

The Complexity of Some Problems in Cryptography

Presenter: Rave Harpaz *

March 12, 2002

*Computer Science Department, The Graduate Center,
City University of New York.

Outline:

- NP and coNP
 - Definitions
 - Testing Primality and compositeness
 - * Showing that $PRIMES \in coNP$
 - * Fermat's little theorem
 - * Pratt's theorem, $PRIMES \in coNP \cap NP$
 - Summery and open problems
- Function problems, FNP, TFNP
 - Definitions
 - Total functions and the FACTORING problem
 - Summery
- Randomized Computations
 - Definitions
 - RP and coRP (randomized poly-time)
 - * $PRIMES \in coRP$ (Rabin-Miller)
 - ZPP (zero-error probabilistic poly-time)
 - * $PRIMES \in ZPP$ (Adleman-Huang)
 - BPP (bounded-error probabilistic poly-time)
 - PP (probabilistic poly-time)
 - Summery and open problems

Definition 1.

$NP = \{L \mid \text{there exists a polynomial time NDTM that decides } L\}$

Definition 2.

$coNP = \{\bar{L} \mid L \in NP\}$

As opposed to to NP, only "no" instances (disqualifications) of problems in coNP possess short proofs (certificates).

Examples:

VALIDITY, HAMILTON PATH COMPLEMENT $\in coNP$

Some results:

- $L \in NP\text{-complete} \Rightarrow \bar{L} \in coNP\text{-complete}$
- $P = NP \Rightarrow NP = coNP$
- $L \in coNP\text{-complete} \wedge L \in NP \Rightarrow NP = coNP$
- $L \in NP \cap coNP \Rightarrow L \notin NP\text{-complete} \text{ (unless } NP = coNP)$

Testing Primality and Compositeness

- The trivial $O(\sqrt{N})$ deterministic algorithm for solving PRIMES is misleading, since its time bound is not polynomial in the length of the input, which is $\log N$.
- At present, it is not known whether there is a deterministic polynomial algorithm for PRIMES.
- However primality testing has advanced to a stage where the primality of a 100-digit can be checked very fast. Which is why current research efforts are directed towards proving that the problem is in P.
- More evidence that primality testing might be in P is by the fact that it is in $NP \cap coNP$.
- Moreover, Miller (1976) proved that if the "extended Riemann's hypothesis" will be proved to be correct then $PRIMES \in P$.

Claim: $PRIMES \in coNP$

Proof. Consider the following nondeterministic algorithm :

```
begin {input: n}
  guess  $a$  in  $\{2, \dots, n - 1\}$ 
  if  $n \bmod a = 0$  then reject else accept
end
```

(Checking that $n \bmod a = 0$ requires $O(\log^2 n)$ steps.)

Corollary 1. $COMPOSITE \in NP$

Claim: $PRIMES \in NP$

We know: $p \in PRIMES \Leftrightarrow \forall(1 < a < p)(p \bmod a \neq 0)$
which suggests that the problem is in coNP.

To show that $PRIMES \in NP$ we need to use an alternative characterization of primes, one that uses only the "exists" quantifier, or alternatively show that $COMPOSITE \in coNP$

Theorem 1 (Fermat's little theorem).

$$p \in PRIMES \Rightarrow \forall(1 < a < p)(a^{p-1} \equiv 1 \bmod p)$$

Corollary 2.

$$\exists(1 < a < p)(a^{p-1} \not\equiv 1 \bmod p) \Rightarrow p \in COMPOSITE$$

(In fact we have: $p \in PRIMES(p) \Leftrightarrow \forall(1 < a < p)(a^{p-1} \equiv 1 \bmod p)$)

Are any of the above theorems sufficient to prove our goal?

No, they can only show what we already know, that is $PRIMES \in coNP$. However:

Theorem 2.

$$p \in PRIMES \Leftrightarrow \exists r \text{ s.t. : } \begin{array}{l} 1. \quad 1 < r < p \\ 2. \quad r^{p-1} \equiv 1 \pmod{p} \\ 3. \quad r^{\frac{p-1}{q}} \not\equiv 1 \pmod{p} \\ \quad (\forall \text{ prime divisors } q \text{ of } p-1) \end{array}$$

Corollary 3 (Pratt's Theorem). $PRIMES \in NP \cap coNP$

Proof: We already know that $PRIMES \in coNP$. Theorem 2 says that every prime possesses a certificate that is missing from all composites. Thus if we show that this certificate is polynomially succinct and polynomially checkable then $PRIMES \in NP$.

How long does it take to verify condition no.2?

Let $l = \lceil \log p \rceil$ be the length of p represented in binary. In order to calculate $r^{p-1} \equiv 1 \pmod{p}$ we do not need to perform $p-2$ multiplications (which will not be polynomial in l), instead we use the repeated squaring modulo p algorithm. This will result an $O(l^3)$ time bound to verify the second condition.

How long does it take to verify condition no. 3?

We first need to find all the prime divisors of $p-1$ (q_1, \dots, q_k), how do we find them? guess them. Now we need to verify that they are prime, how do we accomplish this? use the same verification algorithm in a recursive manner.

An NP algorithm to test primality

```

prime(p)
begin
  if p=2 then accept
  else
    begin
      guess  $r$  in  $\{2, \dots, p-1\}$ 
      if  $r^{p-1} \not\equiv 1 \pmod{p}$  then reject
      else
        begin
          guess  $(q_1, \dots, q_k)$  in {set of prime divisors of p-1}
          verify that p-1 can be reduced to 1 by repeated divisions of  $q_i$ 's
          if  $\text{prime}(q_1) \wedge \dots \wedge \text{prime}(q_k)$  then
            if  $r^{\frac{p-1}{q_1}} \not\equiv 1 \pmod{p} \wedge \dots \wedge r^{\frac{p-1}{q_k}} \not\equiv 1 \pmod{p}$  then accept
            else reject
          else reject
        end
      end
    end
  end
end

```

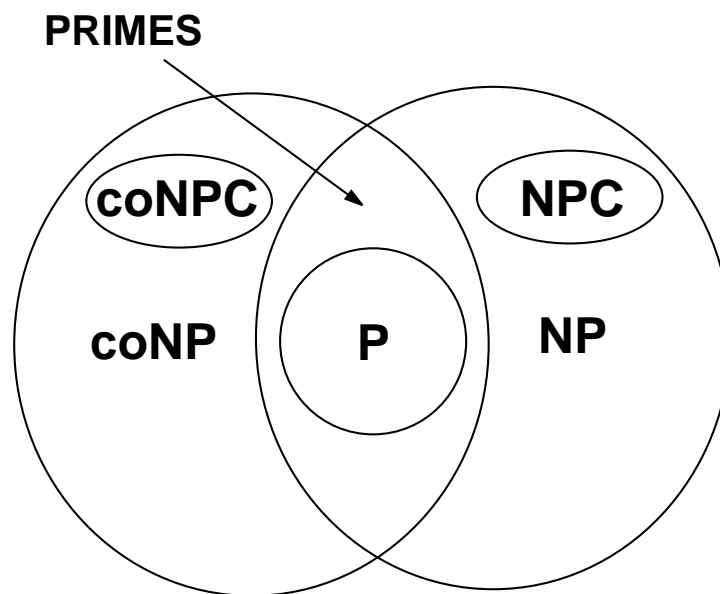
The certificate that a number p is a prime is the following:

$$\begin{aligned}
 p = 2 & \quad C(p) = () \\
 p > 2 & \quad C(p) = (r, q_1, C(q_1), \dots, q_k, C(q_k))
 \end{aligned}$$

Lemma 1. Any $n > 1$ has $k < \log n$ prime divisors.

\therefore Total time required to verify a certificate is $O(l^4)$
 And $|C(p)| = O(l^2)$.

Summary



(assuming $P \neq NP$ and $NP \neq coNP$)

Open problems:

- $NP = coNP$?
- $PRIMES \in P$?
- $P = NP \cap coNP$?
- is there an $NP \cap coNP$ complete problem ?

Function Problems

- Not all computational tasks require decision ("yes or "no") solutions. We call such problems, requiring an answer more elaborate than "yes" or "no", function problems.

- Example.

Definition 3. FSAT

input: A formula ϕ

output: An assignment that satisfies ϕ (if ϕ is satisfiable), "no" otherwise.

- Decision problems are useful surrogates of function problems only in the context of negative complexity results, hence since we know that $SAT \in NPC$ we can conclude that there is no polynomial time algorithm for solving $FSAT$ (unless $P = NP$).
- In general we can say that if FL is the functional a decision problem:

$$FL \in C \Rightarrow L \in C$$

$$L \in C \not\Rightarrow FL \in C$$

- However for some problems such as FSAT:

SAT can be solved in poly-time \Leftrightarrow FSAT can be solved in poly-time

(this property is called self-reducibility)

Definition 4. the class FNP

$FNP = \{L | \exists \text{ a polynomial-time NDTM that on input } x \text{ outputs } y, \text{ if such a } y \text{ exists, else outputs "no"}\}$

(where $L \subseteq \Sigma^* \times \Sigma^*$)

Problems in FNP that can be solved using a poly-time DTM are called **FP**.

Definition 5. Reductions between function problems

$FL' \leq FL$ if $\exists \phi, \varphi$ (computable in LogSpace) s.t for input x , output y : $x \in FL' \Leftrightarrow \phi(x) \in FL, FL'(x) = y \Leftrightarrow FL(\phi(x)) = \varphi(y)$.

Lemma 2. *FP and FNP are closed under reductions.*

Theorem 3. $FSAT \in FNPC$

Corollary 4. $FP = FNP \Leftrightarrow P = NP$

Total functions

There are certain problems in FNP that never return "no", despite that they can be very difficult problems, not known or believed to be in FP. The subclass of FNP containing such problems is denoted **TFNP**. Example:

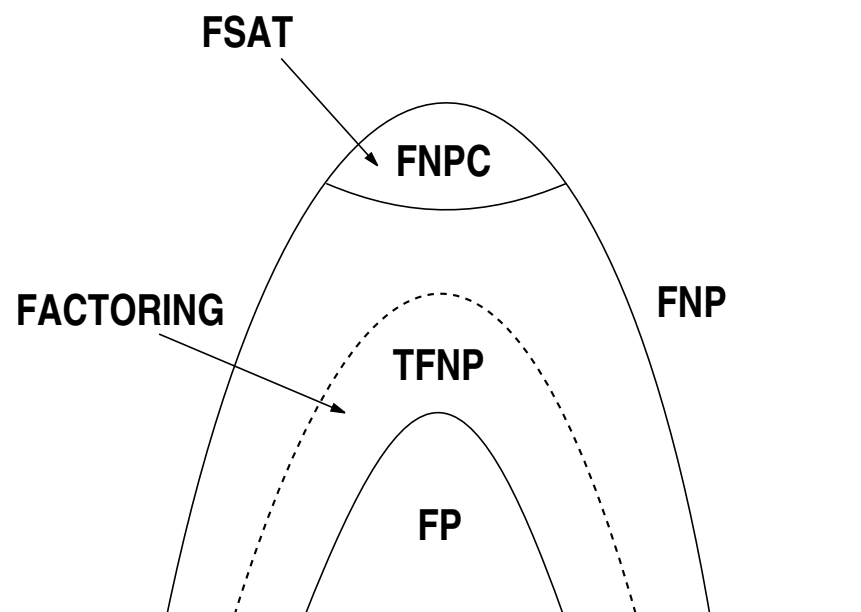
Definition 6. FACTORING

input: an integer N

output: prime decomposition of N , together with primality certificates-
 $q_1, C(q_1), \dots, q_n, C(q_n)$ where $N = q_1^{k_1} q_2^{k_2} \dots q_n^{k_n}$

Despite the fact that these type of problems are easier than FNPC problems such as FSAT, There is still no known polynomial algorithm for the problem FACTORING.

Summary



Randomized Computations

A **Probabilistic algorithm** is an algorithm designed to use the outcome of a random process, typically it would contain an instruction to "**flip a coin**" which would influence the algorithm's subsequent execution and output. Certain problems are more easily solvable by probabilistic algorithms than by deterministic ones.

Definition 7. probabilistic Turing Machine

Is a type of a nondeterministic Turing machine where each step is called a coin-flip step and has two legal next moves. We assign a probability to each branch b of the machine's computation as follows, where k is the number of coin-flips:

$$Pr[b] = 2^{-k}$$

The probability that the machine accepts input w :

$$Pr[M \text{ accepts } w] = \sum_{b \text{ is an accepting branch}} Pr[b]$$

The probability that the machine rejects input w :

$$Pr[M \text{ rejects } w] = 1 - Pr[M \text{ accepts } w]$$

Definition 8. Las Vegas and Monte Carlo Machines

A **Las Vegas** machine will always give the correct answer when it terminates. A **Monte Carlo** machine may sometimes produce an incorrect solution (though we are always able to bound the probability of incorrect solutions). There are two kinds of Monte Carlo machines: a **one-sided error** machine in which it may err on only one of the yes-no answers, and a **two-sided error** machine in which it may err on both yes-no answers. Note that a Las Vegas machine is a Monte Carlo machine with error probability 0.

The classes **RP** And **coRP**

Randomized polynomial time, one-sided error computable languages.

Definition 9. RP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow \Pr[M \text{ accepts } x] \geq \frac{1}{2}$$

$$x \notin L \Rightarrow \Pr[M \text{ rejects } x] = 1$$

(i.e. no false positives, accept by majority and reject unanimously)

Definition 10. coRP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow \Pr[M \text{ accepts } x] = 1$$

$$x \notin L \Rightarrow \Pr[M \text{ rejects } x] \geq \frac{1}{2}$$

(i.e. no false negatives, accept unanimously and reject by majority)

An alternative way to define coRP is:

$$\text{coRP} = \{L | \bar{L} \in \text{RP}\}$$

Theorem 4. $P \subseteq RP$, $P \subseteq coRP$, $RP \subseteq NP$, $coRP \subseteq coNP$

proof: A polynomial deterministic Turing machine is a special case of an RP machine, it is one that ignores the coin-flips and also accepts unanimously (probability 1 is at least as large as $\frac{1}{2}$). An RP machine is by definition a nondeterministic machine, since if half of the computations accept, surely at least one does.

Invariance of the constant

The constant $\frac{1}{2}$ in the definition of RP is arbitrary. We could have chosen any constant ϵ (between 0-1) and still get the same complexity class. If the constant is less than $\frac{1}{2}$ we can, by an **amplification** procedure bring the threshold to a level $\geq \frac{1}{2}$. The amplification procedure runs the RP machine a certain number of times (depending on the constant) and returns "yes" if one of the runs returns "yes". If none of the runs return a "yes" then it is obvious that the probability of getting an incorrect answer falls, thus increasing the probability of getting a correct answer.

It can even be shown that we can replace the constant ϵ by $\frac{1}{p(|x|)}$ or $1 - 2^{-p(|x|)}$, which are thresholds that depend on the length of the input.

Randomized computations for Primality and Compositeness testing.

Definition 11.

$$\mathbb{Z}_n = \{r \mid 0 \leq r \leq n - 1\}$$

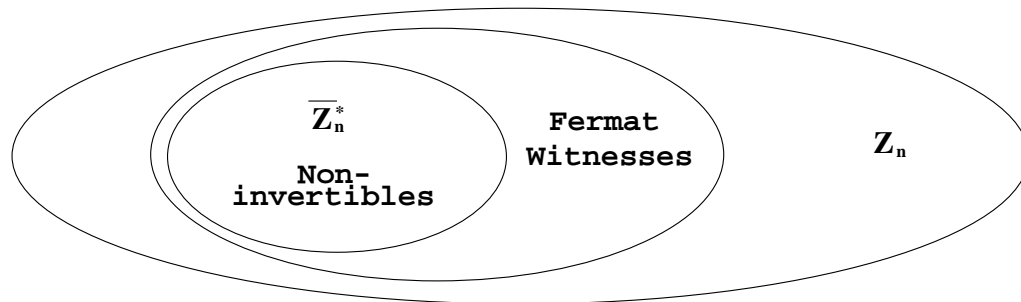
\mathbb{Z}_n is the group of **residues modulo n** , with respect to addition.

$$\mathbb{Z}_n^* = \{r \mid 1 \leq r \leq n - 1 \wedge \gcd(r, n) = 1\}$$

\mathbb{Z}_n^* is the group of **residues modulo n** , with respect to multiplication. \mathbb{Z}_n^* is also defined as $\Phi(n)$ and referred to as the set of **invertible elements modulo n** .

- As a reminder, Fermat's theorem states that a number $n > 2$ is prime iff for every $a \in \mathbb{Z}_n^* \setminus \{0\}$, a passes Fermat's test ($a^{n-1} \equiv 1 \pmod{n}$).
- A number that fails Fermat's test will be called a **Fermat witness** of n 's compositeness.
- Most composites have a lot of Fermat witnesses, but the question is: **does every composite have enough witnesses?** since we do not want to search for a witness (exponential time) but randomly test potential witnesses.

Fermat witness search space



- Every element of $\overline{\mathbb{Z}_n^*} = \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ is a Fermat witness.
- We would like that for every composite n , $\overline{\mathbb{Z}_n^*}$ will be large, or if it is small then \mathbb{Z}_n^* will be rich in witnesses.
- Unfortunately there are some composites, called the **Carmichael numbers** in which $\overline{\mathbb{Z}_n^*}$ is relatively small and \mathbb{Z}_n^* contains no Fermat witness (the first three are 561, 1105, 1729).

Theorem 5. (Rabin-Miller)

If $n > 4$ is composite then the number of witnesses of n 's compositeness is at least $3(n-1)/4$.

Note: the test for n 's compositeness is more sophisticated than Fermat's test, avoiding the problem with the Carmichael numbers. The underlying principle is that composite numbers including the Carmichael numbers have four or more square roots of 1, whereas the prime numbers have exactly two square roots ± 1 .

Corollary 5. $COMPOSITE \in RP$

```
begin {input:  $n > 4$ }
  if  $n$  is even then accept
  else
    begin
       $a := \mathbf{random}(1, n - 1)$ 
      if  $a$  witnesses the compositeness of  $n$  then accept
      else reject
    end
  end
end
```

It can be shown that checking whether a is a compositeness witness of n takes poly-time operations. Further, the error probability of the algorithm is less than $\frac{1}{4}$. That is, if the algorithm accepts we are sure that n is composite, and when it rejects we can say that n is prime with probability at least $\frac{3}{4}$.

Corollary 6. $PRIMES \in coRP$

The class **ZPP**

zero-error probabilistic polynomial time, modeling Las Vegas computations.

Definition 12. ZPP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M containing an additional final state called "don't know", such that

$$x \in L \Rightarrow \Pr[M \text{ accepts } x] > \frac{1}{2} \wedge \Pr[M \text{ rejects } x] = 0$$

$$x \notin L \Rightarrow \Pr[M \text{ rejects } x] > \frac{1}{2} \wedge \Pr[M \text{ accepts } x] = 0$$

(i.e. no false positives, no false negatives, but we may get a "don't know" answer)

Again the value $\frac{1}{2}$ is arbitrary and can be replaced as before with any value between $2^{-p(|x|)}$ and $1 - \frac{1}{p(|x|)}$.

Theorem 6. $ZPP = RP \cap coRP$

Proof: *To show that $ZPP \subseteq RP$ we label the "don't know" states in the ZPP machine with "reject" to obtain an RP machine deciding the same language. Similarly to show that $ZPP \subseteq coRP$ we label "don't know" states with "accept". Thus $ZPP \subseteq RP \cap coRP$. To show that $RP \cap coRP \subseteq ZPP$ we construct a ZPP machine from the RP and coRP machines (M_1, M_2) as follows:*

*run M_1 , if it accepts then accept
run M_2 , if it rejects then reject
otherwise return "don't know"*

Theorem 7. $P \subseteq ZPP$

Since $P \subseteq RP \cap coRP$.

Theorem 8. $PRIMES \in RP$ (Adleman, Huang 1987)

Corollary 7. $PRIMES \in ZPP$

The class **BPP**

bounded-error probabilistic polynomial time, two-sided error computable languages.

Definition 13. BPP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow Pr[M \text{ accepts } x] \geq \frac{1}{2} + \epsilon$$

$$x \notin L \Rightarrow Pr[M \text{ rejects } x] \geq \frac{1}{2} + \epsilon$$

(i.e. accept by "clear majority" and reject by "clear majority")

The BPP machine makes mistakes but returns the correct answer most of the time. By running the machine a large number of times and returning the majority of the answers we can guarantee that the probability of a mistake is very low. Although the amplification process is less trivial as before we can still build BPP machines with thresholds ranging from $\frac{1}{2} + \frac{1}{p(|x|)}$ to $1 - 2^{-p(|x|)}$.

Even though BPP machines can have exponentially small error probabilities, no such machine is known to solve a decision problem, not solvable by more constrained machines.

Theorem 9. $RP \cup coRP \subseteq BPP$

$RP \subseteq BPP$ and $coRP \subseteq BPP$ because a one-sided error machine is a special case of a two-sided error machine.

Theorem 10. $coBPP = BPP$

This is because the definition of a BPP machine is symmetric.

The class **PP**

probabilistic polynomial time, two-sided error computable languages.

Definition 14. PP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow Pr[M \text{ accepts } x] > \frac{1}{2}$$

$$x \notin L \Rightarrow Pr[M \text{ rejects } x] \geq \frac{1}{2}$$

(i.e. accept by "regular majority" and reject by "regular majority")

The PP class is wider than what we have seen so far. In the BPP case we had a gap between the number of accepting and rejecting computations, wide enough to enable us to invoke the machine polynomially many times to notice the difference between inputs that are in the language and those that are not. PP machines do not put the gap restriction which may be very small, hence amplification to a desired level can not always be achieved in polynomial time. Therefore languages in PP have no realistic computational content, similar to NP.

Theorem 11. $BPP \subseteq PP$

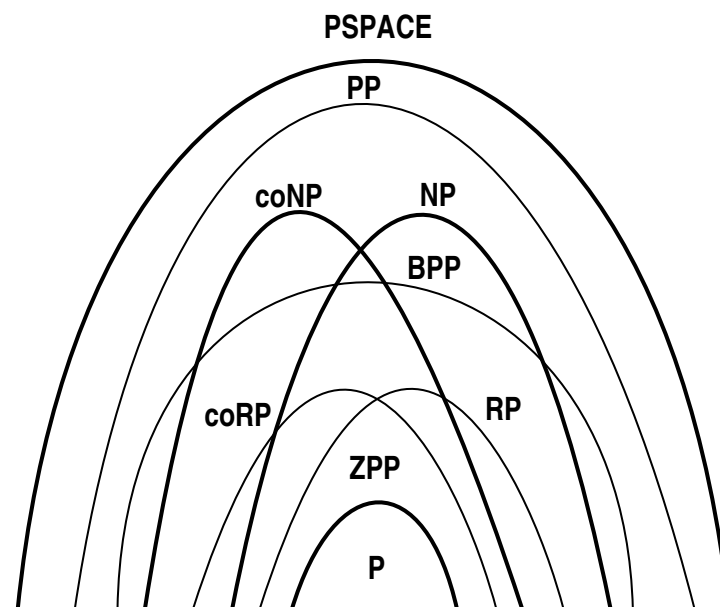
Trivial, because $\frac{1}{2} + \epsilon > \frac{1}{2}$.

Theorem 12. $PP \subseteq PSPACE$

Run all possible computation paths with length $p(|x|)$ reusing the space and count the number of accepting and rejecting states.

Theorem 13. $NP \cup coNP \subseteq PP$

Summary



We Know:

- $P \subseteq ZPP$
- $ZPP = RP \cap coRP$
- $RP \cup coRP \subseteq BPP$
- $RP \subseteq NP$ and $coRP \subseteq coNP$
- $BPP \subseteq PP$
- $PP \subseteq PSPACE$

Open problems:

- $P = NP?$
- $NP = coNP?$
- $P = NP \cap coNP?$
- $PRIMES \in P?$
- $RP = coRP?$
- $RP = NP \cap coNP?$
- $BPP \subseteq NP?$

It is believed that $BPP = P$, hence the following hierarchy $P \subseteq ZPP \subseteq RP \subseteq BPP$ will collapse and randomized computations will be of no real use.

References

1. C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. pp 219-278.
2. D.P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice hall, 1994. pp 130-153, 178-201.
3. O. Goldreich. *Introduction to Complexity Theory*. Lecture notes.
4. T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990. pp 801-852.