Numerics/Math

Lecture Objectives

- Understanding Fundamental Math Concepts for Algorithmic Problem-solving
- Utilizing Common Numerical Algorithms in DSA Problem Solving

Lecture Outline

Prerequisites

- Time and Space Complexity Analysis
- Recursion I
- Loops and Conditionals

Divisibility

Divisibility

- We can describe any number N as the product of the **divisor d** and **the quotient q**, plus a **remainder r**.
  - `N = q * d + r,    0 <= r < d`

- Python syntax
  - **`N % d == r`**
  - **`N // d == q`**
  - **`divmod(N, d) == (q, r)`**, divmod is a function that returns `(q, r)`

Divisibility - Modular Arithmetic

- `(a + b) % m = (a%m + b%m) % m`
- `(a - b) % m = (a%m - b%m) % m`
- `(a*b)%m = (a%m * b%m) % m`
- If `(a - b) % m = 0`, then `a%m = b%m`
- Division is a little complicated.

Divisibility - Modular Arithmetic

- Sometimes, it could be difficult to work with modular arithmetic, especially under time pressure.
- If we need to express the modulo of an unknown variable in modular arithmetic, we can follow these steps
  - Get rid of the modulo operator from each term
  - Rearrange the terms in the usual arithmetic rules so that you can express the unknown variable on one side and the knowns on the other
  - Modulo every term
- Note: This works only if the involved operations are addition, subtraction and multiplication (of integers)

Divisibility - Modular Arithmetic

Problem Link

- Example: solve for x%k
  - `(a + x)%k = 0`
  - Answer: `x%k = (-a)%k`

Prime Numbers

Prime Numbers

- What are prime numbers?
- 2, 3, 5, 7 . . .

Prime Numbers

- How can we check if a number is prime?
- We could check if the number is divisible by all the numbers before it?

```
def isPrime(x: int) -> bool:
    d = 2
    while d < x:
        if x % d == 0:
            return False
        d += 1
    return True
```

Primality Test

- If `x = d1*d2`.
- `Either d1 <= d2 or d2 < d1.`
- `Thus d1*d1 <= d1*d2 = x or d2*d2 < d1*d2 = x.`
- What does this mean?

```python
def isPrime(x: int) -> bool:
    d = 2
    while d*d <= x:
        if x % d == 0:
            return False
        d += 1
    return True
```

Primality Test - Time and Space Complexity

- The loop runs until we reach the first time `d*d = n`
- Hence, the time complexity is `O(d) = O(sqrt(n))`

Fundamental Theorem Of Arithmetic

- Every positive integer can be written in a unique way as a product of primes:
- `n = p1 * p2 * p3 * . . . pn`
- Example:
  - `84 = 2*2*3*7`
  - `52 = ?`

Prime Factorization

How can we factorize a given number n into prime factors?

- Start from d = 2.
- Divide n by d as long as you can. Append d.
- Increment d by 1 and try again.

Prime Factorization

```python
def trial_division_simple(n: int) -> list[int]:
    factorization: list[int] = []
    d = 2
    while d * d <= n:
        while n % d == 0:
            factorization.append(d)
            n //= d
        d += 1
    if n > 1:
        factorization.append(n)

    return factorization
```

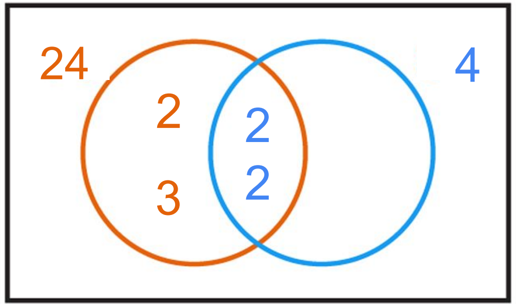
```
Worst case: n is prime, O(d) = O(sqrt(n))
Best case: n = 2**k, Ω(k) = Ω(log(n))
```

Exercise

Almost Prime

Divisibility and Prime Factors

How does prime factorization relate to divisibility?



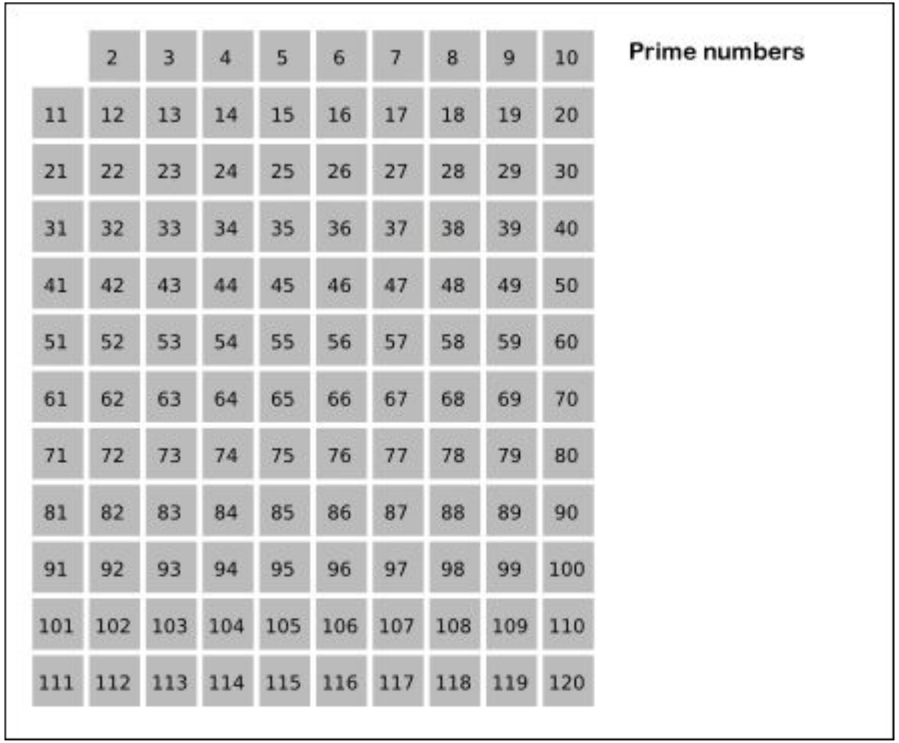
Example: 24 is divisible by 4

4 = 2 * 2

24 = 2 * 2 * 2 * 3

Generating Primes

Sieve of Eratosthenes



Prime numbers

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Sieve of Eratosthenes

- To determine all the primes in the range `[2, n]`
    - Start with two and mark all its multiples until n not including it
    - Then mark the multiples of the next unmarked number not including it
    - At the end the unmarked elements are the primes

Sieve of Eratosthenes

```python
def prime_sieve(n: int) -> list[bool]:
   is_prime: list[bool] = [True for _ in range(n + 1)]
   is_prime[0] = is_prime[1] = False

   i = 2
   while i <= n:
       if is_prime[i]:
           j = 2 * i
           while j <= n:
               is_prime[j] = False
               j += i
       i += 1

   return is_prime
```

Sieve of Eratosthenes - Optimization

- How can we optimize it?
    - Sieve till root
    - Consider only proper multiples greater than square of the number
    - Sieving by the odd numbers only

Sieve of Eratosthenes

```
def prime_sieve(n: int) -> list[bool]:
   is_prime: list[bool] = [True for _ in range(n + 1)]
   is_prime[0] = is_prime[1] = False

   i = 2
   while i * i <= n:
       if is_prime[i]:
           j = i * i
           while j <= n:
               is_prime[j] = False
               j += i
       i += 1

   return is_prime
```

Time Complexity: O(n log log n)

[Proof of the bound on sum of the reciprocal of primes]()

Exercise

Count Primes

GCD

Greatest Common Divisor

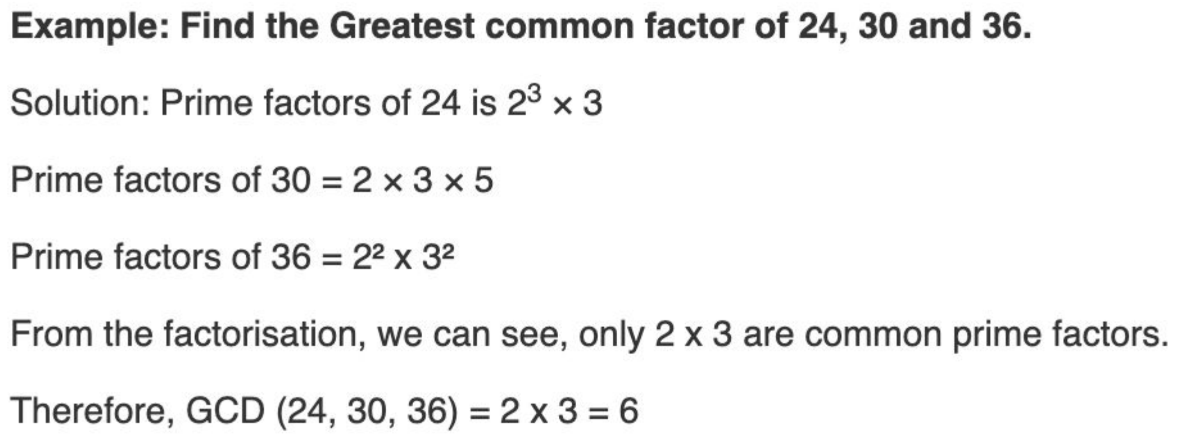
- GCD (Greatest Common Divisor) of two numbers a and b is the greatest number that divides both a and b.
  - Naive algorithm
  - Fast algorithm (`log(n)`)

Greatest Common Divisor

- How would you calculate gcd by hand?

Greatest Common Divisor

- In the prime factorisation method, each given number is written as the product of prime numbers and then find the product of the smallest power of each common prime factor. Why?



Example: Find the Greatest common factor of 24, 30 and 36.

Solution: Prime factors of 24 is $2^3 \times 3$

Prime factors of 30 = $2 \times 3 \times 5$

Prime factors of 36 = $2^2 \times 3^2$

From the factorisation, we can see, only 2 x 3 are common prime factors.

Therefore, GCD (24, 30, 36) = 2 x 3 = 6

Greatest Common Divisor

- Find the GCD of 32 and 28

Greatest Common Divisor

Let a = b*q + r, and m be a common divisor

```
●   r%m = (a - b*q)%m
        = (a%m - (b*q)%m)%m
        = (a%m - (b%m)*(q%m))%m
        = (0 - 0)%m = 0
```

Thus m also divides r.

If r = 0, then a is a multiple of b and thus b is the GCD.

Euclid’s Algorithm

```
fun gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)
```

Formal Proof

Greatest Common Divisor

- Find the GCD of 32 and 28

Least Common Multiples

- `LCM(a, b)`: the smallest positive integer that is a multiple of a and b
- How would you calculate LCM by hand?
- `LCM(a, b) x GCD(a, b) = a x b`
  - Why?
- How can we implement the LCM function?

Inclusion-Exclusion Principle

Inclusion-Exclusion Principle

- used to count the number of elements in the union of multiple sets.
  - $n(A \cup B) = n(A) + n(B) - n(A \cap B)$
  - $n(A \cup B \cup C) = n(A) + n(B) + n(C)$

    $- \; n(A \cap B) - n(B \cap C) - n(A \cap C)$

    $+ \; n(A \cap B \cap C)$

Inclusion-Exclusion Principle

- Find the total number of integers between 1 and 100 that are either divisible by 2 or by 3.
    - How many multiples of 2 are there between 1 and 100?
    - How many multiples of 3 are there between 1 and 100?
    - How many multiples of 6 are there between 1 and 100?

What about the number of integers that are divisible by either of 2, 3 and 5?

Exercise

[Complicated GCD]
[Find Greatest Common Divisor of Array]
[Number of Subarrays With GCD Equal to K]
[Count Primes]
[Divisibility by 2^n]
[Block Game]
[Divide and Equalize]

Quote of the Day

"The enchanting charms of this sublime science reveal only to those who have the courage to go deeply into it."

~ Carl Friedrich Gauss