COL202: Discrete Mathematical Structures

Ragesh Jaiswal, CSE, IIT Delhi

Number Theory and Cryptography

Theorem

Let b be an integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0,$$

where k is a nonnegative integer, $a_0, a_1, ..., a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

- What is the running time of each of the following operations:
  - Adding an m bit number with an n bit number.
  - Multiplying an m bit number with an n bit number.

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

- Solution 1: Use long multiplication.
- What is the running time of the algorithm that uses long multiplication?

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

- Solution 1: Use long multiplication.
- What is the running time of the algorithm that uses long multiplication? $O(n^2)$
- Is there a faster algorithm?

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

- Solution 1: Algorithm using long multiplication with running time $O(n^2)$.
- Solution 2: (Assume n is a power of 2)
  - Write $A = A_L \cdot 2^{n/2} + A_R$ and $B = B_L \cdot 2^{n/2} + B_R$.
  - So, $A \cdot B = (A_L \cdot B_L) \cdot 2^n + (A_L \cdot B_R + A_R \cdot B_L) \cdot 2^{n/2} + (A_R \cdot B_R)$
  - Main Idea: Compute $(A_L \cdot B_L)$, $(A_R \cdot B_R)$, and $(A_R \cdot B_L)$, and $(A_L \cdot B_R)$ and combine these values.

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

- Solution 1: Algorithm using long multiplication with running time $O(n^2)$.
- Solution 2: (Assume n is a power of 2)
  - Write $A = A_L \cdot 2^{n/2} + A_R$ and $B = B_L \cdot 2^{n/2} + B_R$.
  - So, $A \cdot B = (A_L \cdot B_L) \cdot 2^n + (A_L \cdot B_R + A_R \cdot B_L) \cdot 2^{n/2} + (A_R \cdot B_R)$
  - Main Idea: Compute $(A_L \cdot B_L)$, $(A_R \cdot B_R)$, and $(A_R \cdot B_L)$, and $(A_L \cdot B_R)$ and combine these values.

Algorithm

`DivideAndConquer`(A, B)
- If $(|A| = |B| = 1)$ return$(A \cdot B)$
- Split A into A_L and A_R
- Split B into B_L and B_R
- $P \leftarrow$ `DivideAndConquer`(A_L, B_L)
- $Q \leftarrow$ `DivideAndConquer`(A_R, B_R)
- $R \leftarrow$ `DivideAndConquer`(A_L, B_R)
- $S \leftarrow$ `DivideAndConquer`(A_R, B_L)
- return(`Combine`(P, Q, R, S))

- What is the recurrence relation for the running time of the above algorithm?

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

Algorithm

DivideAndConquer(A, B)

- If $(|A| = |B| = 1)$ return$(A \cdot B)$
- Split A into A_L and A_R
- Split B into B_L and B_R
- $P \leftarrow$ DivideAndConquer(A_L, B_L)
- $Q \leftarrow$ DivideAndConquer(A_R, B_R)
- $R \leftarrow$ DivideAndConquer(A_L, B_R)
- $S \leftarrow$ DivideAndConquer(A_R, B_L)
- return(Combine(P, Q, R, S))

- What is the recurrence relation for the running time of the above algorithm? $T(n) = 4 \cdot T(n/2) + O(n)$ for $n > 1$ and $T(1) = O(1)$.
- What is the solution to the above recurrence relation?

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

Algorithm

DivideAndConquer(A, B)
- If $(|A| = |B| = 1)$ return$(A \cdot B)$
- Split A into A_L and A_R
- Split B into B_L and B_R
- $P \leftarrow$ DivideAndConquer(A_L, B_L)
- $Q \leftarrow$ DivideAndConquer(A_R, B_R)
- $R \leftarrow$ DivideAndConquer(A_L, B_R)
- $S \leftarrow$ DivideAndConquer(A_R, B_L)
- return(Combine(P, Q, R, S))

- What is the recurrence relation for the running time of the above algorithm? $T(n) = 4 \cdot T(n/2) + O(n)$ for $n > 1$ and $T(1) = O(1)$.
- What is the solution to the above recurrence relation? $T(n) = O(n^2)$.

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

- Solution 1: Algorithm using long multiplication with running time $O(n^2)$.
- Solution 2: Naïve Divide and Conquer with running time $O(n^2)$.
- Solution 3:
  - Write $A = A_L \cdot 2^{n/2} + A_R$ and $B = B_L \cdot 2^{n/2} + B_R$.
  - So, $A \cdot B = (A_L \cdot B_L) \cdot 2^n + (A_L \cdot B_R + A_R \cdot B_L) \cdot 2^{n/2} + (A_R \cdot B_R)$
  - Main Idea: Compute $(A_L \cdot B_L)$, $(A_R \cdot B_R)$, and $(A_L + B_L) \cdot (A_R + B_R) - (A_L \cdot B_L) - (A_R \cdot B_R)$.

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

Algorithm

`Karatsuba`(A, B)
- If $(|A| = |B| = 1)$ return$(A \cdot B)$
- Split A into A_L and A_R
- Split B into B_L and B_R
- $P \leftarrow$ `Karatsuba`(A_L, B_L)
- $Q \leftarrow$ `Karatsuba`(A_R, B_R)
- $R \leftarrow$ `Karatsuba`$(A_L + A_R, B_L + B_R)$
- return(`Combine`(P, Q, R))

- What is the recurrence relation for the running time of the above algorithm?

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

Algorithm

Karatsuba(A, B)
- If $(|A| = |B| = 1)$ return$(A \cdot B)$
- Split A into A_L and A_R
- Split B into B_L and B_R
- $P \leftarrow \texttt{Karatsuba}(A_L, B_L)$
- $Q \leftarrow \texttt{Karatsuba}(A_R, B_R)$
- $R \leftarrow \texttt{Karatsuba}(A_L + A_R, B_L + B_R)$
- return$(\texttt{Combine}(P, Q, R))$

- Recurrence relation: $T(n) \leq 3 \cdot T(n/2) + cn$; $T(1) \leq c$.
- What is the solution of this recurrence relation?

Problem

Multiplying two n-bit numbers: Given two n-bit numbers, A and B, Design an algorithm to output $A \cdot B$.

Algorithm

$\texttt{Karatsuba}(A, B)$
- If $(|A| = |B| = 1)$ return$(A \cdot B)$
- Split A into A_L and A_R
- Split B into B_L and B_R
- $P \leftarrow \texttt{Karatsuba}(A_L, B_L)$
- $Q \leftarrow \texttt{Karatsuba}(A_R, B_R)$
- $R \leftarrow \texttt{Karatsuba}(A_L + A_R, B_L + B_R)$
- return$(\texttt{Combine}(P, Q, R))$

- Recurrence relation: $T(n) \leq 3 \cdot T(n/2) + cn$; $T(1) \leq c$.
- What is the solution of this recurrence relation?
  $T(n) \leq O(n^{\log_2 3})$

Theorem

Let b be an integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0,$$

where k is a nonnegative integer, $a_0, a_1, ..., a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

- What is the running time of each of the following operations:
  - Adding an m bit number with an n bit number.
  - Multiplying an m bit number with an n bit number.
  - Dividing an m bit number by an n bit number.
  - Computing an m bit number modulo an n bit number.

Definition

An integer p greater than 1 is called *prime* if the only positive factors of p are 1 and p. A positive integer that is greater than 1 and is not prime is called *composite*.

Theorem (Fundamental theorem of arithmetic)

Every integer greater than 1 can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of nondecreasing size.

Theorem

If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

- How can we find all prime numbers ≤ 100?
  - Show that any composite number ≤ 100 are divisible by $2, 3, 5, 7$.
  - Sieve of Eratosthenes uses this idea to eliminate all composites and list all primes.

Theorem

There are infinitely many primes.

Definition

Let a and b be integers, not both zero. The largest integer d such that $d|a$ and $d|b$ is called the *greatest common divisor* of a and b. The greatest common divisor of a and b is denoted by $gcd(a, b)$.

Definition

The integers a and b are *relatively prime* if their greatest common divisor is 1.

Definition

The integers $a_1, a_2, ..., a_n$ are pairwise relatively prime if $gcd(a_i, a_j) = 1$ whenever $1 \leq i < j \leq n$.

Definition

The *least common multiple* of the positive integers a and b is the smallest positive integer that is divisible by both a and b. The least common multiple of a and b is denoted by $lcm(a, b)$.

Theorem

Let a and b be positive integers. Then $ab = gcd(a, b) \cdot lcm(a, b)$.

Theorem

Let $a = bq + r$, where a, b, q, and r are integers. Then $gcd(a, b) = gcd(b, r)$.

- Using the above theorem, design an algorithm to compute gcd of two n bit numbers. What is the worst-case running time of your algorithm?

Theorem

Let $a = bq + r$, where a, b, q, and r are integers. Then $gcd(a, b) = gcd(b, r)$.

- Using the above theorem, design an algorithm to compute gcd of two n bit numbers. What is the worst-case running time of your algorithm?

Euclid-GCD(a, b)
  If ($b = 0$) then return(a)
  else return(Euclid-GCD($b, a \ (mod \ b)$))

Euclid-GCD(a, b)
If ($b = 0$) then return(a)
else return(Euclid-GCD($b, a \ (mod \ b)$))

- How many recursive calls are made by the algorithm?
- What is the worst-case time complexity of the algorithm?

Theorem

Let a, b be positive integers. Then there exists integers x, y such that $xa + yb = gcd(a, b)$. Furthermore, $gcd(a, b)$ is the smallest positive integer that can be expressed in this way.

End