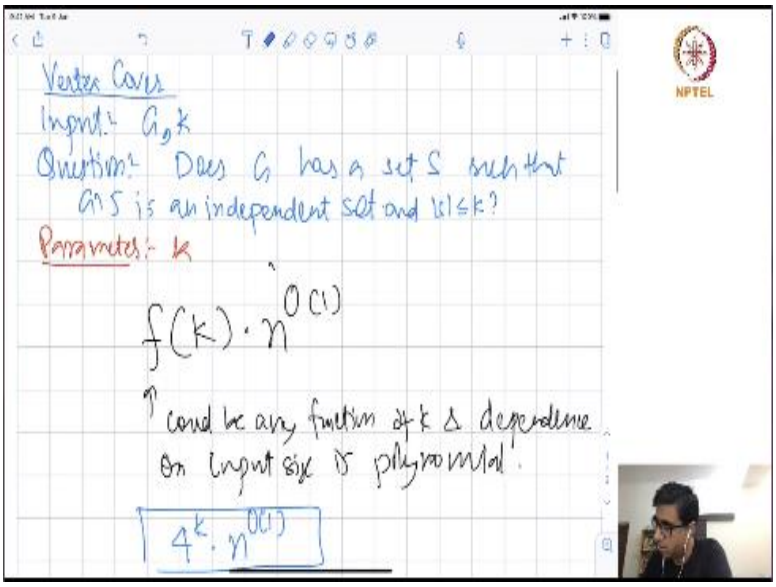
Parameterized Algorithms
Neeldhara Misra and Saket Saurabh
The Institute of Mathematical Sciences
Indian Institute of Technology – Ghandhinagar

Lecture - 02
Formalizing FPT

(Refer Slide Time: 00:16)



Welcome back. So, we will continue giving basic definitions of the course and before we move on to the techniques the course. So, in the last lecture, we saw the following problems. We saw the following problem of vertex cover. So, what are the problems of the vertex cover? So, problem of the vertex cover was; input was G, k, question was: does G has a set S such that G minus S is an independent set.

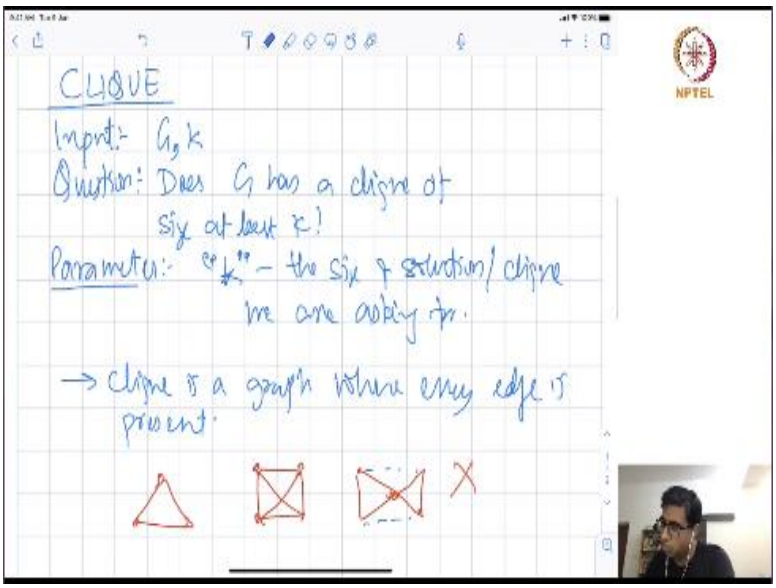
Set S independent set and mod S is less than or equal to k. So, basically we are looking for a set S such that for every edge, one of the endpoint is present in f and that is that f for also called a vertex cover and then we introduced this notion of parameter. And parameter was k and our objective was to design an algorithm with running time f of k times n to the power big O of 1.

And where f could be any function of k and dependence on input size is polynomials. And for vertex cover, we did make an algorithm with running time 4 power k n to the power of big O of 1. And then we asked that in this course, we will see a lot of techniques to design such an

algorithm we will define FPT today formally. But this is the kind of follow them, we would like to design.

And we said there are other problems for the same problem either could be different parameters with respect to some parameter, the problem could be hard meaning we cannot expect to have in such an algorithm f of k n to power of big O of 1 kind of algorithm. And there could be some other parameters for which such an algorithm do exist. So, let us look at a close related problem of a; we will look at the closed related problem of clique.

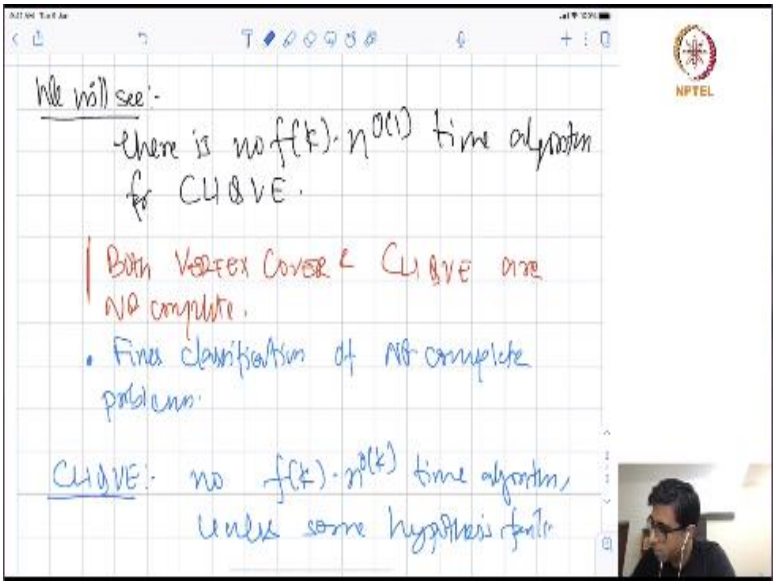
(Refer Slide Time: 03:17)



So, what is a clique problem? So, in the clique problem input edge G, k. Question is: does G has a clique of size at least k? And our parameter: k, the size of solution or the clique we are seeking for, they are asking. And what is a clique? Basically, clique is a graph where every edge is present. Example, triangle is a clique. Example, look at all the edges are present here, right, this is a clique.

But this is not a clique because, we could, there would be edges here which is absent; there is edges. So, basically, a graph is called a clique if look at any pair of vertex, there is an edge present there. So, this is vertex clique.

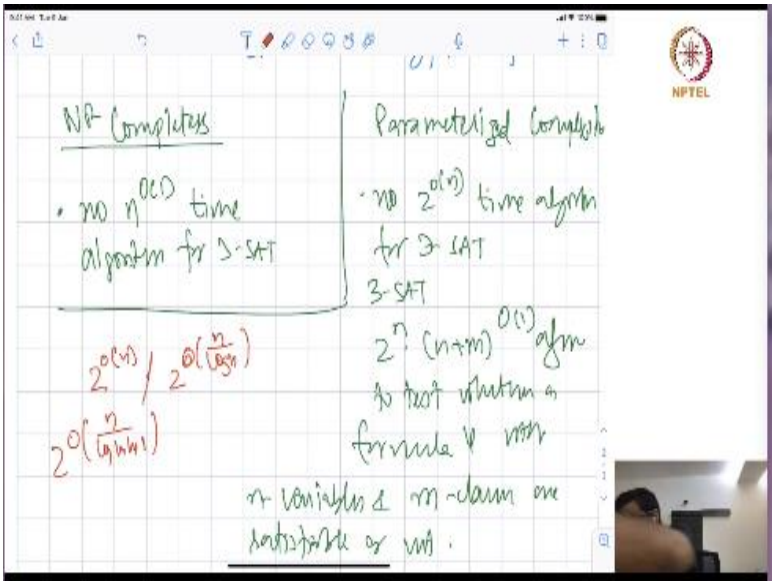
(Refer Slide Time: 05:11)



Now, as we will see in the course, that for this problem, we will see, there is no f of k n to the power big O of 1 time algorithm for clique. So, notice this field is quite interesting that while we suffer a vertex cover; there is an algorithm of running time f of k polyend. We will see that clique on the other hand does not have such an algorithm. But notice so, just last remark both vertex covered and clique are NP complete.

But, in the world of parameterize complexity or multivariate algorithm, there is so different behaviour. So, in some sense, the field also does finer classification of NP complete problems. In fact, what is, what can be show is even more. So, in fact for clique, we can show; for clique: no f of k n to the power little o of k algorithm time algorithm. In fact, even this is possible, unless some hypothesis fails. So, let us talk about what kind of hypothesis, are we talking about a little bit.

(Refer Slide Time: 07:16)



So, notice that the world of NP completeness in some sense. Let us say, NP completeness world is the basically on the following hypothesis. No n to the power. So, 1 time algorithm for 3 SAT. And we can build a complexity theory or we can build a theory. So, the whole NP completeness is based on that there is no polynomial time algorithm for satisfiability basically, 3 satisfiability where it is a; where every clause has at most 3 literals.

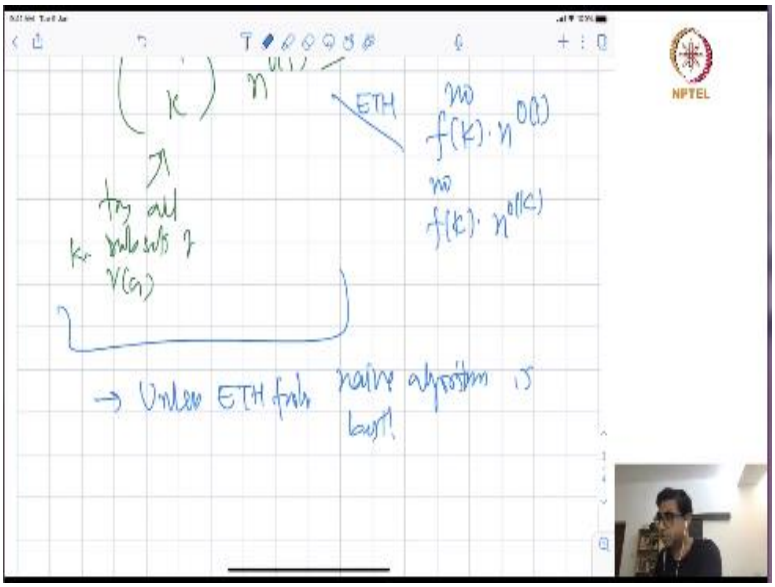
And so, on the other hand, we can show, clique cannot have such an algorithm by assuming the following hypothesis. So, this is what is called; so, in the one of the hypotheses in parameterized complexity is that no to the power little O of n time algorithm for 3 SAT. Notice that for 3 SAT for that matter any SAT, you have a 2 power n n plus m times big O of 1 algorithm to test whether a formula v with n variables and m clauses are satisfiable or not.

So, because you look at each variable, you try all Boolean assignments of 0,1 and check whether the formula is satisfiable. Now, the 2 power n of them and you can check this. But what this hypothesis tells us that look for 3 SAT, we do have such an algorithm. In fact, for 3 SAT, you have much faster algorithm than 2 power n in terms of base of exponent. But what it tells us that you cannot have, if you assume that you cannot have an algorithm with running time 2 to power little of n.

For example, you cannot have algorithm running time 2 to power n by log n or 2 to the power big O of n by log log log n. If you assume that, you cannot have 2 little of n algorithm or in particular you cannot have such an algorithm like 2 power big O of n over log n or 2 over big

O of n log log n then we will be able to show. In fact that clique, you cannot even have any algorithm better than f of k n to the power little of k.

(Refer Slide Time: 10:30)

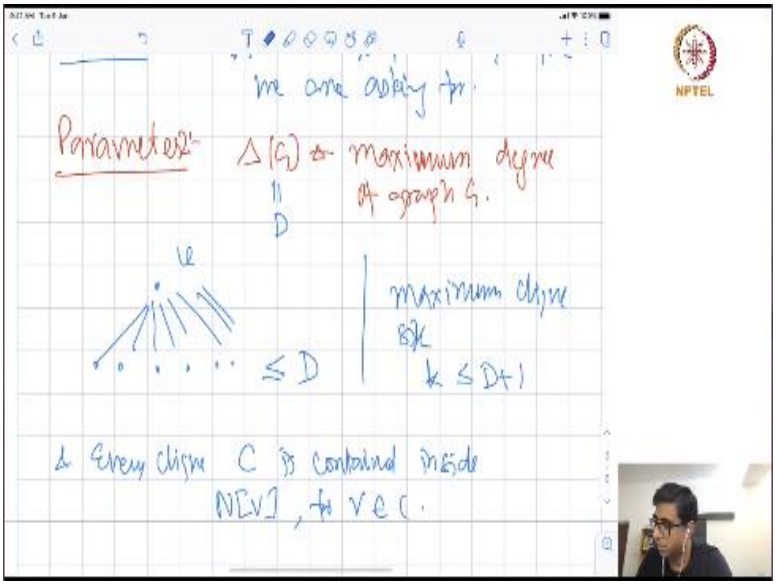


So, imagine, what are the best what can be one good algorithm for the clique? One good algorithm for the clique or the trivial algorithm for the clique is try all subsets of; try all k subsets of V of G that is it. And n to the power big O form, you check whether that forms a clique or not by looking at every pair of edges there is an edge or not. So, similar to vertex cover we had this in an algorithm, but for vertex cover, we did make 4 power k n to the power 1 algorithm. This was for the vertex cover.

And if assuming, what is this? This hypothesis is also called exponential time hypothesis. We will talk about it much more in the course exponential time hypothesis. In fact, what we are saying that look not only, you cannot have f of k n to the power big O of no such algorithm. In fact, you cannot even have n, no f of k into the power little o of k. So, hoping to do better than this is by is not possible.

So, what we can say is unless ETH fails, naive algorithm is best. So, this is what so, assuming this hypothesis, we can start showing. So, if you assume like much stronger running time for satisfiability problem, you can so much quantitative lower bounds or much more qualitatively better lower bounds on problems which are NP hard. But now, I will change my track and look at the clique problem.

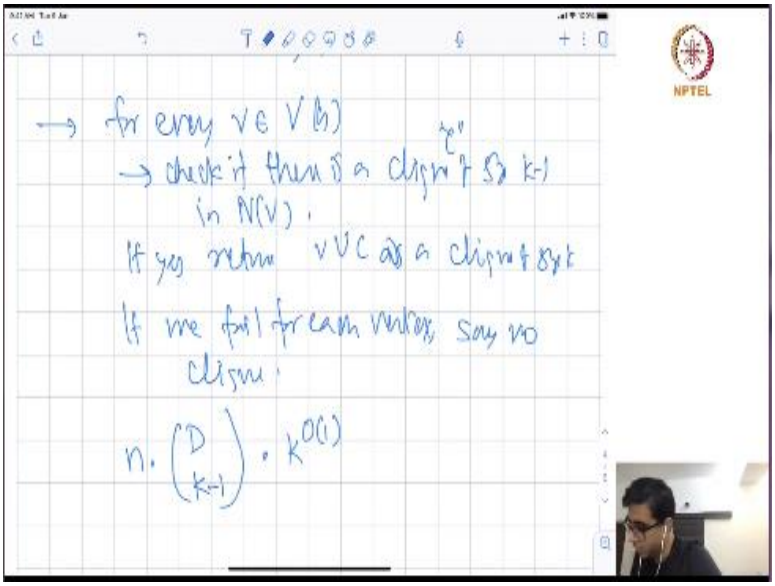
(Refer Slide Time: 12:49)



Now, rather than assuming this parameter, let us use another parameter. So, what is the parameter? Another parameter is delta of G. This is; what is this? Maximum, so, this let us call this parameter 2, parameter 1, maximum degree of graph G. Now, let us see, what happens this then? Now, notice that with respect to this look at any vertex v, what is its degree is, let us call this D; is at most D.

So, if you are looking for a case is clique, what can be the maximum clique size? Maximum clique size is notice; a clique must be in a neighbourhood of a particular vertex. Because, we must have neighbours to all those vertices which participates into this, which implies k is less than equal to D plus 1. In fact, and every clique C is contained inside an N of v close neighbourhood for v in C.

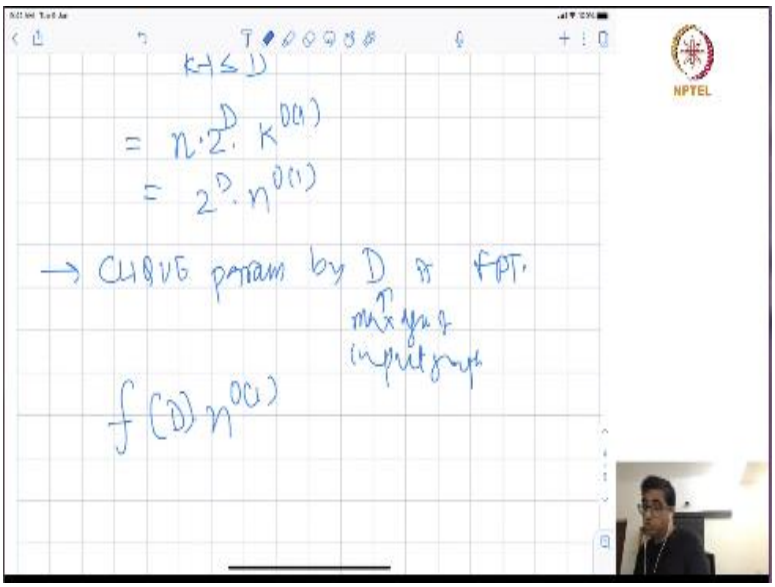
(Refer Slide Time: 14:48)



So, now, our algorithm is very simple. You for every v in V G; check if there is a clique of size k minus 1 in open neighbourhood N of v, clique C of size k minus 1. If yes, return v union C as a clique of size k. So, if we fail for each vertex say no clique then this is easy. So, what is the running time of this algorithm? The running time of this algorithm is, you go over each vertex.

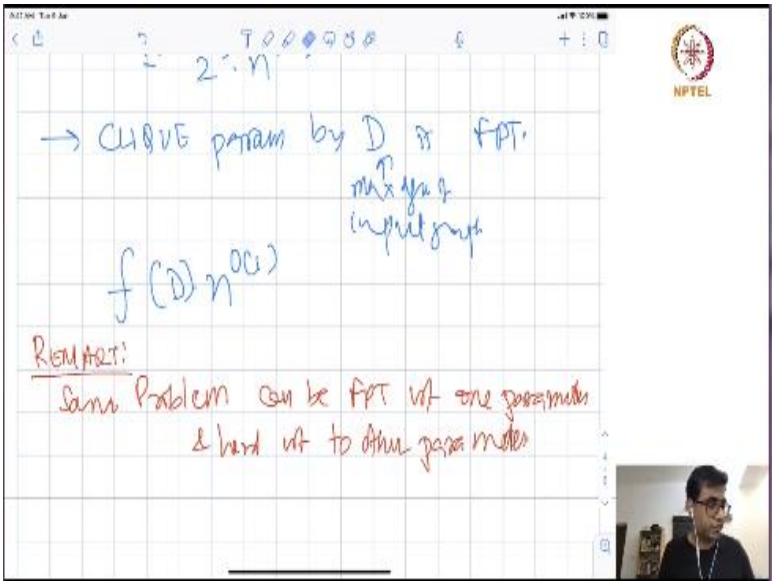
For each vertex, you try D choose k – 1, whether they clique of size at most k – 1 is clique of size k – 1. And then you check whether this forms a clique or not. So, that is like actually k to power big O of 1, because the k vertex and you have to check with it.

(Refer Slide Time: 16:13)



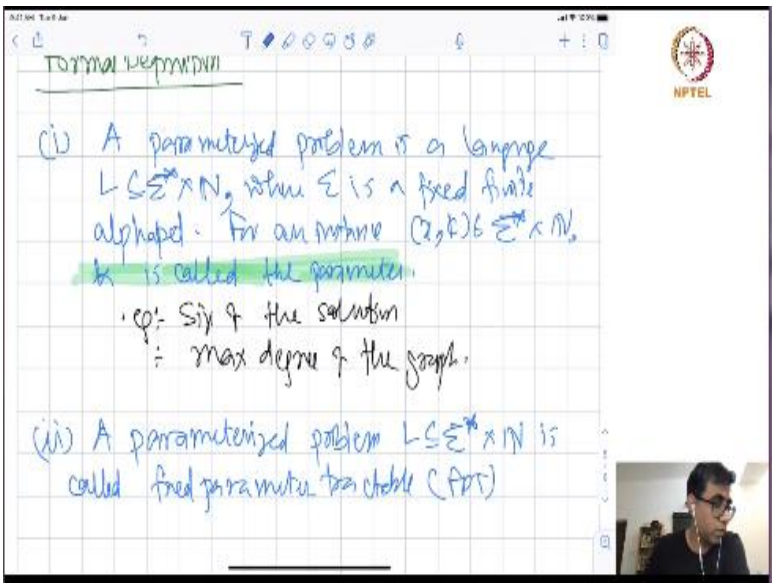
So, notice this algorithm, but what we know, k minus 1 is less than equal to D. So, this algorithm is actually n to the power 2 power D times k to the power big O of 1 which is 2 to the power D n to the power big O of 1. So, now notice that for parameterized by D. Clique parameterized by D max degree of input graph is fixed parameter tractable. Why? Because we made an algorithm which running, time this.

(Refer Slide Time: 17:08)



So, notice that for a problem there could be several parameters; with respect to one parameter, the problem could be intractable. With respect to other parameter, the problem could be track. So, remark: Same problem can be FPT with respect to one parameter and hard with respect to other parameter. I think now, it is time for me to have given you some notion. To define some formal definition that we will use in the course, just to make sure that we are on the same page.

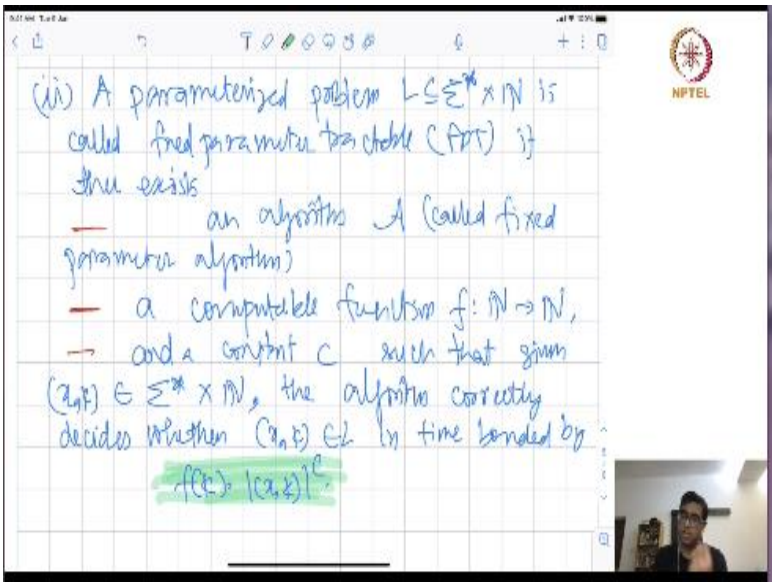
(Refer Slide Time: 18:14)



So, the formal definitions: Let me give you first is that I defined you a parametrized problem is a language L subset of sigma cross natural number, where sigma is a fixed alphabet, fixed finite alphabet. Let us not say just alphabet, with a fixed finite alphabet. For an instance x, k in sigma cross natural numbers, k is called the parameter.

So, for example, size of the solution; maximum degree of the graph. So, this is my first definition that I would like to tell you. Second another important definition is about FPT fixed parameter tractability which we have using up until now, you know not so like in an informal way, but let us define it formally a parameterize problem L cross n. It is called fixed parameter tractable or FPT.

(Refer Slide Time: 21:08)

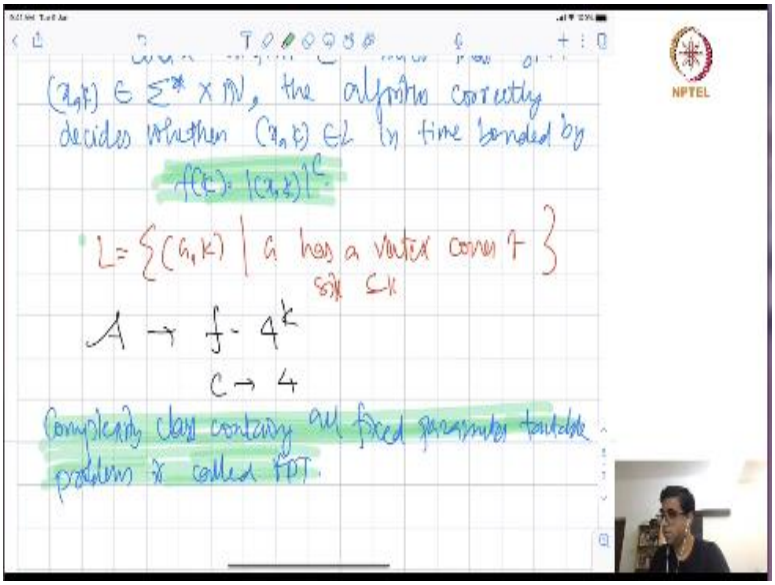


If there exists an algorithm currently A called fixed parameter algorithm. A computable function, this is important for us, I will tell you in a minute. A computable function f from natural number to a natural number and a constant C such that given x, k in L or not L, but given x, k in sigma cross natural numbers. The algorithm correctly decides whether x, k belongs to k whether in time bonded by f of k size of the instance time C that is it.

So, what is an FPT algorithm? So, given a language L if you have an algorithm A and a computable function N and from natural number to national. And a constant C such that if you get an, if you give this instance x, k to algorithm A, curly A, then it can decide whether x, k belongs to L or not in time f of k, x, k to the power C. For example, for vertex cover, we made such an algorithm; we had a computable function which is nothing but 4 power k.

So, given an integer k, in 4 power k type and our constant was something like 3. So, in 4 power k times some the size of the input to the power 3. We were able to decide whether a given instance has a vertex cover of size at most k or not and the language there will be.

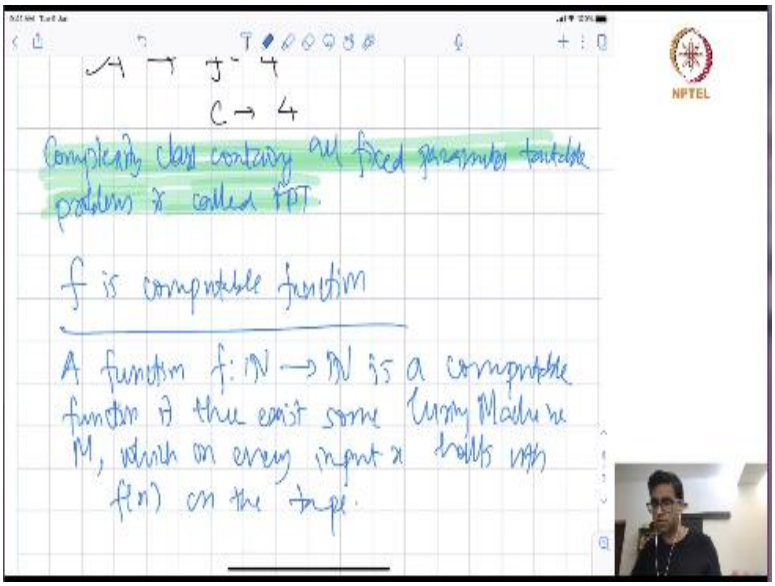
(Refer Slide Time: 24:38)



So, for example for what is the language for vertex cover? Let us understand. So, the language for vertex cover is G, k. G has a vertex cover of size at most k. So, now, our algorithm we made an algorithm where f was 4 power k and our constant was roughly like maybe 4 **(()) (25:15)**. So, in 4 power k times some n to the power 4, we were able to decide whether given a graph G contains a vertex cover of size at most k or not.

So, this just implies even with respect to this definition, the problem is FPT. So, another thing complexity clause, this is the first time containing all fixed parameter algorithm, fixed parameter tractable problems is called FPT. So, this is another thing which you like to remember that the clause FPT is basically all load problems for which there exists an FPT algorithm that is it.

(Refer Slide Time: 26:20)

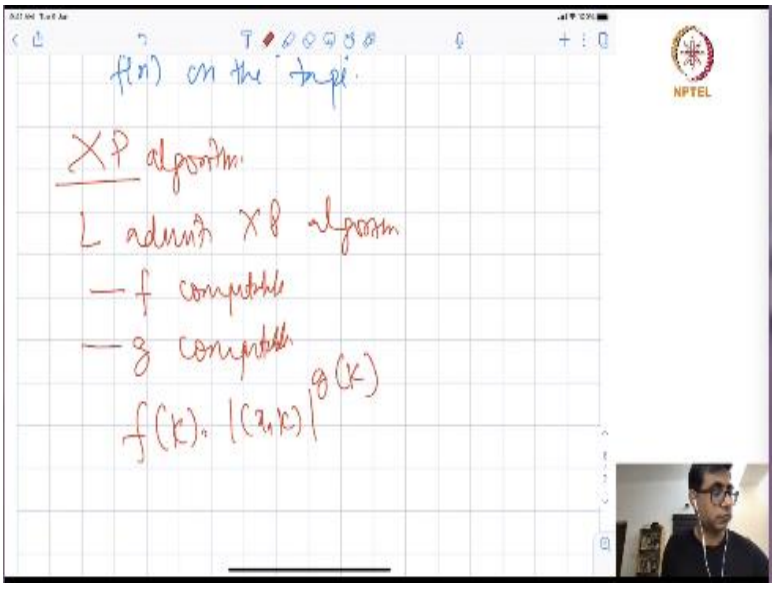


Now, one notion which we have defined, we have said that the function f should be computable. So, what is the meaning of that? So, basically, what is the meaning that f is computable function. So, basically if we do not assume that f is a computable function, then we get into some sort of trouble when we are trying to design reductions and hardness. So, it is better to assume and it will not take away anything from us.

So, basically, what is computable function? A function f from natural number to natural number is a computable function, if that exist some turing machine M. If there is some M which given an input x which on every input x halts with f of x on the tape. So, basically there is an algorithm which can compute this function that like that is all that it means.

So, even if you do not understand what computable function is, ignore this aspect from this whole course, it was just to make things formal. So, if you do not understand it is perfectly fine just that every function or every algorithm that we will design in this course will be computable. So, let us not worry about this if you do not understand. And if you understand, this will be useful tool to have.

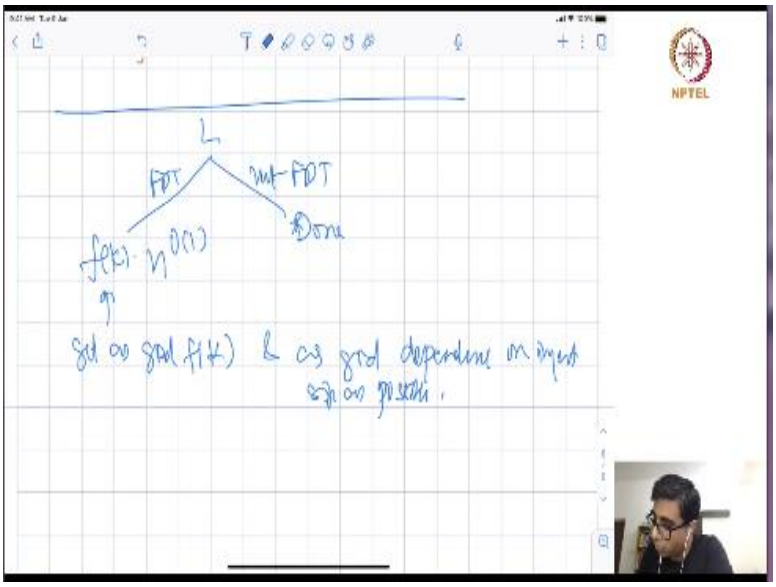
(Refer Slide Time: 28:22)



So, another clause; so, that is all the basic definition that I would like to define. Another clause which has been useful is what is called XP algorithm. So, basically XP algorithm is like, I will say that L admits X P algorithm, if, first of all you should have f computable; another function g computable. And the running time is like f of k n to the power g of k time that is it. n to the power or rather I think that we should say this, f of k x power k, x, k the input size, g of k.

So, n choose k algorithms are XP algorithm because n to the power k, n is the input size and this so, basically, so, this is also clause of algorithms that. So, in general for most of the problems, we will have a simple XP algorithm and the hard part will be to design whether they admit FPT algorithm or not.

(Refer Slide Time: 29:44)



So, given now that we have defined everything formally, so, given a problem L, so, first thing that we will show that whether; so, what will be so? So, the flow of research is very simple given an L, FPT or not FPT done. FPT, it means f of k n to the power big O 1 for algorithm. Get as good f of k and as good dependence on input size as possible. And so, this is what our basic goal is and then there are other kinds of things we do.

If we have got some FPT algorithm with running time f of k, is this f of k and optimal? If not, can be shown that we can improve it or under some complexity assumptions can be shown that this is the best we can hope for. Similarly, in the other side when you show that the problem is not FPT can be shown, find some other interesting parameters with respect to which I can show the problem is FPT and things of the kind of questions that we will be dealing with in this course.

But basically, we will be providing the tools at this point of time to show whether a problem belongs to a FPT or it does not belong to a FPT and with that, I will end my today's lecture and we will move to the first technique of a doing designing FPT or not; in designing FPT in the speak which is called kernelization. Thank you.