

# Complexity Theory in Cryptography

Dhananjoy Dey

Indian Institute of Information Technology, Lucknow  
[ddey@iiitl.ac.in](mailto:ddey@iiitl.ac.in)

December 23, 2022

# Outline

- 1 Introduction
  - Time Estimation
- 2 Notations
- 3 Complexity Classes
  - Time Complexity

# Definition

- *Computational complexity theory* is the study of the minimal resources needed to solve computational problems.

# Definition

- *Computational complexity theory* is the study of the minimal resources needed to solve computational problems.
  - Two fundamental questions:
    - ❶ Is a problem  $P$  intrinsically “*easy*” or “*difficult*” to solve?

# Definition

- *Computational complexity theory* is the study of the minimal resources needed to solve computational problems.
  - Two fundamental questions:
    - ❶ Is a problem  $P$  intrinsically “*easy*” or “*difficult*” to solve?
    - ❷ Given two problems,  $P_1$  and  $P_2$ , which is easier to solve?

# Definition

- *Computational complexity theory* is the study of the minimal resources needed to solve computational problems.
  - Two fundamental questions:
    - ❶ Is a problem  $P$  intrinsically “*easy*” or “*difficult*” to solve?
    - ❷ Given two problems,  $P_1$  and  $P_2$ , which is easier to solve?
- **Running time** - the number of basic (or primitive) operations (or steps) taken by an algorithm.

# Definition

- *Computational complexity theory* is the study of the minimal resources needed to solve computational problems.
  - Two fundamental questions:
    - ❶ Is a problem  $P$  intrinsically “*easy*” or “*difficult*” to solve?
    - ❷ Given two problems,  $P_1$  and  $P_2$ , which is easier to solve?
- **Running time** - the number of basic (or primitive) operations (or steps) taken by an algorithm.
  - The running time of an algorithm usually depends on **the size of the input**.

# Definition

- *Computational complexity theory* is the study of the minimal resources needed to solve computational problems.
  - Two fundamental questions:
    - ❶ Is a problem  $P$  intrinsically “*easy*” or “*difficult*” to solve?
    - ❷ Given two problems,  $P_1$  and  $P_2$ , which is easier to solve?
- **Running time** - the number of basic (or primitive) operations (or steps) taken by an algorithm.
  - The running time of an algorithm usually depends on **the size of the input**.
- **Space complexity** - to measure the amount of temporary storage used when performing a computational task.



# Representation of a Number

- **Numbers in different bases**

# Representation of a Number

- **Numbers in different bases**

Any number  $n$  between  $b^{k-1}$  and  $b^k$  is a  $k$ -digit number to the base  $b$ .

# Representation of a Number

- **Numbers in different bases**

Any number  $n$  between  $b^{k-1}$  and  $b^k$  is a  $k$ -digit number to the base  $b$ .

- **Number of digits**

# Representation of a Number

- **Numbers in different bases**

Any number  $n$  between  $b^{k-1}$  and  $b^k$  is a  $k$ -digit number to the base  $b$ .

- **Number of digits**

$$= \lceil \log_b n \rceil + 1.$$

# Representation of a Number

- **Numbers in different bases**

Any number  $n$  between  $b^{k-1}$  and  $b^k$  is a  $k$ -digit number to the base  $b$ .

- **Number of digits**

$$= \lceil \log_b n \rceil + 1.$$

- **Number of bits**

# Representation of a Number

- **Numbers in different bases**

Any number  $n$  between  $b^{k-1}$  and  $b^k$  is a  $k$ -digit number to the base  $b$ .

- **Number of digits**

$$= \lceil \log_b n \rceil + 1.$$

- **Number of bits**

$$= \lceil \log_2 n \rceil + 1 \approx \lceil 1.44 \times \ln n \rceil + 1.$$

# Size of Some Mathematical Objects

## Example

- 1 If  $\mathbf{A} = [\mathbf{a}_{ij}]_{r \times s}$  is a matrix with  $r$  rows,  $s$  columns, where  $\mathbf{a}_{ij} \in \mathbb{Z}_n$ , then the size of  $\mathbf{A}$

# Size of Some Mathematical Objects

## Example

- ① If  $\mathbf{A} = [\mathbf{a}_{ij}]_{r \times s}$  is a matrix with  $r$  rows,  $s$  columns, where  $\mathbf{a}_{ij} \in \mathbb{Z}_n$ , then the size of  $\mathbf{A}$

$$= rs(1 + \lceil \log_2 n \rceil) \text{ bits.}$$



# Size of Some Mathematical Objects

## Example

- ① If  $\mathbf{A} = [\mathbf{a}_{ij}]_{r \times s}$  is a matrix with  $r$  rows,  $s$  columns, where  $\mathbf{a}_{ij} \in \mathbb{Z}_n$ , then the size of  $\mathbf{A}$

$$= rs(1 + \lceil \log_2 n \rceil) \text{ bits.}$$

- ② If  $f$  is a polynomial of degree  $d$ , each coefficient  $\in \mathbb{Z}_n$ , then the size of  $f$

# Size of Some Mathematical Objects

## Example

- ① If  $\mathbf{A} = [\mathbf{a}_{ij}]_{r \times s}$  is a matrix with  $r$  rows,  $s$  columns, where  $\mathbf{a}_{ij} \in \mathbb{Z}_n$ , then the size of  $\mathbf{A}$

$$= rs(1 + \lceil \log_2 n \rceil) \text{ bits.}$$

- ② If  $f$  is a polynomial of degree  $d$ , each coefficient  $\in \mathbb{Z}_n$ , then the size of  $f$

$$= (d + 1)(1 + \lceil \log_2 n \rceil) \text{ bits.}$$

# Outline

- 1 Introduction
  - Time Estimation
- 2 Notations
- 3 Complexity Classes
  - Time Complexity

# Number of Steps for Doing Arithmetic

Number of steps required to add 2 integers  $a$  &  $b$

# Number of Steps for Doing Arithmetic

**Number of steps required to add 2 integers  $a$  &  $b$**

**Input:** integers  $a \geq b \geq 0$

**Output:**  $a + b$

**Algorithm:**

```
while ( $b \neq 0$ ){  
     $a = a + +$   
     $b = b - -$   
}  
output  $a$ 
```

# Number of Steps for Doing Arithmetic

**Number of steps required to add 2 integers  $a$  &  $b$**

**Input:** integers  $a \geq b \geq 0$

**Output:**  $a + b$

**Algorithm:**

```
while ( $b \neq 0$ ){  
     $a = a + +$   
     $b = b - -$   
}  
output  $a$ 
```

Number of operations

# Number of Steps for Doing Arithmetic

**Number of steps required to add 2 integers  $a$  &  $b$**

**Input:** integers  $a \geq b \geq 0$

**Output:**  $a + b$

**Algorithm:**

```
while ( $b \neq 0$ ){  
     $a = a + +$   
     $b = b - -$   
}  
output  $a$ 
```

Number of operations =  $3b + 1$

# Bit Operation for Doing Arithmetic

Number of bit operations required to add 2  $k$ -bit integers  $n$  &  $m$



# Bit Operation for Doing Arithmetic

Number of bit operations required to add 2  $k$ -bit integers  $n$  &  $m$

- i. Look at the top and bottom bit and also at whether there's a carry above the top bit.
- ii. If both bits are 0 and there is no carry, then put down 0.
- iii. If either both bits are 0 and there is a carry; or one of the bits is 0, the other is 1 and there is no carry, then put down 1.
- iv. If either one of the bits is 0, the other is 1, and there is a carry; or both bits are 1 and there is no carry then put down 0, put a carry in the next column.
- v. If both bits are 1 and there is a carry, then put down 1, put a carry in the next column.

# Bit Operation for Doing Arithmetic

Number of bit operations required to add 2  $k$ -bit integers  $n$  &  $m$

- i. Look at the top and bottom bit and also at whether there's a carry above the top bit.
- ii. If both bits are 0 and there is no carry, then put down 0.
- iii. If either both bits are 0 and there is a carry; or one of the bits is 0, the other is 1 and there is no carry, then put down 1.
- iv. If either one of the bits is 0, the other is 1, and there is a carry; or both bits are 1 and there is no carry then put down 0, put a carry in the next column.
- v. If both bits are 1 and there is a carry, then put down 1, put a carry in the next column.

$\text{Time}(n + m) = k\text{-bit operations.}$

# Bit Operation for Doing Arithmetic

Number of bit operations required to add 2  $k$ -bit integers  $n$  &  $m$

**Input:**  $n = n_k n_{k-1} \cdots n_2 n_1$  &  $m = m_k m_{k-1} \cdots m_2 m_1$

**Output:**  $n + m$  in binary.

**Algorithm:**  $c \leftarrow 0$

for( $i = 1$  to  $k$ ) {

    if  $sum(n_i, m_i, c) = 1$  or  $3$

        then  $d_i \leftarrow 1$

        else  $d_i \leftarrow 0$

    if  $sum(n_i, m_i, c) \geq 2$

        then  $c \leftarrow 1$

        else  $c \leftarrow 0$  }

if  $c = 1$  then output  $1d_k d_{k-1} \cdots d_2 d_1$

else output  $d_k d_{k-1} \cdots d_2 d_1$ .

# Bit Operation for Doing Arithmetic

- Number of bit operations required to add 2  $k$ -bit integers  $n$  &  $m$

iv. If either

- one of the bits is 0, the other is 1, and there is a carry, or
- both bits are 1 and there is no carry,

then put down 0, put a carry in the next column.

- v. If both bits are 1 and there is a carry, then put down 1, put a carry in the next column.

# Bit Operation for Doing Arithmetic

- Number of bit operations required to add 2  $k$ -bit integers  $n$  &  $m$

iv. If either

- one of the bits is 0, the other is 1, and there is a carry, or
- both bits are 1 and there is no carry,

then put down 0, put a carry in the next column.

- v. If both bits are 1 and there is a carry, then put down 1, put a carry in the next column.

$\text{Time}(n + m) = k\text{-bit operations.}$

# Bit Operation for Doing Arithmetic

- Number of bit operations required to multiply a  $k$ -bit integer  $n$  by an  $\ell$ -bit integer  $m$

# Bit Operation for Doing Arithmetic

- Number of bit operations required to multiply a  $k$ -bit integer  $n$  by an  $\ell$ -bit integer  $m$ 
  - i. at most  $\ell$  rows can be obtained
  - ii. each row consists of a copy of  $n$  shifted to the left a certain distance
  - iii. suppose there are  $\ell' \leq \ell$  rows.
  - iv. multiplication task can be broken down into  $\ell' - 1$  additions
  - v. moving down from the  $2^{nd}$  row to the  $\ell'^{th}$  row, adding each new row to the partial sum of all of the earlier rows
  - vi. each addition takes at most  $k$ -bit operations
  - vii. total number of bit operations is at most  $\ell \times k$ .

# Bit Operation for Doing Arithmetic

- Number of bit operations required to multiply a  $k$ -bit integer  $n$  by an  $\ell$ -bit integer  $m$ 
  - i. at most  $\ell$  rows can be obtained
  - ii. each row consists of a copy of  $n$  shifted to the left a certain distance
  - iii. suppose there are  $\ell' \leq \ell$  rows.
  - iv. multiplication task can be broken down into  $\ell' - 1$  additions
  - v. moving down from the  $2^{nd}$  row to the  $\ell'^{th}$  row, adding each new row to the partial sum of all of the earlier rows
  - vi. each addition takes at most  $k$ -bit operations
  - vii. total number of bit operations is at most  $\ell \times k$ .

Time( $n \times m$ )  $< k\ell$ -bit operations.



# Bit Operation for Doing Arithmetic

- Number of bit operations required to multiply two  $n$ -bit integers  $x$  &  $y$

# Bit Operation for Doing Arithmetic

- Number of bit operations required to multiply two  $n$ -bit integers  $x$  &  $y$
- Let  $n = 2t$ . Then

$$x = 2^t x_1 + x_0 \text{ \& } y = 2^t y_1 + y_0$$

# Bit Operation for Doing Arithmetic

- Number of bit operations required to multiply two  $n$ -bit integers  $x$  &  $y$
- Let  $n = 2t$ . Then

$$x = 2^t x_1 + x_0 \text{ \& } y = 2^t y_1 + y_0$$

•

$$x.y = u_2.2^{2t} + u_1.2^t + u_0$$

where  $u_0 = x_0.y_0$ ,  $u_2 = x_1.y_1$  &  $u_1 = (x_0 + x_1).(y_0 + y_1) - u_0 - u_2$ .

# Bit Operation for Doing Arithmetic

## Complexity of Integer Multiplication

Method	Year	# Operations
Karatsuba Ofman	1962	$\leq C.n^{\log_2 3}$
Toom	1963	$\leq C.n^{1+\epsilon}$
Schönhage Strassen	1971	$\leq C.(n \log n \log \log n \dots 2^{\leq C_1 \log^* n})$ $\leq C.(n \log n \log \log n)$
Fürer	2007	$\leq C.(n \log n \cdot 2^{\leq C_1 \log^* n})$
Anindya De	2008	$\leq C.(n \log n \cdot 2^{\leq C_1 \log^* n})$

# Bit Operation for Doing Arithmetic

## Example

An upper bound for the number of bit operations required to compute  $n!$ .

# Bit Operation for Doing Arithmetic

## Example

An upper bound for the number of bit operations required to compute  $n!$ .

- i. At the  $(j - 1)^{th}$  step ( $j = 2, 3, \dots, n - 1$ ), you are multiplying  $j!$  by  $j + 1$ .
- ii.  $n - 2$  steps requires to compute  $n!$ , where each step involves multiplying a partial product by the next integer.
- iii. Product of  $n$   $k$ -bit integers will have at most  $nk$  bits.
- iv. At each step, we require multiplication of an integer with at most  $k$  bits by an integer with at most  $nk$  bits.
- v. The total number of bit operations is bounded by  $(n - 2)nk^2$ .

# Bit Operation for Doing Arithmetic

## Example

An upper bound for the number of bit operations required to compute  $n!$ .

- i. At the  $(j - 1)^{th}$  step ( $j = 2, 3, \dots, n - 1$ ), you are multiplying  $j!$  by  $j + 1$ .
- ii.  $n - 2$  steps requires to compute  $n!$ , where each step involves multiplying a partial product by the next integer.
- iii. Product of  $n$   $k$ -bit integers will have at most  $nk$  bits.
- iv. At each step, we require multiplication of an integer with at most  $k$  bits by an integer with at most  $nk$  bits.
- v. The total number of bit operations is bounded by  $(n - 2)nk^2$ .

$$\text{Time(to compute } n!) \leq n^2(\ln n)^2.$$

# The Traveling Salesman Problem



# The Traveling Salesman Problem

## Problem

Given a list of  $n$  cities,  $c_1, c_2, \dots, c_n$  and an  $n \times n$  symmetric matrix  $\mathbf{D}$  of distances, such that

$$D_{ij} = \text{distance from city } c_i \text{ to city } c_j,$$

determine an optimal shortest tour visiting each of the cities exactly once.

# The Traveling Salesman Problem

## Problem

Given a list of  $n$  cities,  $c_1, c_2, \dots, c_n$  and an  $n \times n$  symmetric matrix  $\mathbf{D}$  of distances, such that

$$D_{ij} = \text{distance from city } c_i \text{ to city } c_j,$$

determine an optimal shortest tour visiting each of the cities exactly once.

## Solution

Try all possible tours in turn and choose the shortest one.

# The Traveling Salesman Problem

## Problem

Given a list of  $n$  cities,  $c_1, c_2, \dots, c_n$  and an  $n \times n$  symmetric matrix  $\mathbf{D}$  of distances, such that

$$D_{ij} = \text{distance from city } c_i \text{ to city } c_j,$$

determine an optimal shortest tour visiting each of the cities exactly once.

## Solution

Try all possible tours in turn and choose the shortest one.

No. of possible tours

# The Traveling Salesman Problem

## Problem

Given a list of  $n$  cities,  $c_1, c_2, \dots, c_n$  and an  $n \times n$  symmetric matrix  $\mathbf{D}$  of distances, such that

$$D_{ij} = \text{distance from city } c_i \text{ to city } c_j,$$

determine an optimal shortest tour visiting each of the cities exactly once.

## Solution

Try all possible tours in turn and choose the shortest one.

No. of possible tours =  $n!$ .

# Big- $O$

## Definition

Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ ,  $g(x) > 0 \ \forall x \geq a$ , where  $a \in \mathbb{N}$ . Then  $f = O(g)$  means that  $\frac{f(x)}{g(x)}$  is bounded  $\forall x \geq a$ , i.e.,  $\exists$  a constant  $M > 0$  such that

$$|f(x)| \leq M \cdot g(x) \quad \forall x \geq a.$$

# Big- $O$

## Definition

Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ ,  $g(x) > 0 \forall x \geq a$ , where  $a \in \mathbb{N}$ . Then  $f = O(g)$  means that  $\frac{f(x)}{g(x)}$  is bounded  $\forall x \geq a$ , i.e.,  $\exists$  a constant  $M > 0$  such that

$$|f(x)| \leq M \cdot g(x) \quad \forall x \geq a.$$

## Example

Let  $f(n) = 2n^3 + 3n^2 + 4n + 5$  &  $g(n) = n^3$ .

# Big- $O$

## Definition

Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ ,  $g(x) > 0 \forall x \geq a$ , where  $a \in \mathbb{N}$ . Then  $f = O(g)$  means that  $\frac{f(x)}{g(x)}$  is bounded  $\forall x \geq a$ , i.e.,  $\exists$  a constant  $M > 0$  such that

$$|f(x)| \leq M.g(x) \quad \forall x \geq a.$$

## Example

Let  $f(n) = 2.n^3 + 3.n^2 + 4.n + 5$  &  $g(n) = n^3$ .

Then  $f = O(g)$ , for take  $a = 5$ ,  $M = 3$ .

The notation **Big  $O$**  represents an upper bound of the computational complexity of an algorithm in the **worst-case scenario**.

# Big- $O$

- $g$  is simpler function than  $f$  and it does not increase much faster than  $f$ .



# Big- $O$

- $g$  is simpler function than  $f$  and it does not increase much faster than  $f$ .

## Example

1  $n^2 = O(n^3 + n^2 \ln n + 595)$

2  $n^2 = O(e^{n^2})$

3  $e^{-n} = O(n^2)$

# Big- $O$

- $g$  is simpler function than  $f$  and it does not increase much faster than  $f$ .

## Example

- 1  $n^2 = O(n^3 + n^2 \ln n + 595)$

- 2  $n^2 = O(e^{n^2})$

- 3  $e^{-n} = O(n^2)$

- 4  $f(n) (= a_0 + a_1 n + \dots + a_d n^d) = O(n^d)$

# Big- $O$

- $g$  is simpler function than  $f$  and it does not increase much faster than  $f$ .

## Example

- 1  $n^2 = O(n^3 + n^2 \ln n + 595)$

- 2  $n^2 = O(e^{n^2})$

- 3  $e^{-n} = O(n^2)$

- 4  $f(n) (= a_0 + a_1 n + \dots + a_d n^d) = O(n^d)$

- 5  $\ln n = O(n^\delta)$  for any  $\delta \in \mathbb{R}^+$

# Complexity of Arithmetic Operations

- 1 Time( $k$ -bit +  $k$ -bit) =  $O(k)$
- 2 Time( $k$ -bit -  $k$ -bit) =  $O(k)$
- 3 Time( $k$ -bit  $\times$   $k$ -bit) =  $O(k^2)$
- 4 Time( $k$ -bit  $\div$   $k$ -bit) =  $O(k^2)$
- 5 Time( $n!$ ) =  $O(n^2(\ln n)^2)$

# Complexity of Integer Multiplication

Method	Year	# Operations
Karatsuba Ofman	1962	$O(n^{\log_2 3})$
Toom	1963	$O(n^{1+\epsilon})$
Schönhage Strassen	1971	$O((n \log n \log \log n \dots 2^{O(\log^* n)}))$ $O((n \log n \log \log n))$
Fürer	2007	$O((n \log n \cdot 2^{O(\log^* n)}))$
Anindya De	2008	$O((n \log n \cdot 2^{O(\log^* n)}))$

# Euclidean Algorithm

**Input:**  $(a, b)$  [ $a \geq b$ ]

**Output:**  $\gcd(a, b)$

$r_0 \leftarrow a;$

$r_1 \leftarrow b;$

$m \leftarrow 1;$

**while**{ $r_m \neq 0$ } {

$q_m \leftarrow \lfloor \frac{r_{m-1}}{r_m} \rfloor;$

$r_{m+1} \leftarrow r_{m-1} - q_m \cdot r_m;$

$m \leftarrow m + 1;$

}

# Euclidean Algorithm

```
 $m \leftarrow m - 1;$   
return  $q_1, \dots, q_m; r_m$   
 $\mathbf{r}_m = \mathbf{gcd}(\mathbf{a}, \mathbf{b})$ 
```

# Euclidean Algorithm

```
 $m \leftarrow m - 1;$   
return  $q_1, \dots, q_m; r_m$   
 $\mathbf{r}_m = \mathbf{gcd}(\mathbf{a}, \mathbf{b})$ 
```

Time to compute  $\mathbf{gcd}(\mathbf{a}, \mathbf{b}) = O((\ln \mathbf{a})^2)$ .



# Extended Euclidean Algorithm

## Proposition

Let  $d = \gcd(a, b)$ , where  $a > b$ .

Then  $\exists u \ \& \ v \in \mathbb{Z} : d = u.a + b.v$ .

$\Rightarrow$   *$\gcd$  of 2 numbers can be expressed as a linear combination of the numbers with integer coefficients.*

# Extended Euclidean Algorithm

## Proposition

Let  $d = \gcd(a, b)$ , where  $a > b$ .

Then  $\exists u \ \& \ v \in \mathbb{Z} : d = u.a + b.v$ .

$\Rightarrow \gcd$  of 2 numbers can be expressed as a linear combination of the numbers with integer coefficients.

*Time to compute  $u \ \& \ v = O((\ln a)^2)$ .*

# Complexity in $\mathbb{Z}_n$

Let  $m_1$  &  $m_2 \in \mathbb{Z}_n$  and  $size(n) = k$ -bit.

## Result

- ①  $Time(m_1 + m_2) \bmod n = O(k)$
- ②  $Time(m_1 - m_2) \bmod n = O(k)$
- ③  $Time(m_1 \times m_2) \bmod n = O(k^2)$
- ④  $Time(m_1)^{-1} \bmod n =$

# Complexity in $\mathbb{Z}_n$

Let  $m_1$  &  $m_2 \in \mathbb{Z}_n$  and  $size(n) = k$ -bit.

## Result

- ①  $Time(m_1 + m_2 \bmod n) = O(k)$
- ②  $Time(m_1 - m_2 \bmod n) = O(k)$
- ③  $Time(m_1 \times m_2 \bmod n) = O(k^2)$
- ④  $Time(m_1^{-1} \bmod n) = O(k^2)$
- ⑤  $Time(m_1^c \bmod n) =$

# Complexity in $\mathbb{Z}_n$

Let  $m_1$  &  $m_2 \in \mathbb{Z}_n$  and  $size(n) = k$ -bit.

## Result

- ①  $Time(m_1 + m_2 \bmod n) = O(k)$
- ②  $Time(m_1 - m_2 \bmod n) = O(k)$
- ③  $Time(m_1 \times m_2 \bmod n) = O(k^2)$
- ④  $Time(m_1^{-1} \bmod n) = O(k^2)$
- ⑤  $Time(m_1^c \bmod n) = O((\ln c)k^2)$

# Small- $o$

## Definition

Let  $f$  and  $g$  be 2 +ve real valued functions such that

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \rightarrow 0.$$

Then we say that  $f = o(g)$ ,  $\Rightarrow f(n) \ll g(n)$  when  $n$  is large.

- A function  $f$  is **negligible** if  $f = o(1/g)$  for any polynomial  $g(n) = n^c$
- The notation  $g = \Omega(f)$  means exactly the same thing as  $f = O(g)$ .
- If  $f = O(g)$  and  $f = \Omega(g)$  then we use the notation  $f = \Theta(g) \Rightarrow C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$  for  $n \geq n_0, C_i \in \mathbb{R}^+$ .

## Example

$$① \sqrt{n} = o(n)$$

$$② n \ln \ln n = o(n \ln n)$$

$$③ n \ln n = \Omega(3n \ln n + 5n \ln \ln n + 2)$$

$$④ \text{length}(n!) = \Theta(n \ln n)$$

# From Polynomial to Exponential Time

## Definition

- 1 **Polynomial time algorithm:** computational complexity is  $O(n^k)$ , where  $n$  is the size of the input in bits and  $k \in \mathbb{R}^+$ .
- 2 **Exponential time algorithm:** computational complexity is of the form  $O(c^{f(n)})$  where  $c > 1$  is a constant and  $f$  is a polynomial function on the size of the input  $n \in \mathbb{N}$ .
- 3 **Subexponential time algorithm:** computational complexity for input  $q \in \mathbb{N}^a$  is

$$L_q(\alpha, c) = O(e^{(c+o(1))(\ln q)^\alpha (\ln \ln q)^{1-\alpha}}),$$

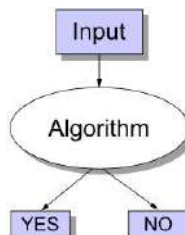
where  $\alpha \in \mathbb{R}$ ,  $0 < \alpha < 1$  and  $c$  is a positive constant.

---

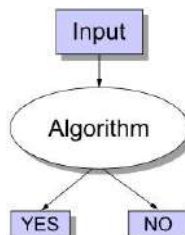
<sup>a</sup>Note that  $q$  is the input to the algorithm and not the size of the input.



# Decision Problem



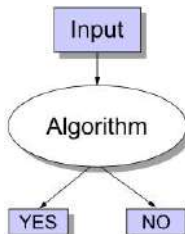
# Decision Problem



## Example

- 1 **Input:**  $N \in \mathbb{Z}^+$   
**Question:** Is  $N$  a prime number?

# Decision Problem



## Example

- 1 **Input:**  $N \in \mathbb{Z}^+$   
**Question:** Is  $N$  a prime number?
- 2 **Input:**  $N, k \in \mathbb{Z}^+$   
**Question:** Does  $N$  have a factor  $M$ , where  $2 \leq M \leq k$ ?

# Outline

- 1 Introduction
  - Time Estimation
- 2 Notations
- 3 Complexity Classes**
  - Time Complexity**

# $\mathcal{L}$ and $\mathcal{NL}$ -Space

## $\mathcal{L}$ – Logarithmic Space

- The class of decision problems solvable by a Turing machine restricted to use an amount of memory logarithmic in the size of the input,  $n$ .

## $\mathcal{NL}$ – Nondeterministic Logarithmic-Space

- Has the same relation to  $\mathcal{L}$  as  $\mathcal{NP}$  does to  $\mathcal{P}$ .
- $\mathcal{NL} = \text{co-}\mathcal{NL}$ .

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Definition

A decision problem  $P \in \mathcal{P}$  of polynomial time problems if  $\exists$  a polynomial  $p$  and an algorithm such that if for an instance of  $P$  with input size  $\leq n$ , the algorithm answer the question correctly in time  $\leq p(n)$ .

## Example

**Instance:**  $n \in \mathbb{Z}^+$

**Question:** Is  $n$  prime?

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Definition

A decision problem  $P \in \mathcal{P}$  of polynomial time problems if  $\exists$  a polynomial  $p$  and an algorithm such that if for an instance of  $P$  with input size  $\leq n$ , the algorithm answer the question correctly in time  $\leq p(n)$ .

## Example

**Instance:**  $n \in \mathbb{Z}^+$

**Question:** Is  $n$  prime?

**Answer:** Yes,  $[O((\log n)^6)$  using AKS algo]

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Definition

A decision problem  $P \in \mathcal{NP}$  if for given any instance of  $P$  with some extra information, the **YES** answer can be verified in polynomial time.

## Definition

A decision problem  $P \in \text{co-}\mathcal{NP}$  if for given any instance of  $P$  with some extra information, the **NO** answer can be verified in polynomial time.

## Example

**Instance:**  $n \in \mathbb{Z}^+$

**Question:** Is  $n$  composite?



# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Definition

A decision problem  $P \in \mathcal{NP}$  is said to be  $\mathcal{NP}$ -complete if every other problem  $Q \in \mathcal{NP}$  can be reduced to  $P$  in polynomial time, i.e.,  $Q \leq_P P$ .



## Definition

Let  $\Pi$  and  $\Gamma$  be 2 decision problems. We say that  $\Pi \leq_P \Gamma$ , if  $\Pi$  can be solved in polynomial time given access to an oracle that solves  $\Gamma$ .

Informally, if  $\Pi \leq_P \Gamma \Rightarrow \Pi$  is not harder than  $\Gamma$ .

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Example

-  **Input:** A quadratic polynomial  $p \in \mathbb{Z}[x]$   
**Question:** Does  $p(x)$  have 2 distinct real roots?
-  **Input:**  $N \in \mathbb{Z}$   
**Question:** Is  $N \in \mathbb{Z}^+$

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Example

- Input:** A quadratic polynomial  $p \in \mathbb{Z}[x]$   
**Question:** Does  $p(x)$  have 2 distinct real roots?
- Input:**  $N \in \mathbb{Z}$   
**Question:** Is  $N \in \mathbb{Z}^+$

## Example

$\text{PRIMALITY} \leq_P \text{FACTORING}$

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard

## Definition

A decision or search problem  $P$  is *NP-hard* if  $\exists$  some NP-complete problem  $Q$  that poly-time reduces to  $P$ , i.e.,  $Q \leq_P P$

## Example

**Input:** A finite cyclic group  $\mathcal{G}$  of order  $n$ , a generator  $\alpha$  of  $\mathcal{G}$  and an element  $\beta \in \mathcal{G}$ .

**Question:** Find the integer  $x$ ,  $0 \leq x \leq n - 1$ , such that  $\alpha^x = \beta$ .

# NP-hard Problems

## Example

- **Integer Factorization:** Given  $n \in \mathbb{Z}^+$ , find  $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$  where the  $p_i$  are pairwise distinct primes and each  $e_i \geq 1$  for  $1 \leq i \leq k$ .

# NP-hard Problems

## Example

- **Integer Factorization:** Given  $n \in \mathbb{Z}^+$ , find  $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$  where the  $p_i$  are pairwise distinct primes and each  $e_i \geq 1$  for  $1 \leq i \leq k$ .
- **Discrete Logarithm Problem:** Given an abelian group  $(G, \cdot)$  and  $g \in G$  of order  $n$ . Given  $h \in G$  such that  $h = g^x$  find  $x$  ( $DLP(g, h) \rightarrow x$ ).

# NP-hard Problems

## Example

- **Integer Factorization:** Given  $n \in \mathbb{Z}^+$ , find  $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$  where the  $p_i$  are pairwise distinct primes and each  $e_i \geq 1$  for  $1 \leq i \leq k$ .
- **Discrete Logarithm Problem:** Given an abelian group  $(G, \cdot)$  and  $g \in G$  of order  $n$ . Given  $h \in G$  such that  $h = g^x$  find  $x$  ( $DLP(g, h) \rightarrow x$ ).
- **Computational Diffie-Hellman Problem:** Given  $a = g^x$  and  $b = g^y$  find  $c = g^{xy}$ . ( $CDH(g, a, b) \rightarrow c$ ).

# NP-hard Problems

## Example

- **Elliptic Curve Discrete Logarithm Problem (ECDLP):**  $\mathbb{E}$  denotes the collections of points on a elliptic curve and  $P \in \mathbb{E}$ . Let  $\mathcal{S}$  be the cyclic subgroup of  $\mathbb{E}$  generated by  $P$ . Given  $Q \in \mathcal{S}$ , find an integer  $x$  such that  $Q = x.P$ .



# NP-hard Problems

## Example

- **Elliptic Curve Discrete Logarithm Problem (ECDLP):**  $\mathbb{E}$  denotes the collections of points on a elliptic curve and  $P \in \mathbb{E}$ . Let  $\mathcal{S}$  be the cyclic subgroup of  $\mathbb{E}$  generated by  $P$ . Given  $Q \in \mathcal{S}$ , find an integer  $x$  such that  $Q = x.P$ .
- **Subset Sum Problem (SSP):**  $\{a_1, a_2, \dots, a_n\} \subset \mathbb{Z}^+$  called a knapsack set,  $s \in \mathbb{Z}^+$ , determine whether or not  $x_i \in \{0, 1\}$ , for  $1 \leq i \leq n$  such that

$$\sum_{i=1}^n x_i \cdot a_i = s.$$

# NP-complete Problem

## Example

- **Hilbert's Nullstellensatz (HN):** Given  $p_1, p_2, \dots, p_m \in \mathbb{K}[x_1, x_2, \dots, x_n]$ , where  $\mathbb{K}$  is a finite field. Does there exist  $(\alpha_1, \alpha_2, \dots, \alpha_n) \in \mathbb{K}^n$  such that:

$$p_1(\alpha_1, \alpha_2, \dots, \alpha_n) = 0, \dots, p_m(\alpha_1, \alpha_2, \dots, \alpha_n) = 0.$$

# NP-complete Problem

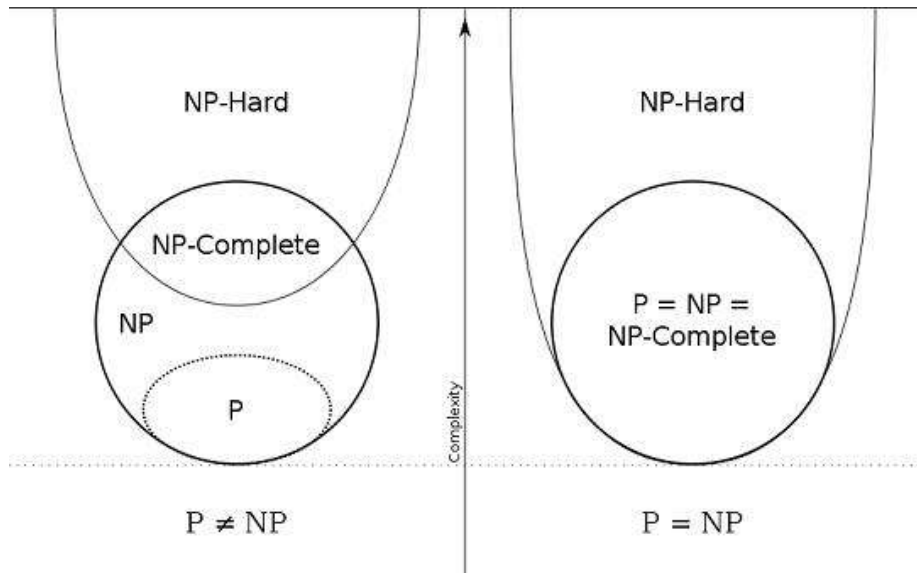
## Example

- **Hilbert's Nullstellensatz (HN):** Given  $p_1, p_2, \dots, p_m \in \mathbb{K}[x_1, x_2, \dots, x_n]$ , where  $\mathbb{K}$  is a finite field. Does there exist  $(\alpha_1, \alpha_2, \dots, \alpha_n) \in \mathbb{K}^n$  such that:

$$p_1(\alpha_1, \alpha_2, \dots, \alpha_n) = 0, \dots, p_m(\alpha_1, \alpha_2, \dots, \alpha_n) = 0.$$

It remains NP-complete with input polynomials of total degree 2.

# $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ $\mathcal{NP}$ -hard



$\mathcal{RP}, \mathbf{co}\text{-}\mathcal{RP}, \mathcal{ZPP}, \mathcal{BPP}$ 

A language  $L$  belongs to  $\mathcal{RP}$  iff  $\exists$  a PTM,  $M$ , with polynomial running time such that on any input  $x \in \Sigma_0^n$ :

- if  $x \in L$  then  $\Pr[M \text{ accepts } x] \geq 1/2$ ;
- if  $x \notin L$  then  $\Pr[M \text{ accepts } x] = 0$ .

$RP, co-RP, ZPP, BPP$ 

A language  $L$  belongs to  $RP$  iff  $\exists$  a PTM,  $M$ , with polynomial running time such that on any input  $x \in \Sigma_0^n$ :

- if  $x \in L$  then  $Pr[M \text{ accepts } x] \geq 1/2$ ;
- if  $x \notin L$  then  $Pr[M \text{ accepts } x] = 0$ .

## Theorem

$$P \subseteq RP \subseteq NP.$$

$RP, co-RP, ZPP, BPP$ 

Input:  $n \in \mathbb{Z}^+$ .

Question: is  $n$  composite?

### Theorem

$COMPOSITE \in RP$ .

# The Miller-Rabin Primality Test

**Input:** an odd integer  $n \geq 3$ .

**Algorithm:**

choose  $a \in_R \mathbb{Z}_n^+$

if  $\gcd(a, n) \neq 1$  output 'composite'

let  $n - 1 = 2^k \cdot m$ , with  $m$  odd

if  $a^m \equiv 1 \pmod n$  output 'prime'

for  $i = 0$  to  $k - 1$

    if  $a^{m \cdot 2^i} \equiv -1 \pmod n$  then output 'prime'

next  $i$

output 'composite'.



# The Miller-Rabin Primality Test

## Theorem

*The Miller-Rabin primality test is a probabilistic polynomial time algorithm. Given input  $n$*

- (i) if  $n$  is prime then the algorithm always outputs 'prime';*
- (ii) if  $n$  is composite then*

$$\text{Pr}[\text{the algorithm outputs 'composite'}] \geq \frac{1}{2}.$$

# The Miller-Rabin Primality Test

## Theorem

*The Miller-Rabin primality test is a probabilistic polynomial time algorithm. Given input  $n$*

- (i) if  $n$  is prime then the algorithm always outputs 'prime';*
- (ii) if  $n$  is composite then*

$$Pr[\text{the algorithm outputs 'composite'}] \geq \frac{1}{2}.$$

**Hence**

*COMPOSITE*  $\in RP$  or equivalently *PRIME*  $\in co-RP$ .

$RP, co-RP, ZPP, BPP$ 

- If  $L \in RP$  then  $\exists$  a polynomial time PTM for  $L$  which is always correct when it accepts an input but which will sometimes incorrectly reject an input  $x \in L$ .
- ||ly if  $L \in co-RP$  then  $\exists$  a polynomial time PTM for  $L$  which is always correct when it rejects an input but which will sometimes incorrectly accept an input  $x \notin L$ .

# $RP, co-RP, ZPP, BPP$

- If  $L \in RP$  then  $\exists$  a polynomial time PTM for  $L$  which is always correct when it accepts an input but which will sometimes incorrectly reject an input  $x \in L$ .
- |||y if  $L \in co-RP$  then  $\exists$  a polynomial time PTM for  $L$  which is always correct when it rejects an input but which will sometimes incorrectly accept an input  $x \notin L$ .

## Definition

A language  $L$  is decidable in **zero-error probabilistic polynomial time** or equivalently belongs to  $ZPP$  iff  $\exists$  a PTM,  $M$ , with polynomial expected running time such that for any input  $x \in \Sigma_0^*$ :

- (i) if  $x \in L$  then  $Pr[M \text{ accepts } x] = 1$ ;
- (ii) if  $x \notin L$  then  $Pr[M \text{ accepts } x] = 0$ .

$RP, co-RP, ZPP, BPP$ 

## Proposition

$$ZPP = RP \cap co-RP.$$

## Definition

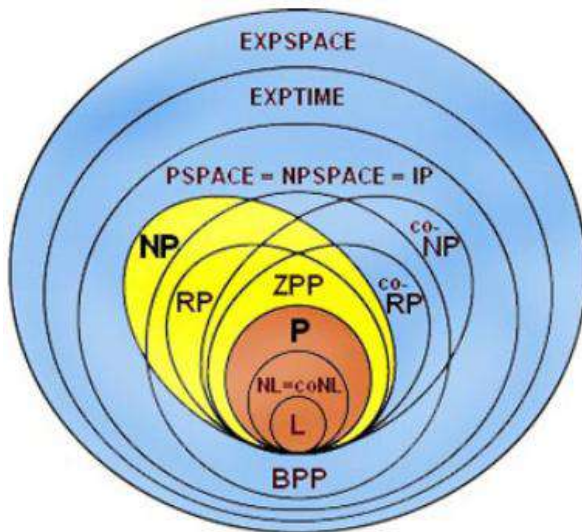
A language  $L$  belongs to  $BPP^a$  iff there is a PTM,  $M$ , with polynomial running time such that on any input  $x \in \Sigma_0^*$ :

- (i) if  $x \in L$  then  $Pr[M \text{ accepts } x] \geq 3/4$ ;
- (ii) if  $x \notin L$  then  $Pr[M \text{ accepts } x] \leq 1/4$ .

---

<sup>a</sup>Bounded-Error Probabilistic Polynomial-Time

# Complexity Class



# $\mathcal{P}$ vs. $\mathcal{NP}$ & Cryptography

- $\mathcal{P} \neq \mathcal{NP}$  &  $\mathcal{NP} \not\subseteq \mathcal{BPP}$  are necessary for most of the modern cryptography.
- They are not sufficient for cryptography.
- **Worst-case complexity**  $\Rightarrow$  the maximum number of steps taken on any instance of size  $n$ .
- **Best-case complexity**  $\Rightarrow$  the minimum number of steps taken on any instance of size  $n$ .
- **Average-case complexity**  $\Rightarrow$  the average number of steps taken on any instance of size  $n$ .



## Complexity Zoo

[http://qwiki.stanford.edu/index.php/Complexity\\_Zoo](http://qwiki.stanford.edu/index.php/Complexity_Zoo)



*T. Apostol,*

Introduction to Analytic Number Theory, *Narosa Publishing House*, 1998.



*N. Koblitz,*

A Course in Number Theory and Cryptography, *Springer*, 1994.



*A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone,*

Handbook of Applied Cryptography, *CRC Press*, 1996. Available online at

<http://www.cacr.math.uwaterloo.ca/hac/>



*J. Talbot & D. Welsh,*

Complexity and Cryptography - An Introduction, *Cambridge University Press*, 2006.



# The End

**Thanks a lot for your attention!**