwise two different input permutations would reach the same answer (one gets sorted incorrectly).
- Thus $\boldsymbol{k \geq \log_2 n!}$. We have to do at least $\log_2 n!$ comparisons in the worst-case.

Stirling's Approximation

- A result called **Stirling's approximation** says that as n grows large, $\log_2 n! \approx n \log_2 n$
- Consequently, any comparison-based sorting algorithm must make (approximately) $n \log_2 n$ comparisons.
- Thus there cannot be a comparison-based sorting algorithm that does less than $\Theta(n \log_2 n)$ work making comparisons.
- Since some comparison-based sorting algorithms have this runtime (mergesort, heapsort, etc.), these algorithms are **asymptotically optimal**.

Why All This Matters

- We've spent the last few weeks exploring proof techniques and defining mathematical structures.
- These techniques make it possible to reason about fundamental questions in computing.
- In about a week, we'll begin exploring more elaborate models of computation using similar techniques.

Next Time

- **Propostional Logic**
  - How do we start formalizing our intuitions about mathematical truth?
  - How do we justify proofs by contradiction and contrapositive?