Cryptosystems and Symmetric Encryption/Decryption)

- Need for improved Security

Finite Fields and Number Theory

- will now introduce finite fields
- of increasing importance in cryptography
  - AES, Elliptic Curve, IDEA, Public Key
- concern operations on “numbers”
  - where what constitutes a “number” and the type of operations varies considerably
- start with concepts of groups, rings, fields from abstract algebra

Group

- a set of elements or “numbers”
- with some operation whose result is also in the set (closure)
- obeys:
  - associative law: $(a.b).c = a.(b.c)$
  - has identity e: $e.a = a.e = a$
  - has inverses a^{-1}: $a.a^{-1} = e$
- if commutative $a.b = b.a$
  - then forms an **abelian group**
- $x^n=x*x*\ldots*x$ (n times) for $n\in Z^+$
- $X^{-n}=(x^{-1})^{|n|}=x^{-1}*x^{-1}*x^{-1}*\ldots*x^{-1}$.(n times), for $n\in Z^-$

Cayley table of a group (* operation)

*	e	a	b	c
e	e	a	b	c
a	a	b	c	e
b	b	c	e	a
c	c	e	a	b

$a^1=a$, $a^2=b$, $a^3=c$ ve $a^4=e$ ‘
$b=a^2=a^6=a^{-2}$.
Each elements of {e,a,b,c} can be defined as a^n
a is a generator of group.

Cyclic Group

- define **exponentiation** as repeated application of operator
  - example: $a^3 = a.a.a$
- and let identity be: $e=a^0$
- a group is cyclic if every element is a power of some fixed element
  - ie $b = a^k$ for some a and every b in group
- a is said to be a generator of the group

Ring

- a set of "numbers"
- with two operations (addition and multiplication) which form: {R,+,X)
- an abelian group with addition operation
- and multiplication:
  - has closure
  - is associative
  - distributive over addition: `a(b+c) = ab + ac`
- if multiplication operation is commutative, it forms a **commutative ring**
- if multiplication operation has an identity and no zero divisors, it forms an **integral domain**

Ring

- (H_1)Closure, if $a,b \in R$, $ab \in R$.
- (H_2) associative law. For $\forall a,b,c \in R$, $a(bc) = (ab)c$.
- (H_3) Distributive,, $a(b+c) = ab + ac$, $(a+b)c = ac +bc$ for $\forall a,b,c \in R$.
- (H_4) Commutative., $ab = ba$, for $\forall a,b \in R$.
- integral domain.
- (H_5) ineffective elm. $a1 = 1a =a$ for $\forall a \in R$.
- (H_6) Not zero divider if $ab=0$, $a=0$ or $b=0$, for $\forall a,b \in R$

Field

- a set of numbers
- with two operations which form:
  - abelian group for addition
  - abelian group for multiplication (ignoring 0)
  - ring
- have hierarchy with more axioms/laws
  - group -> ring -> field

Field Axioms

- (F,+,X) , a *F* field is a set proving the following axioms with addition and multiplication for $\forall$a,b,c $\in$ F.
- Group (G_1-G_5) and Ring axioms(H_1-H_6)
- integral domain.
- (H_7) Multiplicative invers . For $\forall$a $\in$F (except zero) $a^{-1} \in$F and $aa^{-1}= (a^{-1})a =1$.

Modular Arithmetic

- define **modulo operator** "`a mod n`" to be remainder when a is divided by n
- use the term **congruence** for: `a = b mod n`
  - when divided by *n,* a & b have same remainder
  - eg. 100 = 34 mod 11
- b is called a **residue** of a mod n
  - since with integers can always write: `a = qn + b`
  - usually chose smallest positive remainder as residue
    - ie. `0 <= b <= n-1`
  - process is known as **modulo reduction**
    - `eg. -12 mod 7 = -5 mod 7 = 2 mod 7 = 9 mod 7`

Divisors

- say a non-zero number `b` **divides** `a` if for some `m` have `a=mb` (`a,b,m` all integers)
- that is `b` divides into `a` with no remainder
- denote this `b|a`
- and say that `b` is a **divisor** of `a`
- eg. all of 1,2,3,4,6,8,12,24 divide 24
- if $a|1$, $a= \pm 1$
- if $a|b$ and $b|a$, $a= \pm b$.
- any $b \neq 0$ divides zero.
- if, $b|g$ and $b|h$, for any integer m,n $b|(mg +nh)$

Modular Arithmetic Operations

- is 'clock arithmetic'
- uses a finite number of values, and loops back from either end
- modular arithmetic is when do addition & multiplication and modulo reduce answer
- can do reduction at any point, ie
  - `a+b mod n ≡ [a mod n + b mod n] mod n`
- if, $n|(a-b)$ then $a \equiv b$ ***mod*** n .
- $a \equiv b$ ***mod*** n , means $b \equiv a$ ***mod*** n .
- $a \equiv b$ ***mod*** n and $b \equiv c$ ***mod*** n, means $a \equiv c$ ***mod n***

Modulo 8 Addition Example

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Properties of Mod operation

- **Add** $(a+b) \bmod n \equiv [(a \bmod n) + (b \bmod n)] \bmod n$
- **Sub** $(a-b) \bmod n \equiv [(a \bmod n) - (b \bmod n)] \bmod n$
- **Mult.** $a x b \bmod n \equiv [(a \bmod n) \times (b \bmod n)] \bmod n$
- Derived with repeated addition
- Neither a nor b not equal zero may be $a.b \text{ modn} = 0$
  - Ex. $2.5 \bmod 10$
- **Div.** $a/b \bmod n$
- Like multiply inverse of b: $a/b \equiv a.b^{-1} \bmod n$
- İf n prime there is $b^{-1} \bmod n$ and $b.b^{-1} \equiv 1 \bmod n$
  - Ex. $2.3 \equiv 1 \bmod 5$ so,$4/2=4.3 \equiv 2 \bmod 5$.

Modular Arithmetic

- can do modular arithmetic with any group of integers: $Z_n = \{0, 1, \ldots, n-1\}$
- Z_n is defined as residue class $[r] = \{ a : a \text{ integer}; \text{ such as } a \equiv r \bmod n$
- form a commutative ring for addition
- with a multiplicative identity
- note some peculiarities
  - if $(a+b) \equiv (a+c) \bmod n$
    then $b=c \bmod n$
  - but if $(a.b) \equiv (a.c) \bmod n$
    then $b \equiv c \bmod n$ only if a is relatively prime to n

$\in$

properties

property	description
Commutative	$(a + b) \bmod n \equiv (b + a) \bmod n$ $(a \times b) \bmod n \equiv (b \times a) \bmod n$
Associative	$[(a+b) + c] \bmod n \equiv [a + (b + c)] \bmod n$ $[(a \times b) \times c] \bmod n \equiv [a \times (b \times c)] \bmod n$
Distributive	$[a \times (b + c)] \bmod n \equiv [(a \times b) + (a \times c)] \bmod n$
Identity element	$(0 + a) \bmod n \equiv a \bmod n$ $(1 \times a) \bmod n \equiv a \bmod n$
Additive invers(-a)	For $\forall a \;\; Z_n$ there is a b such as; $a + b \equiv 0 \bmod n$.

if order of finite field is p^n, (p prime number)

This field called **Galois Field modulo p** and shown as **GF(p^n)**

Greatest Common Divisor (GCD)

- a common problem in number theory
- GCD (a,b) of a and b is the largest number that divides evenly into both a and b
  - eg GCD(60,24) = 12
- often want **no common factors** (except 1) and hence numbers are **relatively prime**
  - eg GCD(8,15) = 1
  - hence 8 & 15 are relatively prime

Euclidean Algorithm

- an efficient way to find the GCD(a,b)
- uses theorem that:
  - GCD(a,b) = GCD(b, a mod b)
- Euclidean Algorithm to compute GCD(a,b) is:
- ***GCD (a,n) is given by:***
- ***let g_o=n g_1=a***
- ***g_{i+1} = g_{i-1} mod gi***
- ***when g_i=0 then GCD(a,n) = gi-1***
- ex. Find GCD (56,98) ‘.
- g_0=98 g_1=56 g_2 = 98 mod 56 = 42 g_3 = 56 mod 42 = 14 g_4 = 42 mod 14 = 0 as a result GCD(56,98)=14

Example GCD(1970,1066)

```
1970 = 1 x 1066 + 904     gcd(1066, 904)
1066 = 1 x 904 + 162      gcd(904, 162)
904 = 5 x 162 + 94        gcd(162, 94)
162 = 1 x 94 + 68         gcd(94, 68)
94 = 1 x 68 + 26          gcd(68, 26)
68 = 2 x 26 + 16          gcd(26, 16)
26 = 1 x 16 + 10          gcd(16, 10)
16 = 1 x 10 + 6           gcd(10, 6)
10 = 1 x 6 + 4       gcd(6, 4)
6 = 1 x 4 + 2             gcd(4, 2)
4 = 2 x 2 + 0             gcd(2, 0)
```

Galois Fields

- finite fields play a key role in cryptography
- can show number of elements in a finite field **must** be a power of a prime p^n
- known as Galois fields
- denoted $GF(p^n)$
- in particular often use the fields:
  - $GF(p)$
  - $GF(2^n)$

Galois Fields GF(p)

- GF(p) is the set of integers {0,1, … , p-1} with arithmetic operations modulo prime p
- these form a finite field
  - since have multiplicative inverses
- hence arithmetic is “well-behaved” and can do addition, subtraction, multiplication, and division without leaving the field GF(p)

GF(7) Multiplication Example

$\times$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

GF(7)additive and multip. inverse

For $\forall w \in Z_p$ there is z in Z_p and $w x Z = 1 \bmod p$

w	-w	w^{-1}
0	0	-
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

Polynomial Arithmetic

- can compute using polynomials

  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = \sum a_i x^i$

  - nb. not interested in any specific value of x
  - which is known as the indeterminate
- several alternatives available
  - ordinary polynomial arithmetic
  - poly arithmetic with coords mod p
  - poly arithmetic with coords mod p and polynomials mod m(x)

Ordinary Polynomial Arithmetic

- add or subtract corresponding coefficients
- multiply all terms by each other
- Eg
- $f(x) = \sum a_i x^{i\ (i=1,n)}$, $g(x) = \sum b_j x^{j\ (j=1,m)}$

  let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 - x + 1$

  $f(x) + g(x) = x^3 + 2x^2 - x + 3$

  $f(x) - g(x) = x^3 + x + 1$

  $f(x) \times g(x) = x^5 + 3x^2 - 2x + 2$

Polynomial Arithmetic with Modulo Coefficients

- when computing value of each coefficient do calculation modulo some value
  - forms a polynomial ring
- could be modulo any prime
- but we are most interested in mod 2
  - ie all coefficients are 0 or 1
  - eg. let $f(x) = x^3 + x^2$ and $g(x) = x^2 + x + 1$

$$f(x) + g(x) = x^3 + x + 1$$

$$f(x) \text{ x } g(x) = x^5 + x^2$$

Polynomial Division

- can write any polynomial in the form:
  - $f(x) = q(x)\, g(x) + r(x)$
  - can interpret $r(x)$ as being a remainder
  - $r(x) = f(x) \bmod g(x)$
- if have no remainder say $g(x)$ divides $f(x)$
- if $g(x)$ has no divisors other than itself & 1 say it is **irreducible** (or prime) polynomial
- arithmetic modulo an irreducible polynomial forms a field

Polynomial GCD

- can find greatest common divisor for polys
  - *c(x)* = GCD(*a(x), b(x)*) if *c(x)* is the poly of greatest degree which divides both *a(x), b(x)*
- can adapt Euclid's Algorithm to find it:

EUCLID[$a(x)$, $b(x)$]

1. $A(x) = a(x)$; $B(x) = b(x)$

2. if $B(x) = 0$ **return** $A(x) = \gcd[a(x), b(x)]$

3. $R(x) = A(x) \bmod B(x)$

4. $A(x) \leftarrow B(x)$

5. $B(x) \leftarrow R(x)$

6. goto 2

Modular Polynomial Arithmetic

- can compute in field $GF(2^n)$
  - polynomials with coefficients modulo 2
  - whose degree is less than n
  - hence must reduce modulo an irreducible poly of degree n (for multiplication only)
- form a finite field
- can always find an inverse
  - can extend Euclid's Inverse algorithm to find

Example GF(2^3)

Table 4.6 Polynomial Arithmetic Modulo ($x^3 + x + 1$)

	+	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

(a) Addition

	×	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

(b) Multiplication

Computational Considerations

- since coefficients are 0 or 1, can represent any such polynomial as a bit string
- addition becomes XOR of these bit strings
- multiplication is shift & XOR
  - cf long-hand multiplication
- modulo reduction done by repeatedly substituting highest power with remainder of irreducible poly (also shift & XOR)

Computational Example

- in GF(2^3) have (x^2+1) is 101_2 & (x^2+x+1) is 111_2
- so addition is
  - $(x^2+1) + (x^2+x+1) = x$
  - 101 XOR 111 = 010_2
- and multiplication is
  - $(x+1).(x^2+1) = x.(x^2+1) + 1.(x^2+1)$
    $= x^3+x+x^2+1 = x^3+x^2+x+1$
  - 011.101 = (101)<<1 XOR (101)<<0 =
    1010 XOR 101 = 1111_2
- polynomial modulo reduction (get q(x) & r(x)) is
  - (x^3+x^2+x+1) mod $(x^3+x+1) = 1.(x^3+x+1) + (x^2) = x^2$
  - 1111 mod 1011 = 1111 XOR 1011 = 0100_2

Using a Generator

- equivalent definition of a finite field
- a **generator** g is an element whose powers generate all non-zero elements
  - in F have 0, g^0, g^1, …, g^{q-2}
- can create generator from **root** of the irreducible polynomial
- then implement multiplication by adding exponents of generator

Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
- eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
61 67 71 73 79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179 181 191
193 197 199
```

Prime Factorisation

- to **factor** a number `n` is to write it as a product of other numbers: `n=a x b x c`
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorisation** of a number `n` is when its written as a product of primes
  - eg. `91=7x13 ; 3600=`2^4`x`3^2`x`5^2

$$a = \prod_{p \in \mathrm{P}} p^{a_p}$$

Relatively Prime Numbers & GCD

- two numbers `a, b` are **relatively prime** if have **no common divisors** apart from 1
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor
- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
  - eg. $300=2^1\text{x}3^1\text{x}5^2$ $18=2^1\text{x}3^2$ hence $\text{GCD}(18,300)=2^1\text{x}3^1\text{x}5^0=6$

Fermat's Theorem

- $a^{p-1} = 1 \pmod{p}$
  - where p is prime and $\gcd(a,p)=1$
- also known as Fermat's Little Theorem
- also $a^p = a \pmod{p}$
- useful in public key and primality testing

- If elements of Z_p multiply with {0,1,....,(p-1)} a , modulo p residues sequences of $Z_{p\,.}$ In addition, a x 0 = 0mod p ., So array of { $a$mod p, 2$a$modp, ...,(p-1)$a$mod p } is (p-1) number {0,1,....,(p-1)}.
- ax2ax … x ((p-1)a) = [(a modp) x (2$a$modp) x .....x((p-1)a modp]modp
- = [1 x 2 x ... x (p-1)] modp
- = (p-1)! modp
- But, a x 2a x ......x((p-1)a) = (p-1)!a^{p-1} .
- So, (p-1)!a^{p-1} = (p-1)! Modp. Here we can cancel (p-1)! ‘As a result:
- a^{p-1} = 1 modp

Euler Totient Function $\varnothing(\mathtt{n})$

- when doing arithmetic modulo n
- **complete set of residues** is: `0..n-1`
- **reduced set of residues** is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}
- number of elements in reduced set of residues is called the **Euler Totient Function ø(n)**

Euler Totient Function $\varnothing(n)$

- to compute ø(n) need to count number of residues to be excluded
- in general need prime factorization, but
  - for p (p prime) $\varnothing(p) = p-1$
  - for p.q (p,q prime) $\varnothing(pq) = (p-1) x (q-1)$
- eg.

  $\varnothing(37) = 36$

  $\varnothing(21) = (3-1)x(7-1) = 2x6 = 12$

Primitive Roots

- from Euler's theorem have $a^{\varnothing(n)} \bmod n=1$
- consider $a^m=1 \pmod n$, $GCD(a,n)=1$
  - must exist for $m = \varnothing(n)$ but may be smaller
  - once powers reach m, cycle will repeat
- if smallest is $m = \varnothing(n)$ then a is called a **primitive root**
- if p is prime, then successive powers of a "generate" the group $\bmod p$
- these are useful but relatively hard to find

$\phi(n)$ values for n=30

n	$\phi(n)$	**n**	$\phi(n)$	**n**	$\phi(n)$
1	1	11	10	21	12
2	1	12	4	22	10
3	2	13	12	23	22
4	2	14	6	24	8
5	4	15	8	25	20
6	2	16	8	26	12
7	6	17	16	27	18
8	4	18	6	28	12
9	6	19	18	29	28
10	4	20	8	30	8

Euler's Theorem

- a generalisation of Fermat's Theorem
- $a^{\varnothing(n)} = 1 \pmod{n}$
  - for any a,n where gcd(a,n)=1
- eg.

  $a=3; n=10;\ \varnothing(10)=4;$

  hence $3^4 = 81 = 1 \bmod 10$

  $a=2; n=11;\ \varnothing(11)=10;$

  hence $2^{10} = 1024 = 1 \bmod 11$

Primality Testing

- often need to find large prime numbers
- traditionally **sieve** using **trial division**
  - ie. divide by all numbers (primes) in turn less than the square root of the number
  - only works for small numbers
- alternatively can use statistical primality tests based on properties of primes
  - for which all primes numbers satisfy property
  - but some composite numbers, called pseudo-primes, also satisfy the property
- can use a slower deterministic primality test

Miller Rabin Algorithm

- a test based on Fermat's Theorem
- algorithm is:

TEST (n) is:

1. Find integers k, q, k > 0, q odd, so that $(n-1)=2^k q$
2. Select a random integer a, $1<a<n-1$
3. **if** $a^q \bmod n = 1$ **then** return ("maybe prime");
4. **for** j = 0 **to** k – 1 **do**
   5. **if** ($a^{2^j q} \bmod n = n-1$)
      then return(" maybe prime ")
6. return ("composite")

Probabilistic Considerations

- if Miller-Rabin returns “composite” the number is definitely not prime
- otherwise is a prime or a pseudo-prime
- chance it detects a pseudo-prime is < $^1/_4$
- hence if repeat test with different random a then chance n is prime after t tests is:
  - Pr(n prime after t tests) = $1-4^{-t}$
  - eg. for t=10 this probability is > 0.99999

Prime Distribution

- prime number theorem states that primes occur roughly every (`ln n`) integers
- but can immediately ignore evens
- so in practice need only test `0.5 ln(n)` numbers of size `n` to locate a prime
  - note this is only the "average"
  - sometimes primes are close together
  - other times are quite far apart

Discrete Logarithms

- the inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- that is to find x such that $y \equiv g^x \pmod{p}$
- this is written as $x \equiv \log_g y \pmod{p}$
- if g is a primitive root then it always exists, otherwise it may not, eg.

  $x \equiv \log_3 4 \bmod 13$ has no answer

  $x \equiv \log_2 3 \bmod 13 = 4$ by trying successive powers
- while exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem

Chinese Remainder Theorem

- used to speed up modulo computations
- if working modulo a product of numbers
  - eg. $\mathrm{mod}\ M = m_1 m_2 .. m_k$
- Chinese Remainder theorem lets us work in each moduli m_i separately
- since computational cost is proportional to size, this is faster than working in the full modulus M

- **Chinese Remainder Theorem:** For relatively prime moduli *m* and *n*, the congruences
  $x \equiv a \pmod{m}$
  $x \equiv b \pmod{n}$

have a unique solution *x* modulo *mn*. Our example problem would have a unique solution modulo .

- $x \equiv 2 \pmod{3}$
  $x \equiv 3 \pmod{5}$

Our example problem would have a unique solution modulo 27.16

Chinese Remainder Theorem

- can implement CRT in several ways
- to compute $\mathtt{A}$ $(\mathtt{mod\ M})$
  - first compute all $\mathtt{a_i = A \ mod \ m_i}$ separately
  - determine constants $\mathtt{c_i}$ below, where $\mathtt{M_i = M/m_i}$
  - then combine results to get answer using:

$$A \equiv \left(\sum_{i=1}^{k} a_i c_i \right) (\bmod M)$$

$$c_i = M_i \times (M_i^{-1} \bmod m_i) \quad \text{for } 1 \le i \le k$$

- **Chinese Remainder Theorem example:**
- $x \equiv 2 \pmod 3$

  $x \equiv 3 \pmod 5$

Sol: $t \in Z$, $x = 3.t+2$

$3.t+2 \equiv 3 \pmod 5$

$3.t \equiv 1 \pmod 5$ (if we multiply eqn. by 2)

$t \equiv 2 \pmod 5$, then $t = 5.s+2$

$x = 3.t+2 = 3(5s+2)+2 = 15s+8$

For $s=0$, $x=8$.

Computational Complexity Theory

- study of how hard a problem is to solve in general
- allows classification of types of problems
- some problems intrinsically harder than others, eg
  - multiplying numbers $O(n^2)$
  - multiplying matrices $O(n^{(2)(2n-1)})$ (n^(2)(2n-1))
  - solving crossword O(26^(n))
  - recognizing primes O(n^(log log n))
- deal with worst case complexity
  - may on average be easier

Computational Complexity Theory

- an **instance** of a problem is a particular case of a general problem
- the **input length** of a problem is the number **n** of symbols used to characterize a particular instance of it
- the **order** of a function **f(n)** - is some **O(g(n))** of some function **g(n)** s.t.
  - $f(n) \le c \cdot |g(n)|$, for all $n \ge 0$, for some c

Polynomial Time

- An algorithm is said to be **polynomial time** if its running time is upper bounded by a polynomial in the size of the input for the algorithm, i.e., $T(n) = O(n^k)$ for some constant *k*..
- The quicksort sorting algorithm on *n* integers performs at most An^2 operations for some constant *A*. Thus it runs in time $O(n^2)$ and is a polynomial time algorithm.
- Maximum matchings in graphs can be found in polynomial time

Computational Complexity Theory

- a **polynomial time** algorithm (**P**) is one which solves any instance of a particular problem in a length of time O(p(n)), where p is some polynomial on input length
- an **exponential time** algorithm (**E**) - is one whose solution time is not so bounded
- a **non-deterministic polynomial time** algorithm (**NP**) - is one for which any guess at the solution of an instance of the problem may be checked for validity in polynomial time

Computational Complexity Theory

- **NP-complete** problems - are a subclass of NP problems for which it is known that if any such problem has a polynomial time solution, then **all NP** problems have polynomial solutions. These are thus the **hardest NP** problems
- **Co-NP** problems - are the complements of **NP** problems, to prove a guess at a solution of a Co-NP problem may well require an exhaustive search of the solution space

Computational Complexity Theory