Modular Arithmetic → wrap around result of arithmetic operation or conversion on integer type.

- ➢ Next number after the largest in an integer type os the smallest
  - ✓ (byte) 128 == (byte) (127+1) == (byte) -128
- ➢ Result of some arithmetic subexpression with Type T, an n-bit integer type,
  - ✓ x = the real (mathematical) value of computation
  - ✓ x' = x modulo 2^n → x' = x in the range of T
  - ✓ i.e. x - x' is a multiple of 2^n → x' = x - a multiple of 2^n
- ➢ a ≡ b (mod n) → a - b = kn for some integer k ~ n = 2^(# of bits) && b = a - kn
- ➢ Binary operation a mod n → b such that a ≡ b (mod n) and 0 <= b < n for n > 0
- ➢ Different from % operation
- ➢ Various facts (a' = a mod n)
  - ✓ a'' = a'
  - ✓ a' + b'' = (a' + b)' = a + b'
  - ✓ (a' - b')' = (a' + (-b)')' = (a - b)'
  - ✓ (a' x b')' = a' x b' = a x b'
  - ✓ (a^k)' = ((a')^k)' = (a x (a^(k-1))')' for k > 0

Modular Arithmetic and Bits

- ➢ Bit n, counting from 0 at the right, corresponds to 2^n
- ➢ Bits to the left of the vertical bars represents multiples of 2^(# of bits)
- ➢ Arithmetic modulo 2^(# of bits) → throwing away left of the vertical bar
- ➢ Representation for -1 → bits of 1s

Bit twiddling

- ➢ Java allows handling integer types as sequences of bits w/o conversion
- ➢ Operations
  a. Mask (&) → AND operation
  b. Set (|) → OR operation
  c. Flip (^) → XOR operation ~ true iff only one of bits is true
  d. Flip all (~) → NOT operation
- ➢ Shifting
  a. Left (<<) → append 0 at the left ~ equivalent to multiplying 2^k
  b. Arithmetic Right (>>) → append 1 at the right ~ equivalent to dividing low(2^k) (i.e. rounded down)
  c. Logical Right (>>>) → append 0 at the right ~ equivalent to ?????

Asymptotic analysis → Input bound

- ➢ Cost Measures (Time)
    a. Dynamic statement counts of # of times statements are executed
    b. Symbolic execution times → formulas for execution times as functions of input size ~ let us see the shape of the cost function
- ➢ Asymptotic behavior → growth of the function f(n) as n gets large
- ➢ Approximation ~ Behavior on small inputs and constant factors are pointless
- ➢ Asymptotic time unchanged by the constant factor
- ➢ No input → Works in a constant time
- ➢ Other than inputs are considered as constant

Order notation → produce families of functions w/ similarly behaved magnitudes

- ➢ Upper limits → Big Oh (O(g)) - Express the upper bound of an algorithm's running time. ~ Measures the worst case (largest possible time)
- ➢ Lower limits → Omega (Ω(g)) → Express the lower bound of an algorithm's running time. ~ Measures the best case (smallest possible time)
- ➢ θ(g) = O(g) ∩ Ω(g) → set of functions bracketed in magnitude by two members of g's family ~ f is θ(g(n)) if g is an accurate characterization of f for large n
- ➢ Theta bound → can be scaled so is both an upper and a lower bound of function
- ➢ Use order notation to specify bounds of functions
- ➢ At most N operations ~ Worst case time is N tests → O(N) &&O(N^2), since N ∈ O(N^2)
- ➢ Average case analysis ~ calculate the expected time spent on a randomly chosen input - involves probabilistic arguments and often require assumptions about the distribution of inputs

Bubble sort → sort a list by repeatedly looping through the list and swapping adjacent items if they are out of order, until the entire sorting is complete.

Θ bound is a tight bound and applies when a function has the same Ω and O bounds. For example, consider searching an array for a specific element. In the best case, we'll find the element immediately, so the best case will be θ(1). Meanwhile, the worst case will be θ(n) since we're only considering inputs that will result in the worst case. As you can see, the best and worst cases have θ bounds, but the runtime of the function as a whole would be Ω(1) and O(n).

For mathematical functions, there aren't any notion of "best" or "worst" case inputs. The input is just x: no subset of inputs x can make this function behave differently

Theta bound is a tight bound. All functions with higher or equal degree than the function of Theta bound can be the valid function of the Upper Bound (Big Oh) and all functions with lower or equal degree can be the valid function of the Lower Bound (Omega) ~ For f(x) = x^2, Ω(1), Ω(log x), Ω(x) can be considered as simple lower asymptotic bounds

Sometimes we can't find a tight theta bound on a function's runtime. So, what we do is find the lower bound and upper bound of the function's runtime by Best and worst-case analysis.

Math.pow(2,i) $\rightarrow i \in N$ where $1 \leq i \leq 31$. Otherwise, it's trivially true that none of them are actually equivalent to Math.pow(2, i) for 32-bit integers.

The worst and the best case runtime includes all bounds, not just Θ bounds, but also the O and Ω bounds. The "worst case" doesn't have to be unique. It can be the same as the "best case", if the runtime can be tightly bounded.