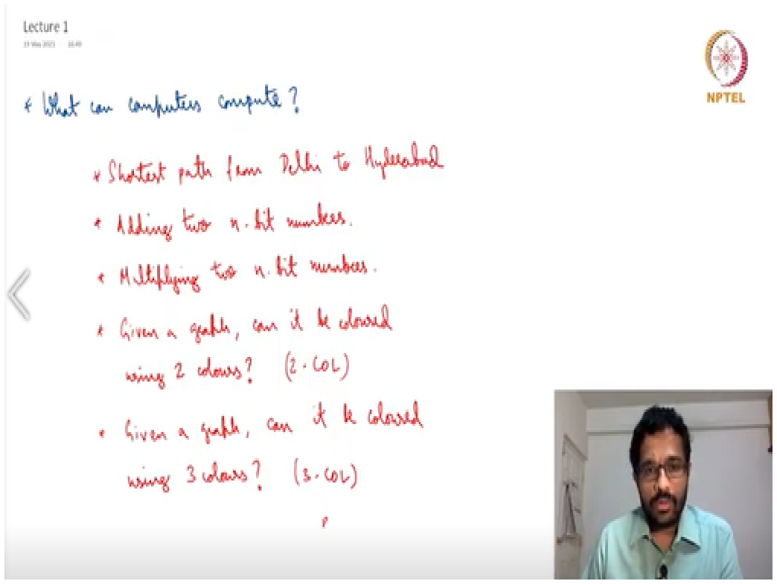
Computational Complexity

Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture -01
Introduction to Computational Complexity

(Refer Slide Time: 00:15)



Lecture 1

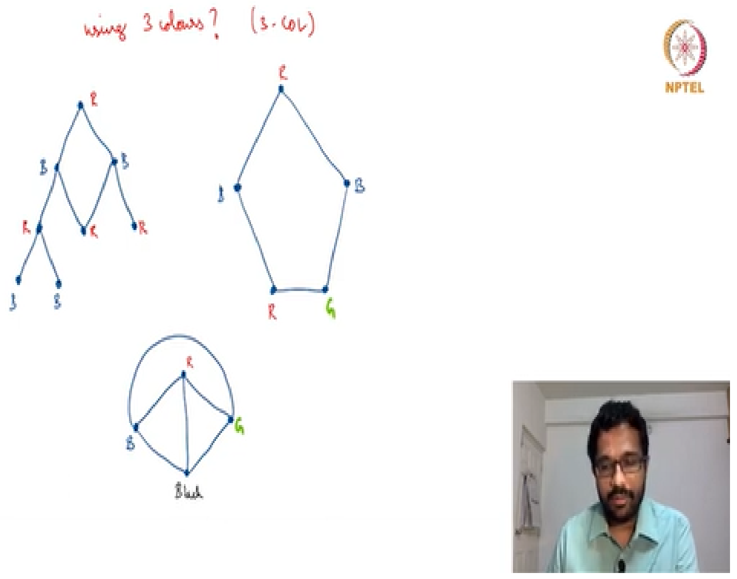
* What can computers compute?
  * Shortest path from Delhi to Hyderabad
  * Adding two n-bit numbers.
  * Multiplying two n-bit numbers.
  * Given a graph, can it be colored using 2 colors? (2-COL)
  * Given a graph, can it be colored using 3 colors? (3-COL)

Welcome to the first lecture of the course computational complexity. So, let me just go through some parts that I had already covered in the introduction video that you must have seen on the Swayam portal. And so, I will explain very briefly what is the scope of the course and what we set out to do during the next 12 weeks. So, the basic goal of this course is to set up a framework to understand the power of computation.

As I had said in the introduction video, there are many computations that we perform ourselves using our brain. But the scope of the course is to set up a model of computing or set up several models of computation and see what can these models of computation achieve. So, even though we will see abstract models the overall goal is to understand what can computers compute.

So, I mentioned several of these problems like what is the shortest path from one city to another? How do you add two n digit numbers? How do you multiply two n digit numbers? So, you may know that or you may guess by just by practice of adding and multiplying numbers that multiplying seems to be a harder problem than adding two numbers. You have more things to compute, more digit wise products to compute, more additions to perform. So, multiplying multiplication seems a harder problem. But then how do you formulize that?

(Refer Slide Time: 01:53)

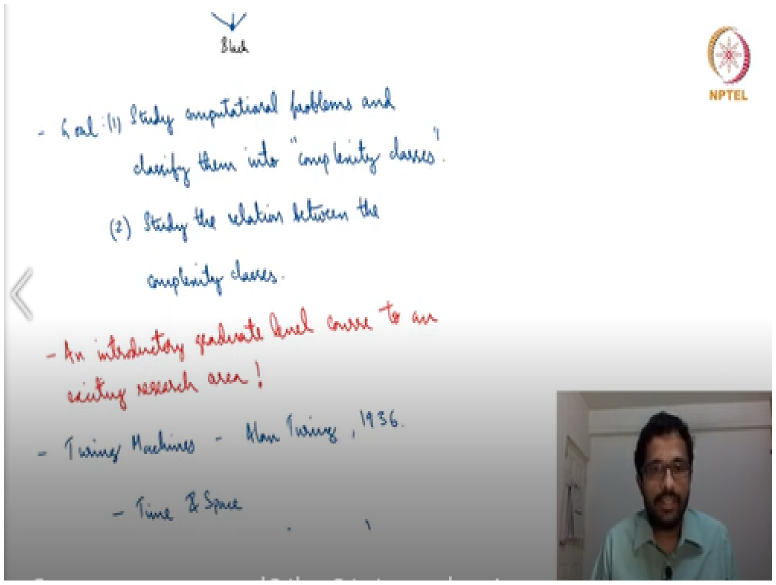


Another problem is given a graph can it be coloured using two colours. So, this tree in the left-hand side can be coloured using two colours. Whereas these five cycles in the right-hand side, the pentagon shaped graph cannot be coloured using two colours. We need red R stands for red and blue, B stands for blue, in addition to red and blue we also need a third colour green. So, now if I give you a graph and ask you can it be coloured using three colours.

So, by the way when I say coloured meaning, coloured in such a way that no two neighbours have the same colour. So, now the top vertex red, and the bottom left vertex red is ok, because these two are not directly connected by an edge. So, these two reds are ok and these two blues are also ok. But then this red and this blue have to be two different colours.

So, once we colour these four vertices red, blue, blue, red, this vertex cannot get red or blue, because one neighbour is red and another neighbour is blue, so it has to be coloured green. So, these are computational problems: given a graph can it be coloured using two colours, can it be coloured in three colours. So, now one of them turns out to be easy and one of them turns out to be harder. So, why is one easy and why is one harder. So, we will try to explore these questions during this course.

(Refer Slide Time: 03:26)



Black

- Goal: (1) Study computational problems and classify them into "complexity classes".
  (2) Study the relation between the complexity classes.
- An introductory graduate level course to an exciting research area!
- Turing Machines - Alan Turing, 1936.
  - Time & Space

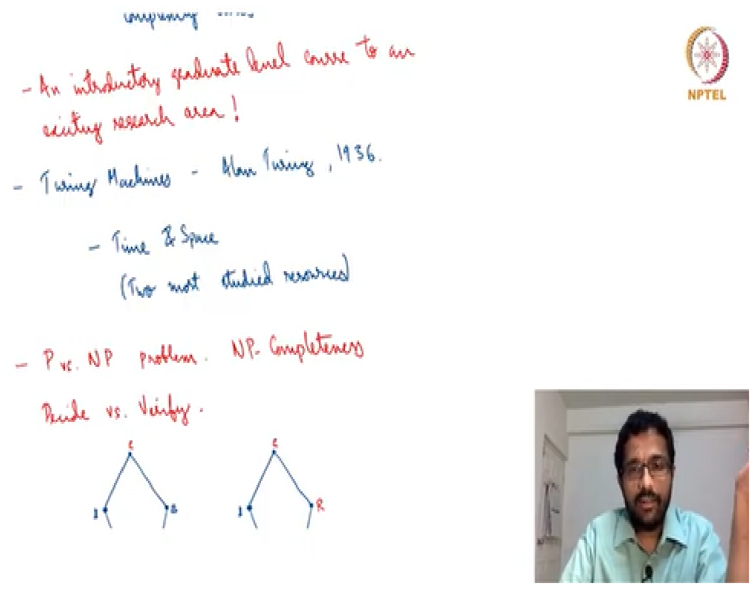
So, overall, what I can say is we will look at various computational problems and classify them into what is called complexity classes. So, we will have several complexity classes that we will see and different models of computation also. So, based on a certain model of computation we will arrive at several different complexity classes based on some other model of computation we will arrive at some other complexity classes.

Sometimes we will also try to see how these classes relate to each other. Does this class contain this class? Does this class is a subset of this class? Or the superset of this class? Are these two classes disjoint like no problem can be in this class as well as that class? So, the goal is to understand what computers can perform. So, to formally understand this we will take the help of these things called complexity classes.

So, and this course is perhaps the first graduate level course, if you are interested in computational complexity as a research area. It is a vast research area with lots of exciting work happening. It is people started thinking about it during the 70s, picked up during the 80s and people have been, it has been taking various paths and various directions throughout the last 30 or 40 or 50 years.

So, it is an exciting research area and if you like to understand this research area and start to like to get started on this area this course would be very good and appropriate for you to get started on it. So, it is like an introductory graduate level course, graduate level in since it is a not an undergraduate it is a PG level course.

(Refer Slide Time: 05:18)



- An introductory graduate level course to an exciting research area!
- Turing Machines - Alan Turing, 1936.
    - Time & Space
      (Two most studied resources)
- P vs. NP problem. NP- Completeness
  Decide vs. Verify.

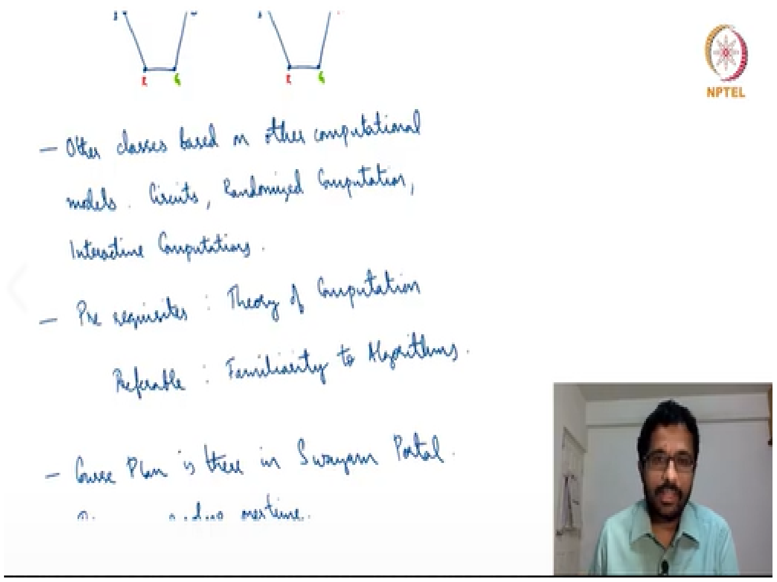
So, the most basic model of computation that are used commonly to model computers are what are called Turing machines. This was developed by Alan Turing in 1930s and he was a British scientist. And we will see the Turing machine model, and the two main resources that pertain to the Turing machine model of computation are time and space. If you have taken an algorithm class.

I hope most of you have taken an algorithms course or at least watched NPTEL video of an algorithms course you would have come across the notion of time complexity like this algorithm sorting takes $nlogn$ $nlogn$ time whereas bubble sort some other sort takes n^2 n^2 time. So, we already have this notion of time complexity, so you might have already been familiar with this.

Another equally important resource that we will also study is the space, how much space does a certain computation take? Even this also you may have seen, you might have seen how much extra memory does a certain computation use. So, we will see all this more formally. One of the most famous problems in computational complexity or even all of computer science is the P versus NP problem.

So, we will see this also during this course, so you may also have heard P versus NP maybe you do not know what it is maybe you know very superficially but we will very formally have a good understanding of what P versus NP is. So, that when somebody comes along and asks you what it is as computer scientists you can explain what is P versus NP problem.

(Refer Slide Time: 07:12)



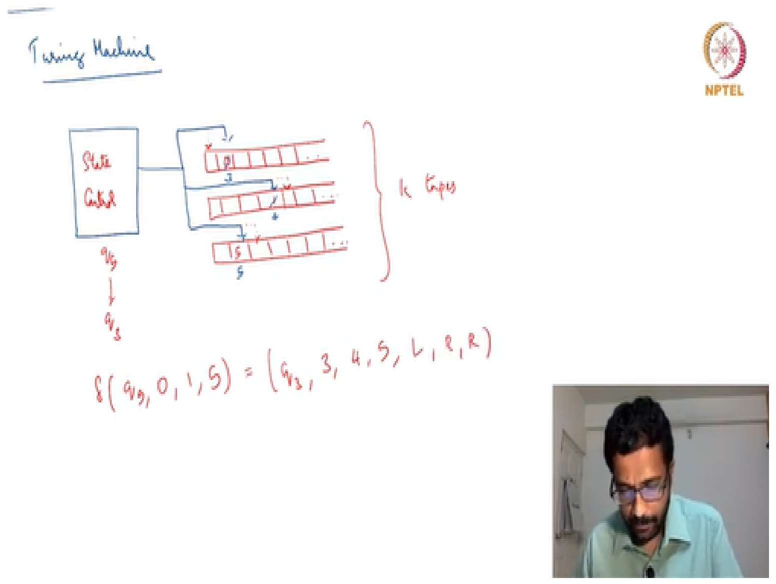
- Other classes based on other computational models. Circuits, Randomized Computation, Interactive Computations.
- Pre requisites : Theory of Computation
  Preferable : Familiarity to Algorithms.
- Course Plan is there in Swayam Portal.
  [illegible] meetime.

So, we will start with the Turing machine model of computation. Soon we will see other models of computation also like randomized computation where we use random bits. We will also see circuits which is another very important model of computation where we do not have a Turing machine with the tape and heads moving and all that but instead you have gates, logic gates, Boolean gates that make the computation.

There are also other paradigms of computing like interactive computation. We will see some of that and corresponding to each of these models like randomized circuits, interactive computation, we will see different complexity classes. So, in randomized computation for instance bits the number of random bits is a resource. In circuits the resources that people bother about are number of gates, size of the circuit and something called depth of the circuit and so on.

So, pertaining to each of these computational models there are different resources of interest and computational complexity classes based on how much resources are taken to achieve a certain computation or to solve a certain computation problem will be studied. As indicated in the Swayam portal the prerequisite one of the courses Theory of Computation I hope most of you have taken a theory of computation course or at least see gone through the materials of a theory of computation course and are familiar with it. A good indication would be to try to solve the assignment zero and see how comfortable you are with the concepts. If you are familiar with Introduction to Algorithms, and have done an undergraduate algorithms course like basic algorithms course that would also be very good. But I am not insisting on that you can you can pick that up along the way.

(Refer Slide Time: 09:09)



The course plan is there and this Swayam portal. You can go through the Swayam portal there is a course plan already there. And let me just say that it may we may modify it slightly as we go along because we are just beginning the course. So, perhaps we may end up spending a bit more time on some topics or a bit less time on some topics or some slight reconfiguration may happen.

So, maybe it is possible that some topic gets stopped or some but largely we will stick to that. So, with that out of the way let me go to the first part of the course which is a Turing machine model. Again, if you have seen the theory of computation course you would have seen the Turing machine model but let me just briefly recap some of the topics. That are important and perhaps it is good to revise.

So, Turing machine contains a state control, so this is the state control this is the brain of the Turing machine. And then there are three or not three, in this picture there are three. But there are k tapes, in this picture there are three tapes but it could be k tapes where k is some finite number it could be one. It could be more than one also in this picture there are three.

And there are tape heads that read the tapes. And these tapes are usually infinite and we have depicted them as infinite to one side. So, they will start there is a left endpoint but then from there it is infinite tapes. And what happens is that these heads read something from these tapes, and then they will be in a certain state, so they could be in a state called let say q_5 q_5 and here they read a 0.

So, let us say here they read a 1 and here they read let us say 5. So, if you are at q_5 q_5 and you read a 0, 1 and 5 maybe the instruction is that you move the first one to the right to the left let us say, so I am putting it in dotted line the second tape to the right let us say, and the third tape to the right let us say, and in from q_5 q_5 you move to let us say q_3 q_3.

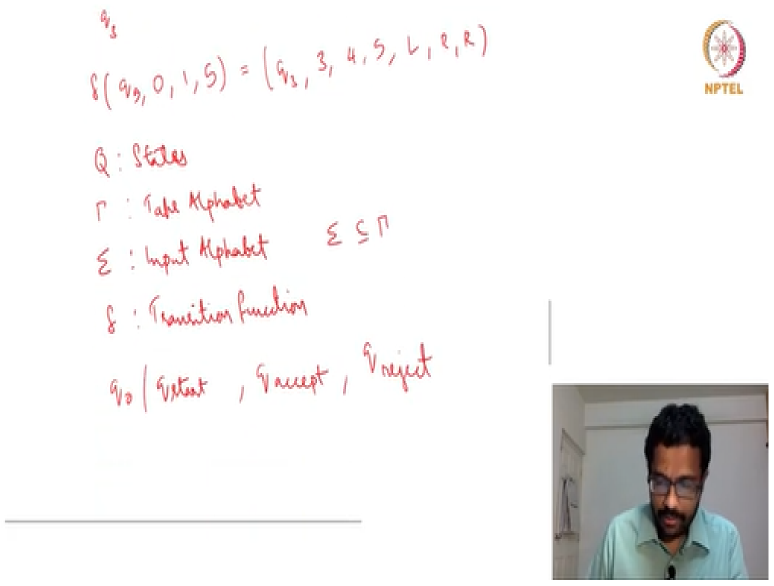
So, this could be a move. So, this would be depicted like-:

$$\delta(q_5, 0, 1, 5) = (q_3, 3, 4, 5, L, R, R)\ \delta(q_5, 0, 1, 5) = (q_3, 3, 4, 5, L, R, R)$$

δ δ is a symbol for the transition function. So, this means if you are at q_5 q_5 state 0, 1, 5 is what you read and this equals to q_3 q_3. So, in addition it can also do one more thing. It could modify the tape content so maybe zero was modified to let say 3, 1 was modified to 4 and let us say 5 was modified or not modified let us say it remains 5.

So, you will say so first you would say the first tape head moves left, second tape it moves right and the third tape head moves right. I think we denote it like a bit differently you first write what you 3, 4, 5 because that is what you write on the tape heads 3, 4, and 5 and then left right and right. So, this is how you depict the movement of the Turing machine.

(Refer Slide Time: 13:30)



q_3

$\delta(q_5, 0, 1, 5) = (q_3, 3, 4, 5, L, R, R)$

Q : States

Γ : Tape Alphabet

Σ : Input Alphabet $\quad \Sigma \subseteq \Gamma$

δ : Transition function

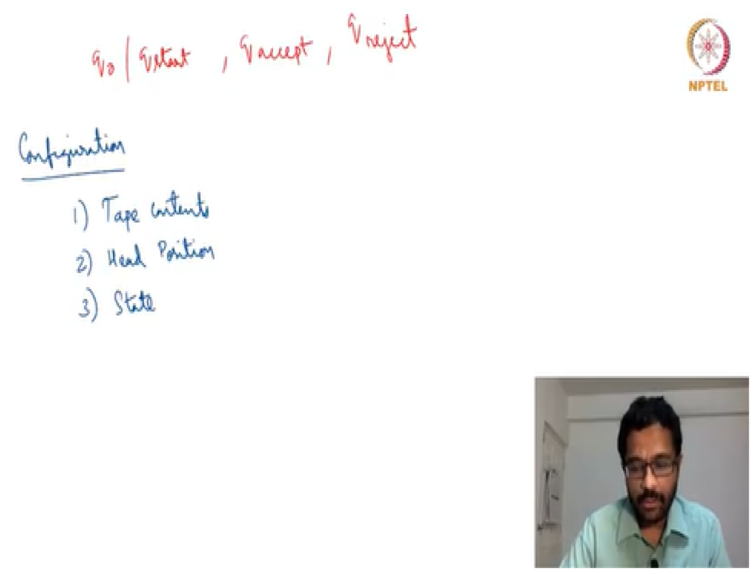
$q_0 \mid q_{start}$, q_{accept} , q_{reject}

So, this is so there is something called states, so the set of states is called Q. There is a tape alphabet Γ Γ, Σ Σ is the input alphabet and Σ Σ must necessarily be contained in Γ Γ. So, you could use some extra symbols for computation to be written on the tape and δ δ is the transition function then there are special states called q_0 q_0 or sometimes it is called q_{start} q_{start} which is the starting state then q_{accept} q_{accept} which is the accepting state and q_{reject} q_{reject} which is the rejecting state.

So, what we will initially see is what is called decision problem. So, you will have to decide. So, like we saw here: Is this graph colourable using three colours? So, that is a yes or no question these things are called decision problems. So, like multiplying two numbers is also a computational problem, but that is not a yes or no problem. It is a problem whose answer is another number. So, we will initially we look at decision problems.

So, which means we give something we give an input and then the machine has to decide yes or no. So, that is why we have accept and reject states. So, the machine once it enters the accept or reject state it stops the computation. So, this is very briefly the overview of a Turing machine it is a very quick recap. So, for more details please revise the theory of computation.

(Refer Slide Time: 15:30)



q_0 | q_{start} , q_{accept} , q_{reject}

Configuration

1) Tape contents
2) Head Position
3) State

And we have one other thing called configuration. So, this is a snapshot of the Turing machine so if somebody has to continue the computation from where we leave the computation. Let us say we temporarily stop the computation let us say I am running the computation and I want you to continue the computation from where I left. So, what information should I convey to you so that you can continue the computation.

So, you can think of it as a board game let us say we both are playing a board game. Now let us say we are playing for instance chess, but then due to whatever reason we have to stop this game now. And we will decide to resume the game at a later date but then you do not want to keep the chessboard open at your home like that with all the pieces and all. So, what will you record? What information do you have to record?

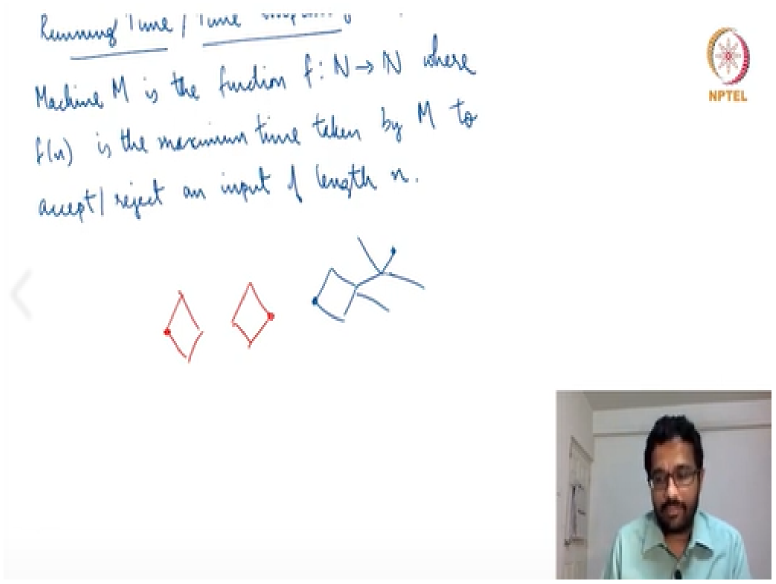
So, that next time when we meet, we can resume the game. So, you have to know which are the pieces that are there on the board? Where these pieces are exactly? Which is the location of each of these pieces? And perhaps you also need to know whose turn is it next? So, like that if you stop the computation of the Turing machine and later want to resume what are the information that you need.

So, there are three pieces of information that will be required: one is tape contents, much like what are there in the chess board. So, this is not an exact analogy but you get the idea, tape contents what are the all these? So, all these tapes will have some content let us say maybe the first one says 1, 3 let us say x, a, 3, 5, some something like this. So, we need to specify what are there in each of the tape. Then head position.

So, here in this picture the heads are at point 2 and maybe later it moved to 1 here. It was in 1, 2, 3, 4th position and then it moved to 5, here it was in second position and then moved to 3. So, whichever point the heads are we should be able to convey we should convey. And finally, what is the state that it is in. So, these things like $q_3, q_5, q_{accept}, q_{reject}$ $q_3, q_5, q_{accept}, q_{reject}$. It is some device by which the machine remembers what it is doing. It is kind of indicators for the machine. So, the configuration is tape content, head position and state. These are the three main ingredients that we need to remember if you want to resume the computation. And so, this is like a snapshot of where the Turing machine is now.

So, this is what is called the configuration. So, the reason I am saying later we will use it, so this is a quick recap.

(Refer Slide Time: 19:10)



Running Time / Time

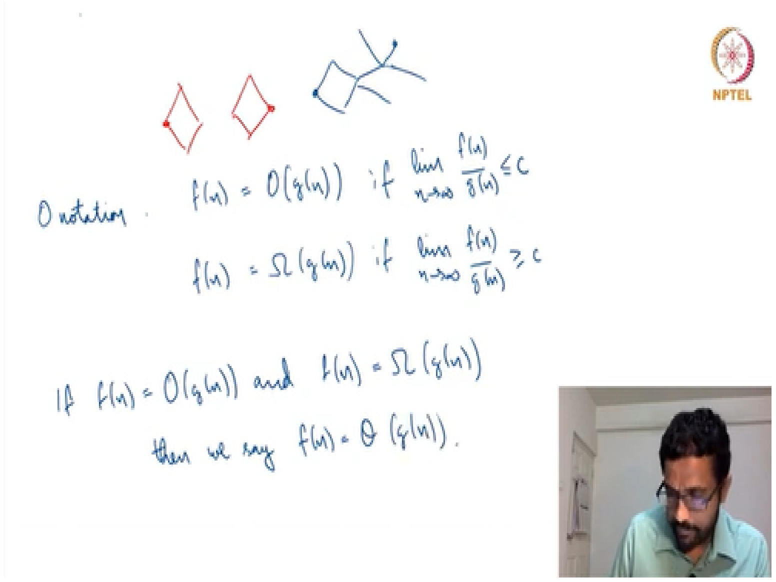
Machine M is the function f : N → N where f(n) is the maximum time taken by M to accept/reject an input of length n.

What is a Running time? Again, you would have seen it. So, given a particular problem instance how long does it take to compute? So, running time sometimes it is called time complexity of a Turing machine M is a function $f : N \rightarrow N$ $f : N \rightarrow N$ where $f(n)$ $f(n)$ is the maximum time taken by M to accept or reject an input of length n. So, basically it is the maximum time taken to accept or reject an input of length n.

So, we will see examples of this. So, for example to decide whether a graph is connected. Let us say given a graph, does the graph look like this where everything is connected to everything else or does the graph look like this where everything is not connected. So, the red graph is in two pieces, if you start from here you cannot go to here. Whereas a blue graph is connected, so if you start here you can go to any other vertex, there is a path to any other vertex.

So, if I ask you whether a graph is connected this can be completed in some time. So, what is the time? That would be the running time. So, if there is a graph on n vertices then you can do this in order n square time checking the connectivity. So, you can you can do better you can specify this better but I am not getting into vertices versus edges and so on.

(Refer Slide Time: 21:54)



O notation. $f(n) = O(g(n))$ if $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \leq c$

$f(n) = \Omega(g(n))$ if $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \geq c$

If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

then we say $f(n) = \Theta(g(n))$.

So, these are all basic definitions, so I am not really going into the definitions. Another important notation that you and you need to do need you to revise is the O notation. So, we say:

$$f(n) = O(g(n))\ f(n) = O(g(n)) \text{ if } \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$$

If constant c is 0 then you would say it is $o(n)\ o(n)$.

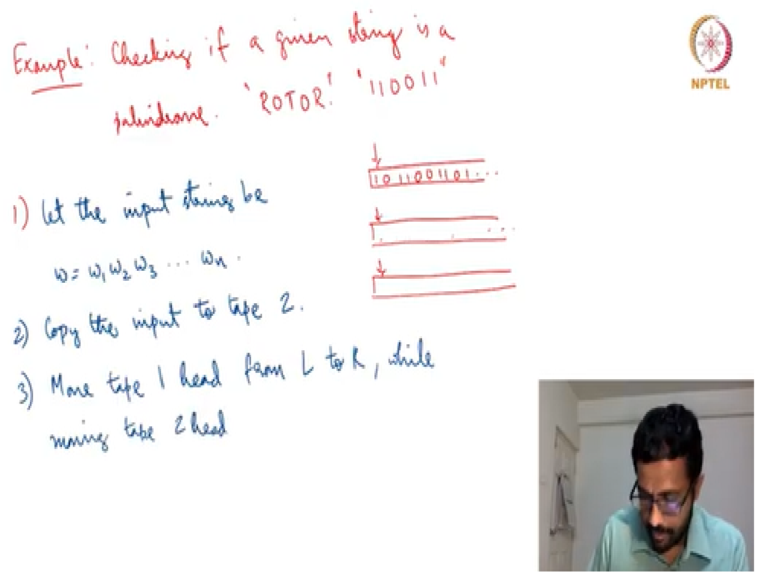
$$f(n) = \Omega(g(n))\ f(n) = \Omega(g(n)) \text{ if } \lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c \lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c$$

So, $O(g(n))\ O(g(n))$ is used for upper bounds, and $\Omega(g(n))\ \Omega(g(n))$ is used for lower bounds, again you would have seen these definitions. Another important definition is:

If $f(n) = O(g(n))\ f(n) = O(g(n))$ and $f(n) = \Omega(g(n))\ f(n) = \Omega(g(n))$

then we say $f(n) = \Theta(g(n))\ f(n) = \Theta(g(n))$

(Refer Slide Time: 23:35)


Example: Checking if a given string is a palindrome. 'ROTOR' '110011'

1) Let the input string be

$w = w_1 w_2 w_3 \cdots w_n$.

2) Copy the input to tape 2.

3) Move tape 1 head from L to R, while moving tape 2 head

So, this O, Ω, Θ O, Ω, Θ these are all basic definitions. So, maybe we will do a very quick example.

Example: Checking if a given string, is a palindrome.

So, palindrome is words like 'rotor' which reads the same if you go from right to left or left to right. So, 'rotor' is a palindrome. So, whether a given word or a given string is a palindrome is the question. So, maybe another string is for instance 1 1 0 0 1 1 is a palindrome.

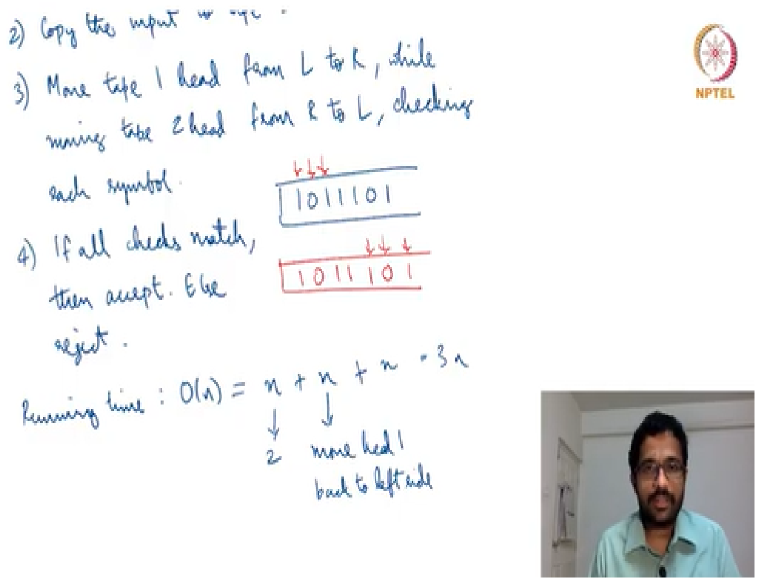
So, if you read left to right and right to left 1 1 0 0 1 1 reads the same. So, how will a Turing machine accomplish this. Again, one thing that I did not say here is that it starts off by having in the first step contains the input. And all the other tapes are usually empty or it starts off being empty the first step contains the input the other tapes are empty. And all the heads are in the left most position this is the starting position of a Turing machine.

So, suppose we give a Turing machine this situation let us say the first step contains the string 0 1 0 1 something. So, I do not know it may be extending and the other tape heads are empty and all the heads start here at the leftmost point. How can this be checked whether it is a palindrome? So, maybe let the input string be w. And w is a string which means so it could contain many symbols like binary or alphabets or whatever.

1) Let the input string be $w = w_1 w_2 w_3 \dots w_n$ $w = w_1 w_2 w_3 \dots w_n$
2) Copy the input to tape 2.
3) Move tape 1 head from L to R while moving tape 2 head from R to L, while checking each symbol.
4) If all checks match then accept else reject.

If the word is rotor, then w_1 w_1 is ‘r’ then w_2 w_2 is ‘o’ and so on. Now what you can do is copy the input to tape 2. And then move tape 1 head from left to right while moving tape 2 head from right to left, checking each symbol.

(Refer Slide Time: 27:09)



2) Copy the input w [illegible]

3) Move tape 1 head from L to R, while moving tape 2 head from R to L, checking each symbol.

4) If all checks match, then accept. Else reject.

Running time : $O(n) = \underset{2}{n} + \underset{\substack{\text{move head 1} \\ \text{back to left side}}}{n} + n = 3n$

1011101
1011101

So, you see the point. So, now suppose this string is 1 0 1 1 1 0 1, this is a palindrome. So, the first tape head contains this and I am asking to copy this to the second tape 1 0 1 1 1 0 1. And then you check whether this is same and this is same from left to right. And then if these are the same you go to the next point, whether this is the same as this and these two are the same so you continue.

So, if it is a palindrome then everything should be same. If all checks match then accept else reject. So, if there is some check that does not match you reject. So, if it is not a palindrome there should be some place where this match fails. So, this is a very simple algorithm and what is the running time of this? The running time of this is $O(n)$ $O(n)$, where *n* is the length of the input, again another convention is that *n* is the usually the length of the input.

The running time here is $O(n)$ $O(n)$. Let us see why, because the input string is w, step one is not really a step. So, copying the input to tape 2 how long does that take? So, you move the tape head from in the tape 1 from left to right and while copying the same thing from left to right in the tape 2.

So, if there are n locations this also takes something like n n time. So, usually time means the number of Turing machine steps. So, where each step the computations does something like this.

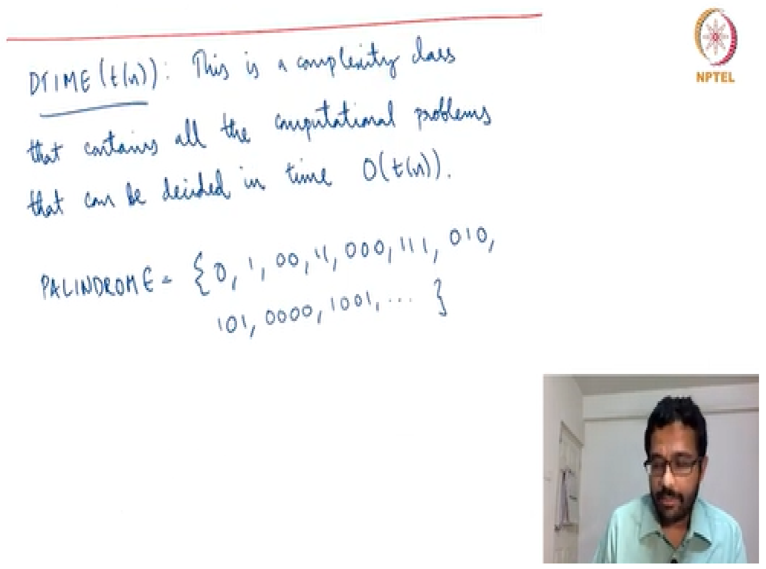
So, first copying takes n n time and then step three you need to move the tape 1 head from left to right, but after copying both the heads would be in the right side.

So, we need to move the tape 1 head back to the left side which takes another n n time. And then now head 1 is at the left side head 2 is in the right side and then you again move step by step and while matching the symbols that you see. So, this also takes order n n time.

$$O(n) = n + n + n = 3n \; O(n) = n + n + n = 3n$$

So, this is total $3n$ $3n$ which is which is $O(n)$ $O(n)$. So, what we will bother about is the function of n not these constants that is another way to remember the O notation. Again, you can revise all that. So, this is the running time of the simple algorithm.

(Refer Slide Time: 31:15)



DTIME(t(n)) : This is a complexity class that contains all the computational problems that can be decided in time O(t(n)).

PALINDROME = {0, 1, 00, 11, 000, 111, 010, 101, 0000, 1001, ... }

So, let me just make some definitions and then we will move. So, another definition that we will use is the $DTIME(t(n))$ $DTIME(t(n))$. This is a complexity class, may be the first complexity class that we are seeing in this course. That contains all the computational problems that can be decided in time $O(t(n))$ $O(t(n))$. So, this $DTIME(t(n))$ $DTIME(t(n))$ is a complexity class where $t(n)$ $t(n)$ is a function.

So, $t(n)$ $t(n)$ is like $f(n)$ $f(n)$, so this is a complexity class of all the computational problems that can be decided in $O(t(n))$ $O(t(n))$ so palindrome here it took order n time. So, $DTIME(t(n))$ $DTIME(t(n))$ will contain the language or the problem palindrome. Just another point again this is an introductory lecture where I will say some basics, so computational problems are sometimes called are languages because every problem can be written as a string.

So, for instance palindrome, so palindrome is a set of all strings let us say in binary 0, 1, 0 0, 1 1, 0 0 0,111, 010, 101. So, these are all the 1, 2, 3-bit palindromes. So, and you can write like 0000, 1001and so on. So, it is a set of all the strings, so to determine if it is a palindrome is the same as asking whether it is a member of the set.

So, sometimes so this set is sometimes called a language. So, sometimes we will use the word language to refer to a computational problem because if it is yes, no question. If it is a yes, no question, the computational problem is a decision problem then we are just simply asking whether a given string is a member of this set or not or a member of this language or not. So, it is a computational problem can be also thought of as a set or a language.

So, we will use the word sometimes language to refer to a decision problem. So I can give you a set of all the connected graphs. So, now to determine whether a graph is connected is the same as checking whether this given graph is in the set of all connected graphs. So, it is just checking membership so sometimes it is called language. So, $DTIME(t(n))$ $DTIME(t(n))$ is a complexity class that contains all the computational problems that can be decided in $O(t(n))$ $O(t(n))$.

So, if it takes time n^3 n^3 , all the problems that take time n^3 n^3 or lesser will be in $DTIME(n^3)$ $DTIME(n^3)$. So, I think we are about half an hour so I will stop this lecture and we will continue in the next lecture. Thank you.