Let's look at a little example of a problem that becomes trivially easy to solve when we understand modular arithmetic:

Problem: Find **all values** x such that $(4 * x)\%11 = 2$

First we might ask how many solutions there could be? It's conceivable that there are none, one, more than one but a finite number, or infinitely many.

Suppose x is a solution. Then it's not hard to see that $x + 11$ is also a solution, since if
$(4 * x)\%11 = 2$
then $(4 * (x + 11))\%11 = (4 * x + 4 * 11)\%11 = (4 * x)\%11 = 2$ also.

In fact if x is a solution then values such as
$x - 77, x - 11, x + 22, x + 33, x + 55,$ and in general $x + k * 11 \quad \forall\ k$ are all solutions.

From this it follows that if there is at least one solution, then there are infinitely many.

It's also easy to show – though I didn't do this in class – that if there is a solution, then there must be a solution in $\mathbb{Z}_{11}$

Proof: Suppose x is a solution. Consider $r = x\%11$ By definition, $r \in \mathbb{Z}_{11}$. We know $x = k * 11 + r,$ so $r = x - k * 11.$ By the argument given above, r is also a solution.

So to find all solutions to the problem, we can start by finding the solution in $\mathbb{Z}_{11}$

$(4 * x)\%11 = 2$ is equivalent to $4 \otimes x = 2$ in $\mathbb{Z}_{11}$

Note that 4 is invertible in $\mathbb{Z}_{11}$ because $gcd(4, 11) = 1$

Multiplying both sides by 4^{-1} we get $4^{-1} \otimes 4 \otimes x = 4^{-1} \otimes 2$

But $4^{-1} \otimes 4 = 1$ so this reduces to $x = 4^{-1} \otimes 2$

Now we just need to know what 4^{-1} is

In $\mathbb{Z}_{11}$, $4^{-1} = 3$ ($4 * 3 = 12$, and $12\%11 = 1$)

So $x = 3 \otimes 2$, ie. $x = 6$

But remember that if x is a solution to $(4 * x)\%11 = 2$, then so is $x + 11 * k$ for all integer values of k

So our final answer is:

all integers of the form $6 + 11 * k$, where k is any integer

In that example finding 4^{-1} in $\mathbb{Z}_{11}$ was easy enough to do in my head. But what if the question were "Find all solutions to $(82 * x)\%100019 = 322$" It turns out that 100019 is prime (I looked it up) so we know that 82^{-1} exists in $\mathbb{Z}_{100019}$, but how do we find it?

The answer to this question is a bit outside the scope of this course at this point, but if you are interested you will find a good description of different methods at

https://en.wikipedia.org/wiki/Modular_multiplicative_inverse

and you will also find much good knowledge in the monumental 4-volume work "The Art of Computer Programming" by living legend Donald Knuth.

Our textbook discusses using Euclid's algorithm to find inverses. We won't be doing that.

If n is small, there is a simple method for finding a^{-1}.

We know we are looking for a value b such that $a \otimes_n b = 1$

But this is equivalent to saying $(a * b)\ \%\ n = 1$,
which is the same as $a * b = k * n + 1$

In other words, we are looking for a multiple of a that is 1 more than a multiple of n. Let's just list the multiples of a and see what we find!

For example, suppose we want to compute 3^{-1} in $\mathbb{Z}_7$

b	Multiples of 3	Multiples of 7
0	0	0
1	3	7
2	6	**14**
3	9	21
4	12	28
5	**15**	35

Whoa, there it is! $3 * 5 = 15$, and $15 = 14 + 1$. So $3 * 5 = 2 * 7 + 1$, which means $(3 * 5)\ \%\ 7 = 1$, which means $3 \otimes_7 5 = 1$, which means $3^{-1} = 5$ in $\mathbb{Z}_7$

Another example: what is 8^{-1} in $\mathbb{Z}_{11}$?

b	Multiples of 8	Multiples of 11
0	0	0
1	8	11
2	16	22
3	24	33
4	32	44
5	40	**55**
6	48	66
7	**56**	

So we see that $8^{-1} = 7$ in $\mathbb{Z}_{11}$

This method gets tedious when n is large, but it still works. This method really is the same as working out the appropriate row of the multiplication table, without applying the $\%\ n$ to every calculation.

Let's return to problems involving numbers that are small enough to work with easily.

Consider this problem : find all solutions to $(4 * x)\%9 = 8$

> We can apply the same technique as before: in $\mathbb{Z}_9$, $4^{-1} = 7$
>
> $4 \otimes x = 8$ in $\mathbb{Z}_9$ gives us $x = 7 \otimes 8$, so $x = 2$
>
> Thus the set of all solutions is $2 + k * 9$ for any integer k

Easy enough, but consider this one: find all solutions to $(6 * x)\%9 = 8$

6^{-1} does not exist in $\mathbb{Z}_9$ so we cannot apply the method we have used successfully so far. In fact, a bit of experimentation shows there is no solution to the equation $6 \otimes x = 8$ in $\mathbb{Z}_9$ and therefore no solution in general.

These problems all have the general form "Find all solutions to $(a * x)\%n = b$"

The first two examples are solvable because a and n are relatively prime, so a^{-1} exists in $\mathbb{Z}_n$. In the third example, $a = 6$ and $n = 9$, which are not relatively prime … and the problem has no solution.

But does this mean that we can **only** solve equations of the form $(a * x)\%n = b$ when a and n are relatively prime?

Alas, that would be too simple. Consider this example:

> Find all solutions to $(6 * x)\%9 = 3$.

This looks like it might suffer the same fate as the last example since we know 6^{-1} does not exist in $\mathbb{Z}_9$. But in fact, the equation $6 \otimes x = 3$ does have solutions in $\mathbb{Z}_9$: $x = 2, x = 5$ and $x = 8$ are all solutions. We can verify this by looking at the multiplication table for $\mathbb{Z}_9$

Thus any integer of the form

$$2 + k * 9, \quad 5 + k * 9 \quad \text{or} \quad 8 + k * 9$$

is a solution to the original problem.

A bit of exploration will reveal that we can find solutions to
$(6 * x)\%9 = b$ only when $b = 0$ or $b = 3$ or $b = 6$

What is the magical property that makes these problems solvable? It turns out that we need a very specific relationship between all three numbers that define the problem.

In class I gave this summary:

We can find solutions to $(a * x)\%n = b$ **(where** $b \in \mathbb{Z}_n$**)**
if and only if b **is a multiple of** $gcd(a, n)$

This is not very hard to prove and I recommend it as an exercise if you are intrigued by modular arithmetic. Or … see this proof if you get stumped.

In class we talked about variations on this type of problem. We have now learned how to determine the existence of solutions to problems such as

Find all solutions to $(5 * x)\%7 = 4$
(there are infinitely many because 5 and 7 are relatively prime so $gcd(5, 7) = 1$, and 4 is obviously a multiple of 1)

and

Find all solutions to $(9 * x)\%12 = 7$
(there are none because 7 is not a multiple of $gcd(9, 12)$)

and

Find all solutions to $(9 * x)\%12 = 6$
(there are infinitely many because 6 is a multiple of $gcd(9, 12)$)

When solutions exist, we know how to find them.

Here's another type of question:

For what values of b can the following equation be solved:
$(12 * x) \ \% \ 30 = b$

We know this question can be solved if and only if $b \in \mathbb{Z}_{30}$ and b is a multiple of $gcd(12, 30)$. Now $gcd(12, 30) = 6$, and the multiples of 6 that are elements of $\mathbb{Z}_{30}$ are $\{0, 6, 12, 18, 24\}$

So $(12 * x) \ \% \ 30 = b$ has solutions if and only if $b \in \{0, 6, 12, 18, 24\}$

Exponentiation in Modular Arithmetic

Here's an interesting question: what is the value of $(4^{156})\%7$?

Well you may not find it interesting, but hopefully you will agree that it is challenging. We could certainly just work out the value of 4^{156} and compute its remainder when divided by 7 … but that isn't really feasible because 4^{156} is a ridiculously large number and our computing hardware usually places a limit on the largest integer we can store.

Fortunately we can apply some of our smarts about modular arithmetic and avoid the integer overflow problem.

Remember this fact about modular arithmetic: $(a * b)\%n = (a\%n * b\%n)\%n$

In other words, we can carry the "$\%n$" operation inside the brackets without changing the result. We will use this principle heavily in CISC-235 when we look at the hash-table data structure.

So how does that help us with $(4^{156})\%7$?

$$\begin{aligned}(4^{156})\%7 &= (4 * 4 * \cdots * 4)\%7 \\ &= (\ldots(((4 * 4)\%7) * 4)\%7) * 4)\%7) * \cdots * 4)\%7\end{aligned}$$

In other words, we could start with 4, then repeatedly multiply by 4 and apply $\%7$ to the result until we have multiplied 4 by itself 155 times. In pseudo-code it would look like this:

```
answer = 4
for c = 1 to 155:
      answer = (answer * 4) % 7
print answer
```

This approach completely avoids large numbers – the largest we will ever deal with is 24. But we are doing a lot of work, and there are much better methods.

Consider this:

$$\begin{aligned}(4^{156})\%7 &= (4 * 4 * \cdots * 4)\%7\\ &= ((4*4)\%7 * (4*4)\%7 \cdots * (4*4)\%7)\%7\\ &= (16\%7 * 16\%7 \cdots * 16\%7)\%7\\ &= (2 * 2 * \cdots * 2)\%7\\ &= (2^{78})\%7\end{aligned}$$

It is (2^{78}) because we started with 156 4's and we combined them into 78 pairs.

Continuing …

$$\begin{aligned}(2^{78})\%7 &= (2 * 2 * \cdots * 2)\%7\\ &= ((2*2*2)\%7 * (2*2*2)\%7 \cdots * (2*2*2)\%7)\%7\\ &= (8\%7 * 8\%7 \cdots * 8\%7)\%7\\ &= (1 * 1 * \cdots * 1)\%7\\ &= (1^{26})\%7\end{aligned}$$

where it is 1^{26} because we started with 78 2's and we combined them into 26 trios

But $1^{26} = 1$ and $1\%7 = 1$ …. so our final answer is

$$\begin{aligned}(4^{156})\%7 &= (2^{78})\%7\\ &= (1^{26})\%7\\ &= 1\%7\\ &= 1\end{aligned}$$

This method arrived at the answer with very little calculation but quite a bit of brain work. We were "lucky" that 156 divided by 2, and "lucky" that 78 divided by 3, and "lucky" that $(2*2*2)\%7 = 1$. If we tried to turn this into an algorithm we would have to take care of situations where we are not so lucky – it is doable though.

Tomorrow we will look at two more methods that significantly reduce the number of computations without requiring an excessive amount of analysis. I'm including the notes on the first method here.

Solving Exponentials Using Repeated Squaring

Consider the problem of computing $(19^{54})\%8$?

The first thing we can do is use our rule that $(a*b)\%n = (a\%n * b\%n)\%n$

Since 19^{54} is just the product of 19 with itself over and over, we can write

$$(19^{54})\%8 = ((19\%8)^{54})\%8 = (3^{54})\%8$$

which is looking better already.

As we have seen, we could use something like this

```
x = 3
for p = 2 to 54:
     x = (x*3) % 8
print x
```

But we can do better! Here's how:

We can write 3^{54} as $(3^{27})^2$

Why would we do this???? Well, 3^{27} is certainly easier to compute than 3^{54} and once we have 3^{27}, we can square it to get 3^{54} with one more multiplication.

We can't reduce 3^{27} quite as easily because 27 is odd … but we can "extract" one 3 and write 3^{27} as $3*(3^{13})^2$

We can write 3^{13} as $3*(3^6)^2$

We can write 3^6 as $(3^3)^2$

We can write 3^3 as $3*3^2$

Putting these all together, we get

$$
\begin{aligned}
3^{54} &= (3^{27})^2 \\
&= (3 * (3^{13})^2)^2 \\
&= (3 * (3 * (3^6)^2)^2)^2 \\
&= (3 * (3 * ((3^3)^2)^2)^2)^2 \\
&= (3 * (3 * ((3 * 3^2)^2)^2)^2)^2
\end{aligned}
$$

which we can compute by starting at the middle and working outwards. Recall that we are doing all of this $\%8$, so we can apply $\%8$ after each operation to keep the values small.

Algorithmically the process looks like this:

```
x = 3
x = (3 * x^2 ) %8        this gives us 3^3  %8
x = (x^2) %8             this gives us 3^6  %8
x = (3 * x^2) %8         this gives us 3^13 %8
x = (3 * x^2) %8         this gives us 3^27 %8
x = (x^2) %8             this gives us 3^54 %8
```

This uses a grand total of 8 multiplications (counting each squaring operation as a multiplication) and 5 "mod" operations …. which is a huge improvement over the loop that executed 53 times.

If you are curious about the actual answer to the question (although personally I find the process much more interesting than the answer – sort of "the journey is more important than the destination" thinking) we can easily work it out, line by line

$$
\begin{aligned}
&x = 3 \\
&x = (3 * x^2)\%8 = 27\%8 = 3 \\
&x = (x^2)\%8 = 9\%8 = 1 \\
&x = (3 * x^2)\%8 = 3\%8 = 3 \\
&x = (3 * x^2)\%8 = 27\%8 = 3 \\
&x = (x^2)\%8 = 9\%8 = 1
\end{aligned}
$$

So $(19^{54})\%8 = 1$

Notice that there is no mystery about the decomposition process – if the exponent is even we simply divide it by 2, and if it is odd we "extract" one copy of the value and divide the remaining exponent by 2

Another quick example: what is $47^{485}\%13$?

First we reduce the 47 by applying $\%13$ to it, changing the problem to $8^{485}\%13$. Then we go to work on the exponent.

485 = 1+ 2*242
242 = 2*121
121 = 1 + 2*60
60 = 2*30
30= 2*15
15 = 1+2*7
7 = 1+2*3
3 = 1+2*1

So our computation looks like this:

```
x = 8
x = (8 * x^2) %13              8^3
x = (8 * x^2) %13              8^7
x = (8 * x^2) %13              8^15
x = (x^2) %13                  8^30
x = (x^2) %13                  8^60
x = (8 * x^2) %13              8^121
x = (x^2) %13                  8^242
x = (8 * x^2) %13              8^485
```

In this method we repeatedly square the previous value of x to get the next value of x, and so this solution method is called ***repeated squaring***, or ***the repeated squares method*** (mathematicians are so creative about naming things!)

But in both examples we saw that we occasionally have to throw in another copy of a - basically whenever we need to get an odd exponent. Wouldn't it be wonderful if there were some way to know in advance when we need to do that?

Fear not, citizens … there is a way. To illustrate it, let's look at the first problem again. Let's create a binary string from the lines of computation by writing down "1" if we introduce a 3 on that line, and "0" if we don't. The result is 110110

Now let's represent the exponent we are trying to achieve, in binary notation. $54 = 110110$ in base 2 notation … and it's the same binary string!!!!!

For the second example, the binary string formed from "1" when we bring in an 8, and 0 when we just square is 111100101 … and the base 2 representation of 485 is 111100101

(The proof of this remarkable correspondence is not difficult. Try it as an exercise.)

The net result is that we can construct an algorithm that computes $(a^b)\%n$ for any positive integers a, b and n:

```
start with $x = a$
repeatedly compute $x = (x * x * a^i)\%n$      where $i$ is the next bit of $n$
```

(Note that the first line "uses up" the first bit of n so the "repeatedly" step begins with the second bit of n.)

The use of a^i above is just a clever trick to make the algorithm concise. When $i = 1$, this results in multiplying by a ... and when $i = 0$ it results in multiplying by 1. If we were actually coding this algorithm, we could just as easily use an `if` statement to decide whether to multiply by b or not on each iteration.

Example: let's use this method to compute $(4^{156})\%7$

156 in binary notation = 10011100

$$
\begin{aligned}
&a = 4\\
&a = (4 * 4)\%7 = 16\%7 = 2\\
&a = (2 * 2)\%7 = 4\\
&a = (4 * 4 * 4)\%7 = 64\%7 = 1\\
&a = (1 * 1 * 4)\%7 = 4\\
&a = (4 * 4 * 4)\%7 = 1\\
&a = (1 * 1)\%7 = 1\\
&a = (1 * 1)\%7 = 1
\end{aligned}
$$

which is our answer.

Exercises:

1. Compute $(39^{83})\%4$

2. Compute $(612^{12})\%11$

3. Compute $(16^{103})\%12$