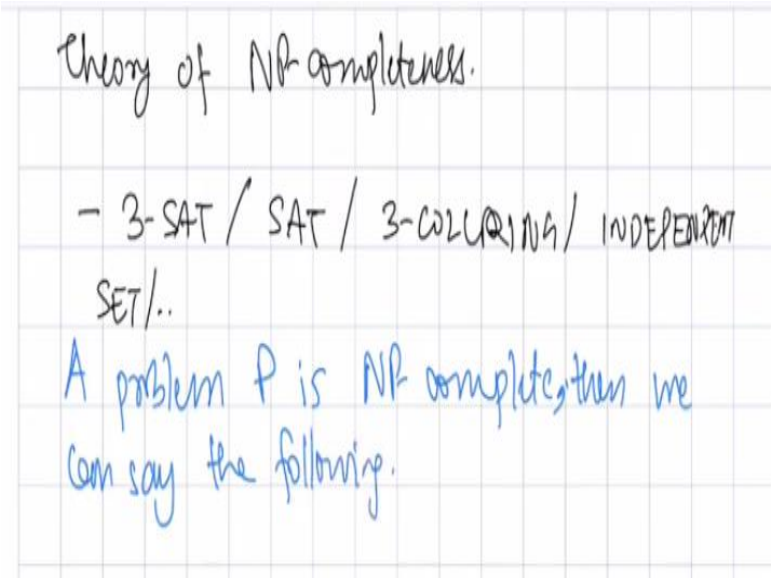
Parameterized Algorithms
Neeldhara Misra and Saket Saurabh
The Institute of Mathematical Sciences
Indian Institute of Technology – Gandhinagar

Lecture - 01
Invitation to FPT

Welcome all of you to the course. And we will start with this course with a small remark that the starting point of our course is going to be theory of NP completeness.

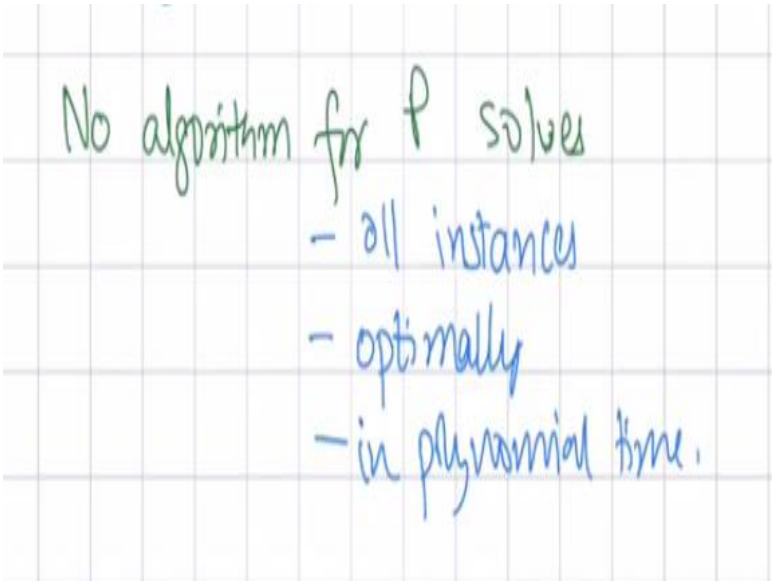
(Refer Slide Time: 00:31)



Some basic examples of problems that are NP complete are like I am sure you must have seen 3 SAT, satisfiability, 3 coloring, independent set, and many other problems. So, for this course, it is good enough for our purpose that we take the following as a definition of NP completeness. It is not formal, but this is something which will be helpful for us as we go forward.

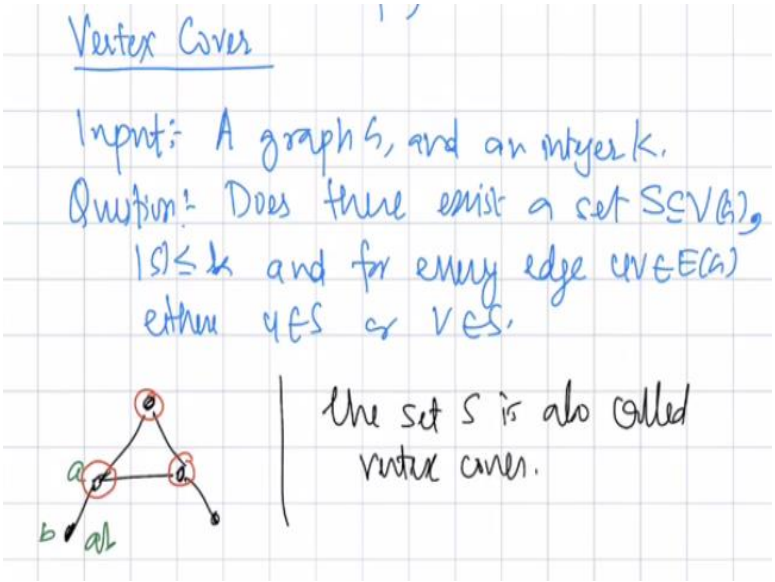
So, basically means a problem P is NP complete, then we can say the following. What we will say? We will say the following.

(Refer Slide Time: 02:03)



No algorithm for P solves all instances optimally in polynomial time. Just to make things concrete, let us fix one NP hard problem. And let us try to see different aspects of that NP hard problem in this lecture. So, the problem which we will consider and which will be fundamental and our most basic problem that as we will go along in the course, is the problem of vertex cover. So, what is the vertex cover problem?

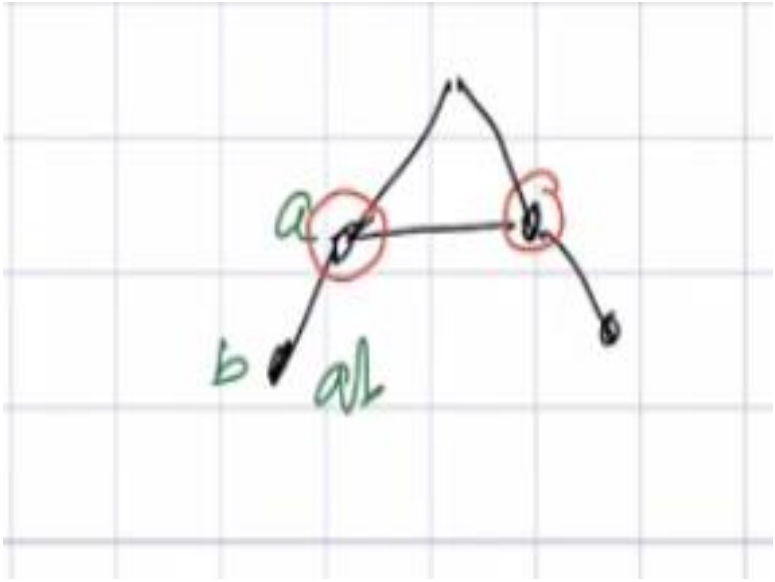
(Refer Slide Time: 03:06)



So, the vertex cover problem is as follows. Input is going to be a graph G and an integer k. And the question is does there exist a set S subset of V G mod S is at **(()) (03:46)** at most k. And for every edge UV in edge set of my graph either U is in S or V is in S. So, for example, let us say let us look at the following example. So, in this example, if I look at this red vertices, do they form a vertex cover? No. Why?

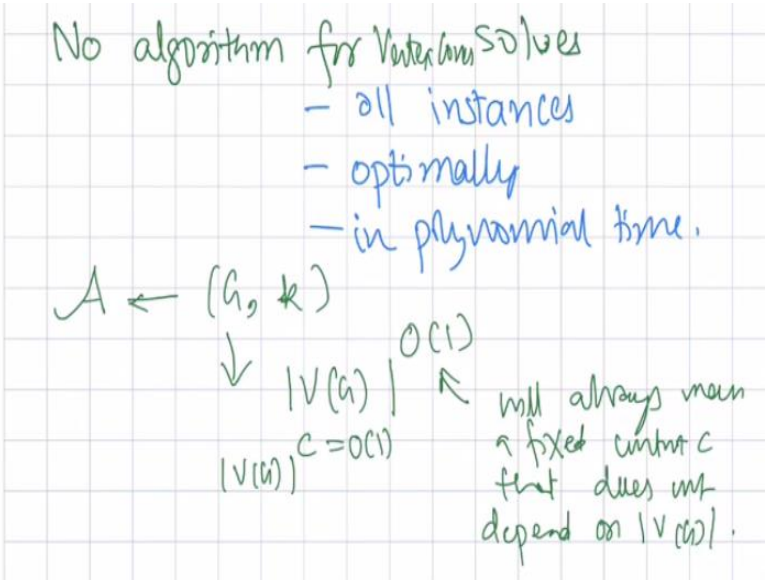
Because there is an edge here, here is an edge say a and b, edge ab, that none of these a and b is one of those red blocks. But if we include a also into our vertex cover, then this is a vertex cover. Because look at any edge, it has at least one endpoint which is red marked. So, colloquially the set S is also called vertex cover. So, for example, for this simple graph that we drew, there is a simple graph that we draw.

(Refer Slide Time: 05:23)



The simple graph we draw. Actually we can find us this is following is also a vertex cover. Because now also if you look at any edge there is at least one endpoint which is red colored or in the red circle. So, this is a very fundamental problem. And if I try to take this basic definition what can we say about vertex cover. So, basically we say we can say we can take back the statement about NP completeness copy and we can say the following.

(Refer Slide Time: 06:18)

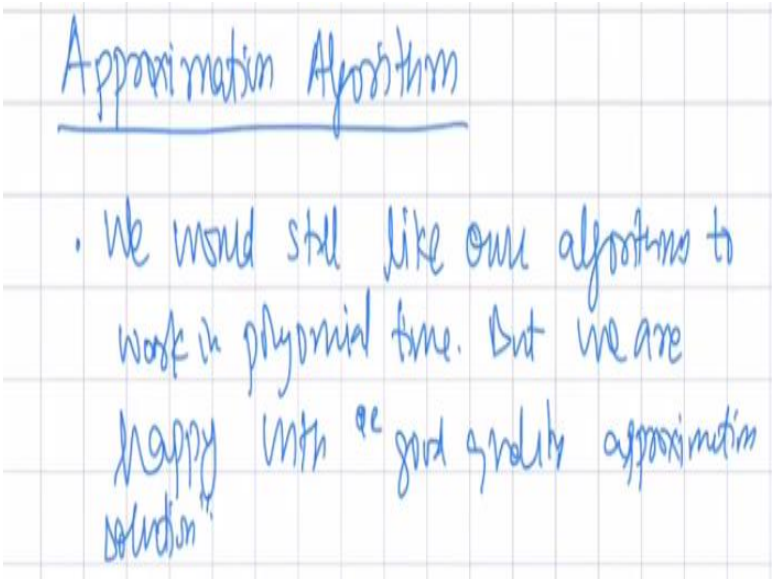


So, there is no algorithm for vertex cover that solves all instances optimally in polynomial time meaning what is a polynomial time for this instance meaning for this there is no algorithm A which takes an instance G, k runs in time some V G to the power big O of 1. So, let us put that remark, what is big O of 1? Big O of 1 will always mean fixed constant C that does not depend on vertex of G.

So, for example, what it means there is no algorithm which takes an instance G, k and in time say V G to the power some C or big O of 1 can tell you whether G has a vertex cover of size at most k or not. So, we do not expect any such algorithm for an NP hard problem and in particular for vertex cover. So, what do people do actually? So, to handle these things people actually has to get away from one of the demands which we have that either we can solve the problem all instances.

Or, we can solve the problem in polynomial time or we can solve the problem optimally. So, one classical field is approximation algorithm.

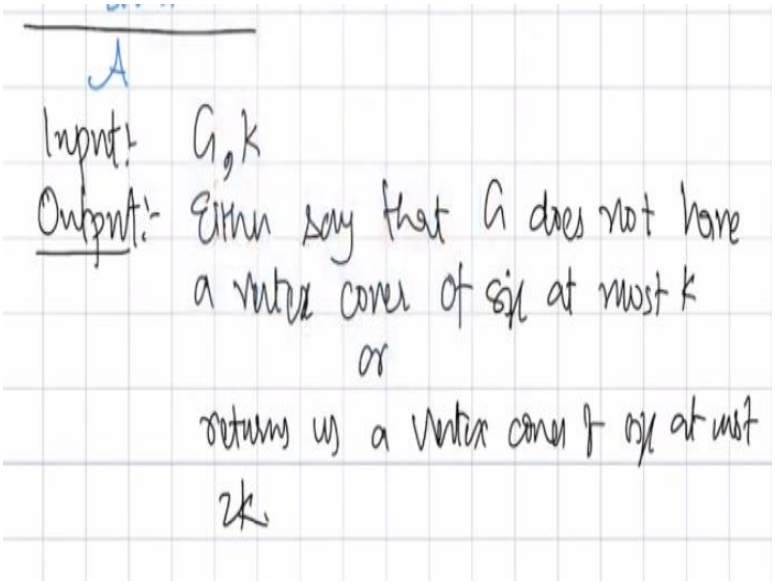
(Refer Slide Time: 08:16)



So, let me talk about a bit approximation algorithm. So, in this field, we would still like our algorithms to work in polynomial time. But we are happy with good quality approximation solution. For example, I said look, it is, I am not able to give you a solution of size at most k. But maybe if there is a solution of size at most k, could you produce me a solution of size at most 2k, 3k or maybe k square or anything.

So, let us look at one very good example of a vertex cover problem and approximation algorithm for vertex cover that runs in polynomial time.

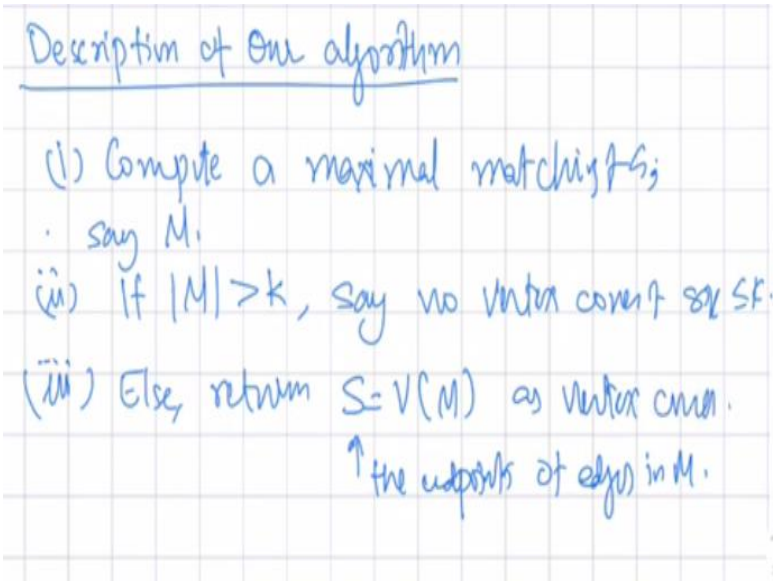
(Refer Slide Time: 09:37)



So, here is our algorithm A. What does what are its steps? So, this is on it takes as input G, k. And what it is going to output us? Either it is going to output us the following. Either say that G does not have a vertex cover of size at most k. This is one thing that our algorithm can return or returns us a vertex cover of size at most 2k. So, this is our algorithm. So, what it is supposed to do. It takes an input G, k.

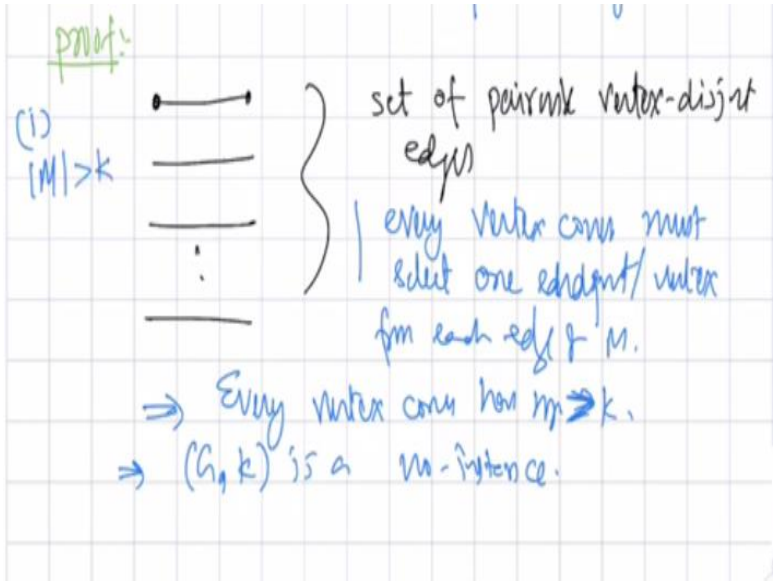
Either, say that G does not have a vertex cover of size at most k or returns a vertex cover of size at most 2k. And here is a very simple algorithm. I am sure many of you would have seen this. So, here is your description of our algorithm.

(Refer Slide Time: 11:03)



So, what it does? Compute a maximal matching of G, say M. Now, sorry if cardinality of matching is more than k, say no vertex cover of size at most k. Third else return S equal to vertex set of M. The endpoints of edges in M. So, just so, what is S is basically you look at every edge of my matching take both endpoints. Else return S as vertex cover. So, first thing first. So, now, let us prove that this is indeed correct.

(Refer Slide Time: 12:47)

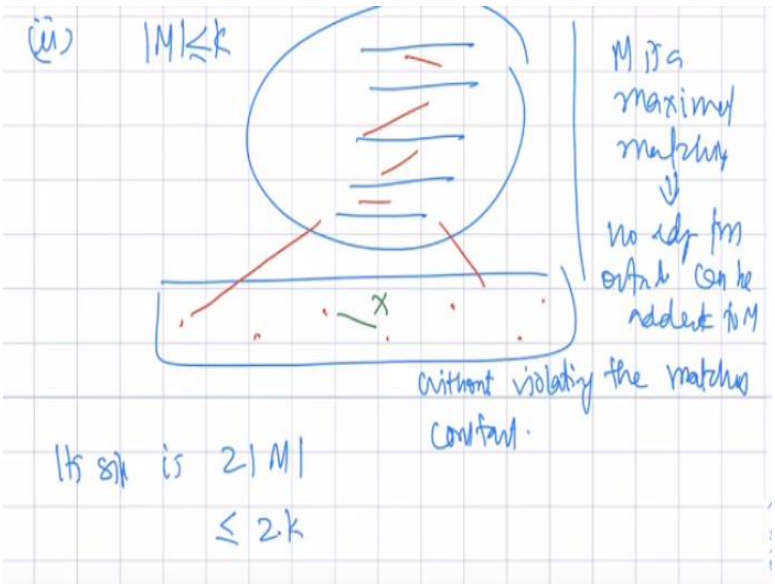


So, here is our proof. So, first case, what happens if we get matching of size greater than equal to k. So, what is matching? Matching is set of pairwise vertex disjoint edges. So, which means, if a vertex appears in the first edge it does not appear in any other edge and so on and so forth. Now, notice what is a vertex cover? Vertex cover is set of vertices that must pick one endpoint from every edge.

And in particular, it must pick one endpoint. Every vertex cover must select one endpoint slash vertex from each edge of M. Now, since these vertices are these, the edges which are the vertices which are involved in the first edge is not part of the second edge and so on and so forth. Any vertex cover must pick any vertex intersection of any vertex cover with any matching edge is at least one and these are disjoint.

So, if you the existence of a matching of size strictly greater than k implies the vertex cover that we would wish to construct. This implies that every vertex cover has size greater than strictly greater than k. This implies that when we say that G, k is a no instance. Meaning it does not have a vertex cover of size at most k. Then this is a correct assertion. Now, so, now look at the second case. So, this is a case when a matching is more than k.

(Refer Slide Time: 15:09)



Now look at second case. Now, what is M? M is a maximal matching. What is the meaning of that? Look at any vertices which are involved here which are not part of this **(()) (15:29).** What is the meaning of a maximal matching? What is the meaning that M is a maximal matching? M is a maximal matching implies no edges from outside can be added to M without violating the matching constraint.

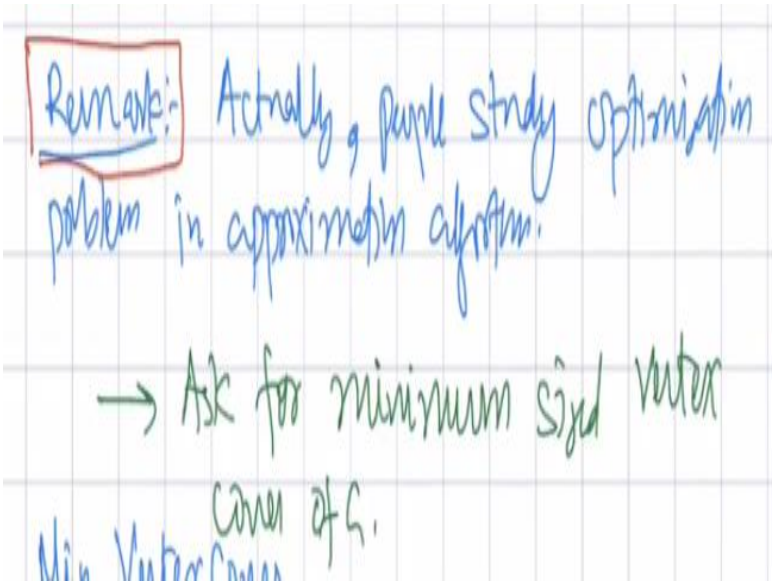
Meaning, if you look at any edge which is not part of this, why we are not able to add it to M, because when we add it, the problem is this edge intersects with one of the earlier selected matching edges. So, it means every other edge after this is going to be either like this or right. But we cannot have an edge entirely consisting of this red vertices because then we could add this edge to M and get a bigger matching.

So, which implies that indeed if I take the end points of every matching edge, then it does form a vertex cover. And what is its size? Its size is 2 times cardinality of M which is 2 times k. So, notice that we have designed an algorithm. And look the running time of this algorithm is polynomial because computing a maximal matching of say M is very simple greedy algorithm.

You start with an edge, then you pick up another edge, which is then you pick up another edge which is not part of this edge, put it so on and so forth. You keep doing this until you cannot add any. So, it is a greedy way of computing a set of edges which is maximal and their pairwise vertex **(()) (17:58)**. And then what are the second step we just computed it size what is which can be done in poly time? If it is large, you say no.

If it is small, then you take both the endpoints and you return it as a vertex cover. And what kind of algorithm we wanted? We wanted to have an algorithm which you wanted to say that either I returned that G does not have a vertex cover of size at most k. And in that case, G should not have a vertex cover of size at most k or we should return a vertex cover of size at most 2k. So, this is a simple algorithm, which does. Let me make it a remark.

(Refer Slide Time: 18:37)



One remark is in order here. So, what is my remark? So, the remark is that actually people study optimization problem in approximation algorithms. So, what I mean by this? So, let us put this remark in red. So, you actually ask for minimum sized vertex cover of G not there is

no k or anything. So, the question will be input **(()) (19:48)** G output. So, let us call this minimum vertex cover problem.

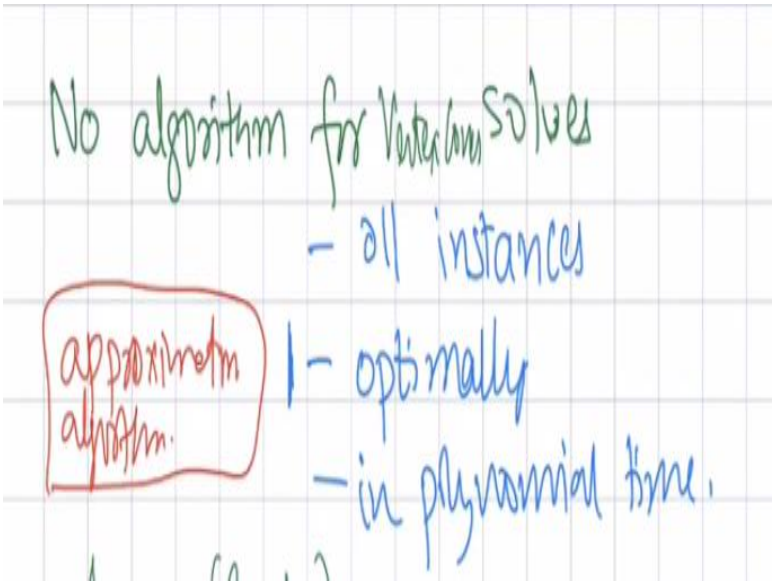
Output minimum sized S such that S is a vertex cover. A minimum size S **(()) (20:18)** or in other words if you look at the graph G **–** S is an independent set. There are no edges after we delete the vertices in S. So, this is what it is studied in classical approximation algorithm that you will study in optimization problem that given a problem you would like to find a set satisfying the property of minimum size or maximum size.

Or, if there are weights on vertices or edges that you would like to find a set satisfying this property of minimum weight maximum weight depending on the problem. So, whatever algorithm I stated before you if you forget about the second step and say compute a maximal matching return both of its endpoints then that will constitute a valid factor to approximation for minimum vertex cover.

But for this particular course, we will only talk about approximation with respect to **(()) (21:32)** version of the problem and approximation with respect to **(()) (21:35)** version of the problem for vertex cover is as follows as stated. So, you compute a maximal matching if the size is large, you return no. If size is small, you take both end points and return that as a particular vertex cover. So, let us go back to our theory that a problem P is NP complete.

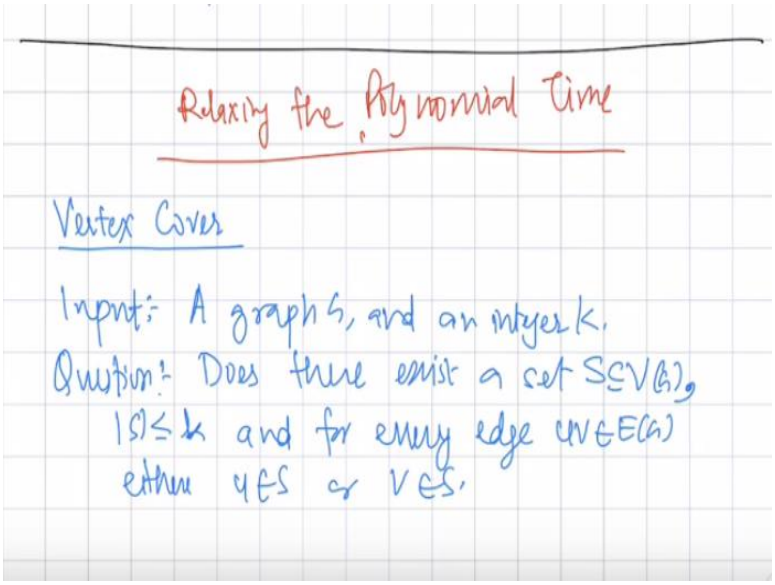
Then we say that for vertex cover we can say, hey, no algorithm for vertex cover solves all instances optimally in polynomial time. So, we went to the world of approximation algorithm.

(Refer Slide Time: 22:14)



When we relaxed the notion of optimality then this leads to the world of approximation algorithm. So, we do not like to but now, what about relaxing the constraints of polynomial time. And this is what will be one of the main theme. This is the main theme of this course that we are going to rather go into allow relaxation in optimality we are going to relax optimality like we are going to relax the condition that I demand you to have an algorithm in polynomial time. So, now, let us go back to the vertex cover.

(Refer Slide Time: 23:03)

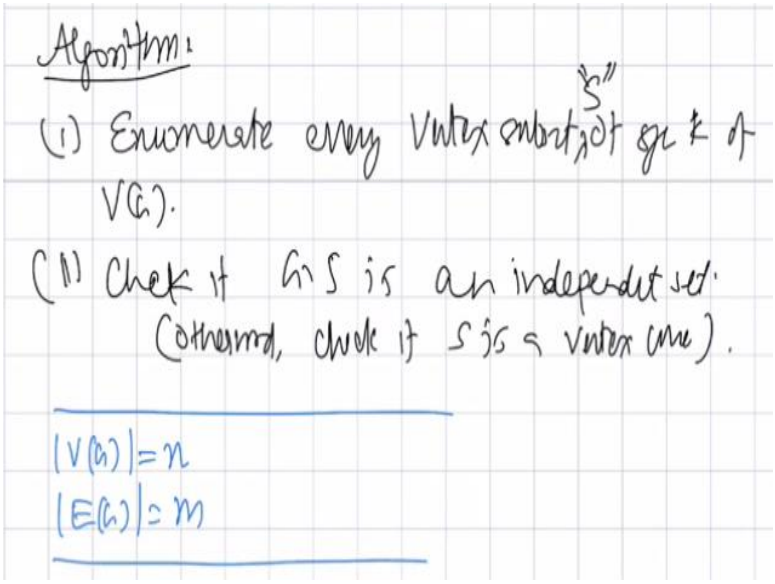


So, let us make a line. And now we are going to look at the notion of relaxing the polynomial time. So, we are going to relax the polynomial time. So, let us ask ourselves what would be an algorithm for a vertex cover? So, let us recall the problem vertex cover problem. If so, let us we can recall it. So, input is the graph G and integer k. And we would like to know

whether there exists a set S of size at most case such that if I delete it, every edge, if I delete this vertex set, then the remaining graph is an independent set.

Or in other words, we are looking for a set S such that every edge has at least one endpoint in this set. So, now look at this vertex cover problem and **(()) (24:25)** look, what would be an simple algorithm or a brute force algorithm to check whether graph contains a vertex cover of size at most k or not. So, here is one algorithm that I could think of is.

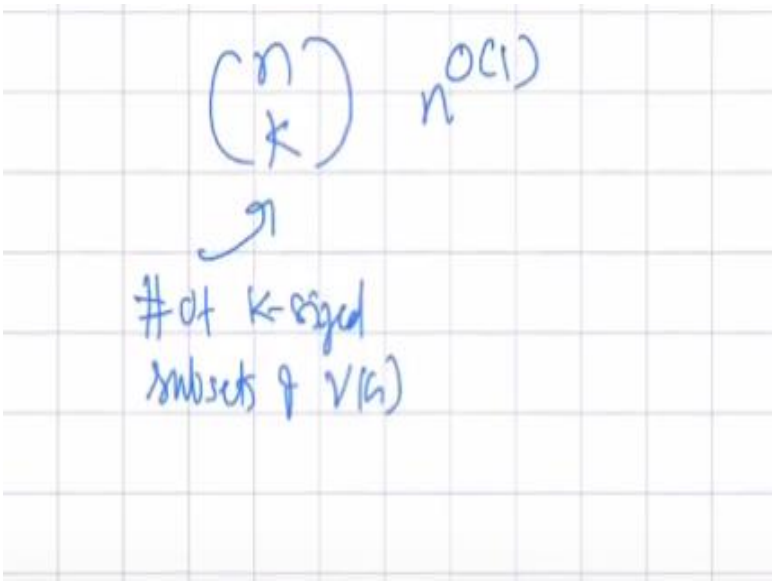
(Refer Slide Time: 24:47)



So, here is an algorithm enumerate every vertex subset of size k of V G. Second, enumerate every vertex subset S of V G of vertex subset. Enumerate every vertex S of size k of V G. Second, check if G – S is an independent set or in other word check if S is a vertex cover. So, here is a very nice algorithm. So, what is the running time of this algorithm? So, since we are going to suppose like so, from now onwards let us take the following notations.

Vertex set of G, size of the vertex set of G will be always n. Size of edge set of G will always be m. So, the graph we will consider will have n vertices and m edges. So, now, how many vertex subsets of size at most k of V G is?

(Refer Slide Time: 26:20)



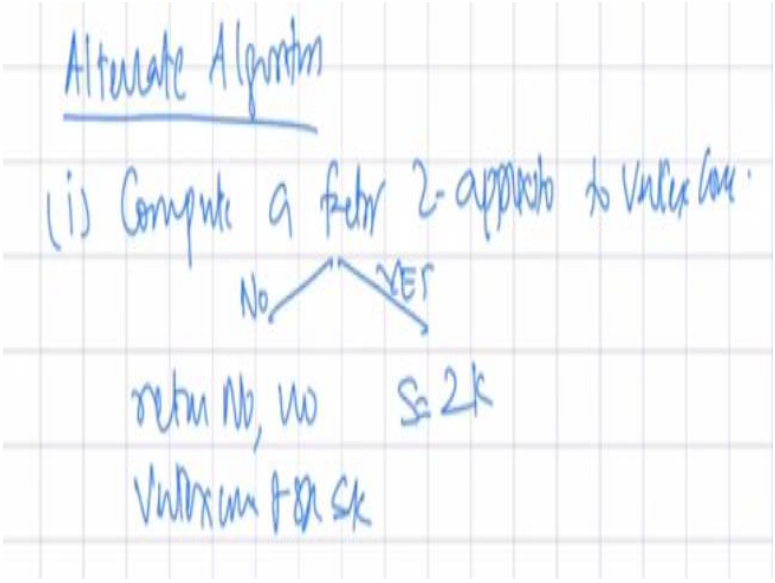
Well, that is n choose k, number of k sized subsets of vertex sets of G. And now, for each, I need to check whether G − S is an independent set. So, basically I go over each edge and check whether S contains one of these vertices **(()) (26:45).** And this we can do it in polynomial time. So, **(()) (26:48).** Recall n to the power big O of 1 is a constant which is like 2 3. In our case, at most 2 or 3 maybe, and that is it. It does not depends on the key.

So, this is one algorithm. So, which we can think of this as like of kind of brute force, because this is not a an algorithm which is doing anything more than the most obvious thing is to try all possible answers. And check if one of these answers is what we need or so for our possible answers for a particular set of size at most k or k. And so, you enumerated all these sets of size k.

And then you checked whether that set forms an independence that forms a vertex cover or not. By checking whether the complement **(()) (27:42)** S is an independent set or not. So, that should be. So, that. So, now, let me try to give you a another algorithm, which says, let us see, which might be slightly better. So, my algorithm is going to start with, so it is inspired by this algorithm.

But rather than enumerating all possible case size subsets of the whole graph or the whole vertex set, we would like to do it slightly selectively. So, what is my algorithm? So, I am going to first run this factor 2 algorithm, factor 2 approximation algorithm A. And so, an alternate algorithm I am trying to make for a vertex cover is the following.

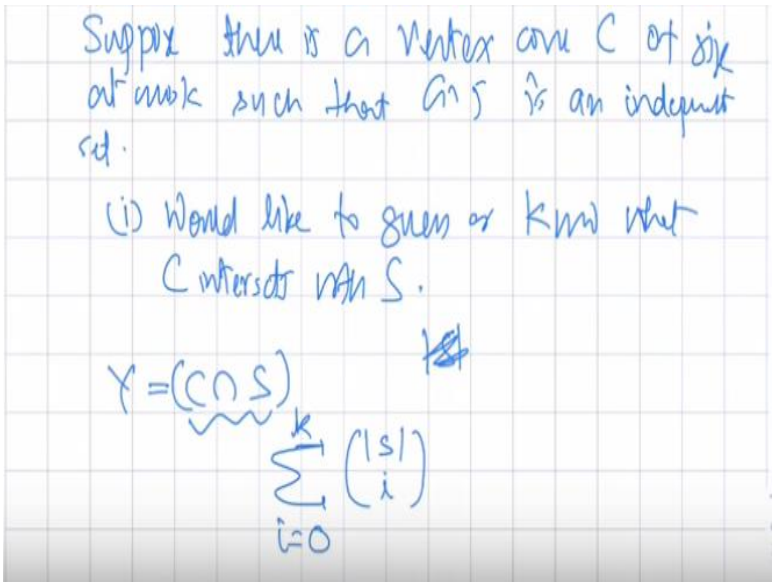
(Refer Slide Time: 28:41)



Alternate algorithm, first, compute a factor 2 approximation to vertex cover. So, remember, the, this algorithm can have 2 answers, No and Yes. No means you also return. Say return No, no vertex cover of size at most k. Yes means you have a S of size 2k. So, this is an interesting case for us. So, rather than writing this algorithm, let us try to understand what this means.

So, my algorithm so look picture wise here is my set S and here is my G – S. There are no edges here. This is an independent set. So, basically all edges are going across or they are contained inside graph **(()) (30:06).** So, I asked myself the following question.

(Refer Slide Time: 30:18)



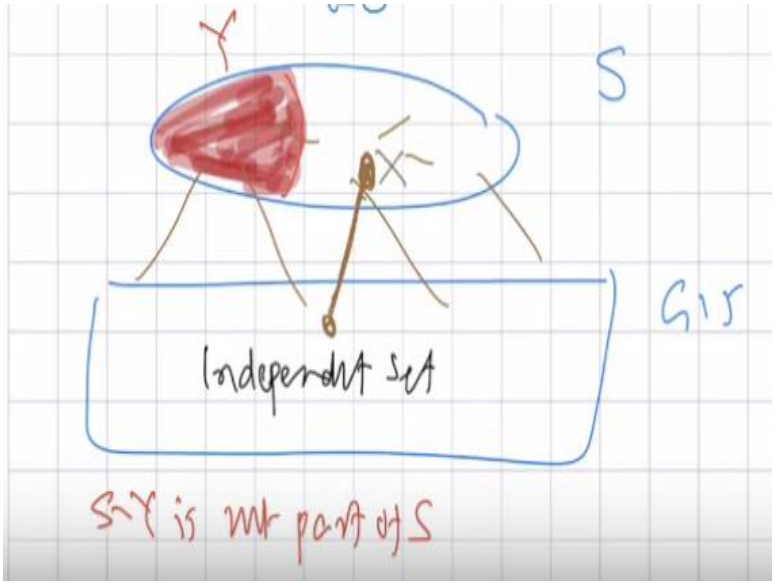
Suppose there is a vertex cover C of size say at most k such that G – S is a vertex cover or G – S is an independent set. Now, I asked myself, fine. So, what, like I asked myself look at that

C. That C might intersects S, it might not intersect S. Now, I want to I would like to guess, what does it intersect from S? So, the first step is would like to guess or know what C intersects with S.

And that is how many such choices are there? So, would like to guess, so, which basically means that we will try all possible sets. So, what is the meaning of this? We will say, let Z be C intersection S. Or rather let us say, yes. This is my C intersection S. How many possible choices of C intersection S is? Well, suppose it is intersection could be empty. Well, so, let us I will guess the cardinality of this.

So, it is going to be at most i equal to 0 to k, mod S choose i, which basically means, well, I am asked, I am checking whether C intersection S is 0 1 2, but it could be up to k. So, I will try all possible subsets as potential this. So, this is it. So, the possible choices of Y is this. So, once we have guessed in Y, let us try to run the algorithm, will try to make an algorithm in polynomial time. So, what we have done? So, we have done is following. Let us cut this.

(Refer Slide Time: 33:36)



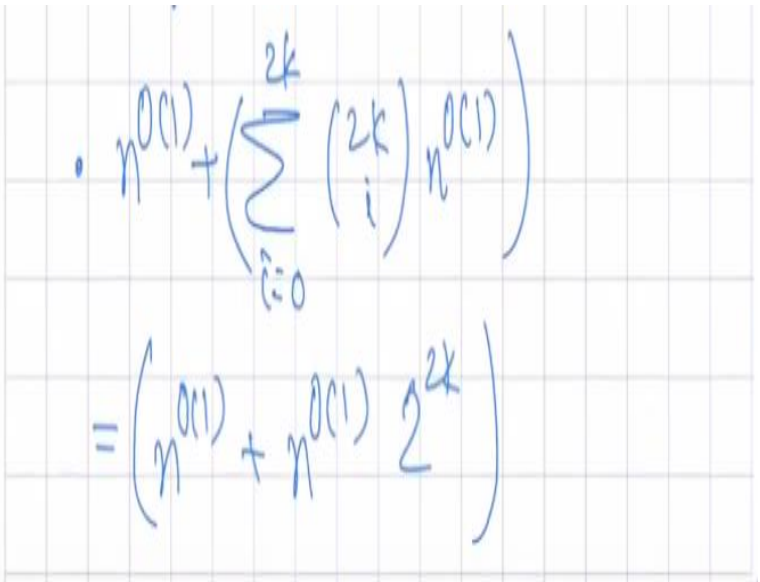
Now, this is my Y. So, how does Y look like? So, let us color that Y with red. So, suppose this is my Y, then what do I know? So, if why is this remaining vertices of S is not part of n, great. So, basically, S – Y is not part of S. But this also gives an useful information towards. Because notice that if S – Y is not part of a vertex cover, then all the edges which are incident to S – 1 S – Y.

There are only 2 choices for each edge which S – Y covers. Since that endpoint, look at an edge here. Look at let us look at an edge here, if this vertex is not part of my C, then this vertex must be part of my C. So, basically given an Y you consider potential vertex cover C as Y cup neighborhood of S – Y cup. Let us call this set I independent set. That is it. So, these are your potential you have to. And notice that because every edge was covered by S.

So, now, this set covers every edge which is covered by S – Y. Y covers every other edge which implies that C is actually a potential vertex cover actually a vertex cover. So, all I need to check if C at most k or not. So, the algorithm is very simple. So, in the second step, what it does for, so, the second step of my algorithm is for every Y subset of S of size at most k check if Y union N S – Y intersection I is a vertex cover of size at most k.

If I succeed in any of for any Y, then I do know there exists a vertex cover of size at most k. And since it is an exhaustive because we tried every possible intersection including empty, if none succeeds, then we know there is no vertex cover of size at most k. So, this algorithm is correct. But what is the running time of this algorithm?

(Refer Slide Time: 37:11)

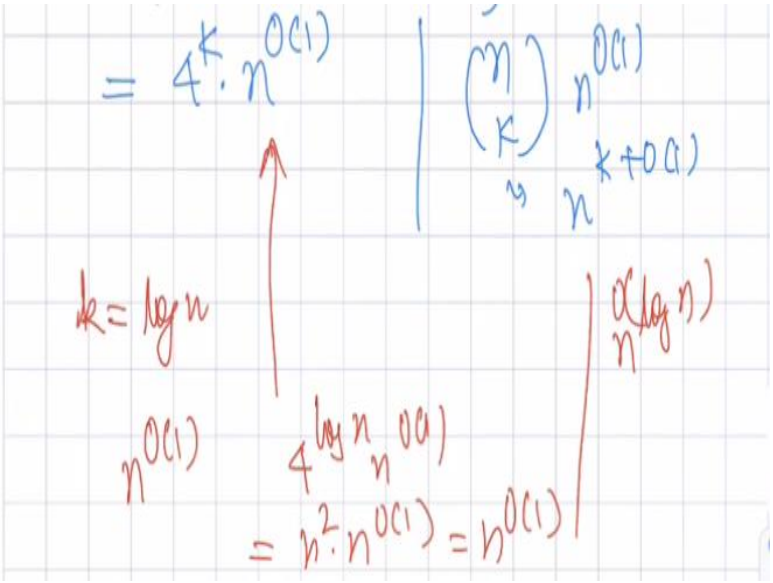


So, the running time of this algorithm is notice in the first step is like n to the power big O of 1. Given this in the second step, we have got a set of size at most 2k. And so, the second step is like for each of the choices, i equal to zero to at most 2k. This is upper bounded by 2k choose i times some n to the power big O of 1 to test. And this is nothing, but if you n to the power big O of 1 plus by binomial I can write down.

This is nothing but at most what, 2 to the power 2k. So, just by binomial because summation i equal to 0 to n, n choose i is to power n. So, 2k takes the role of n, and then you are done. So, we have actually made an algorithm, which runs in time 4 to the power k n to the power big O of 1. Now, notice something good about this algorithm. So, what is good about this algorithm? So, something very good about this algorithm is as follows that we had one algorithm which ran in time.

Let us get that n choose k n to the power big O of 1. And we have another algorithm, which is 4 to the power k times n to the power big O of 1. So, this algorithm is roughly you can say n to the power k plus big O of 1. So, I mean this algorithm is like bad, just for if the number of vertex in the graph is greater than or equal to 4. But another way of looking at is, let us set the parameter set the number k equal to say, log n.

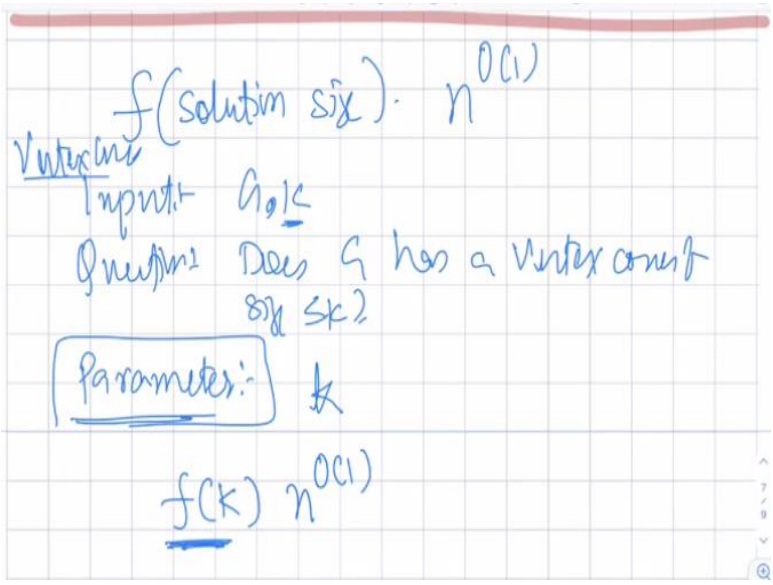
(Refer Slide Time: 39:01)



Then this algorithm is actually n to the power big O of 1. So, up to k equal to log n, we can test with this new algorithm with there exist a vertex cover of size at most k, at most log n in polynomial time. Why? Because, what is 4 to the power log n n to the power big O of 1? This is nothing but n square times n to the power big O of 1, which is n to the power big. But this algorithm, even here is going to run in time n to the power big O of log n.

So, what, so, if you are looking for a vertex cover of size at most k, this brute force algorithm will run in quasi polynomial time. But this nice algorithm will run in. So, this nice algorithm will run in polynomial time. Now, this brings us to the following concept or the following idea, which is a very important and key and central to our concept.

(Refer Slide Time: 40:11)

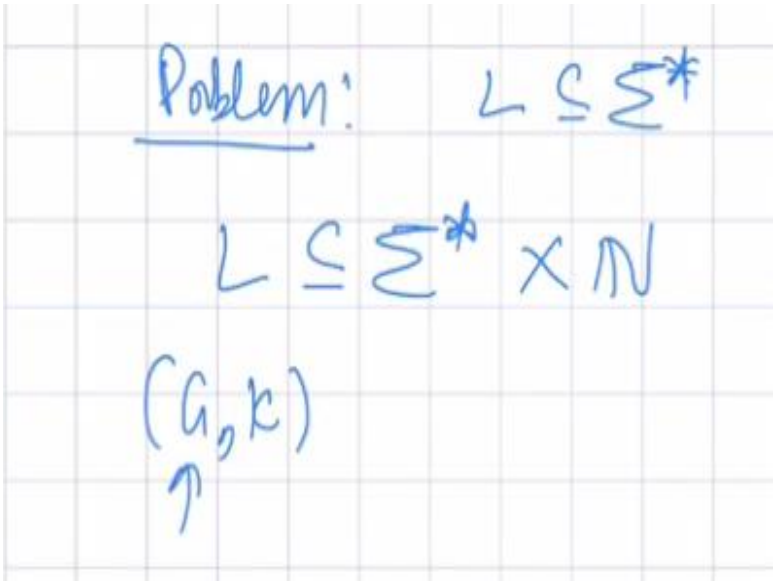


So, what is our basic important definition then we **(()) (40:13).** So, basically what happened here is that we actually design an algorithm which was running in time f of solution size times n to the power big O of 1. Alright, so, basically **(()) (40:36)** are the kind of algorithms that we would like to design. So, the kind of problems you would be interested in. So, for example, if you think of a input for a vertex cover we had G k.

We had a question, does G has a vertex cover of size at most k? The G has a vertex cover of size at most k. Well, now, I can think of this number k as a parameter. What is the meaning of this is like. So, now, I would like to say that look, yes, we do know that the algorithm cannot be solved in polynomial time. But is it possible to design an algorithm whose running time can be bounded by some function which is just depends on the parameter and the dependence on the input side is still polynomial.

So, we have now it is a 2 variable running time of an algorithm. One which is one of function which is allowed to be an exponential in k. But there is another part of the function which is the dependence on the input side which is polynomial as it was classically in the world of polynomial time **(()) (42:13)**. So, in particular, what happens is that so the kind of algorithms we are going to design will have for a problem.

(Refer Slide Time: 42:26)

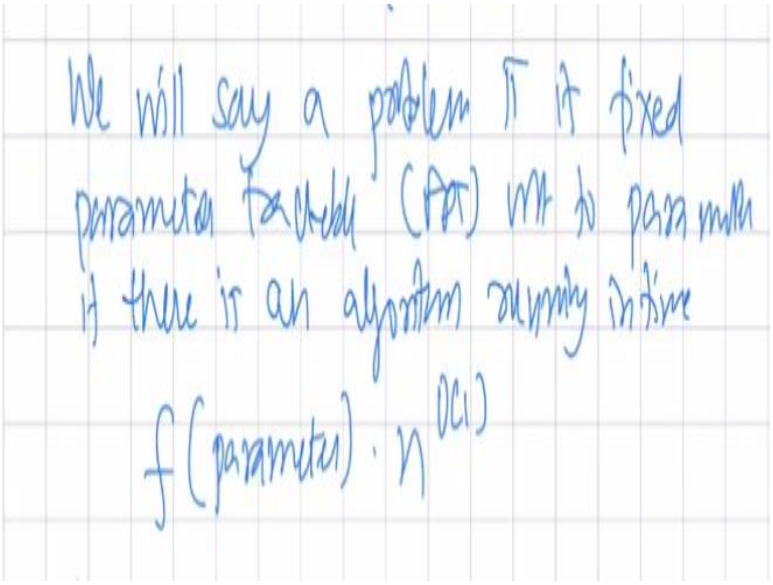


So, the problem generally will be so classically, notice that so k is our part of our input. So, classically, problem used to be called or the language whatever you would like to call it, L used to be subset of sigma star. But the world we will be living in our problem will have will be a subset of sigma star cross natural number. So, for example, for vertex cover, what is our input? G, k.

So, G belongs to our sigma star and n k is a natural number that it represents. So, this is one thing to remember. And so in classical world, so here is a classical world. Let us write it. Classical and the new world or say parameterized world, classical world, parametrized world. So, the classical world, input is given by instance and a question. In the parameterized world, you have to give me an instance which is like G, k. Example, a question.

But also you need to tell me a parameter in the sense, because this is what dictates me that given a problem, what kind of algorithm we are looking for. So, I am looking for an algorithm whose running time can be bounded by function of parameter only. So, this brings to us to the notion of what is called a most important notion of in this course.

(Refer Slide Time: 44:21)

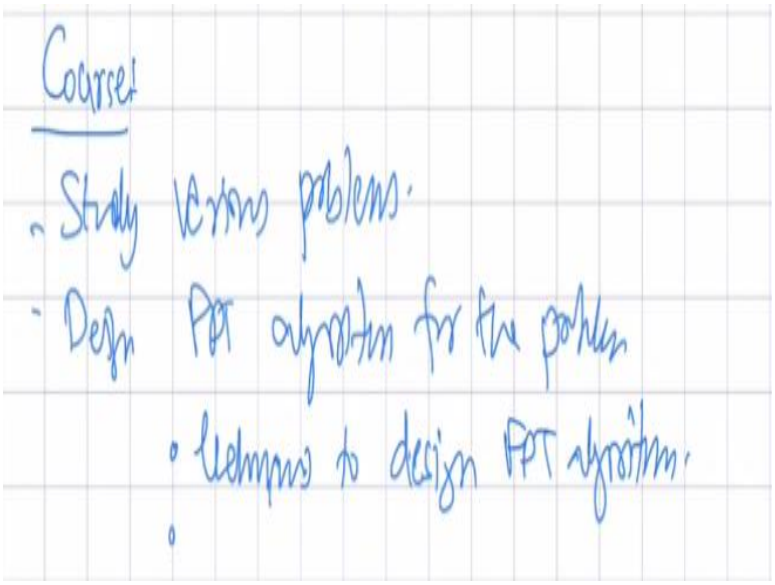


We will say a problem pie is fixed parameter tractable or FPT. With respect to parameter, if there is an algorithm running in time f of parameter times n to the power big O of 1. Here, f only depends on parameter. So, what did we saw? So, we I will define it more formally after 5 minutes, but depends on the parameter. So, what we saw? So, if we saw that vertex cover is fixed parameter tractable with respect to parameter k or solution size.

Now, something which I think before I go further, let me formally define the notion of I would guess that I should formally define the notion of FPT. But, maybe will get to that at the right time. Maybe in the next lecture, we will define formally what FPT means. But for now, let us just live with this definition that there will be an input. There will be a parameter. And we would like to design an algorithm where the running time dependence on parameter could be any function.

And n to the power big O of 1 is the dependence on the input size. Now, what are we going to do in this course?

(Refer Slide Time: 47:14)



So, in this course, we are going to study various problems. Now, design FPT algorithm for the problem. That is our main goal. So, we will learn techniques to design techniques to design FPT algorithms. And that is one thing we will do. But there is more classically there are problems for which there is no FPT algorithm. So, we will also see methods to design hardness in this world. And if you notice, we are always saying the problem is fixed parameter tractable with respect to parameter something.

So, for the same problem, there could be several parameter. And a problem may be FPT with respect to one parameter, but may not be FPT with respect to another parameter. So, in the next lecture, we will define what FPT means formally and define and give you one example of a problem where with respect to one parameter we do not expect to have a such an algorithm, but with respect to some other parameter we do expect such an algorithm.

And I will show you one such algorithm. So, with that, let us conclude the first part of our introduction where we have defined the notion of fixed parameter tractable algorithm which is basically means designing an algorithm where the dependence on the parameter could be exponential, but the dependence on the input size is polynomial, like purely polynomial which does not depends on the parameter. It is just old classical polynomial time, thanks.