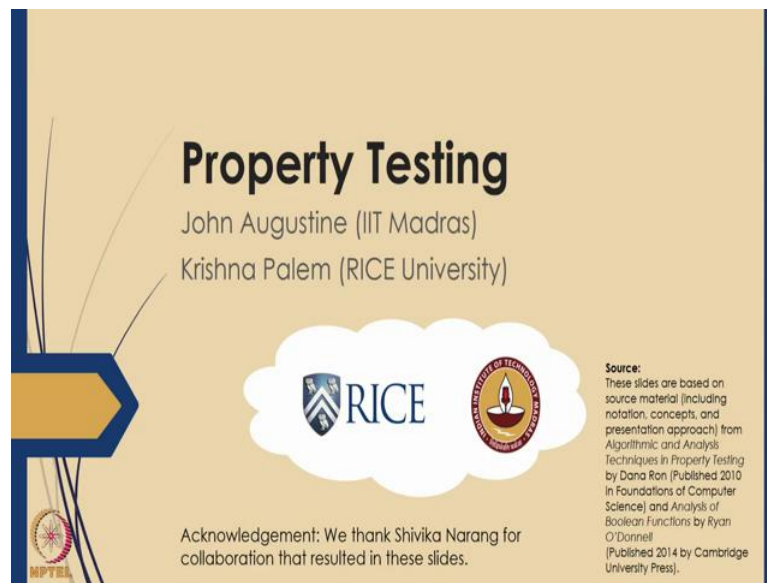


**Algorithms for Big Data**  
**Prof. John Ebenezer Augustine**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

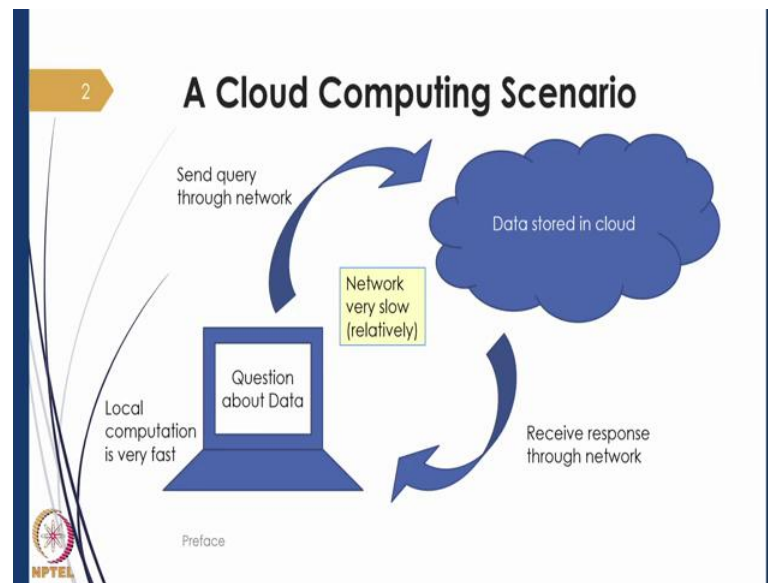
**Lecture – 37**  
**Property Testing Framework**

(Refer Slide Time: 00:15)



Hello everybody. Today we will be talking about property testing. It is an important and interesting computational framework that has been developed in the last 10 to 15 years. Then, some attempts are even earlier than that. But, in the last 10 to 15 years his particular model of computation has become quite popular. And, we will be exploring this model of computation in the context of two examples.

(Refer Slide Time: 00:44)



But, before we get into those examples, let us first motivate the computation model. One scenario that motivates a property testing is as follows. We often have data stored in a, cloud far away from the actual computer that performs a computation.

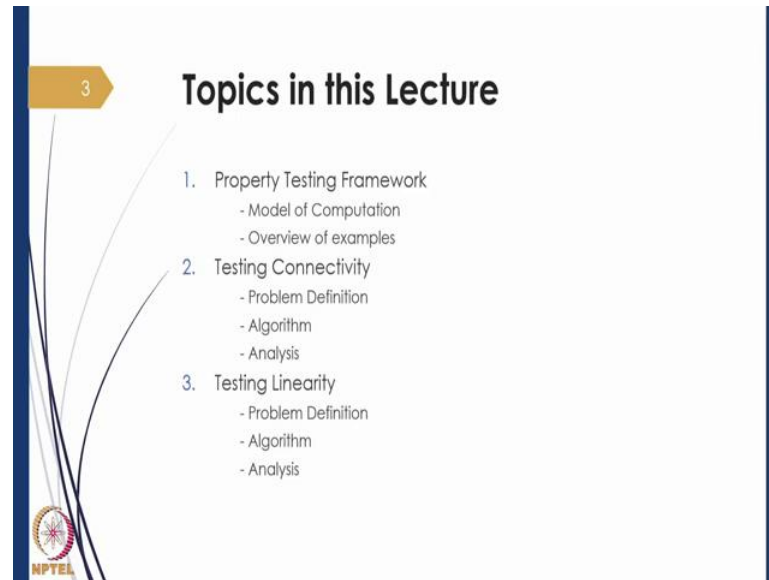
So, this is one scenario that clearly motivates our property testing. We have data. It could be a graph; it could be a large data set; it could be some complex function. This data is somehow stored in a cloud or a data center that is physically far away from the computer that actually performs the local computation. And, this local computer has some question that needs to be answered about the data.

Unfortunately, the only access to the data is through a very slow network. And therefore, any information about the data has to be accessed by queries through the network. And, these queries travel from the computer that is interested in answering the question to the data center. And then, the data center has to process the query and send back a response.

In this context, the local computation performed in the local computer is much faster than the speed at which the queries are answered. So, we will therefore need to design a computational model; that in some sense ignores local computation and is very careful about the number of queries that are sent to the data center and received from the data

center.

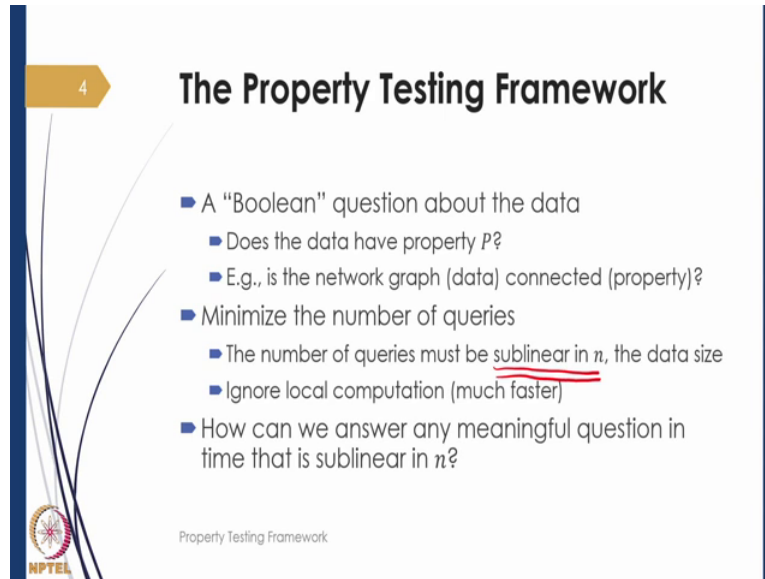
(Refer Slide Time: 02:54)



So, on to this motivation we will first provide a formal computational model and explain all the parameters that governed this computational model and present an overview of couple of examples. That will be the first segment of our lecture.

In the second and third segments, we will consider two examples. One would be the example of testing connectivity in a graph and the other would be the example of testing whether a particular Boolean function is linear or not. And, of course we will define all the terms more carefully, when we look at these examples.

(Refer Slide Time: 03:51)




4

## The Property Testing Framework

- A "Boolean" question about the data
  - Does the data have property  $P$ ?
  - E.g., is the network graph (data) connected (property)?
- Minimize the number of queries
  - The number of queries must be sublinear in  $n$ , the data size
  - Ignore local computation (much faster)
- How can we answer any meaningful question in time that is sublinear in  $n$ ?

Property Testing Framework



So, now let us look at the property testing framework and build an understanding of what the property testing model is. At the core of the model is a "Boolean" question that we need to answer about the data and recall that the Boolean question has to be answered in a local computer, whereas the data is stored in a faraway data center or cloud or whatever you will. And, typically the "Boolean" question we ask is of the form. Does the data have some property? And, let us call that property  $p$ ; an example of such property would be to ask whether a network graph, which is the data is connected, which is the property.

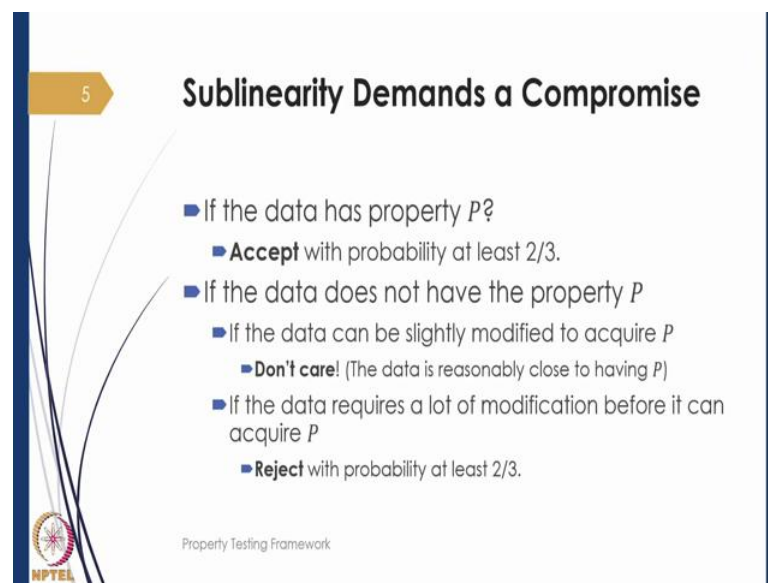
So if the graph is connected, then we are happy. And, if the graph is not connected, obviously we are becoming unhappy. We typically would want if our data represents a network graph, we typically would want that network graph to be connected. So and we would therefore be interested in knowing whether graph is connected or not. And that is the Boolean function that we are interested in.

And with, while trying to answer this Boolean question, we must minimize the number of queries. Too many queries would slow down the entire process because queries have to travel through a very slow network. And for this purpose, we will simply ignore the local computation. This is much faster, and only concerns ourselves with the number of queries. And in particular, in the property testing framework we are interested in very; in

trying to answer the Boolean question with very few queries. In particular, we want to answer the Boolean question using only queries that are sublinear in  $n$ . For example, a constant number of queries over the number of queries are being logarithmic in  $n$  or the number of queries being square root of  $n$  or some such sub linear quantity in  $n$ .

Of course, this would mean that we do not even have enough number of queries to read the entire data set. Clearly, the data is in the cloud because it is large and does not fit within the local computers. So, allowing the local computer to read the entire data set before answering the question would defeat the purpose. So, that is the crucial reason why we insist on the number of queries being sublinear in  $n$ . So, clearly you would be wondering how can we answer any meaningful question about the data set in time or the number of queries here, there is sublinear in  $n$ .

(Refer Slide Time: 07:02)



5

## Sublinearity Demands a Compromise

- If the data has property  $P$ ?
  - **Accept** with probability at least  $2/3$ .
- If the data does not have the property  $P$ 
  - If the data can be slightly modified to acquire  $P$ 
    - **Don't care!** (The data is reasonably close to having  $P$ )
  - If the data requires a lot of modification before it can acquire  $P$ 
    - **Reject** with probability at least  $2/3$ .

Property Testing Framework

NPTTEL

Clearly, this sub linearity requirement will demand a compromise. So, we will not be able to answer the question with 100 percent accuracy. We will have to answer the question in an approximate fashion. And, here is how we will model this part of the requirement. If the data has the property  $P$ , so for example, if the graph that we are concerned with is connected, for example, then we will accept the data set. And, here 'accept' means, we accept that the data has the property  $P$  with probability at least two-

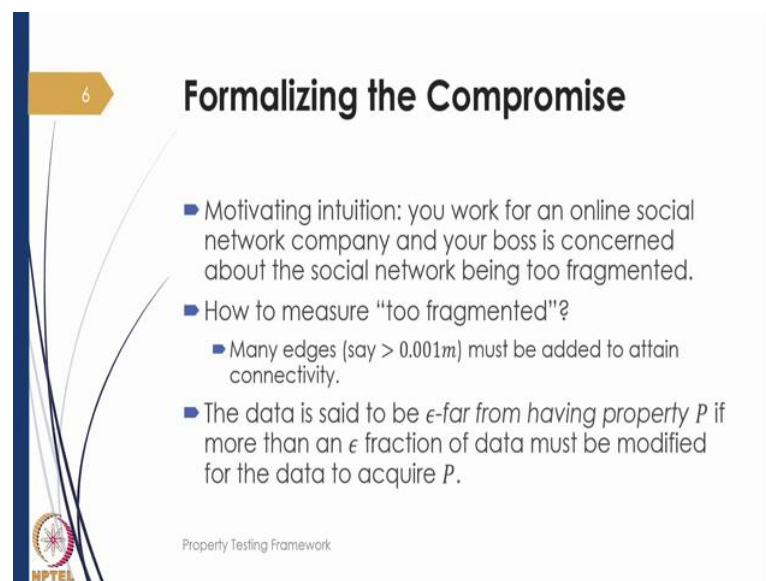
thirds.

On the other hand, if the data does not have the probability  $P$  we need to be more careful. If the data can be slightly modified to acquire the property  $P$ , then it is going to be computationally difficult to identify the fact that this data set does not have the property  $p$ ; because the data set is very close to having the property  $P$ .

And therefore, in order to be able to design good algorithms, we simply ignore these situations. And, they also make sense from practice because typically, we are in willing to live with the little bit of compromise on the accuracy. And, if the data set is close to having the property  $P$  we will be willing to compromise and say we do not care if the question is answer correctly or not.

However, if the data requires lot of modification before it can acquire the property  $P$ . So, then this implies that the data is far from having the property  $P$ . And in this context, we required that our algorithm should reject the data set. Meaning, our algorithm should be able to identify the data set does not confound to property  $P$ , does not satisfy property  $P$ . And therefore, we should be able to reject it with probability at least two-thirds. And, of course we will need to do this with as few queries as possible.

(Refer Slide Time: 09:35)



6

## Formalizing the Compromise

- Motivating intuition: you work for an online social network company and your boss is concerned about the social network being too fragmented.
- How to measure "too fragmented"?
  - Many edges (say  $> 0.001m$ ) must be added to attain connectivity.
- The data is said to be  $\epsilon$ -far from having property  $P$  if more than an  $\epsilon$  fraction of data must be modified for the data to acquire  $P$ .

Property Testing Framework

NPTEL

So, let us think a little bit more carefully about this compromise. And, let us try to formalize this. And towards, let us first start with some motivation. And, let us think of the context where you are working for an online social network company. And, your network, social network is stored in some cloud and your boss concerned about whether the social network is too fragmented or not. If it is perfectly connected, great; if there is some small fragmentation here and there, that is also fine.

But, obviously you and your boss will have to be concerned if the social network has a lot of fragments. And so, your boss asks you, you know, to try and see whether the data is too fragmented. And you ask, “okay, what would constitute too much fragmentation?” And, so then let us say you converse with your boss and you come to the conclusion that many edges must be added to attain connectivity. And if that is the case, then the graph is set to be too fragmented. If you can connect the graph by just adding a few edges, then you do not quite call it ‘too fragmented’.

So, let us formalize this a little bit more carefully. Let say, you and your boss come to the agreement that if there are  $m$  edges and the only way to connect the graph is by adding more than say small fraction;  $0.001$  times  $m$  number of edges. Then, then you would say that the graph is too fragmented. If on the other hand, you can get the graph to be connected with by just adding fewer than  $0.001$  times  $m$  number of edges, then you do not care. You are willing to leave with that amount of fragmentation. And such a motivation leads us to the definition of the notion of being epsilon-far from being, from having property  $P$ .

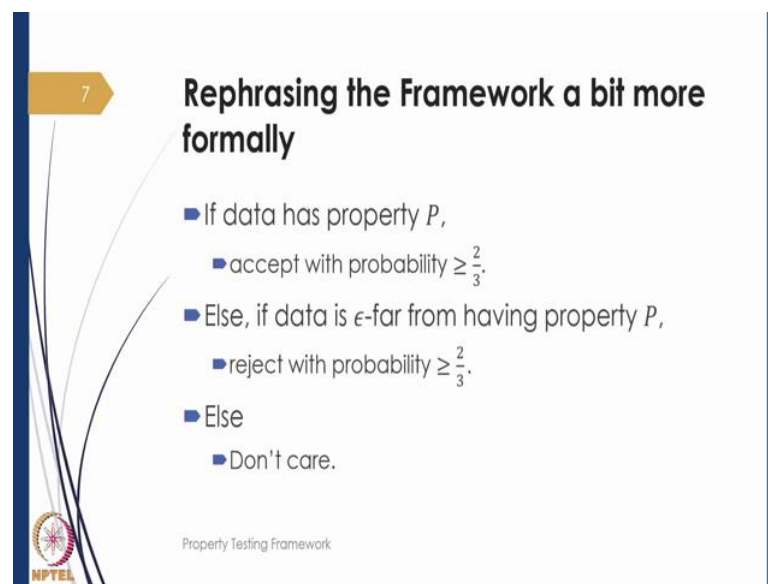
Let us formally define that a data set is set to be epsilon-far from having property  $P$ . If more than an epsilon, fraction of the data must be modified. And, this could include some additions, deletions, edits, so on and so forth modifications for the data to acquire the property  $P$ . Obviously, if the data already has  $P$ , then you do not require any modification at all.

If the data set is close to, you know, satisfying the property  $P$ , then which is the small number of modifications, you should be able to change the data set into having the property  $P$ . But if it is epsilon-far from having the property  $P$ , then you will need to make

a lot of changes; more than an epsilon fraction of the data must be modified. And, this is the notion that we use to define epsilon-far from being, from having property P. And, notice that just to be clear, this is purely a definition of what we mean by the data being far from having property P.

Algorithmically, we would not be concerned about making these modifications to check whether the data set is far from having property P or not. Typically, we want to, want our property testing algorithms to answer the queries without actually modifying the data set, just by querying and getting to know the data set. However, this notion of being epsilon-far from having property P will be exploited in analyzing our algorithm.

(Refer Slide Time: 14:00)



7

### Rephrasing the Framework a bit more formally

- If data has property  $P$ ,
  - accept with probability  $\geq \frac{2}{3}$ .
- Else, if data is  $\epsilon$ -far from having property  $P$ ,
  - reject with probability  $\geq \frac{2}{3}$ .
- Else
  - Don't care.

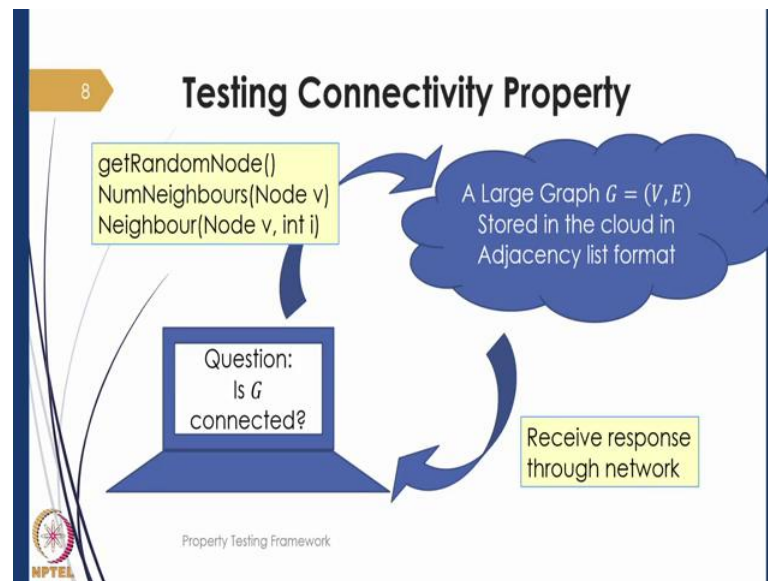
Property Testing Framework

NPTTEL

So, now that we know we have defined the notion of being epsilon-far from having a property P. Let us rephrase the framework, the property testing framework a little bit more formally. If the data has property P, then we would like our property testing algorithm to accept with probability at least two-thirds. Otherwise, if the data is epsilon-far from having the property P, then we would want our algorithm to reject with probability at least two-thirds. Otherwise, we do not care. And, of course all these have to be done with as few queries as possible.

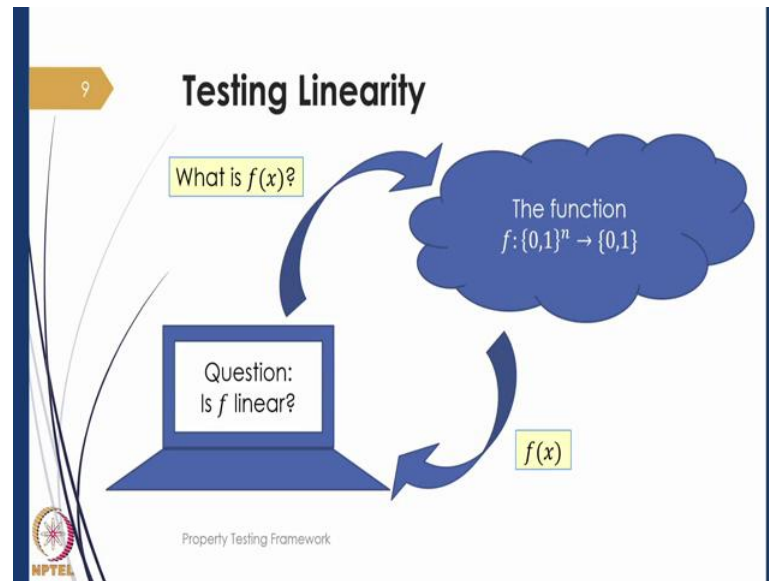


(Refer Slide Time: 14:44)



So, let us briefly look at the two examples that we will be considering in the next two segments of our lecture. The first property that we will be concerned with is testing whether a given graph is connected or not. So, the cloud has a large graph. Let us say it is stored in the form of an (Refer Time: 15:13) list. And locally, we have a question is the graph connected or not. And, we can send out our queries of the form, select a random node, what are the number of neighbors that node has, you know, what is the fifth neighbor of that node and so on and so forth. So, we allowed queries in that form and you will receive appropriate responses. And with as few queries, we need to answer the question whether the graph has a particular, I mean, has the connectivity property or not.

(Refer Slide Time: 15:49)



The second property testing example that we will consider is testing the linearity of a Boolean function  $f$ . The Boolean function  $f$  will be stored in a cloud. And, the local computer has access to the function via queries. And, the queries are of the form what is the value of  $f$  of  $x$ , for some given  $x$ . And using the responses, we will have to answer the question of whether  $f$  is linear. And, of course we will formally define linearity of Boolean functions in the third segment, when we discuss this topic. And that brings us to the end of the first segment. In the second and the third segments, we will look at these two examples in greater detail.

Thank you.