BigInts

By Ruan Schoeman Gr 9, 2nd training camp 2021

BigInts

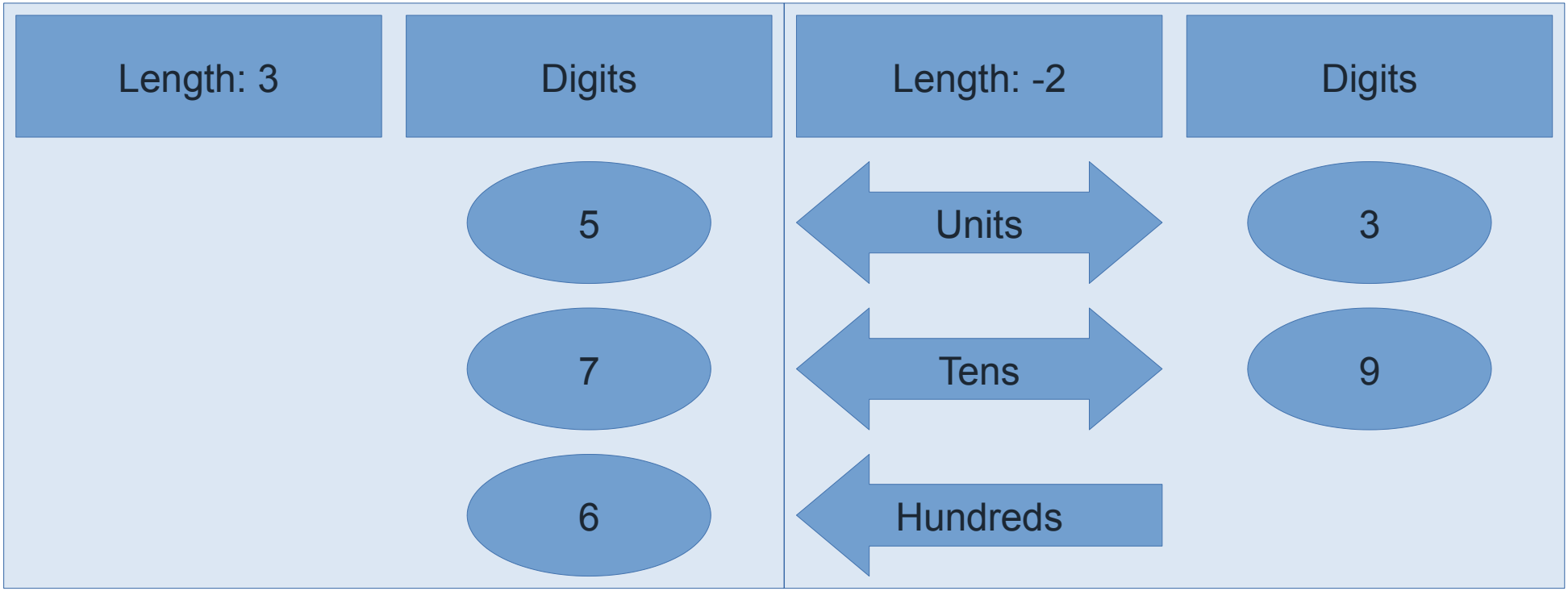
A “big integer” (or BigInt for short, also called an arbitrary-sized integer) is an integer that can theoretically be any size, though it is actually limited by the available memory.
It can be useful when you want to work with really large numbers, like computing the factorial of a number (34! is the largest factorial that will fit into a 128-bit integer), or when you don’t know how large the input to your program will be.

Implementation

- The way I have chosen to implement a BigInt is to store two values:
  - The length of the integer, which is negative for negative numbers
  - An array of digits
- I also store the least significant digit first, as this simplifies arythmatic.
- I use 32-bit signed integers for the digits, but store only use 30 of those bits, for easier handling of overflow during arythmatic.

Implementation: Storage

- If I wanted to store 675 and -93 in a base-10 BigInt it would look something like this:



Implementation: Addition algorithm

- If a is negative and b is positive return b + a
  - (So that if one number < 0, it is the second number)
- If a == -b, then return 0, because x – x = 0
- If a is 0, return b; If b is 0, return a;
- If both numbers are negative, return ***-(-a + -b)***
- If both numbers are positive, then
  - Allocate a list, one digit longer than the largest number, to handle overflow
  - Then for every ***i*** in the range ***[0, # digits in smallest number)*** add ***a.digits[i] + b.digits[i]*** to ***list[i]***
    - If ***list[i]*** >= the base of the BigInt, then set ***list[i+1]*** to one, and set ***list[i]*** to ***list[i] % base***

Implementation: Addition algorithm

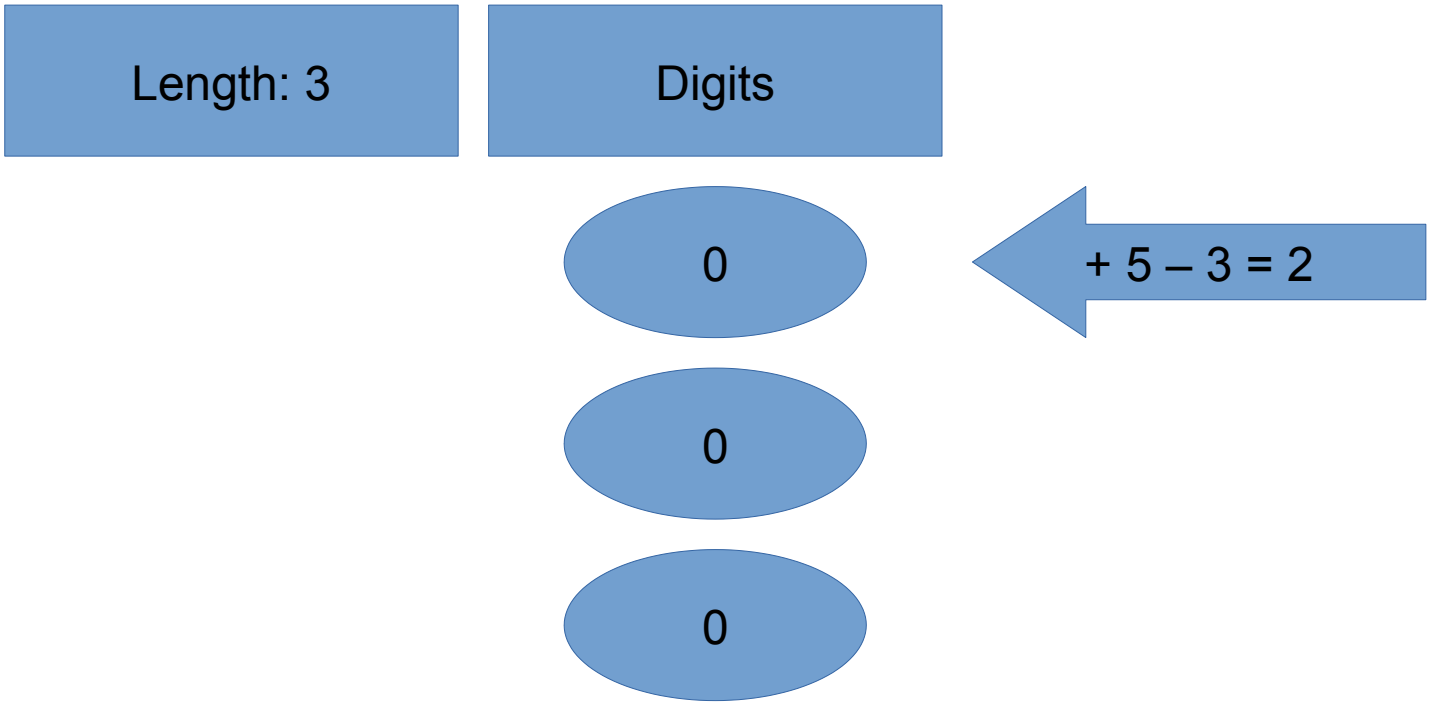
- Next, for every ***i*** in the range ***[# digits in smallest number, # of digits in the largest number)*** add ***larger.digits[i]*** to ***list[i]***
  - If ***list[i]*** >= the base of the BigInt, then set ***list[i+1]*** to one, and set ***list[i]*** to ***list[i] % base***
- While ***list[len-1] == 0***, subtract one from ***len***
  - *(avoid trailing zeroes)*
- return the BigInt constructed from the list
- If one number is negative
- If the absolute value of the negative number is larger than the absolute value of the positive number then return ***-(-neg + -pos)***
- Allocate a list, as long as the largest number

Implementation: Addition algorithm

- Then for every ***i*** in the range ***[0, # digits in b)*** add ***a.digits[i] - b.digits[i]*** to ***list[i]***
  - If ***list[i]*** < 0, then set ***list[i+1]*** to negative one, and add the base of the BigInt to ***list[i]***
- Next, for every ***i*** in the range ***[# digits in b, # of digits in a)*** add ***a.digits[i]*** to ***list[i]***
  - If ***list[i]*** < 0, then set ***list[i+1]*** to negative one, and add the base of the BigInt to ***list[i]***
- While ***list[len-1] == 0***, subtract one from ***len***
  - *(avoid trailing zeroes)*
- return the BigInt constructed from the list

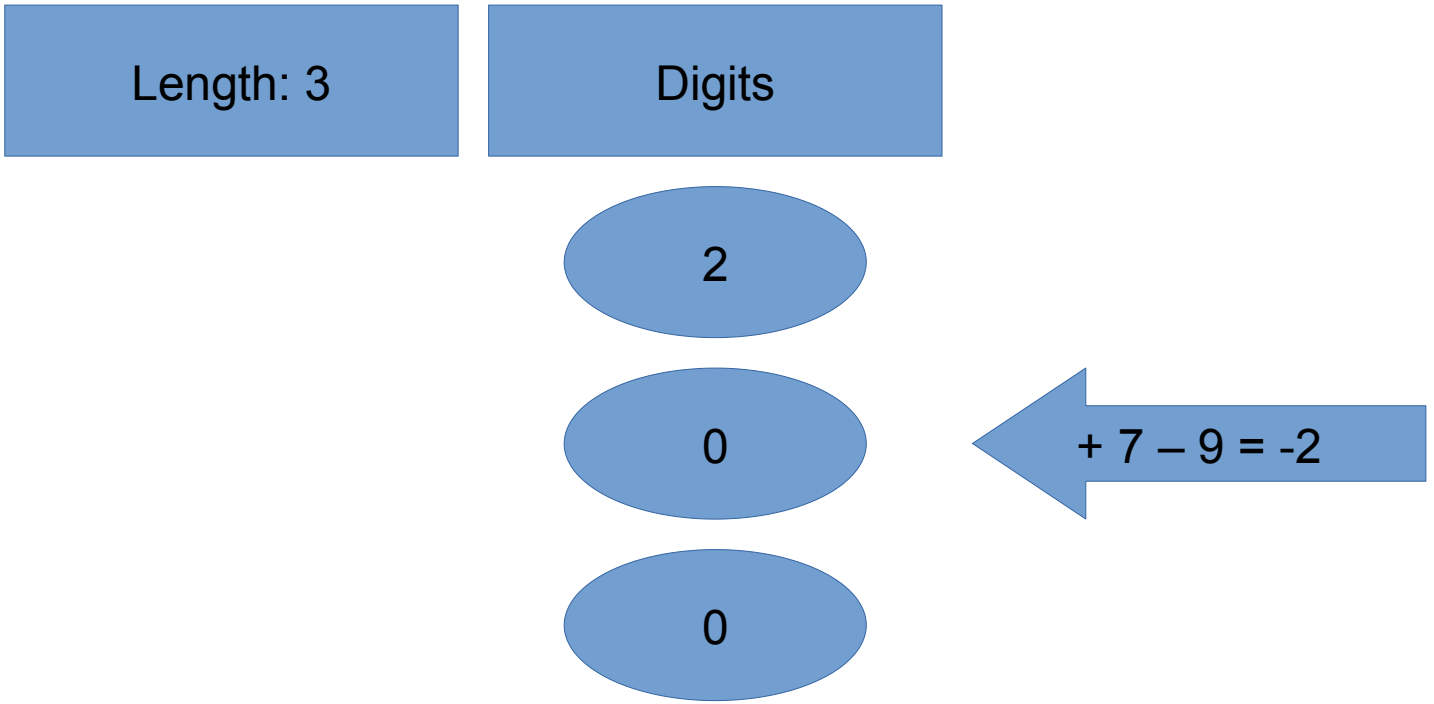
Addition Example

- Now to add 675 and -93 together:



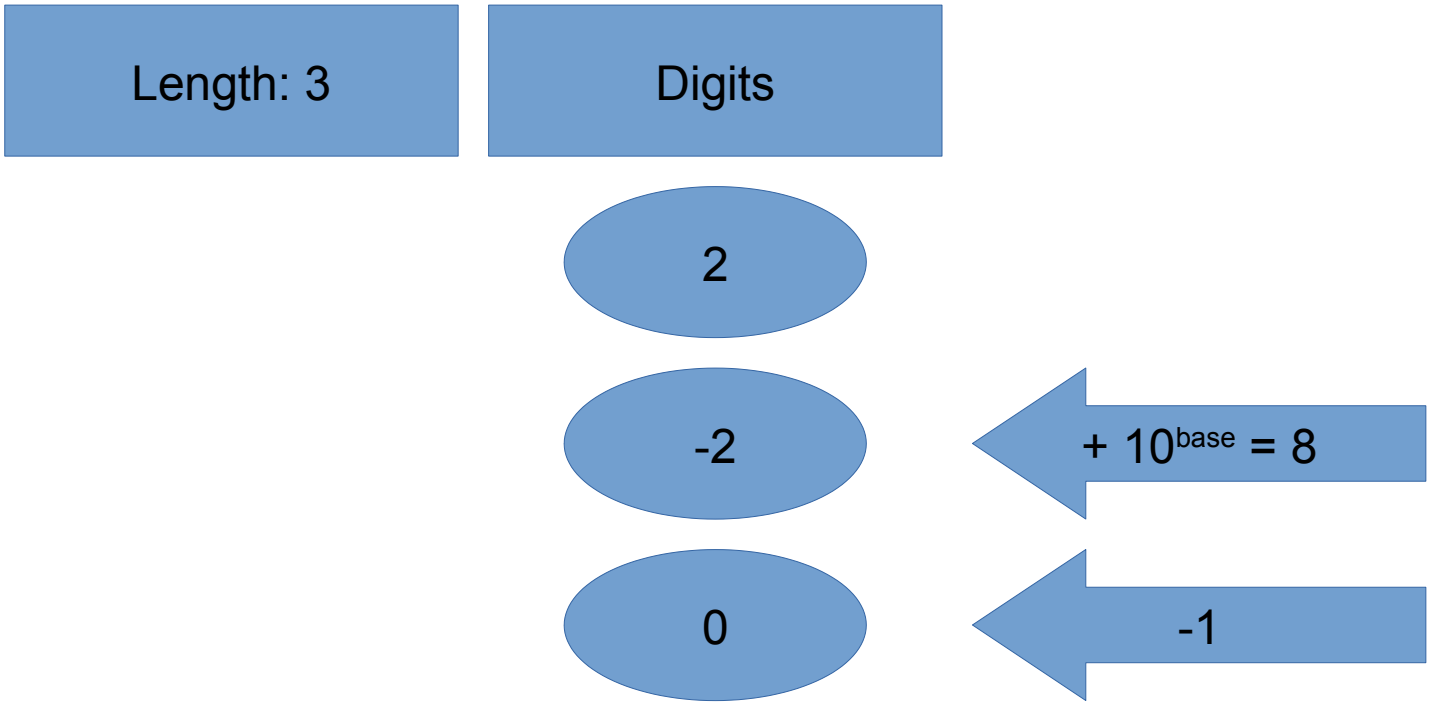
Addition Example

- Now to add 675 and -93 together:



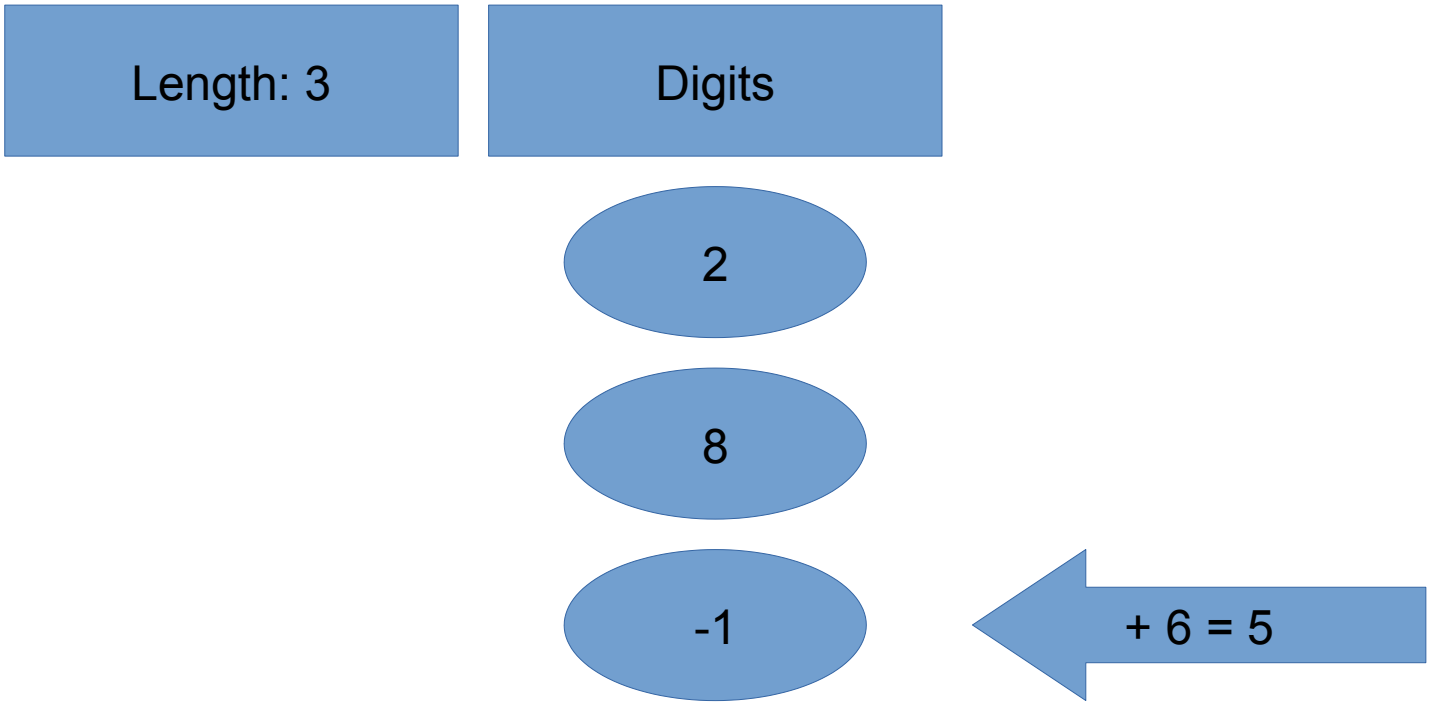
Addition Example

- Now to add 675 and -93 together:



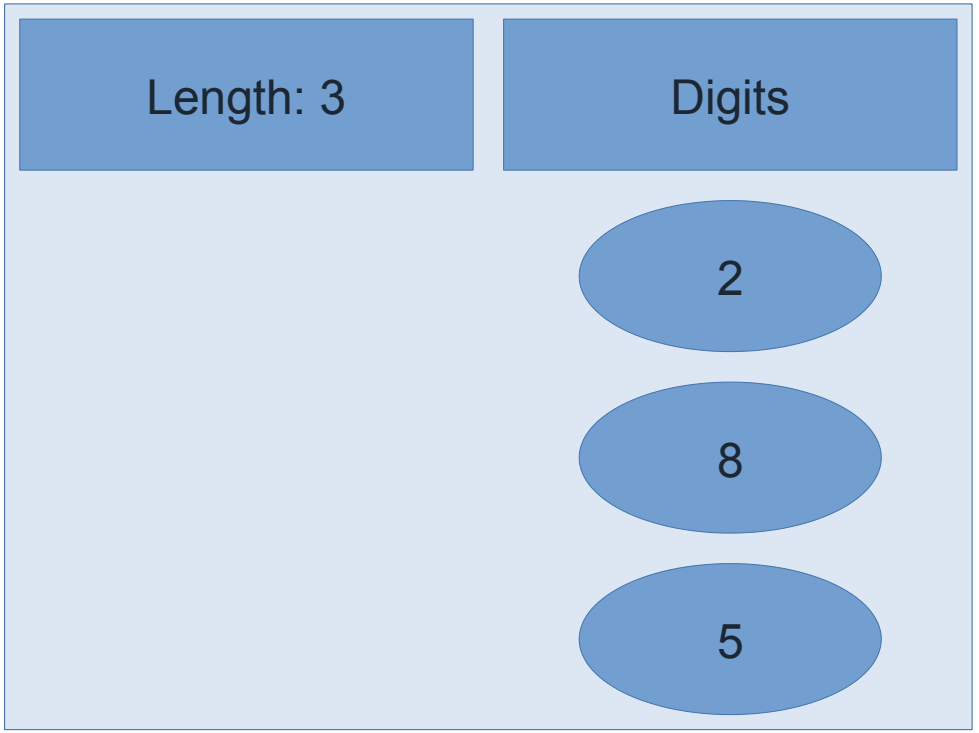
Addition Example

- Now to add 675 and -93 together:



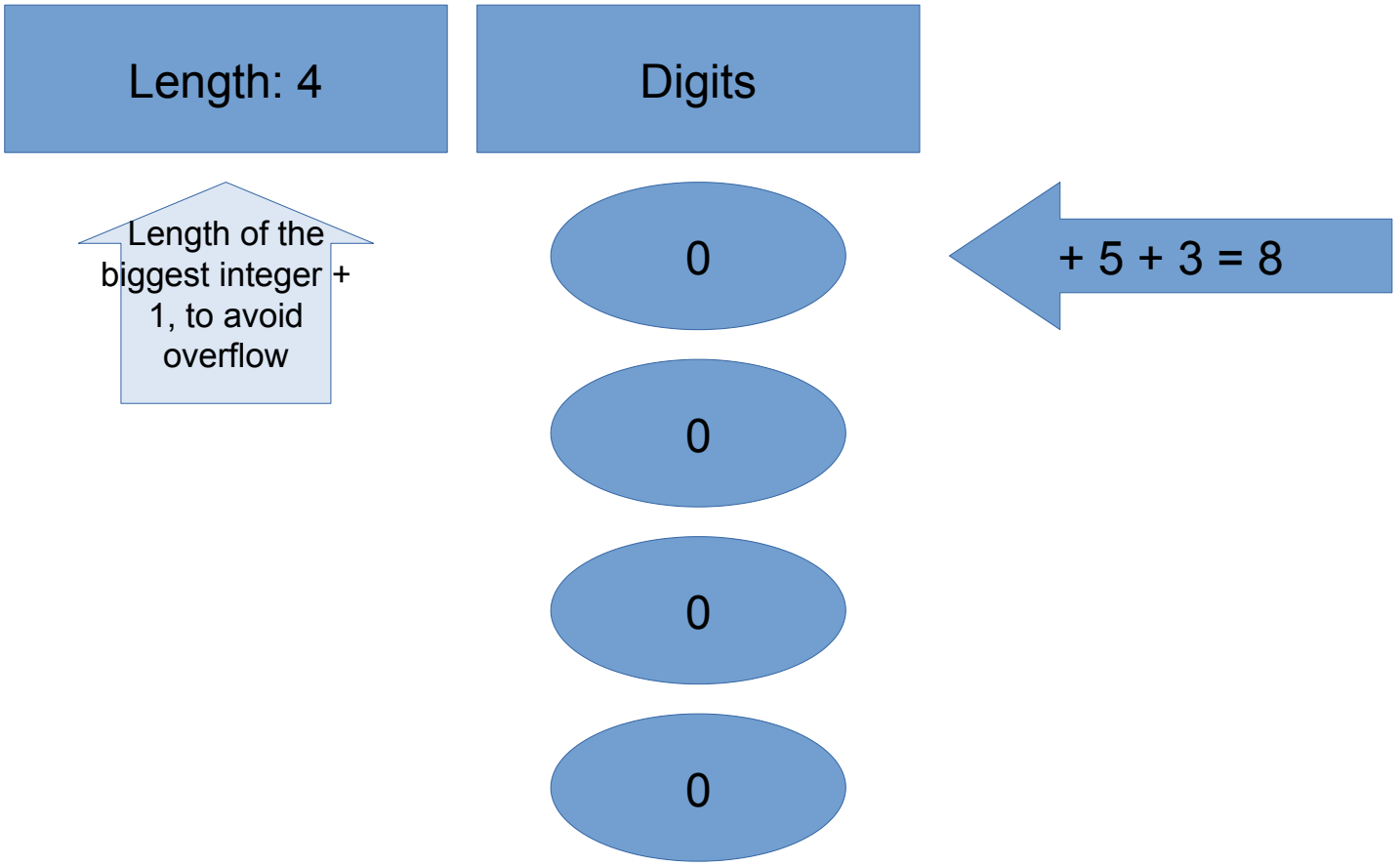
Addition Example

- Now to add 675 and -93 together:



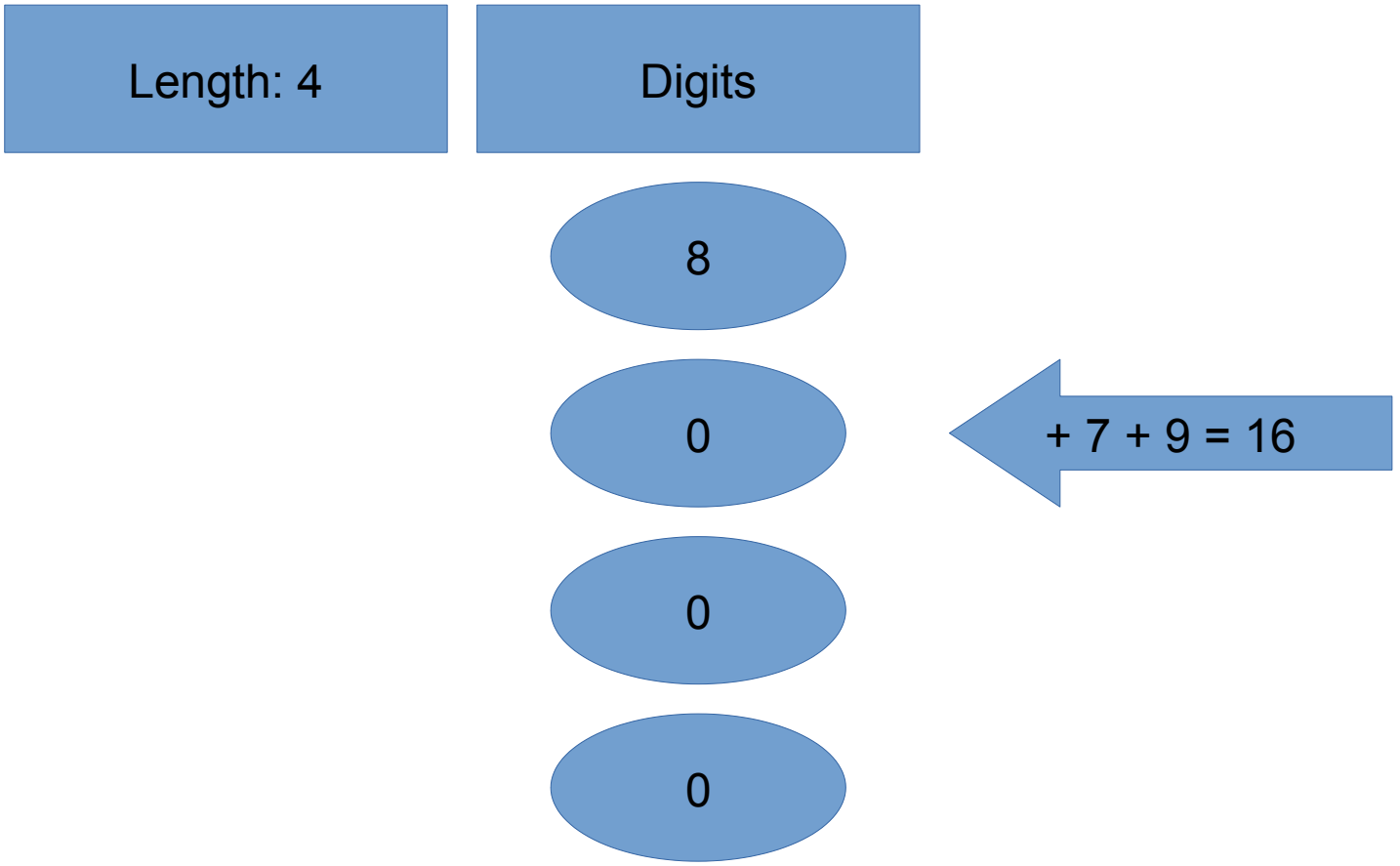
Addition Example

- Now to add 675 and *positive* 93 together:



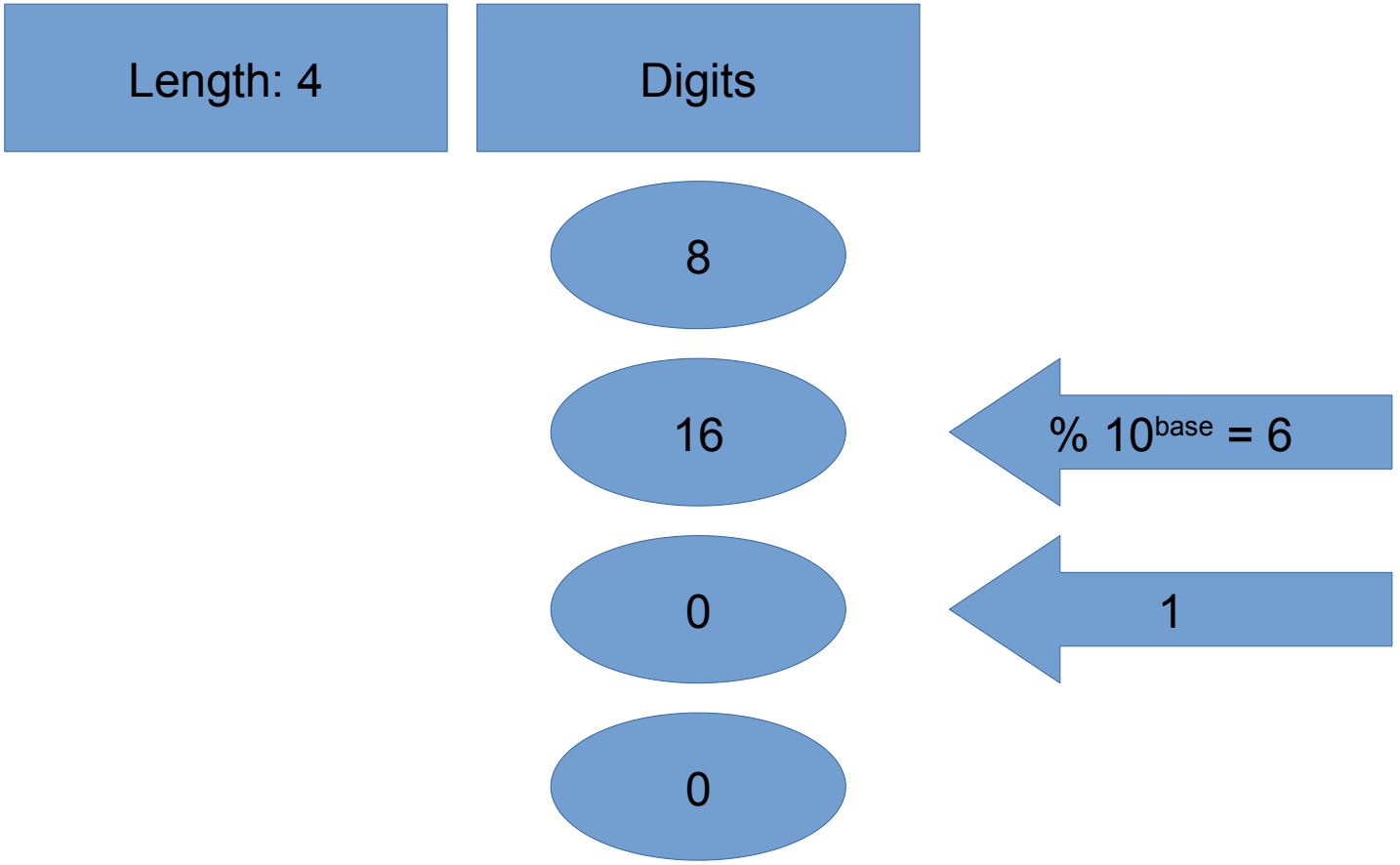
Addition Example

- Now to add 675 and *positive* 93 together:



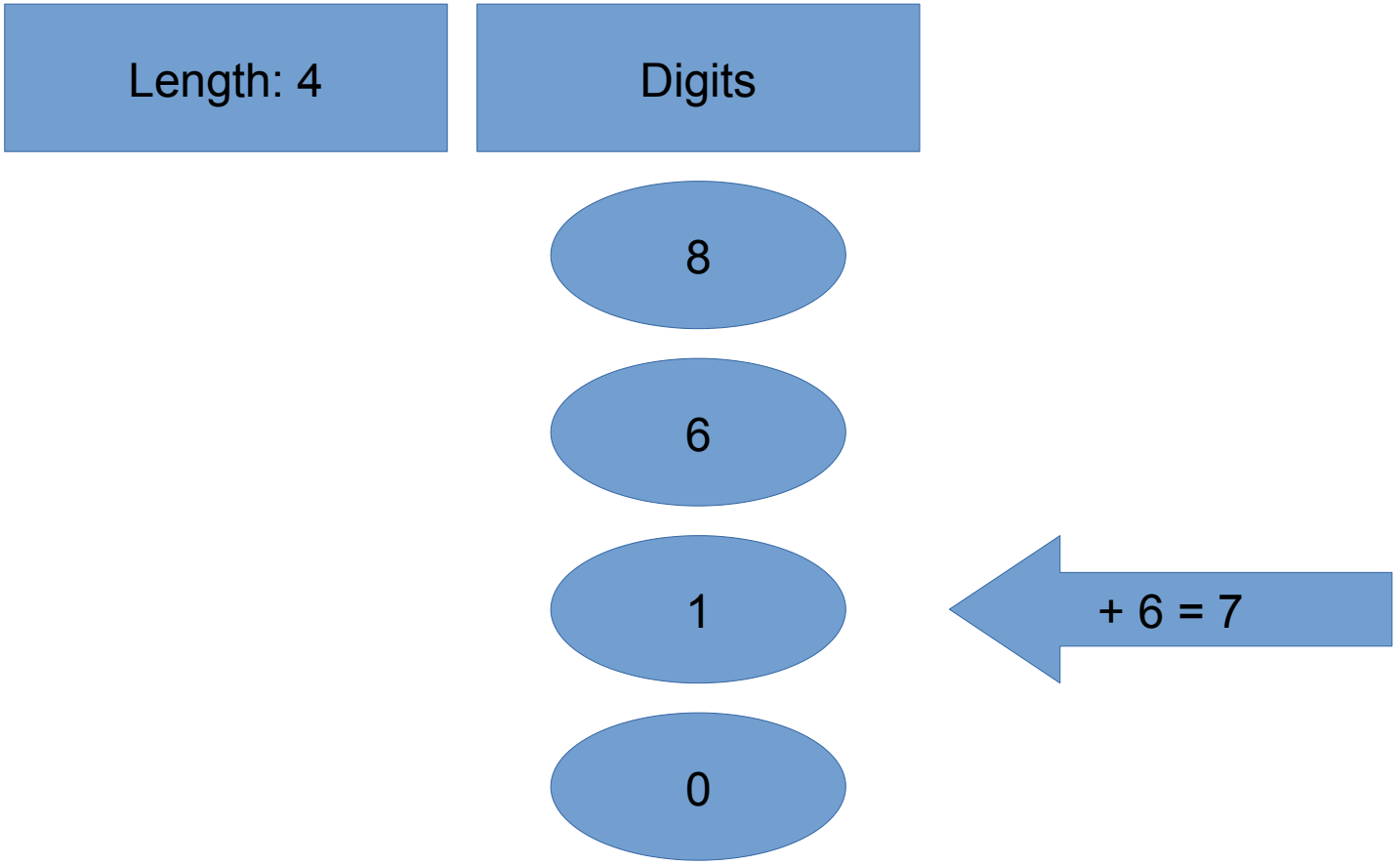
Addition Example

- Now to add 675 and *positive* 93 together:



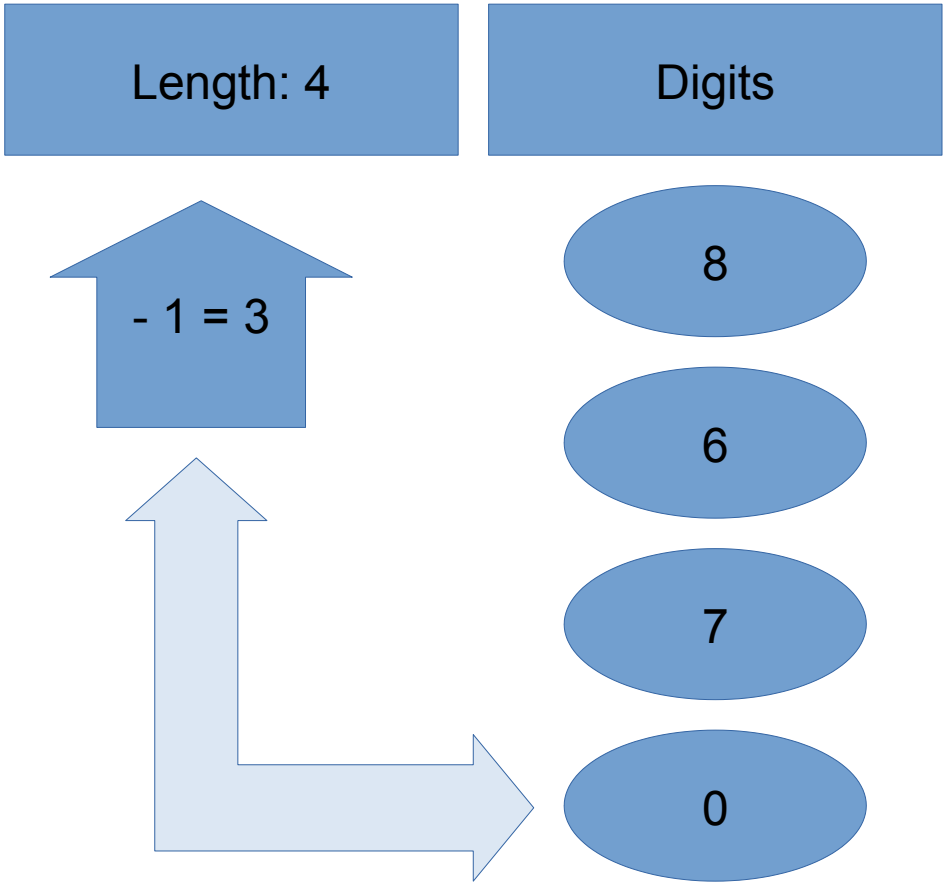
Addition Example

- Now to add 675 and *positive* 93 together:



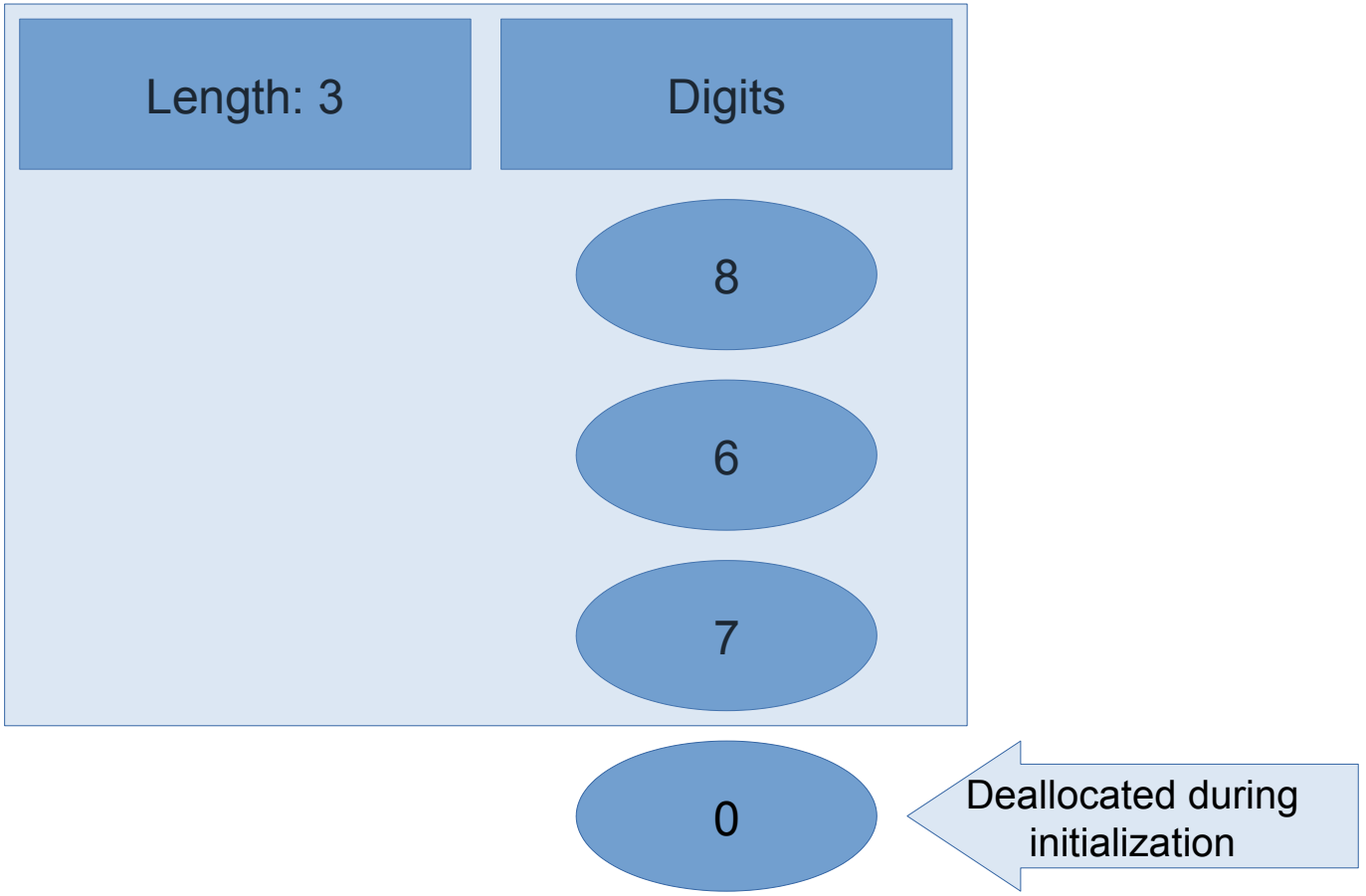
Addition Example

- Now to add 675 and *positive* 93 together:



Addition Example

- Now to add 675 and *positive* 93 together:



Implementation: Multiplication

Implementation: Multiplication algorithm

- If either number is zero, return zero
- If both numbers are negative, return ***-a * -b***
- If ***a*** is negative, return ***-(-a * b)***
- If ***b*** is negative, return ***-(a * -b)***

- Initialize ***result*** to zero
- While ***b*** is more than zero
  - If ***b*** is odd, add ***a*** to ***result***
  - Add ***a*** to itself
  - Shift ***b*** one bit right
- return ***result***
- *(Adapted from Binary Exponentiation)*

Implementation: Division & Modulo

Implementation: Division & Modulo algorithm

- Assert that ***b != 0***
- If ***a < 0*** and ***b < 0***
  - Set ***result*** to ***divMod(-a, -b)***
  - return ***(result[0], -result[1])***
- If ***a < 0***
  - Set ***result*** to ***divMod(-a, b)***
  - return ***(-result[0], result[1])***
- If ***b < 0***
  - Set ***result*** to ***divMod(a, -b)***
  - return ***(-result[0], -result[1])***
- If ***b == 1*** return ***(a, 0)***
- If ***a == b*** return ***(1, 0)***

Implementation: Division & Modulo algorithm

- If ***a*** < ***b*** return ***(0, a)***

- Initialize ***quotient*** to zero
- While ***a*** > ***b***
  - Add one to ***quotient***
  - Subtract ***b*** from ***a***
- Return ***(quotient, a)***

Implementation: Display

Implementation: To $_{\text{(base 10)}}$ string algorithm

- If the BigInt is zero, return ***"0"***
- Initialize ***result*** to an empty string
- Store whether or not the number is negative in variable ***neg***
- Set ***i*** to the absolute value of the number
- While ***i > 0***
  - Add ***'0' + char(i % 10)*** to the start of the string
  - Divide ***i*** by ten
- If the number was negative, return ***"-" + result***, otherwise return ***result***

Questions

Sources

- cppreference.com
- https://cp-algorithms.com/algebra/binary-exp.html
- https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic
- https://cp-algorithms.com/algebra/big-integer.html
- https://rushter.com/blog/python-integer-implementation/

- See my C++ implementation on GitHub:
  https://github.com/Ruan-pysoft/BigInts