CS5800 – ALGORITHMS

MODULE 2. COMPLEXITY (& COMPUTABILITY & CRYPTOGRAPHY)

Lesson 1: Arithmetic Operations

Ravi Sundaram

Topics

- Integers
- Addition/subtraction
- Multiplication/division
- Exponentiation/(Discrete log / root finding)
- Summary

Integers

- Algorithms involve BIG numbers so important to be efficient

36185027886661311069865932815214971104557430211692603585367759320207626861017237846234873269807102970128874356021481964232857782295671675021393065473695394365322208211694158783076964982631058971773918152503322026635065098926803831948392738815054324220771791218388882819961484080523021968898666372006062526501310964926475205090003984176122058711164567946559044971683604424076996342718304654479802116829701349077414009047634829067182274396120369814230709966434551334146376168244238601078897410581312713062262142086360082246515109610189789006815067664901594246966730927620844732714004599013904409378141724958467722895014360827736997469288319568431436186292967922716752485131607758720764878450583672316031730798174741417519051357029671991152963580412838184841733782

Integers

- Sometimes, an arithmetic operation is considered synonymous with a step on the computer
- BUT, in general, arithmetic on numbers is not free
- To a computer a number is a string and so arithmetic operations are string manipulations
- What is the right way to measure the number of steps an algorithm makes (when working with numbers)?

Integers

- Consider A = 56903…….. $\approx 5.7 \text{ x } 10^{75}$ $(5.7 \text{ quattorvigintillion})$
  A is roughly the number of atoms in the universe

- To describe efficiency (runtime/memory) of algorithms we parameterize numbers in terms of the length of their representation

- Definition: #bits to write number A $\simeq \log_2 A = \lg A = \Theta(\text{length}(A) \text{ in decimal})$

- Note: changing logs from one base to another changes the value by a fixed constant: $\log_B A = (\log_B C) * \log_C A$

Integers

- Consider, again A, the number of atoms in the universe 5.7×10^{75}
- lg A = 251 $(10^{75} = (10^3)^{25} \approx (2^{10})^{25} = 2^{250})$
- But this is too small for crypto purposes . Numbers are often more than 4k bits

36185027886661311069865932815214971104557430211692603585367759320207626861017237846234873269807102970128874356021481964232857782295671675021393065473695394365322082116941587830769649826310589717739181525033220266350650989268038319483927388150543242207771791218388882819961484080523021968898666372006062526501310964926475205090003984176122058711164567946559044971683604424076996342718304654479802116829701349077414009047634829067182274396120369814230709966434551334146376168244238601078897410581312713062262142086360082246515109610189789006815067664901594246966730927620844732714004599013904409378141724958467722895014360827736997469288319568431436186292967922716752485131607758720764878450583672316031730798174741751905135702967199115296358041283818484173378 2

1. Addition over the integers

$$
\begin{array}{rl}
36185027886661311069865932815214971104 & A \\
+\ 6574302116926035853677593202076268610 1 & B \\
\hline
10192804905592166960664186483597765720 5 & C
\end{array}
$$

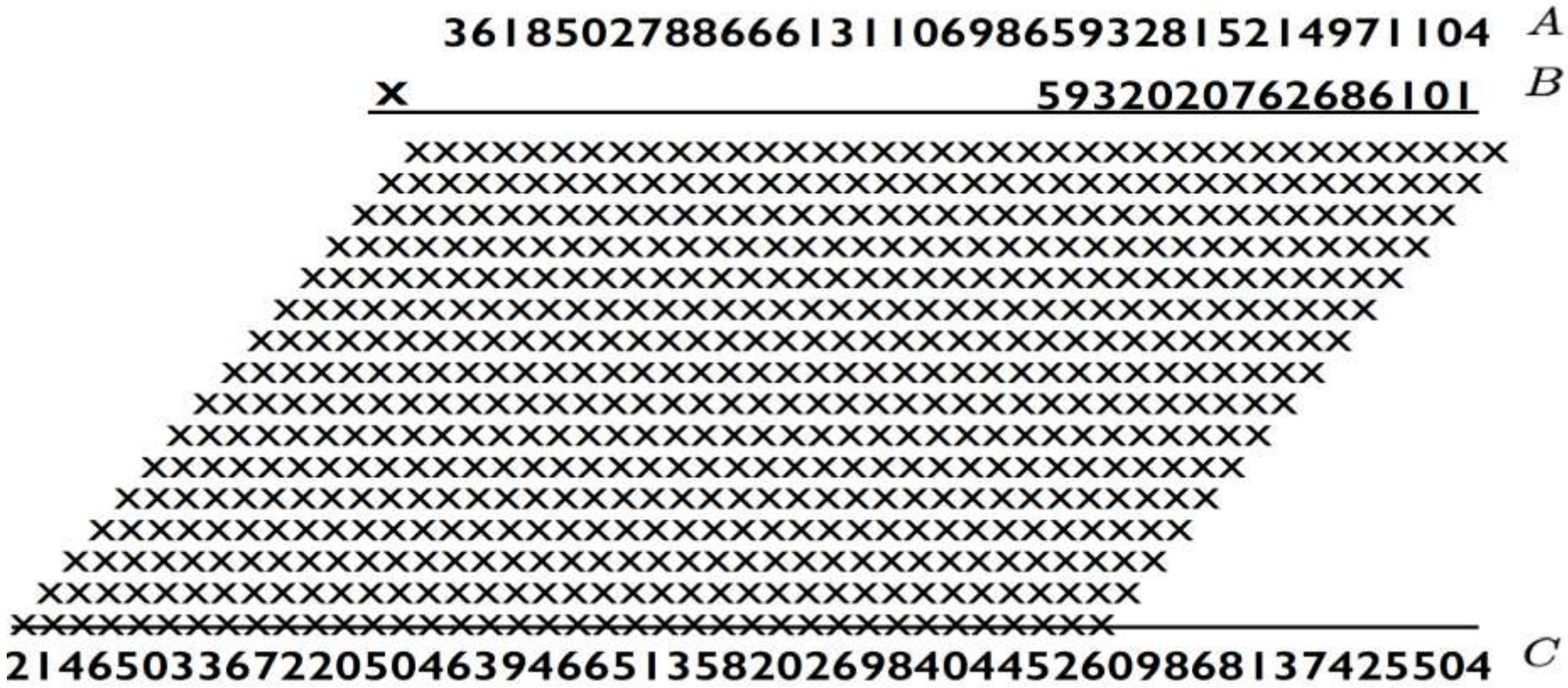
- Align the numbers and run through the positions sequentially, adding and accounting for the carry
- Addition is linear time $\Theta(\lg A + \lg B)$
- If both A and B are n-bit numbers then addition is $\Theta(n)$

2. Subtraction over the integers

$$
\begin{array}{rl}
101928049055921669606641864835977657205 & A \\
- \ 36185027886661311069865932815214971104 & B \\
\hline
65743021169260358536775932020762686101 & C
\end{array}
$$

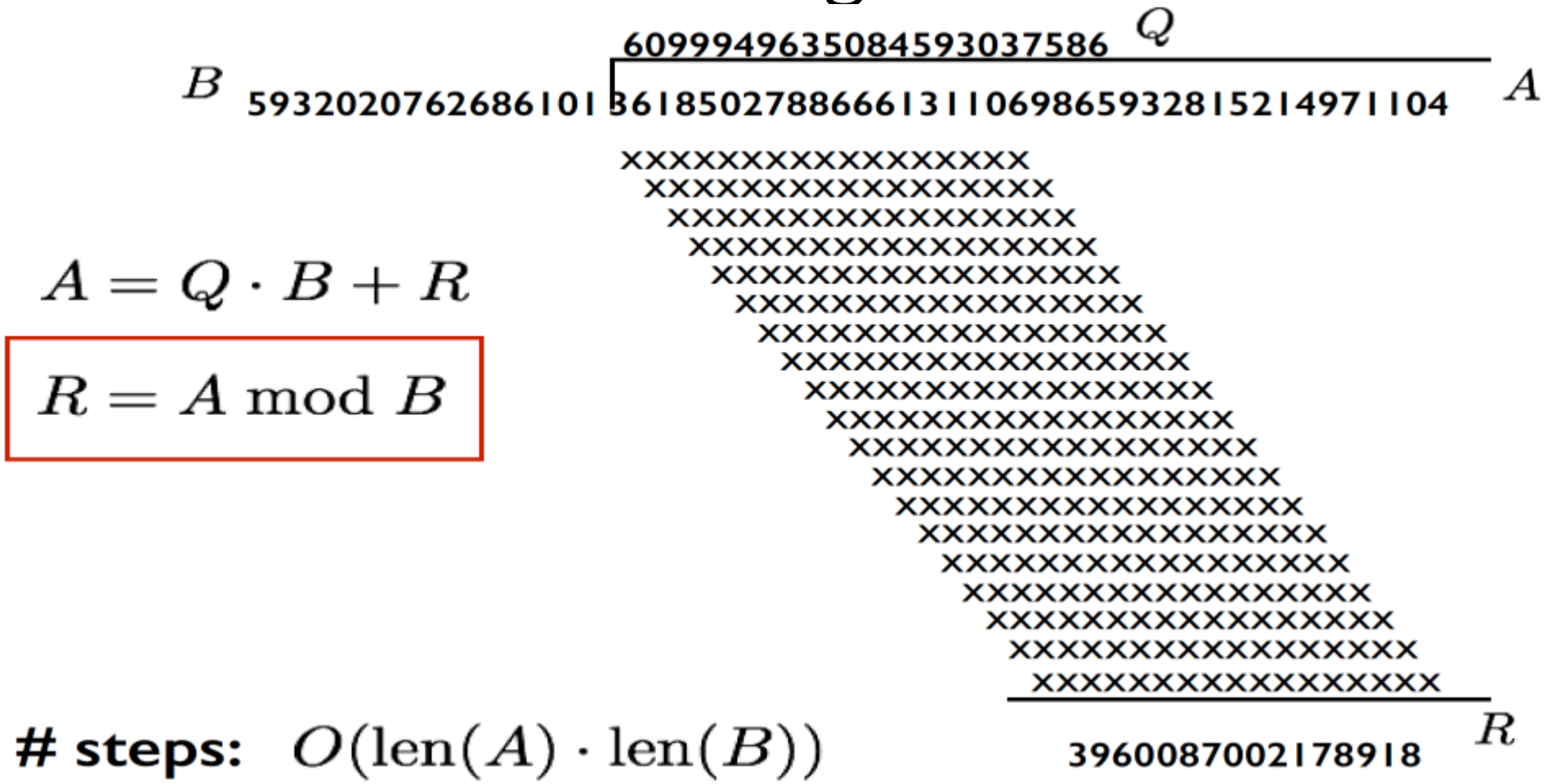
- Subtraction is linear time $\Theta(\lg A + \lg B)$
- If both A and B are n-bit numbers then subtraction is $\Theta(n)$

3. Multiplication over the integers



- Naïve (grade school) multiplication is quadratic time $\Theta(\lg A * \lg B)$
- If both A and B are n-bit numbers then (naïve) multiplication is $\Theta(n^2)$

4. Division over the integers



$$A = Q \cdot B + R$$

$$R = A \bmod B$$

 # steps: $O(\text{len}(A) \cdot \text{len}(B))$

- Naïve (grade school) long division is quadratic time Θ(lg A * lg B)

5. Exponentiation over the integers

- Given A and B compute B^A
- For starters, assume the base B = 2 and A = 56903…….. $\approx 5.7 \times 10^{75}$ (5.7 quattorvigintillion)
- Then, just to write the output 2^A will take $\lg 2^A = A \approx 5.7 \times 10^{75}$ bits which is more than the number of atoms in the universe!
- In other words, exponentiation can be very (exponentially) expensive
- Naïve algorithm: repeatedly multiply by B, complexity is:

$(\lg B)(\lg B + 2\lg B + \ldots + A*\lg B) = \Theta(A^2 \lg^2 B)$

- If both A and B are n-bit numbers then (naïve) exponentiation is:

$\Theta(n^2 * 2^{2n})$

5. Inverse of Exponentiation

- Given A and B exponentiation is computing $C = B^A$
- What is its inverse?

- There are two possible answers, depending on which of A or B is considered the unknown
- If we take A to be the unknown then it is taking logarithms, $A = \log_B C$
- If we take B to be the unknown then it is root-finding, $B = C^{1/A}$
- Without going into details we simply state that computing logs and computing roots are both polynomial-time

Summary

- It is clear that all operations must take at least linear-time since that is the amount of time needed to just look at all the input
- In a recent breakthrough (Harvey, van der Hoeven 2019) the complexity of integer multiplication and division of n-bit numbers is now at O(n log n)
- Main Takeaway: all operations are polynomial-time except for exponentiation