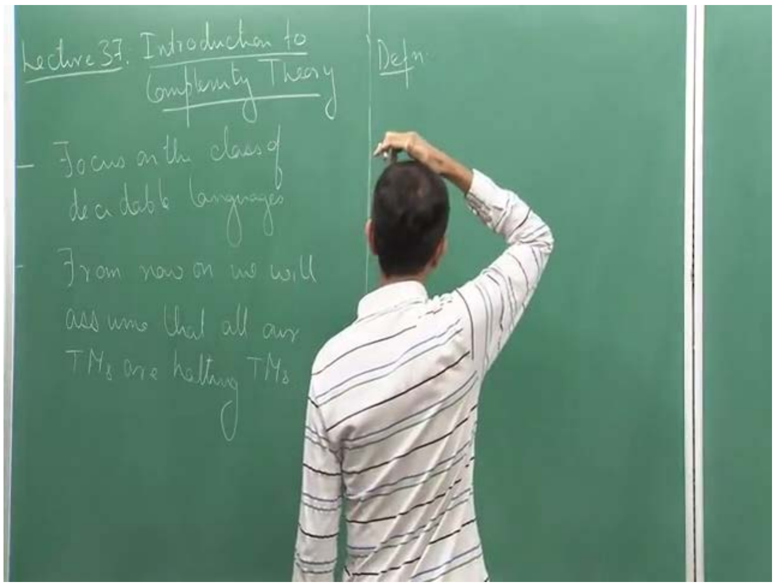
Theory of Computation

Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute Of Technology, Kanpur

Lecture – 37
Introduction to Computational Complexity

Welcome to the 37 lecture. Today, we are going to talk about complexity theory or more specifically will give a brief introduction into what complexity theory is. So, for the rest of this course we will keep our focus only to the set of decidable languages. So far we have seen several class of languages; and the past few lectures we have been our focus was mostly on the set of undecidable languages, we saw various undecidable problems. And last lecture, we saw a mechanism which we can use to show that languages are undecidable. But from now on we only focus on the set of decidable problems, and we will give a finer gradation, we will give a finer classification of this set of decidable languages.

(Refer Slide Time: 01:17)



Lecture 37. Introduction to Complexity Theory

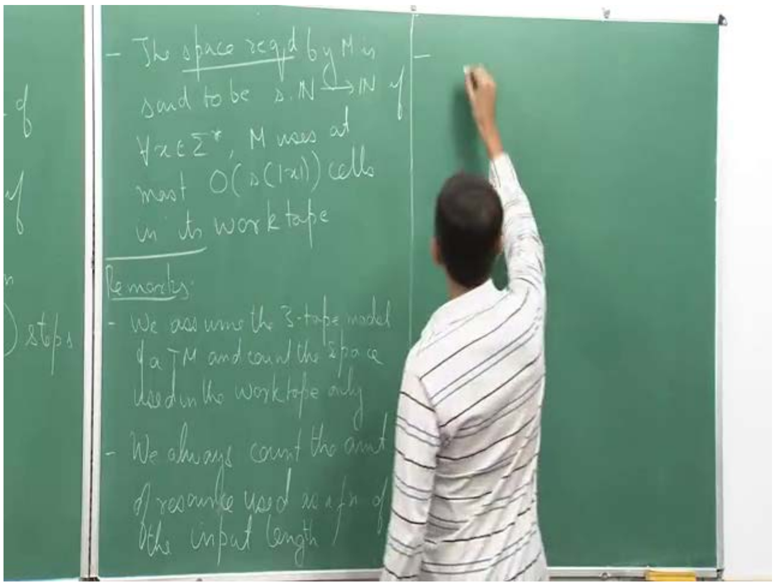
- Focus on the class of decidable languages
- From now on we will assume that all our TMs are halting TMs

Defn.

So as I said that we will focus on the class of decidable languages. So, therefore, from now on we will assume that all our Turing machines are halting Turing machines, of course, unless otherwise specified; if I specify something else then of course, we will assume otherwise, but by default we will assume all the Turing machines are halting Turing machines. So, what do we study in complexity theory, what are the kind of questions that we want to address. So, first what we will do is that within the class of decidable problems, we will classify them based on the amount of resources that a particular language uses or a particular Turing machine of a language uses to rejects answers. So, resources can be many things, so resources can be time, they can be space.

So if you are looking at let say time is a resource, and then we will try to see how much time it takes for a Turing machine to come up with its answer. And depending on the amount of time, we will classify languages in to different classes. So, the class of language which takes more time should be in one class, class of languages which take lesser time should be in other class and so on. Similarly, for resource like space also we give a space based on the amount of cells that the Turing machine uses to come up with its answer. So, this gives us classes, and then we will try and then we will study relation between these classes, so that is the goal of complexity theory that how are these classes connected to each other. Whether we can show the classes are the same, whether we can separate the classes, what kind of separation of are known and so on, these are the kind of questions that we try to addressing complexity theory. So, first we need to define what we mean by running time and space used by a Turing machine.

(Refer Slide Time: 04:15)



- The space reqd by M is said to be $s: \mathbb{N} \rightarrow \mathbb{N}$ if $\forall x \in \Sigma^*$, M uses at most $O(s(|x|))$ cells in its work tape

Remarks:

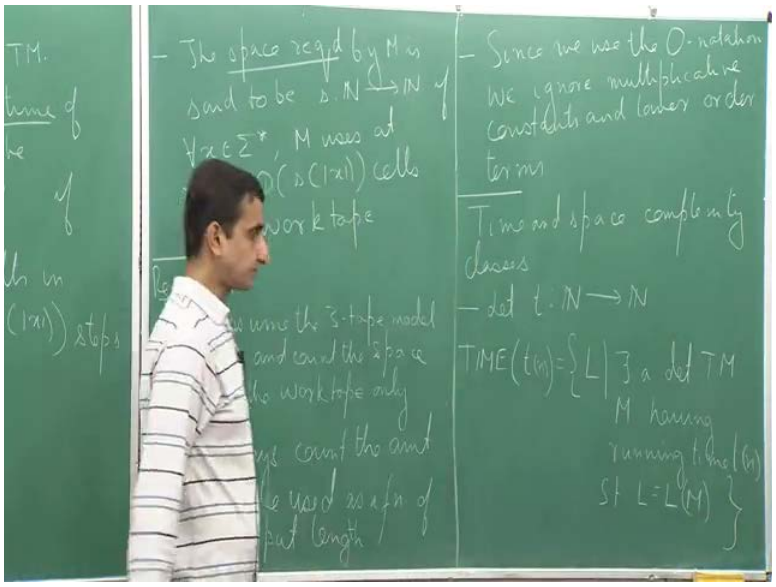
- We assume the 3-tape model of a TM and count the space used in the work tape only
- We always count the amt of resource used as a fn of the input length

So let us look at a few definitions. So, let M be a deterministic Turing machine. The running time of M is said to be T, which is a function from natural numbers to natural numbers, if for all x in sigma star. So, if all x is in sigma star, M halts in at most order T of mode x number of steps. So, we say that the running time of a Turing machine in some function T, if for every x the machine halts in at most order T of mode x steps. So, there are several important points in these definitions. First is that we always compute the running time as a function of the length of the input. And not only for running time, even when we look at other resources such as space, the amount of resource that is used by a Turing machine is always a function of the length of the input. So if x is my input, it is a function of the length.

Next is that here we talk about the running time, we say that the running time is T if input on x, the machine M halts in at most this many number of steps. So, it can halt in much few number of steps also, but it should not take more than this many number of steps. And the third point is the use of the order notation. So, remember that by use of order notation, we are actually ignoring multiplication by constants and lower order terms. For example, if t is let say the function N square, it maps N to N square then we say that the machine as running time N square if for every x the machine halts in at most order N square number of steps. So, it can take for example, N square plus N or 100 N square minus 2 N plus 3 or any such thing which belongs to the class order N square.

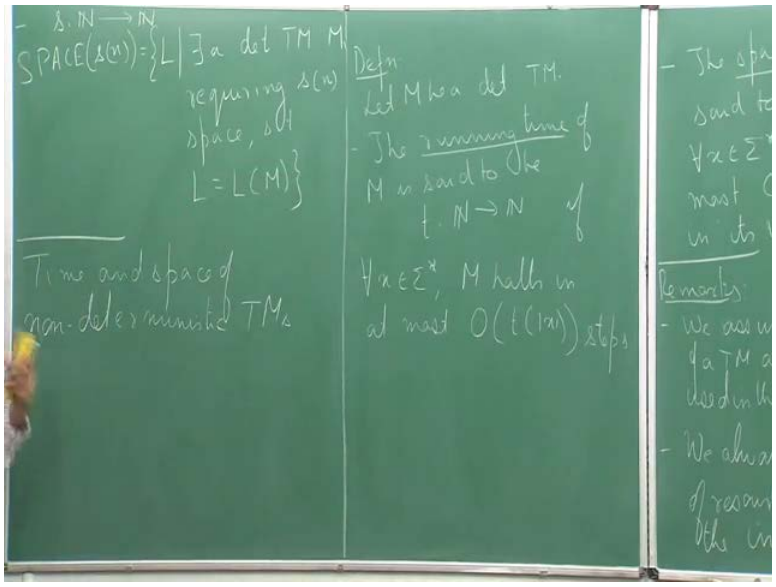
So, similarly we define the space required. The space required by M is said to be the function s going from N to in if for all x in sigma star M uses at most order s of mod x cells in its work tape. So, one additional point that is important here is that when we are looking at space complexity. It is important that we keep in mind the model that we defined or we discussed last time, so we have Turing machine not last time we talked about reduction. So we have a Turing machine which as three tapes - input tape, work tape and output tape. So, when we talk about space complexity of a machine, we do not consider the space used in the input tape, and the output tape. They can be anything that is any arbitrary amount. We only look at the space used in the work tape of the Turing machine, the amount of cells used in the work tape. So remarks, so let me quickly mention what I said. So, we assume the three tapes model of a Turing machine, and count the space used in the work tape only; second is we always count the amount of resource used as a function of the input length. So this is very important.

(Refer Slide Time: 11:20)


– Since we use the O-notation we ignore multiplicative constants and lower order terms
Time and space complexity classes
– det t : N ⟶ N
TIME(t(n)) = { L | ∃ a det TM M having running time t(n) st L = L(M) }
– The space reqd by M is said to be s : N ⟶ N if ∀x ∈ Σ*, M uses at most O(s(|x|)) cells work tape

Since, we use the big O notation. We ignore multiplicative constants and lower order terms. So, now that we have defined time and space complex or we have defined what we mean by running time of a machine and the space used by a machine, we can define the time and space complexity classes. So, let t is a function from natural numbers to natural numbers. So, this is the time function that we will have. We define the class d time t or we will just call it as time t, time t of n has the class of languages L such that there exists a deterministic Turing machine M having running time t of n such that L equals L of M. So, time T of N consists of all languages, for which, there is some Turing machine that have running time t of n and it accepts the language L. So, basically all languages for which we have some algorithm or some program which runs in that many number of steps so that is the idea.

(Refer Slide Time: 14:37).


- $s : \mathbb{N} \longrightarrow \mathbb{N}$

$\text{SPACE}(s(n)) = \{ L \mid \exists$ a det TM M requiring $s(n)$ space, s.t. $L = L(M) \}$

Time and space of non-deterministic TMs

Defn

Let M be a det TM.

- The running time of M is said to be the $t : \mathbb{N} \rightarrow \mathbb{N}$ if $\forall x \in \Sigma^*$, M halts in at most $O(t(|x|))$ steps

- The spa

said to

$\forall x \in \Sigma^*$

most

in its

Remarks:

- We ass

if a TM a

used in th

- We alwa

of resour

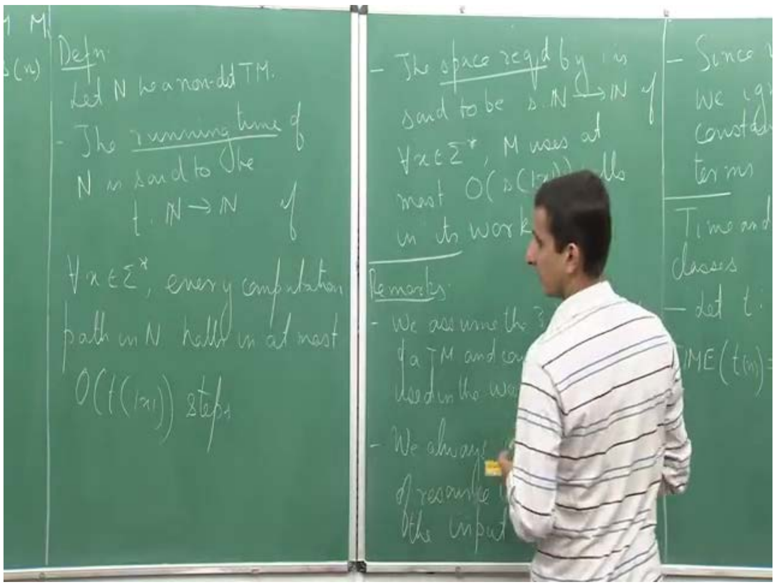
of the in

Similarly, for space also we define space t of n as the class of languages L such that there exists a deterministic Turing machine M requiring, so for space, let me use just for more clarity let me use the function small s of n. Once again s is a function from natural numbers to natural numbers so requiring s of n space such that L equals L of M. So all those languages for which there is a Turing machine that takes order I mean that uses at most order s of n number of cells that runs in space s of m. So, this is how we define the class time t n in the class space s of n. So, in text you will also find them defined as d time and the class d space, but essentially they are the same.

So, now, we can also define all this is for deterministic Turing machine. So, we can also look at time and space of nondeterministic Turing machines. So, in the case of nondeterministic Turing machines, we just have to make a call so how does a non deterministic Turing behave Turing machine behaves.

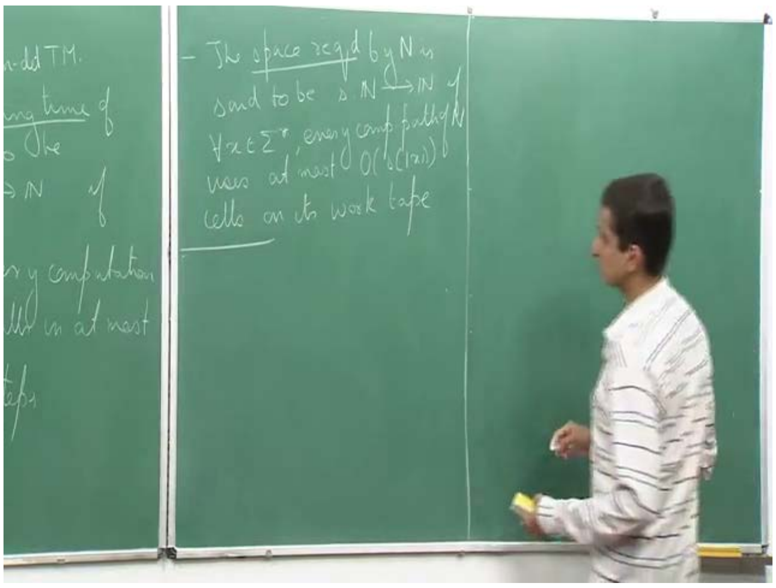
So, recall that in a nondeterministic Turing machine the Turing machine can non-deterministically branch of at every step. So, at every step during its computation, it can non-deterministically decide to do may be one of two different things. So, therefore, it will have more than one computation paths that will lead to a accept or a reject state. And therefore, when we talk about running time and the space required, we have to be little careful. So, the way we defined the running time of a nondeterministic Turing machine is we look at the maximum time that is required over all the computation paths. So, whatever is the maximum running time over all computation paths?

(Refer Slide Time: 18:03)


Defn:
Let N be a non-det TM.
- The running time of N is said to be $t : \mathbb{N} \to \mathbb{N}$ if $\forall x \in \Sigma^*$, every computation path in N halts in at most $O(t(|x|))$ steps
- The space reqd by is said to be $s : \mathbb{N} \to \mathbb{N}$ if $\forall x \in \Sigma^*$, M uses at most $O(s(|x|))$ cells in its work
Remarks:
- We assume the
- We always
- Since
We ig
Time and
classes

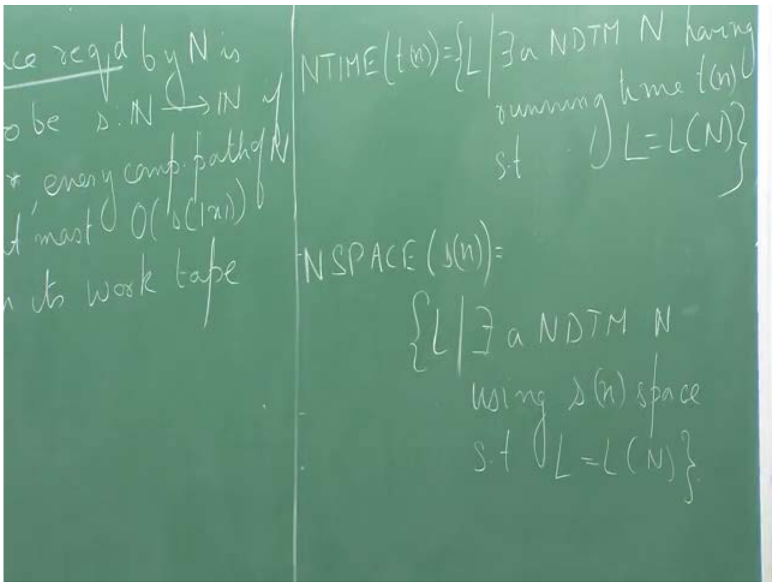
So, let me just define here. So, let N be a nondeterministic Turing machine. The running time of N is said to be a function t from N to N, if for all x belonging into sigma star. Every computation path in N halts in at most order T of x steps.

(Refer Slide Time: 19:18)


- The space reqd by N is said to be $s : \mathbb{N} \to \mathbb{N}$ if $\forall x \in \Sigma^*$, every comp path of N uses at most $O(s(|x|))$ cells on its work tape

Similarly, when we talk about the space we look at the maximum space that is used over the all computation paths. So, the space required by N is said to be a function s from natural numbers to natural numbers such that N uses or such that every computation path of N uses at most O s of mod x cells on its work tape.

(Refer Slide Time: 20:32)



ce reqd by N is

o be $s : \mathbb{N} \rightarrow \mathbb{N}$ if

r every comp. path of N

t most $O(s(|x|))$

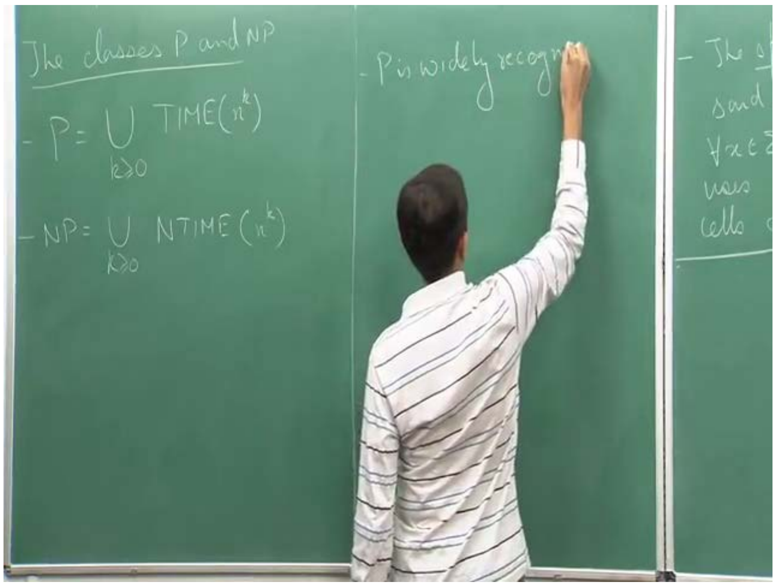
n its work tape

$NTIME(t(n)) = \{L \mid \exists$ a NDTM N having running time $t(n)$ s.t. $L = L(N)\}$

$NSPACE(s(n)) = \{L \mid \exists$ a NDTM N using $s(n)$ space s.t. $L = L(N)\}$

Analogously, we will have the time and space classes also for nondeterministic machines. So, we define the class N time t of n as the class of languages L such that there exists a NDTM - nondeterministic Turing machine, N having running time t of n such that L equals L of N. Similarly, N SPACE s of n is defined as the class of all language is L such that there exists a nondeterministic Turing machine N using s of n space such that L is equal to L of N. So, this is how we defined deterministic time and space classes as well as non-deterministic time and space classes. So, now let us look at some specific deterministic and non-deterministic classes. In fact, we will define the popularly known classes P and NP.

(Refer Slide Time: 22:36)


The classes P and NP

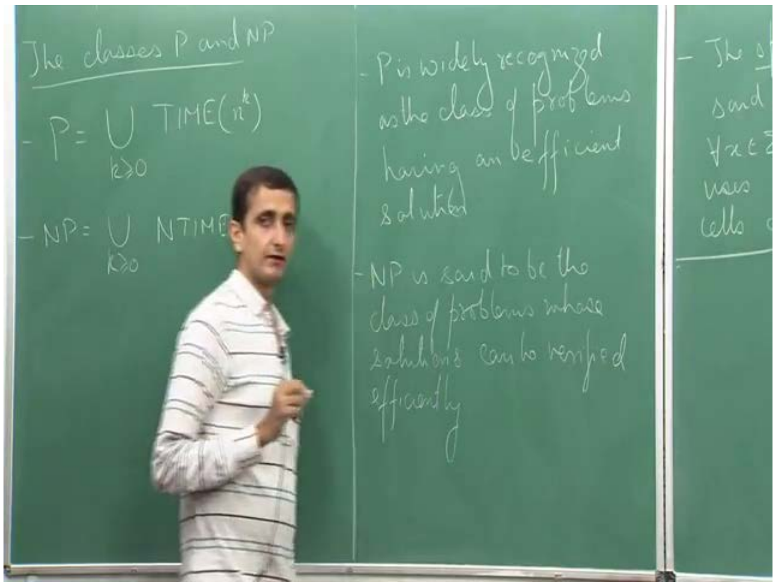
- $P = \bigcup_{k \geq 0} TIME(n^k)$
- $NP = \bigcup_{k \geq 0} NTIME(n^k)$

- P is widely recogn

- The s
said
$\forall x \in \Sigma$
uses
cells

So, the classes P and N P. So, P is defined as the union over all k greater than or equal to let say 0 of time n raise to the power k. So, look at all problems that have a running time either n to the power 0, which is constant or n or n square or n cube or n to the power 4 and so on. And the class of all these problems is basically P, so all problems which have a running time that is polynomial. So, if I say that a language belongs to P that means, it belongs to N to the power k for some k, so it has for some constant k. So, similarly the class NP of N TIME n raise to the power k.

(Refer Slide Time: 25:05).


The classes P and NP

- $P = \bigcup_{k \geq 0} TIME(n^k)$
- $NP = \bigcup_{k \geq 0}$ NTIM

- P is widely recognized as the class of problems having an efficient solution
- NP is said to be the class of problems whose solutions can be verified efficiently

- The s
said
$\forall x \in \Sigma$
uses
cells

So, P is widely recognized as the class of problems having an efficient solution or algorithm. So, I mean it is widely recognized that if we have a problem that is in P, it means that we can actually compute a solution to this problem in an efficient manner. For example, if we have a computer we can actually write down an algorithm or a code for this problem that will produce an answer in a reasonable amount of time quite quickly. On the other hand, NP is said to be the class of problems, whose solutions can be verified efficiently.

So, actually in our next lecture, we will give an alternate definition of NP which will make this idea more clear. But I just want to give a perspective of what we mean here so there are certain problems for which we can actually solve the problem in an efficient manner in some polynomial number of steps. Such as matrix multiplication or give an graph I want to find out whether I can go from one vertex to another or given an array let say I want to find out the minimum element in the array and so on. So, these are the problems that for which we say there is an efficient solution these are the problems which are in P there are some other problems for which it might be difficult to come up with the solution. So, we do not know how complex I mean whatever knowledge that we have it does not seem lightly that a solution can be computed efficiently, but if somebody gives us a solution we can verify it efficiently.

For example, think of the games Sudoku. So, in a game of Sudoku, if somebody asks you to solve Sudoku, it is more challenging than if somebody than gives you solve board of Sudoku and asks you to check whether it is a correct solution or not. So, if somebody fills out the boxes and then asks you whether this is a correct solution or incorrect solution, you can very easily sum up and check whether the solution is correct or not that is very efficient, but if you have to come up with up your own solution that is more difficult. So, there are other such problems for which coming up with a solution is I mean considerably more difficult then verifying whether a solution is correct, so that is the in some sense that is the difference between P and the class NP.

So, next time, we will look at more about these classes, we will look at more examples of problems that are in P and N P. And we will also look at alternate definition of the class NP which is more in line with this fact which I just mentioned that NP is the class of problems whose solutions can be verified in an efficient manner. I will stop here today.

Thank you.