Enough Mathematical Appetizers!

Let us look at something more interesting:

Algorithms

Algorithms

What is an algorithm?

An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

This is a rather vague definition. You will get to know a more precise and mathematically useful definition when you attend CMSC441.

But this one is good enough for now…

Algorithms

Properties of algorithms:

- **Input** from a specified set,
- **Output** from a specified set (solution),
- **Definiteness** of every step in the computation,
- **Correctness** of output for every possible input,
- **Finiteness** of the number of calculation steps,
- **Effectiveness** of each calculation step and
- **Generality** for a class of problems.

Algorithm Examples

We will use a pseudocode to specify algorithms, which slightly reminds us of Basic and Pascal.

Example: an algorithm that finds the maximum element in a finite sequence

```
procedure max($a_1, a_2, \ldots, a_n$: integers)
max := $a_1$
for i := 2 to n
        if max < $a_i$ then max := $a_i$
{max is the largest element}
```

Algorithm Examples

Another example: a linear search algorithm, that is, an algorithm that linearly searches a sequence for a particular element.

```
procedure linear_search(x: integer; $a_1, a_2, \ldots, a_n$: integers)
i := 1
while (i $\le$ n and x $\ne a_i$)
    i := i + 1
if i $\le$ n then location := i
else location := 0
{location is the subscript of the term that equals x, or is zero if x is not found}
```

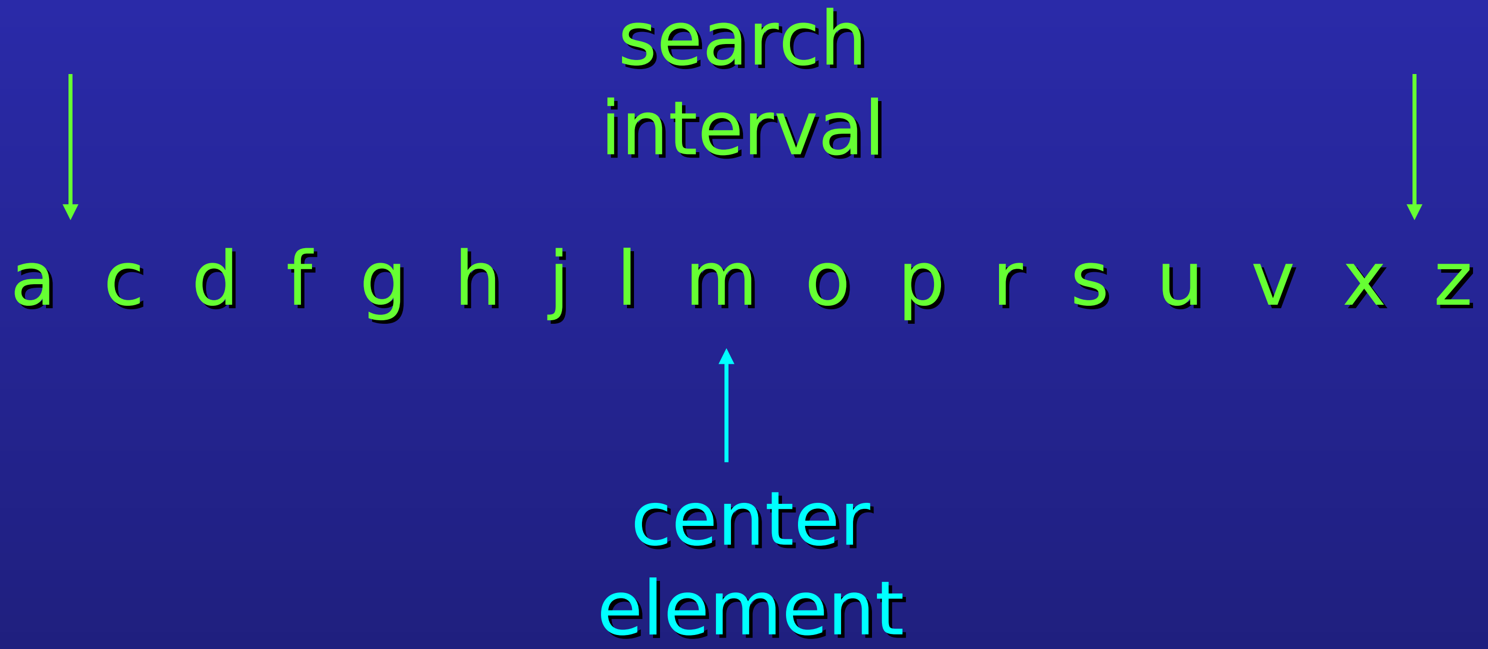
Algorithm Examples

If the terms in a sequence are ordered, a binary search algorithm is more efficient than linear search.

The binary search algorithm iteratively restricts the relevant search interval until it closes in on the position of the element to be located.

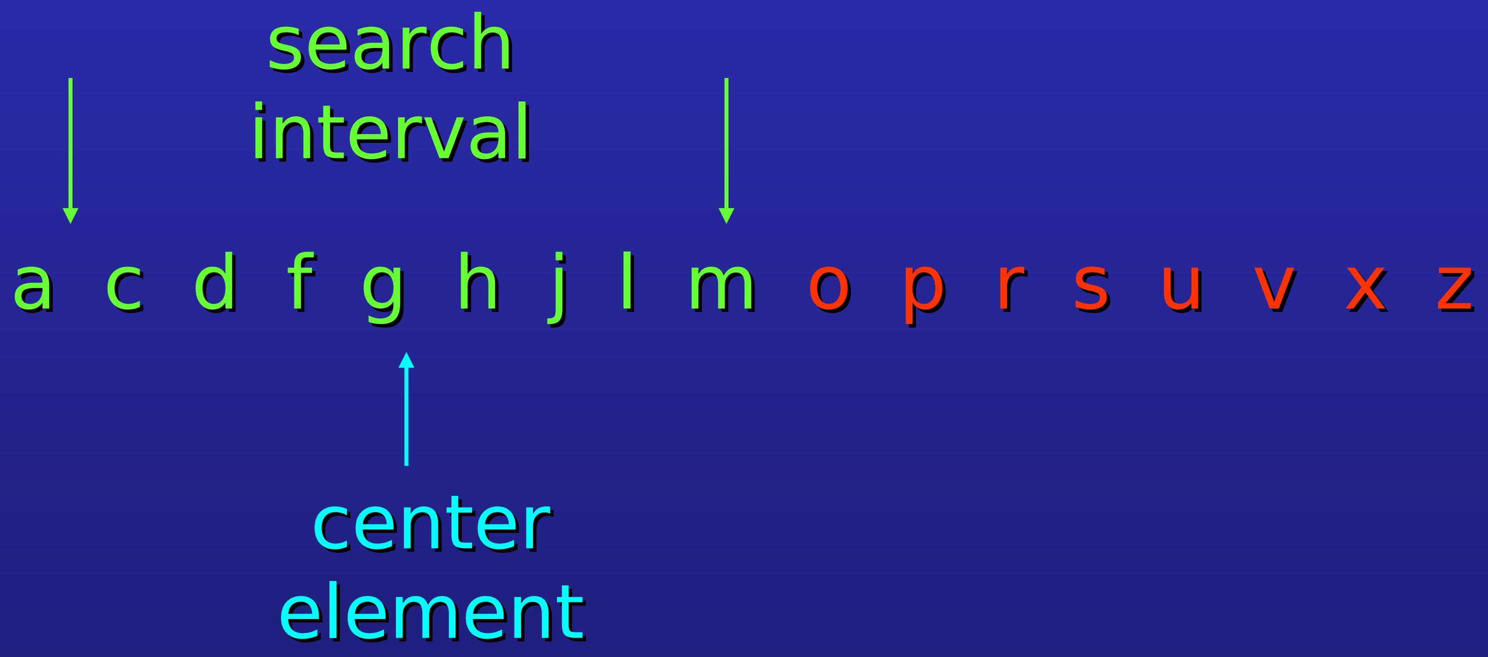
Algorithm Examples

binary search for the letter ‘j’



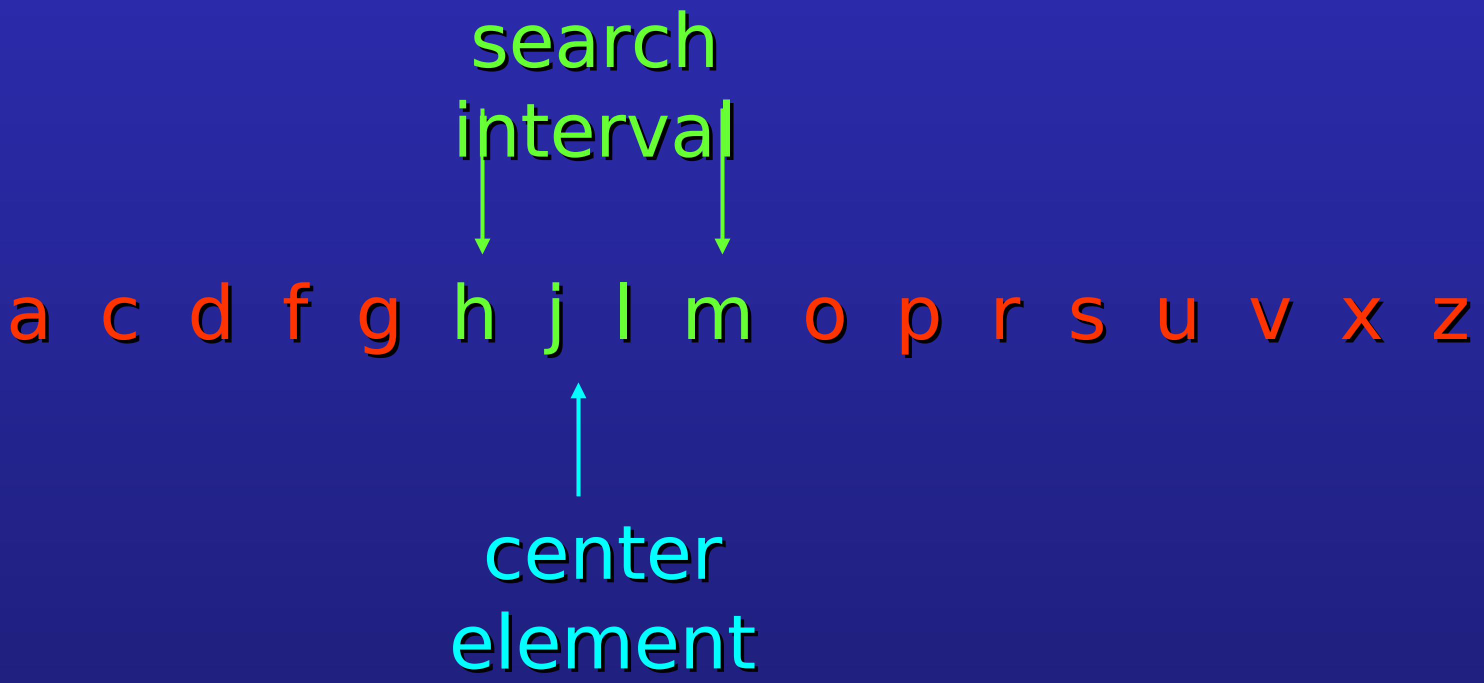
Algorithm Examples

binary search for the letter ‘j’



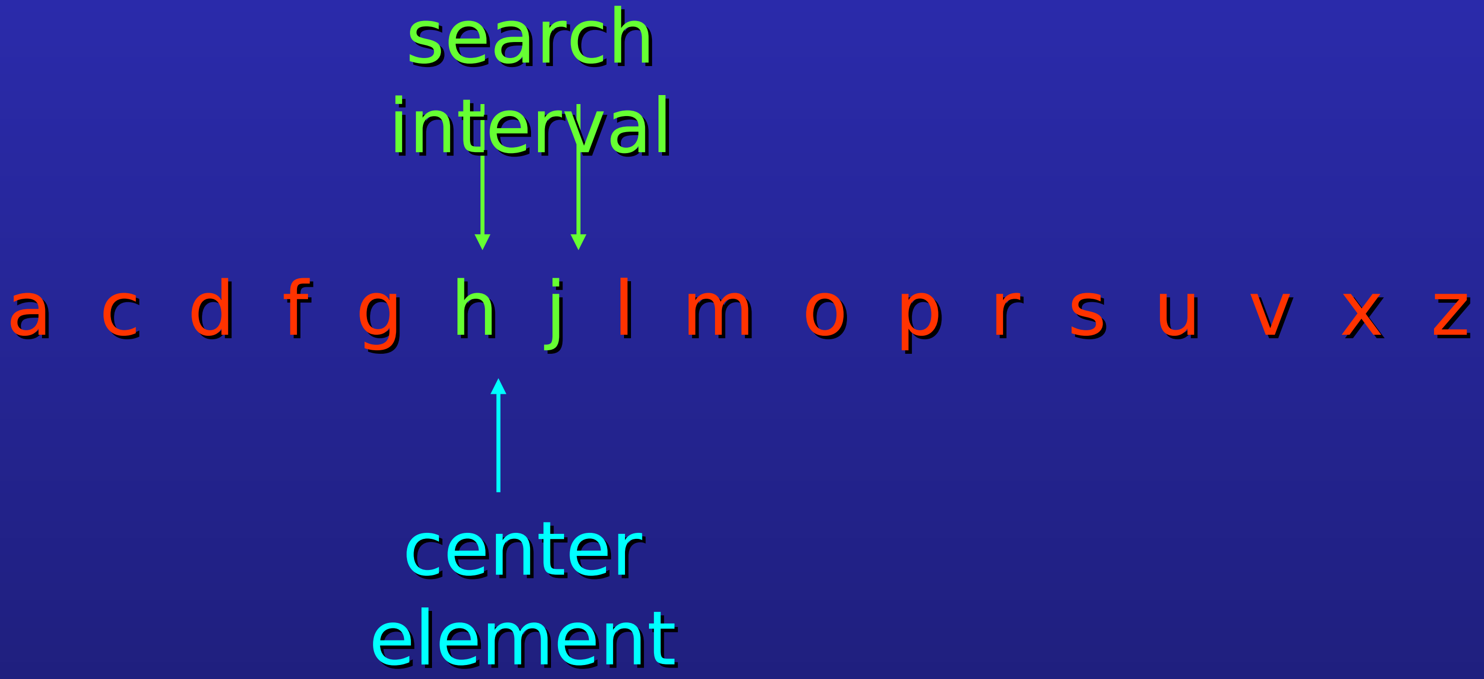
Algorithm Examples

binary search for the letter ‘j’



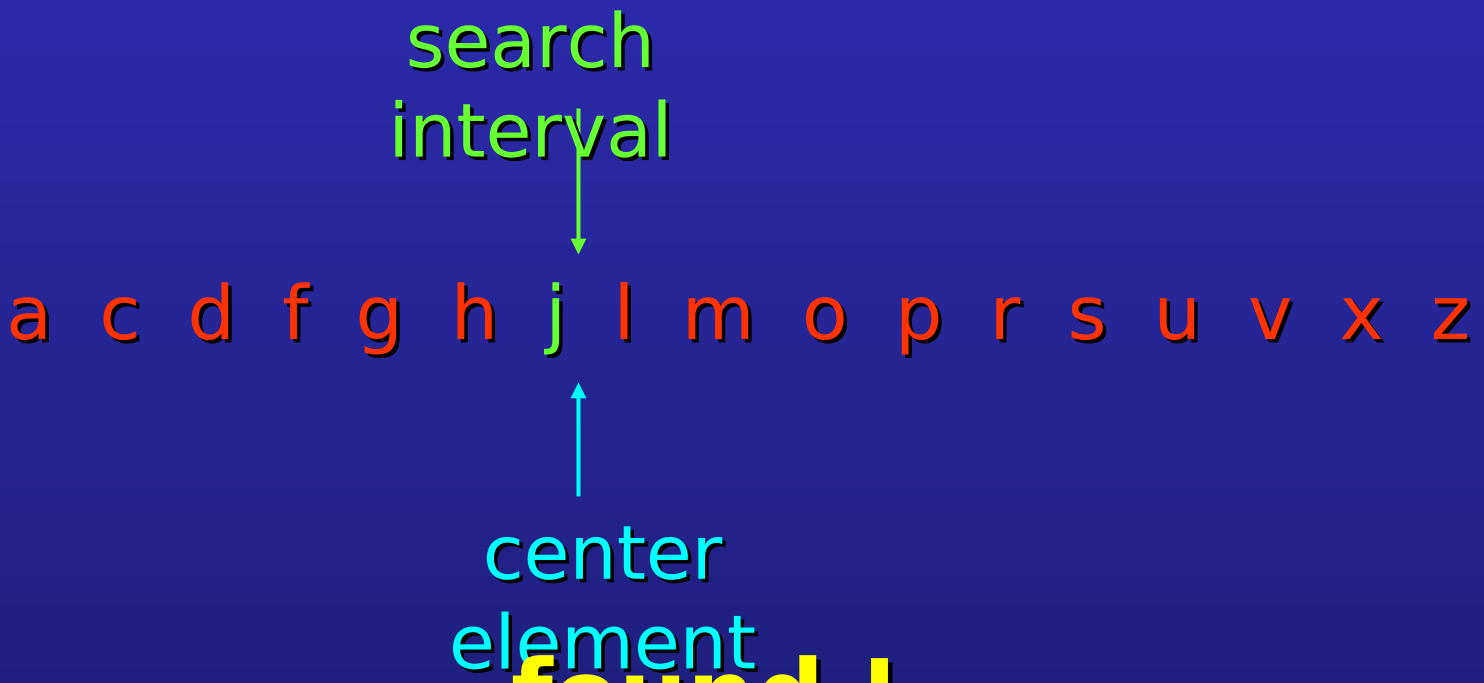
Algorithm Examples

binary search for the letter ‘j’



Algorithm Examples

binary search for the letter ‘j’



found !

Algorithm Examples

```
procedure binary_search(x: integer; $a_1, a_2, \ldots, a_n$: integers)
i := 1   {i is left endpoint of search interval}
j := n   {j is right endpoint of search interval}
while (i < j)
begin
        m := $\lfloor (i + j)/2 \rfloor$
        if x > $a_m$ then i := m + 1
        else j := m
end
if x = $a_i$ then location := i
else location := 0
{location is the subscript of the term that equals x, or is zero if x is not found}
```

Complexity

In general, we are not so much interested in the time and space complexity for small inputs.

For example, while the difference in time complexity between linear and binary search is meaningless for a sequence with $n = 10$, it is gigantic for $n = 2^{30}$.

Complexity

For example, let us assume two algorithms A and B that solve the same class of problems.

The time complexity of A is 5,000n, the one for B is $\lceil 1.1^n \rceil$ for an input with n elements.

For n = 10, A requires 50,000 steps, but B only 3, so B seems to be superior to A.

For n = 1000, however, A requires 5,000,000 steps, while B requires $2.5 \cdot 10^{41}$ steps.

Complexity

This means that algorithm B cannot be used for large inputs, while algorithm A is still feasible.

So what is important is the **growth** of the complexity functions.

The growth of time and space complexity with increasing input size n is a suitable measure for the comparison of algorithms.

Complexity

Comparison: time complexity of algorithms A and B

Input Size	Algorithm A	Algorithm B
n	5,000n	$\lceil 1.1^n \rceil$
10	50,000	3
100	500,000	13,781
1,000	5,000,000	$2.5 \cdot 10^{41}$
1,000,000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

The Growth of Functions

The growth of functions is usually described using the **big-O notation**.

Definition: Let f and g be functions from the integers or the real numbers to the real numbers.
We say that f(x) is O(g(x)) if there are constants C and k such that

$|f(x)| \leq C|g(x)|$

whenever $x > k$.

The Growth of Functions

When we analyze the growth of **complexity functions**, f(x) and g(x) are always positive.

Therefore, we can simplify the big-O requirement to

$f(x) \leq C \cdot g(x)$ whenever $x > k$.

If we want to show that f(x) is O(g(x)), we only need to find **one** pair (C, k) (which is never unique).

The Growth of Functions

The idea behind the big-O notation is to establish an **upper boundary** for the growth of a function f(x) for large x.

This boundary is specified by a function g(x) that is usually much **simpler** than f(x).

We accept the constant C in the requirement

$f(x) \leq C \cdot g(x)$ whenever $x > k$,

because **C does not grow with x.**

We are only interested in large x, so it is OK if $f(x) > C \cdot g(x)$ for $x \leq k$.

The Growth of Functions

Example:

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

For $x > 1$ we have:

$x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$

$\Rightarrow x^2 + 2x + 1 \leq 4x^2$

Therefore, for $C = 4$ and $k = 1$:

$f(x) \leq Cx^2$ whenever $x > k$.

$\Rightarrow f(x)$ is $O(x^2)$.

The Growth of Functions

Question: If f(x) is O(x^2), is it also O(x^3)?

Yes. x^3 grows faster than x^2, so x^3 grows also faster than f(x).

Therefore, we always have to find the **smallest** simple function g(x) for which f(x) is O(g(x)).

The Growth of Functions

"Popular" functions g(n) are
$n \log n$, 1, 2^n, n^2, $n!$, n, n^3, $\log n$

Listed from slowest to fastest growth:

- 1
- $\log n$
- n
- $n \log n$
- n^2
- n^3
- 2^n
- $n!$

The Growth of Functions

A problem that can be solved with polynomial worst-case complexity is called **tractable**.

Problems of higher complexity are called **intractable.**

Problems that no algorithm can solve are called **unsolvable**.

You will find out more about this in CMSC441.

Useful Rules for Big-O

For any **polynomial** $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$, where $a_0, a_1, \ldots, a_n$ are real numbers, $f(x)$ is $O(x^n)$.

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$

If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $O(g(x))$, then $(f_1 + f_2)(x)$ is $O(g(x))$.

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 f_2)(x)$ is $O(g_1(x)\, g_2(x))$.

Complexity Examples

What does the following algorithm compute?

```
procedure who_knows($a_1, a_2, \ldots, a_n$: integers)
m := 0
for i := 1 to n-1
        for j := i + 1 to n
                if $|a_i – a_j| > m$ then m := $|a_i – a_j|$
{m is the maximum difference between any two numbers in the input sequence}
```

Comparisons: n-1 + n-2 + n-3 + … + 1

$$= (n – 1)n/2 = 0.5n^2 – 0.5n$$

Time complexity is $O(n^2)$.

Complexity Examples

Another algorithm solving the same problem:

```
**procedure** max_diff($a_1$, $a_2$, …, $a_n$: integers)
min := a1
max := a1
**for** i := 2 to n
        **if** $a_i$ < min **then** min := $a_i$
        **else** if $a_i$ > max **then** max := $a_i$
m := max - min
```

Comparisons: 2n - 2

Time complexity is O(n).

Let us get into…

Number Theory

Introduction to Number Theory

Number theory is about **integers** and their properties.

We will start with the basic principles of

- divisibility,
- greatest common divisors,
- least common multiples, and
- modular arithmetic

and look at some relevant algorithms.

Division

If a and b are integers with a $\neq$ 0, we say that a **divides** b if there is an integer c so that b = ac.

When a divides b we say that a is a **factor** of b and that b is a **multiple** of a.

The notation **a | b** means that a divides b.

We write **a *χ* b** when a does not divide b (see book for correct symbol).

Divisibility Theorems

For integers a, b, and c it is true that

- if a | b and a | c, then a | (b + c)
  Example: 3 | 6 and 3 | 9, so 3 | 15.

- if a | b, then a | bc for all integers c
  Example: 5 | 10, so 5 | 20, 5 | 30, 5 | 40, …

- if a | b and b | c, then a | c
  Example: 4 | 8 and 8 | 24, so 4 | 24.

Primes

A positive integer p greater than 1 is called prime if the only positive factors of p are 1 and p.
Note: 1 is not a prime

A positive integer that is greater than 1 and is not prime is called composite.

The fundamental theorem of arithmetic:

Every positive integer can be written **uniquely** as the **product of primes**, where the prime factors are written in order of increasing size.

Primes

Examples:

$15 = \quad 3 \cdot 5$

$48 = \quad 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 2^4 \cdot 3$

$17 = \quad 17$

$100 \quad 2 \cdot 2 \cdot 5 \cdot 5 = 2^2 \cdot 5^2$

$512 \quad 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^9$

$515 \quad 5 \cdot 103$

$28 = \quad 2 \cdot 2 \cdot 7$

Primes

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

This is easy to see: if n is a composite integer, it must have at least two prime divisors. Let the largest two be p_1 and p_2. Then $p_1 \cdot p_2 <= n$.

p_1 and p_2 cannot both be greater than $\sqrt{n}$, because then $p_1 \cdot p_2 > n$.

The Division Algorithm

Let **a** be an integer and **d** a positive integer. Then there are unique integers **q** and **r**, with $\mathbf{0 \leq r < d}$, such that $\mathbf{a = dq + r}$.

In the above equation,

- **d** is called the divisor,
- **a** is called the dividend,
- **q** is called the quotient, and
- **r** is called the remainder.

The Division Algorithm

Example:

When we divide 17 by 5, we have

$17 = 5 \cdot 3 + 2.$

- 17 is the dividend,
- 5 is the divisor,
- 3 is called the quotient, and
- 2 is called the remainder.

The Division Algorithm

Another example:

What happens when we divide -11 by 3 ?

Note that the remainder cannot be negative.

$-11 = 3 \cdot (-4) + 1.$

- -11 is the dividend,
- 3 is the divisor,
- -4 is called the quotient, and
- 1 is called the remainder.

Greatest Common Divisors

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).

Example 1: What is gcd(48, 72) ?

The positive common divisors of 48 and 72 are 1, 2, 3, 4, 6, 8, 12, 16, and 24, so gcd(48, 72) = 24.

Example 2: What is gcd(19, 72) ?

The only positive common divisor of 19 and 72 is 1, so gcd(19, 72) = 1.

Greatest Common Divisors

Using prime factorizations:

$a = p_1^{a_1}\ p_2^{a_2} \ldots p_n^{a_n}$, $b = p_1^{b_1}\ p_2^{b_2} \ldots p_n^{b_n}$,

where $p_1 < p_2 < \ldots < p_n$ and $a_i, b_i \in \mathbf{N}$ for $1 \leq i \leq n$

$\gcd(a, b) = p_1^{\min(a_1, b_1)}\ p_2^{\min(a_2, b_2)} \ldots p_n^{\min(a_n, b_n)}$

Example:

$a = 60 \quad 2^2\ 3^1\ 5^1$

$=$

$b = 54 \quad 2^1\ 3^3\ 5^0$

$=$

$\gcd(a, b) \quad 2^1\ 3^1\ 5^0 = 6$

$=$

Relatively Prime Integers

Definition:

Two integers a and b are **relatively prime** if gcd(a, b) = 1.

Examples:

Are 15 and 28 relatively prime?

Yes, gcd(15, 28) = 1.

Are 55 and 28 relatively prime?

Yes, gcd(55, 28) = 1.

Are 35 and 28 relatively prime?

No, gcd(35, 28) = 7.

Relatively Prime Integers

Definition:

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

Examples:

Are 15, 17, and 27 pairwise relatively prime?

No, because gcd(15, 27) = 3.

Are 15, 17, and 28 pairwise relatively prime?

Yes, because gcd(15, 17) = 1, gcd(15, 28) = 1 and gcd(17, 28) = 1.

Least Common Multiples

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.

We denote the least common multiple of a and b by lcm(a, b).

Examples:

lcm(3, 7) 21

=lcm(4, 6) 12

=lcm(5, 10) 10

=

Least Common Multiples

Using prime factorizations:

$a = p_1^{a_1}\ p_2^{a_2} \ldots p_n^{a_n}$, $b = p_1^{b_1}\ p_2^{b_2} \ldots p_n^{b_n}$,
where $p_1 < p_2 < \ldots < p_n$ and $a_i, b_i \in \mathbf{N}$ for $1 \leq i \leq n$

$\mathrm{lcm}(a, b) = p_1^{\max(a_1,\, b_1)}\ p_2^{\max(a_2,\, b_2)} \ldots p_n^{\max(a_n,\, b_n)}$

Example:

$a = 60\ \ 2^2\ 3^1\ 5^1$

$b = 54\ \ 2^1\ 3^3\ 5^0$

$\mathrm{lcm}(a, b)\ \ 2^2\ 3^3\ 5^1 = 4 \cdot 27 \cdot 5 = 540$

=

GCD and LCM

a = 60 2^2 3^1 5^1
=

b = 54 2^1 3^3 5^0
=

gcd(a, b) $2^1\ 3^1\ 5^0$ = 6
=

lcm(a, b) $2^2\ 3^3\ 5^1$ = 540
=

Theorem: $a \cdot b$ gcd(a,b) $\cdot$ lcm(a, b)
=

Modular Arithmetic

Let a be an integer and m be a positive integer. We denote by **a mod m** the remainder when a is divided by m.

Examples:

9 mod 4   1

=

9 mod 3   0

=

9 mod 10   9

=

-13 mod 4   3

=

Congruences

Let a and b be integers and m be a positive integer. We say that **a is congruent to b modulo m** if
m divides a – b.

We use the notation **a ≡ b (mod m)** to indicate that a is congruent to b modulo m.

In other words:
a ≡ b (mod m) if and only if **a mod m = b mod m**.

Congruences

Examples:

Is it true that $46 \equiv 68 \pmod{11}$?
Yes, because $11 \mid (46 – 68)$.

Is it true that $46 \equiv 68 \pmod{22}$?
Yes, because $22 \mid (46 – 68)$.

For which integers z is it true that $z \equiv 12 \pmod{10}$?
It is true for any $z \in \{\ldots, -28, -18, -8, 2, 12, 22, 32, \ldots\}$

Theorem: Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that $a = b + km$.

Congruences

Theorem: Let m be a positive integer.
If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then
$a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

Proof:
We know that $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ implies that there are integers s and t with
$b = a + sm$ and $d = c + tm$.

Therefore,
$b + d = (a + sm) + (c + tm) = (a + c) + m(s + t)$
and
$bd = (a + sm)(c + tm) = ac + m(at + cs + stm)$.

Hence, $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

Congruences

Theorem: Let m be a positive integer. $a \equiv b \pmod{m}$ iff a mod m = b mod m.

Proof:

Let $a = mq1 + r1$, and $b = mq2 + r2$.

Only if part: a mod m = b mod m $\rightarrow$ $r1 = r2$, therefore

$a - b = m(q1 - q2)$, and $a \equiv b \pmod{m}$.

If part: $a \equiv b \pmod{m}$ implies

$$a - b = mq$$

$$mq1 + r1 - (mq2 + r2) = mq$$

$$r1 - r2 = m(q - q1 + q2).$$

Since $0 \leq r1, r2 < m$, $0 \leq |r1 - r2| < m$. The only multiple in that range is 0.

Therefore $r1 = r2$, and a mod m = b mod m.

The Euclidean Algorithm

The **Euclidean Algorithm** finds the **greatest common divisor** of two integers a and b.

For example, if we want to find gcd(287, 91), we **divide** 287 by 91:

$287 = 91 \cdot 3 + 14$

We know that for integers a, b and c,
if a | b and a | c, then a | (b + c).

Therefore, any divisor (including their gcd) of 287 and 91 must also be a divisor of $287 - 91 \cdot 3 = 14$.

Consequently, gcd(287, 91) = gcd(14, 91).

The Euclidean Algorithm

In the next step, we divide 91 by 14:

$91 = 14 \cdot 6 + 7$

This means that gcd(14, 91) = gcd(14, 7).

So we divide 14 by 7:

$14 = 7 \cdot 2 + 0$

We find that 7 | 14, and thus gcd(14, 7) = 7.

Therefore, gcd(287, 91) = 7.

The Euclidean Algorithm

In **pseudocode**, the algorithm can be implemented as follows:

```
**procedure** gcd(a, b: positive integers)
x := a
y := b
while y $\neq$ 0
begin
        r := x mod y
        x := y
        y := r
end {x is gcd(a, b)}
```

Representations of Integers

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

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

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

Example for b=10:

$859 = 8 \cdot 10^2 + 5 \cdot 10^1 + 9 \cdot 10^0$

Representations of Integers

Example for b=2 (binary expansion):

$(10110)_2 = 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^1 = (22)_{10}$

Example for b=16 (hexadecimal expansion):

(we use letters A to F to indicate numbers 10 to 15)

$(3A0F)_{16} = 3 \cdot 16^3 + 10 \cdot 16^2 + 15 \cdot 16^0 = (14863)_{10}$

Representations of Integers

How can we construct the base b expansion of an integer n?

First, divide n by b to obtain a quotient q_0 and remainder a_0, that is,

$n = bq_0 + a_0$, where $0 \leq a_0 < b$.

The remainder a_0 is the rightmost digit in the base b expansion of n.

Next, divide q_0 by b to obtain:

$q_0 = bq_1 + a_1$, where $0 \leq a_1 < b$.

a_1 is the second digit from the right in the base b expansion of n. Continue this process until you obtain a quotient equal to zero.

Representations of Integers

Example:

What is the base 8 expansion of $(12345)_{10}$?

First, divide 12345 by 8:

$12345 = 8 \cdot 1543 + 1$

$1543 = 8 \cdot 192 + 7$

$192 = 8 \cdot 24 + 0$

$24 = 8 \cdot 3 + 0$

$3 = 8 \cdot 0 + 3$

The result is: $(12345)_{10} = (30071)_8$.

Representations of Integers

```
**procedure** base_b_expansion(n, b: positive integers)
q := n
k := 0
**while** q $\neq$ 0
**begin**
        $a_k$ := q mod b
        q := $\lfloor q/b \rfloor$
        k := k + 1
**end**
{the base b expansion of n is $(a_{k-1} \ldots a_1 a_0)_b$ }
```

Addition of Integers

How do we (humans) add two integers?

```
          111        carry
Example:    7583
          +        4932
          ----------
          12515
```

```
                              1 1          carry
Binary expansions:              (1011)_2
                              + (1010)_2
                            ------------
                             (10101)_2
```

$$\begin{array}{r} (1011)_2 \\ +\ (1010)_2 \\ \hline (10101)_2 \end{array}$$

Addition of Integers

Let $a = (a_{n-1}a_{n-2}\ldots a_1a_0)_2$, $b = (b_{n-1}b_{n-2}\ldots b_1b_0)_2$.

How can we **algorithmically** add these two binary numbers?

First, add their rightmost bits:

$a_0 + b_0 = c_0 \cdot 2 + s_0$,

where s_0 is the **rightmost bit** in the binary expansion of a + b, and c_0 is the **carry**.

Then, add the next pair of bits and the carry:

$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$,

where s_1 is the **next bit** in the binary expansion of a + b, and c_1 is the carry.

Addition of Integers

Continue this process until you obtain c_{n-1}.

The leading bit of the sum is $s_n = c_{n-1}$.

The result is:

$a + b = (s_n s_{n-1} \ldots s_1 s_0)_2$

Addition of Integers

Example:

Add $a = (1110)_2$ and $b = (1011)_2$.

$a_0 + b_0 = 0 + 1 = 0 \cdot 2 + 1$, so that $c_0 = 0$ and $s_0 = 1$.

$a_1 + b_1 + c_0 = 1 + 1 + 0 = 1 \cdot 2 + 0$, so $c_1 = 1$ and $s_1 = 0$.

$a_2 + b_2 + c_1 = 1 + 0 + 1 = 1 \cdot 2 + 0$, so $c_2 = 1$ and $s_2 = 0$.

$a_3 + b_3 + c_2 = 1 + 1 + 1 = 1 \cdot 2 + 1$, so $c_3 = 1$ and $s_3 = 1$.

$s_4 = c_3 = 1$.

Therefore, $s = a + b = (11001)_2$.

Addition of Integers

```
procedure add(a, b: positive integers)
c := 0
for j := 0 to n-1
begin
        d := $\lfloor (a_j + b_j + c)/2 \rfloor$
        $s_j := a_j + b_j + c - 2d$
        c := d
end
$s_n := c$
{the binary expansion of the sum is $(s_n s_{n-1} \ldots s_1 s_0)_2$}
```