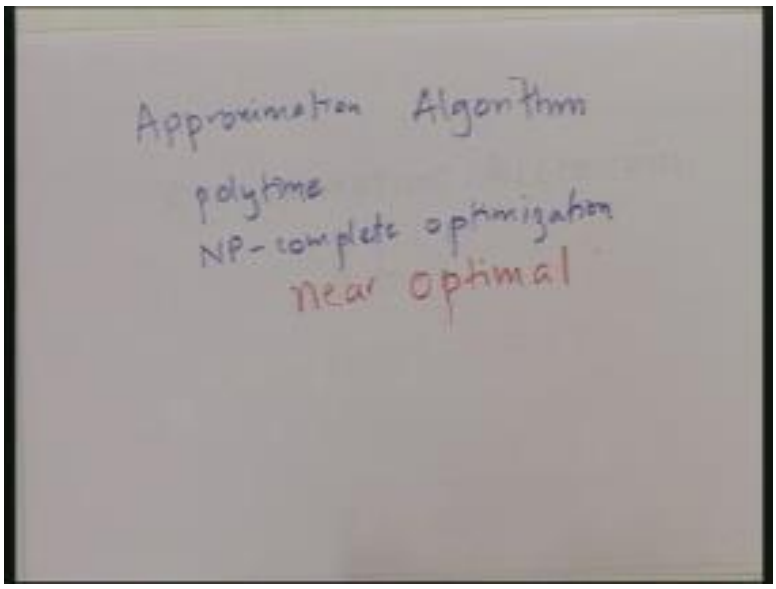
Design & Analysis of Algorithms
Prof. Abhiram Ranade
Department of Computer Science & Engineering
Indian Institute of Technology, Bombay

Lecture - 33
Approximation Algorithms for NP - Complete Problems – II

Welcome to the course on Design and Analysis of Algorithms. This is the second lecture on Approximation Algorithms. We already saw in the previous lecture, what are the approximation algorithms were? We will go over that very quickly today. And we will take two more examples. So, let me quickly remind you what an approximation algorithm is.

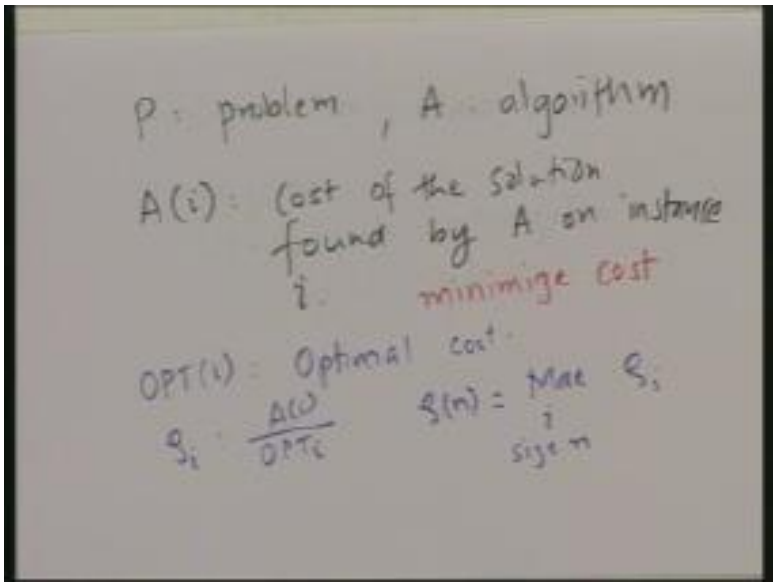
(Refer Slide Time: 01:20)



So, an approximation algorithm is a polynomial time algorithm, first of all. And it solves, an NP complete optimization problem. And it gives near optimal solutions. The solutions it gives are near optimal. Of course, if it gave optimal solutions in polynomial time then, that would prove p equals NP. And of course, we are not here to prove that. And most people in fact believe that, that is not correct.

So, the most people believe that p is not equal to NP. And therefore, polynomial time algorithms which give optimal solutions are unlikely to be there for NP complete problems. So, last time we looked at this and we define some notation for this. So, let me just quickly go over that notation.

(Refer Slide Time: 02:29)



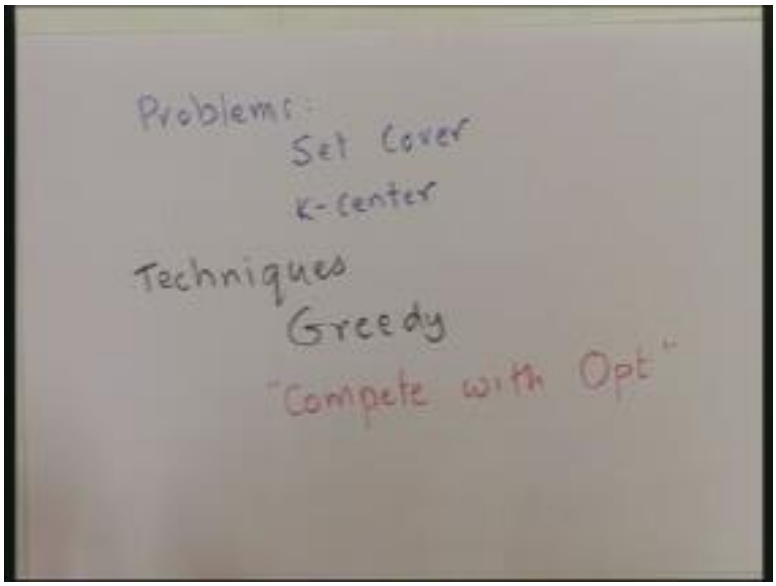
So, let us say p is a problem. And A is an approximation algorithm for solving it. We said last time that, if i is an instance then A of i is the cost of the solution found by A on instance i. So, here we are going to assume that our problem is a cost minimization problem. So, there are some constraints which are specified as a part of the problem. And a cost function is given. And our goal is to minimize this cost.

As we said, this cost in general will not be the optimum A i, will not be the optimal for instance i. But in fact, there is some other cost, which I will call OPT of i which is presumably better than this. So, this would be the cost found by the optimal algorithm. So, clearly we know that A of i is at least as big as optimal as OPT of i. So, we said that rho are the approximation factor on instance i, is simply going to be A of i upon OPT of i.

And we also said that, rho of n is going to be defined was defined as Max over all instances of size n of rho sub i. So, this is the approximation factor of the algorithm. And this is what we want to keep small. So, this is the general framework in which we are going to work. So, we are going to devise algorithms for NP complete problems, which produce a small approximation factor.

In other words the cost that they return, the cost of the solution that they return A of i is going to be reason reasonably close to OPT of i. And of course, they are going to be running in polynomial time in path of i. So, today I am going to look at two problems.

(Refer Slide Time: 05:22)



So, one problem is going to be the Set Cover problem. And another is the, is going to be so called the k Center problem. I will describe these problems in a minute. And I will also give real life examples, corresponding to these problems. In addition to that, we are also going to look at some techniques or some themes. First of all, we are going to use some kind of a greedy idea for solving these problems.

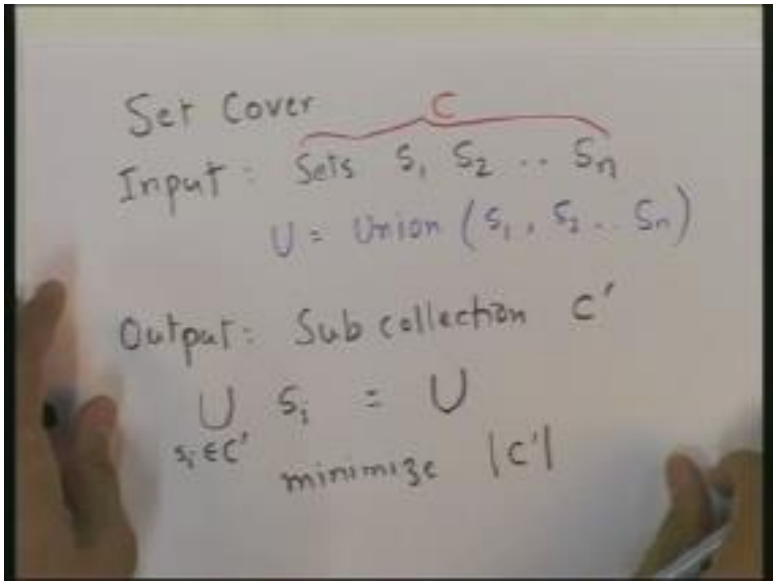
See you have already seen greedy strategies earlier. So, a greedy strategy is, in greedy strategies procedure for solving is to be thought of as a sequence of decisions that you make. Every decision is going to change the cost a little bit. And the goal, the idea of the strategy is that the ith decision that you take, tries to make the ith change in cost the ith increase in cost, as small as possible.

Greedy is also short sighted in the following sense that, we will try to minimize the ith increase in cost using the ith decision. But in doing so, we will not worry about what happens subsequently. In general of course, greedy algorithms will not work. But, in many cases they seem to work. And today's problems are somewhat in that same framework. In addition to that, we are also going to see an interesting proof strategy which I will call Compete with the Optimal.

So, we are going to imagine that, there is an optimal algorithm running along with the algorithm that we design. And we are going to try and do at least reasonably well, as compared to that. We cannot do better than or even as well as the optimal algorithm but,

we will try to see that we do not do too badly. So, let me start off with the set cover problem. So, here is the set cover problem.

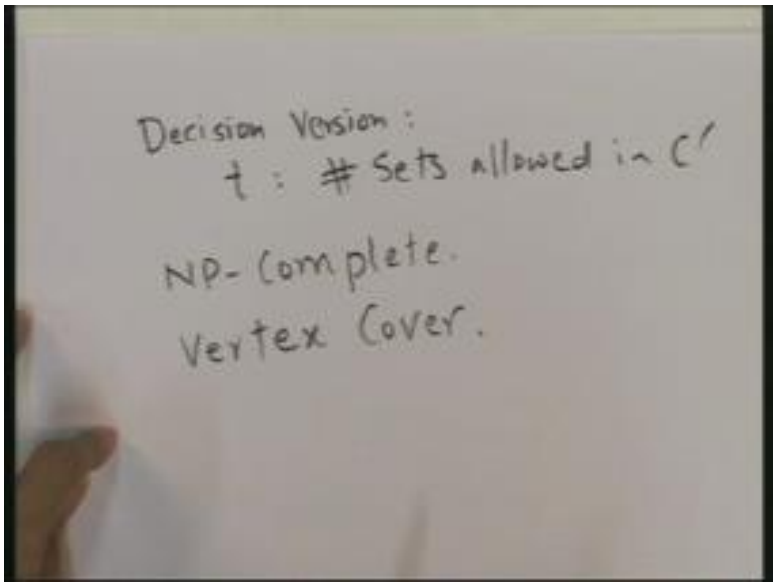
(Refer Slide Time: 08:00)



The input consists of sets, s 1 s 2 all the way till s sub n. Let me use, U to denote union of s 1 s 2 s n. So, let me call this collection this entire collection I am going to call C. So, this is the collection of sets. And our goal, the output that we want is a sub collection. Let me call it C prime. Such that the union of s sub i, where s sub i belongs to C prime has to be exactly equal to U or the union of all the sets.

So, we want a small collection C prime. A small set of sets such that, they contain the same elements as the original connection. So, what is to be there is something to be minimized over here. And we want to minimize the cardinality of C prime. So, this is the set cover problem.

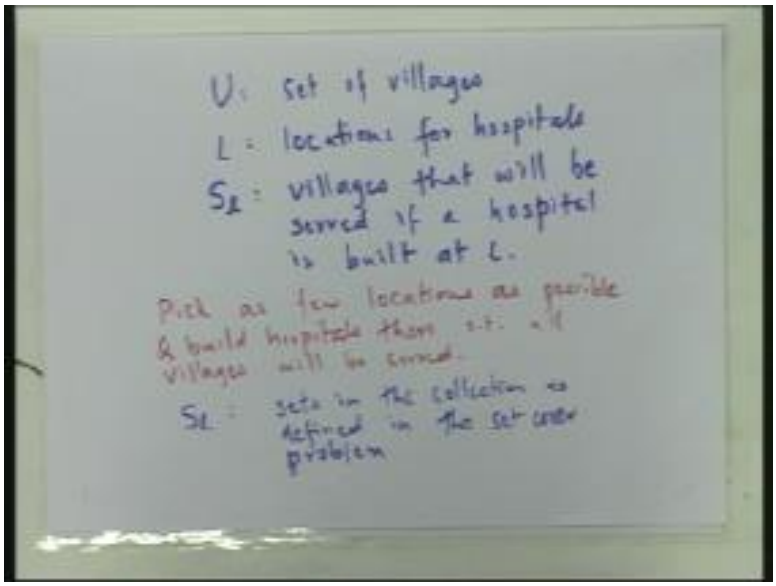
(Refer Slide Time: 09:53)



In the decision version of the problem, so the decision version as usual we are given an additional target t. So, this is the number of sets allowed in C prime. And what happens over here is that this. And we are asking does there exist a C prime with cardinality t. Such that its union, the union of the sets in it is U. And this problem is NP complete. In fact, the decision there is a reasonably simple reduction from the vertex cover problem.

We are not going to prove this. But, we are just going to directly try and find an approximation algorithm for the set cover problem. Now, before going to the algorithm let me give you a brief application, a small application of this problem to tell you that, to tell you how it might arise in real life.

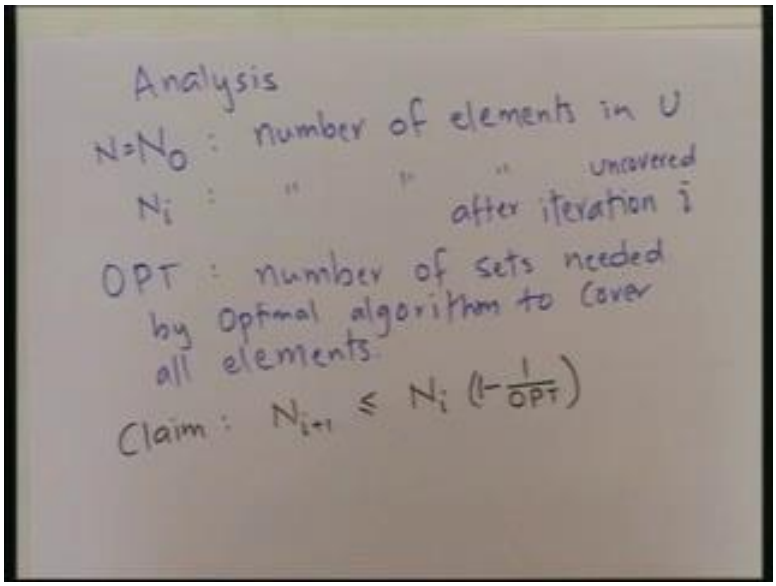
(Refer Slide Time: 11:07)



Suppose U denotes a set of villages. Suppose we are also given a set L of locations, here hospitals can be built. For each location say little l, we are given the set S L. And this is the set of villages that will be served, if a hospital is built at L. The natural problem now is to determine the smallest number of locations such that, all villages can be served. Let me write the term. We want to pick as few locations as possible. And build hospitals there, such that all villages will be served.

Do you see now, how this corresponds to our set cover problem? Well, the correspondence is actually exact our sets S L constitute the collection. These are the sets in the collection as defined in the problem, in the set cover problem. And what we want is to pick a sub collection of this collection such that, all villages will be served or in other words all villages will appear in at least 1 S L. So, the correspondence is exact. So, now we turn to the algorithm.

(Refer Slide Time: 13:45)



So, the basic idea of the algorithm is greedy. What do I mean by that? Well, we are going to think of picking sets when at a time. And every time we pick a set, we will try to cover or to collect together as many elements, which have not yet been collected. So, here is the idea. So, here is basically the algorithm. So, I am going to start by defining my c prime to be null.

So, c prime is going to be the collection our answer eventually. I am also going to say that, all elements of U which is the union of all the sets are initially uncovered. By covering, I mean they are inside a set which is included in this c prime. And now, here is the basic loop. So, while some uncovered elements exist, what do we do? We pick set S i that contains maximum uncovered elements.

So, we will have to maintain some data structure which says, which keeps track of which elements are covered and which elements are not covered. And as soon as we pick elements and we put that, pick a set and we put that set into c prime we will have to cover those elements. So, we pick the set. And then, we set c prime is equal to c prime union S i. And then, we uncover or recover elements in S i.

So, our universe or the ground set, has some elements has all elements initially uncovered. Then, we pick sets as a result of which some elements get covered. And we keep on covering elements. And eventually, we get to a point where all elements are covered. And that is a time, we stop. So, this is where we end the loop and we return c

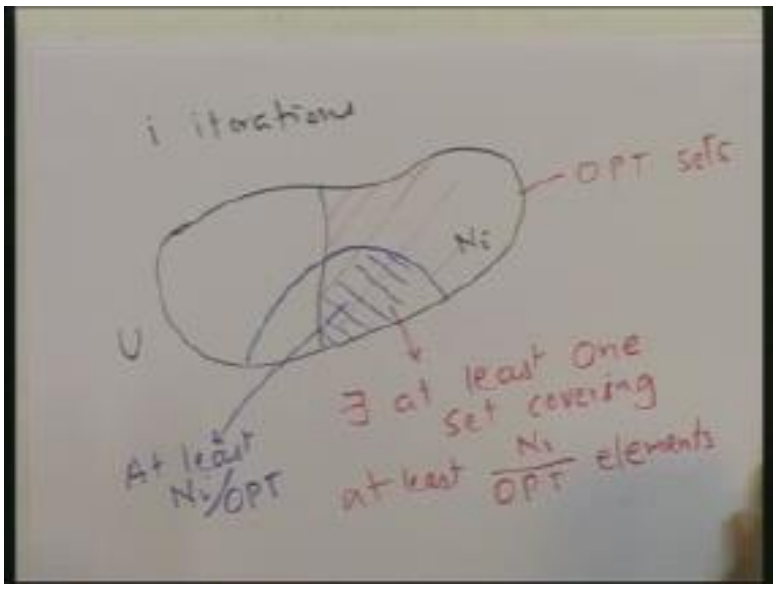
prime. So, we are making the decision at each stage which said to include into our sub collection and at each stage we are saying.

So, let us include a set which gives us which tries to cover maximum elements. So, the analysis of it is actually quite nice and simple. The analysis goes by iterations of this basic algorithm. So, I am going to start by defining some notation. So, let say N sub 0 is the number of elements in U, the total number of elements which I want to cover or this is also equal to N. There is a reason why I want to call it N sub 0 as well.

N sub i is going to be the number of elements, uncovered after iteration i. So, number of elements in U originally, these are all uncovered. So, this is uncovered before the first iteration. N 1 would be elements, which are uncovered after the first iteration after the second iteration, all the way till after the ith iteration. I am going to use OPT to denote, number of sets needed by optimal algorithm to cover all elements.

So, this is my notation. And now I want to state my main claim. So, the main claim is the following. The main claim says that, N sub i plus 1 is less than or equal to N sub i times 1 minus 1 over OPT, very simple claim. And the proof actually is also fairly simple. So, how does the proof work.

(Refer Slide Time: 19:30)



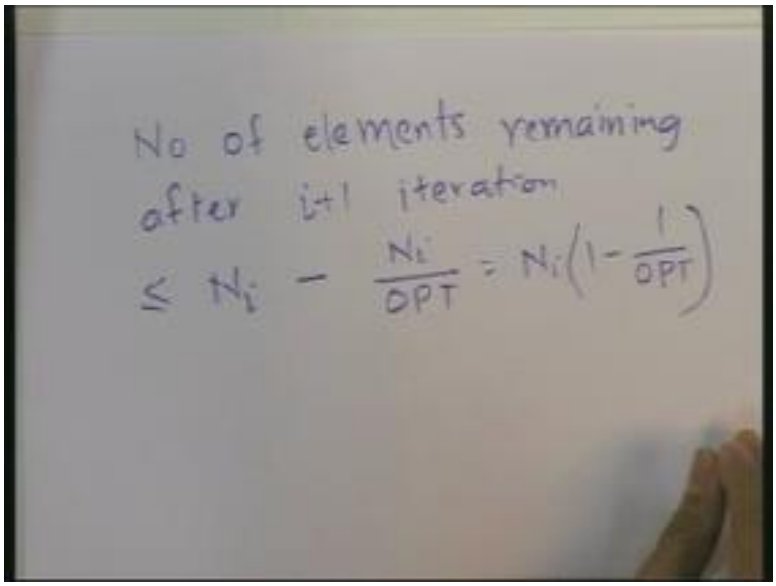
So, our algorithm has executed for i iterations. And this is our set U of which some elements have been covered. And these are the N sub i elements, which are not covered

after the i, after the ith iteration. Now, what do we know about the optimal algorithm. As we said our strategy is going to be, we will try to do we try to compete with the optimal algorithm. So, what we know about the optimal algorithm?

Well, we know that this entire set U is covered by OPT sets, by a number of sets equal to OPT. So, it covers all these elements. So, therefore know that even this region is also covered by OPT sets. So, now if I look at how these sets are covering this region? I must claim that, there has to exist at least one set which covers N sub i upon OPT elements at least. So, at least one set covering at least. Because, if no set covered at least these many then this group of N i elements would not be, it would not be possible to cover this.

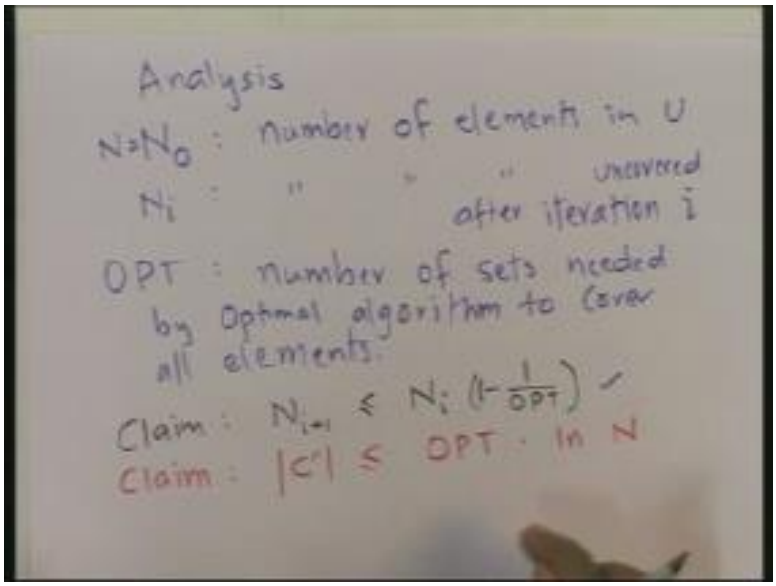
So, there has to exist at least one set which covers at least N sub i upon OPT elements. Here is where the greedy property now comes in. So, at this point we choose a subset, which covers the maximum number of elements from this. So, which? So, what do we know about the set? So, this subset that we choose is this. Then, we know that this region has to contain at least N sub i upon OPT elements.

(Refer Slide Time: 21:53)



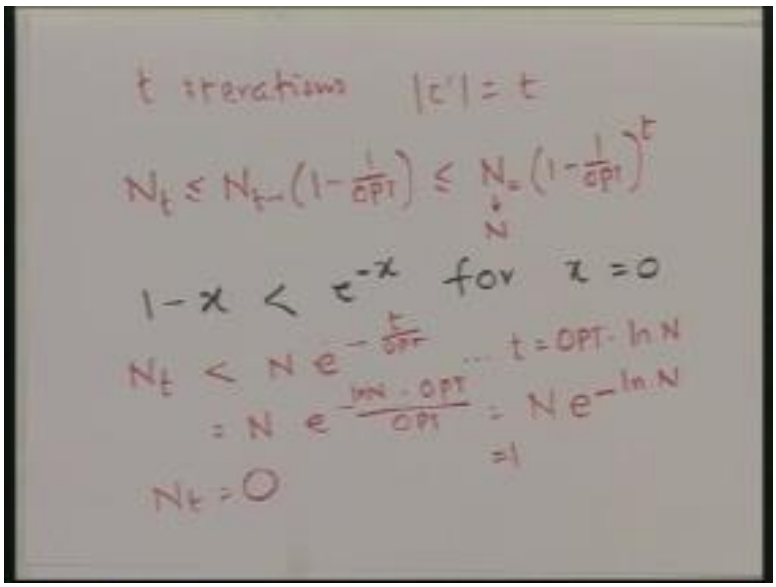
So, what does that mean? That means that number of elements remaining after i plus 1 th iteration has to be less than the N sub I, which were present before the i plus 1 th iteration minus whatever got covered. And these are N sub i upon OPT. So, in other words N sub i times 1 minus 1 over OPT. This is precisely what we claimed a few minutes ago.

(Refer Slide Time: 22:36)



So, this claim has been proved. So, let me write down our second claim just here. Our second claim is that cardinality of c prime, number of sets returned by our algorithm is less than or equal to OPT times l n N. So, it is OPT by log N log of log N factor, log taken to the natural the base E. So, we will return c prime elements. Whereas, OPT will return the optimal algorithm, will return OPT. And we will not be returning too many, too much worse. We will not be doing too much worse. We will be doing only an l n and N factor worse. This is our second claim, this is what we want to prove next.

(Refer Slide Time: 23:41)



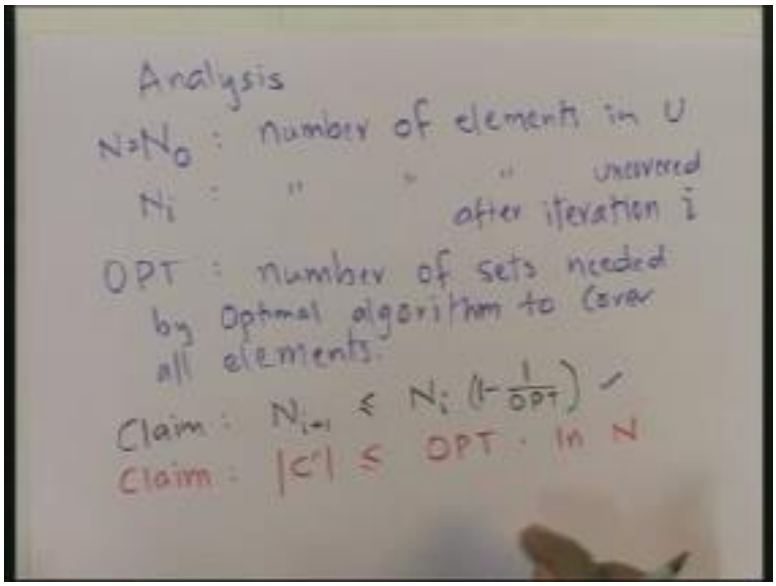
So, how many elements to be returned? Well, if the algorithm runs for t iterations then, we return cardinality of c prime is equal to t. So, what we need to evaluate is how many iterations, does the algorithm run for. So, suppose it runs t iterations then, what do we know about the number of elements that are uncovered after t iterations. Well, we know that N sub t is that number. And we know from our first claim that, N sub t is less than N sub t minus 1 times 1 minus 1 over OPT.

But, we can keep on repeating this. And so, therefore we get this is less than or equal to N sub 0 times 1 minus 1 over OPT whole to the power t because, this factor will keep on repeating. And note of course, that this is just N. Now, nice little inequality comes to our rescue. And that inequality is 1 minus x is less than e to the power minus x for x not equal to 0. So, what does that allow us to conclude?

So, therefore we can conclude N sub t is strictly less than now N times. So, this is 1 minus 1 over OPT. So, 1 over OPT is going to be my x. So, I am going to have e to the power minus t upon OPT. Now, let us check what we what happens if t is equal to OPT times l n N. What happens then? So, this is nothing but N times e to the power minus l n N times OPT upon OPT or this is N times e to the power minus l n N. And therefore, this is strictly less than 1.

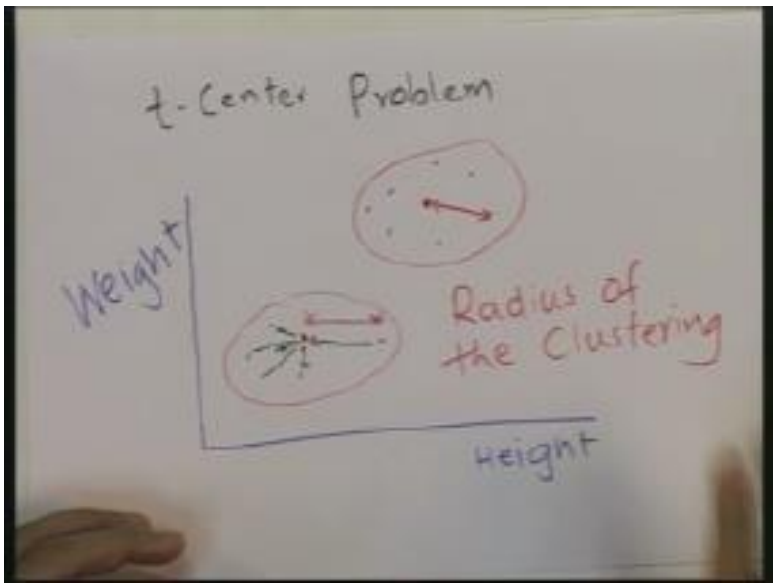
So, we have proved that N t is strictly less than 1. But, if N t is strictly less than 1 what does it mean? That means, that N t is exactly equal to 0. What is that mean? That means, after t iterations all the elements of U have been covered. Nothing has been left uncovered. But, this happens for t equals OPT times l n N. So, in other words cardinality of c prime is OPT times l n N. And that is exactly what we claimed.

(Refer Slide Time: 26:40)



So, this finishes the analysis of the set cover algorithm. So, what we have proved over here is that, we can also find a set cover. The size of that set cover is going to be worse by a factor l n N. We now come to our second problem. The second problem is the so called k center problem. I do not want to use the letter k, because k comes in handy for indices and things like that. So, I am going to call it as the t center problem.

(Refer Slide Time: 27:19)



This time, I am going to describe the problem informally first. And then, I will formalize it. So, let us say let us take an example. The example is that, we are given some statistics

say about students in a class. So, perhaps we are given a plot. So, on one axis we say have the height on the other axis may be we have say the weight. So, may be the class has some number of students and may be the plot looks something like this.

So, this is the plot of the heights of different students. Now, an important question in analyzing this data statistically, is to see if there are any clusters. So, say for example you could think, that this region is consists of students who are somehow similar in their build. This region consists of students, which are who are somehow similar in their build. So, it would be. So, it is interesting statistically.

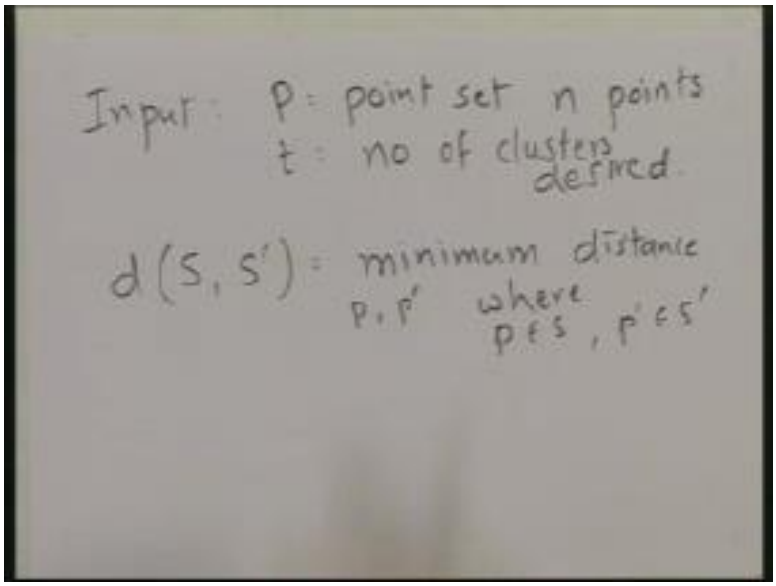
It is see, if this entire distribution can be described just by using say two representatives. And the question then is, is there a good way of choosing those representatives. For example, we could say that this could be one of the representatives. And say, we could say that this is another representative. How do we choose a representative? Well, a representative is chosen, so that its distance from the rest of the elements in the cluster is as small as possible.

And furthermore, we decide to put an element. Say this one into this cluster because, it is closest to its representative. So, the two the thing to the important thing to note over here is the so called radius of each cluster. So, this cluster has this is the farthest element in this. And therefore, this is the radius of this cluster. In this say for example, this might be the farthest distance. And so, this is the radius of this cluster.

And so, the maximum of these two is the so called radius of this clustering. So, what is our goal? Well, we are given how many clusters are needed. And we want to pick the cluster centers and divide the data into clusters. So, in this case this is one cluster and this is going to be another cluster. Here the clusters are very obvious. But, in general they will be mixed and finding representatives and defining the boundary between the clusters, is going to be a little bit hard.

But in any case, we want to define the boundary and find a center the representatives and the radius, the radia of the clusters such that, the largest radius is as small as possible. So, that is how we measure the goodness of the clustering problem. So, we are going to do this but, before that I would like to give you the formal definition of this problem.

(Refer Slide Time: 30:48)

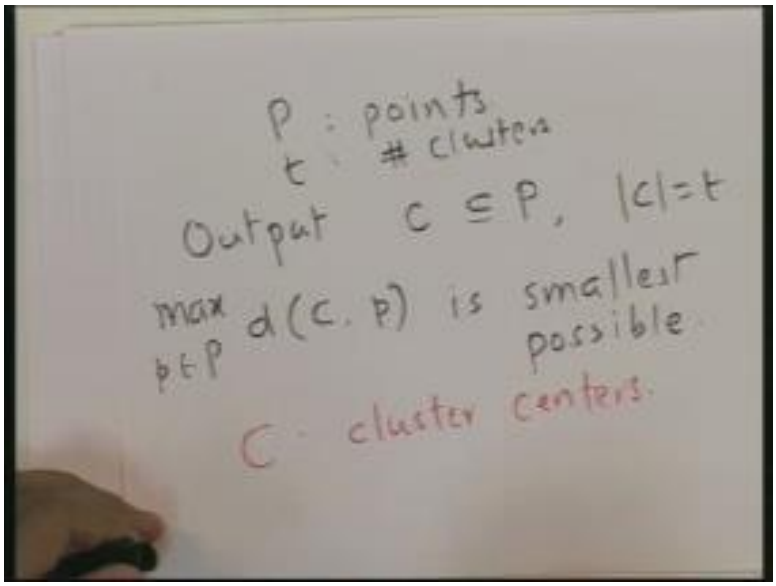


So, the input consists of a set P, which is a point set. Let us say, this point set is in some D dimensions. And it has say some n points. And we are also given an input parameter t. This is the number of clusters desired. Before I describe the output, I need to have good notion of distances and things like that. So, the distance between two points is going to be simply there, Euclidean distance. So, it is a d dimensional space.

It is going to be square root of the sum of the squares of the coordinate differences as usual. But here, we are also going to be talking about the distances between a set of points and a point. So, let me define that or a distance between one set of points and another set of points. So, let us say s is one set of points and s prime is another set of points. So, I am going to define the distance between two point sets. As the minimum, the minimum distance between p p prime, where p is in s and p prime is in s prime.

So, these could be in general arbitrary point sets. But, we want to see are there two points at least which are too close or very close. And that is the distance that if we are going to pay attention to. So, let me now tell you what the output is.

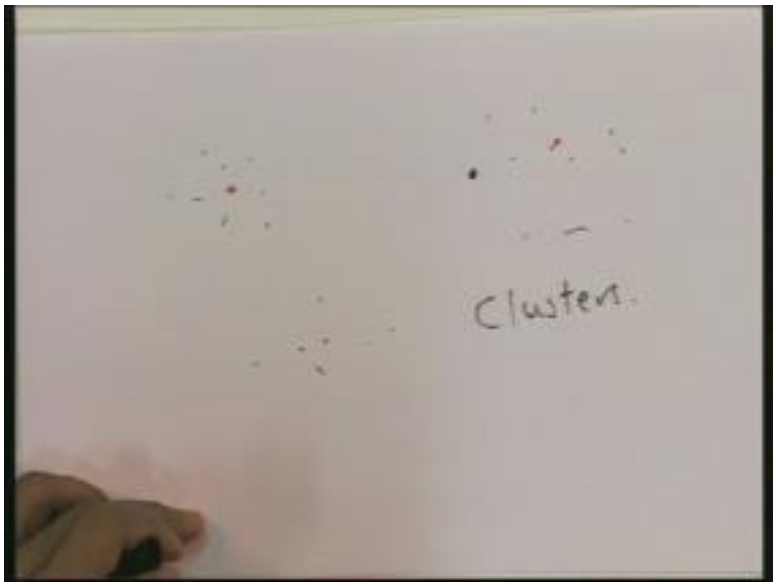
(Refer Slide Time: 32:49)



So, we had the set P of points. So, the output is a set c which is a subset of P such that, cardinality of c is equal to t remember t was number of clusters. And what we want is, that d of c to p should be as small as possible. So, I am going to take the maximum over little p belonging to capital P of this distance. So, I am going to ask which is the point which is farthest from these centers, that is the distance which I want to minimize?

So, max P in p is smallest possible. So, what is the intuition over here? So, the c is the c that we want to select are the cluster centers. Every point is at some distance. So, when we ask d of c comma p, we are asking which is the closest center for this point? So, of the closest centers I am going to take that distance. But, now I am going to ask, which is the point which is farthest from its closest distance? So, which is exactly, what we talked about earlier?

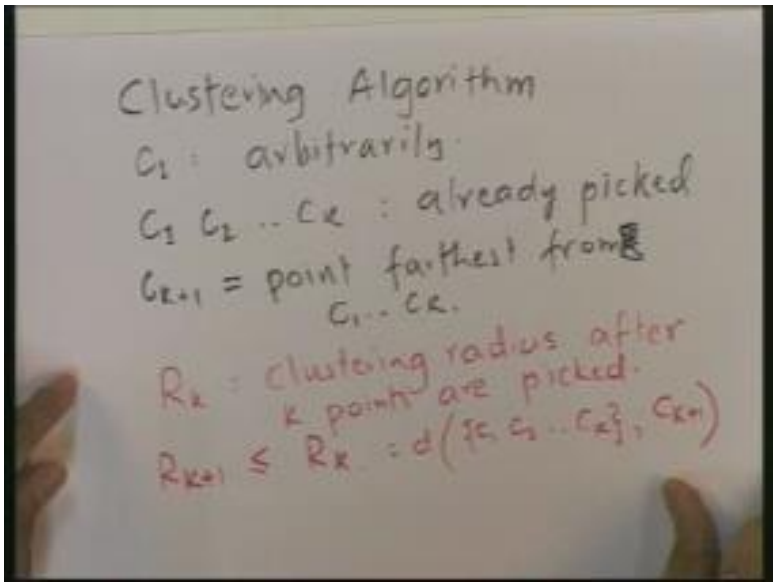
(Refer Slide Time: 35:00)



So, what has happened over here is that, this is our set of points. So, we pick some centers. And for each point we are asking, which is the closest center? And we are taking its distance and we are minimizing this maximum over all distances. So, let me also define the radius of each cluster. So, first of all let me define each cluster. So, if a point is closest to this center then, I am going to say that this point belongs to this cluster.

So, that is how clusters are defined. And of all the points which are belonging to a certain cluster, I just look at the maximum distance. And that is the radius of that cluster. Let us now describe, let me now describe an algorithm for this problem. So, this is the clustering algorithm.

(Refer Slide Time: 36:00)



So, what are we suppose to do? We are supposed to select the centers. We are supposed to select t centers such that, no point is too far away from any of the centers. And we are going to use a greedy strategy for solving this problem. So, how do I select the first center? Well, for that we do not really have really good idea right now. So, let us just say that it is picked arbitrary. But note that, if I pick some k clusters.

So, I have picked c 1 c 2 c k already picked. Then, if I want to pick c k plus 1 there is an interesting greedy idea that can come into play. So, what is the greedy idea? Well we ask, what is the farthest point from these first k clusters? We picked first k centers. And we want to know, which is the point which is farthest from this k centers? So, this point which is farthest is at some distance R.

Then that in fact, is the radius of this clustering that we have produced. So, now we want to reduce this radius of this clustering. How do we do that? Well, here is a simple idea. We pick c k plus 1 to be exactly that point. This is the point, farthest from c 1 all the way till c k. So, let me now have some. So, that is why that is basically the algorithm. So, I will pick c 1 arbitrarily. Then, I will pick c 2 to be the farthest point from c 1.

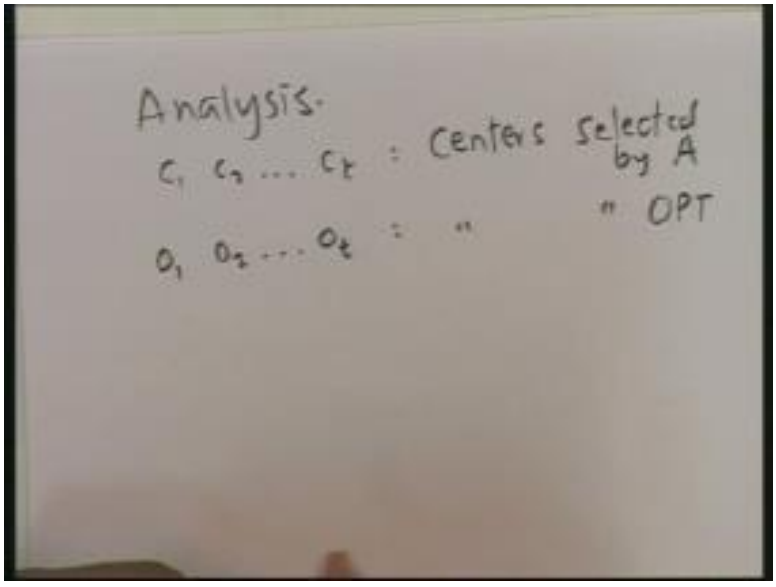
Then, I will pick c 3 to be the farthest point from c 1 as well as c 2. Then, given k points I will pick the k plus 1th point, to be the farthest from all these k points. Let me now define some notation. So, my notation is as follows. I am going to use R sub k to be the clustering radius, which is what we want to keep small after k points are picked. What do

we know about this clustering radius? So, we know that R k plus 1 is less than or equal to R k.

Why? Because, that is what we exactly did. We pick the point, which is farthest and we made it into a cluster. So, very likely now the farthest point the distance of the farthest point has reduced. And therefore, we expect the k plus 1th clustering radius to be smaller than R k. Well, what do we know about R k itself? What is what do we know about the clustering radius R k? This in fact, is the distance from c 1 c 2 c k to c k plus 1. Because that is exactly, what we did.

We looked at which was the farthest point after picking, having picked the first k centers. We picked at we looked at the farthest point. So, this was the point p. And we said, what was that distance. So, this is exactly that distance. And therefore, that must have been the radius after having picked, the first k points. And what we have argued earlier is that, the distance is going to keep on decreasing. So, these two are the important facts about how the radius of clustering change is related to each other and to the distances and to the centers. So, we are going to use these facts in a minute.

(Refer Slide Time: 40:55)



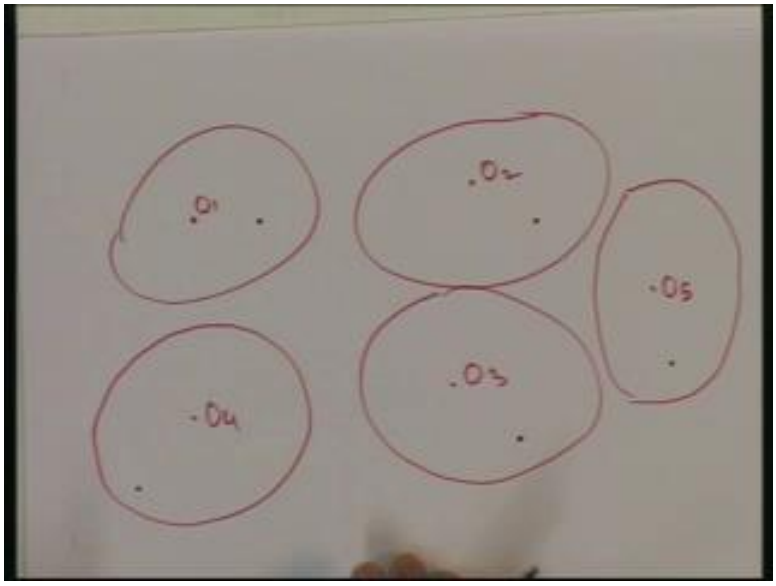
So, let us now analyze this algorithm in more detail. As we said earlier, this algorithm the analysis of both these algorithms is going to be based on this compete with the optimal strategy. So, we are going to try and see what the algorithm, what the optimal

algorithm would do. So, let me just remind you that, c 1 c 2 c t are centers selected by our algorithm A. Let us say, o 1 o 2 o t are centers selected by the optimal algorithm.

So, we would like to see how these centers are related to each other. The interesting thing is that, we do not know what o 1 o 2 o t are. But, nevertheless for the purposes of the proof we will assume that, we know them. So, we are not actually going to compute them any time. But, for the purposes of the proof we know them. And then, based on that we will be able to argue, what the relationship between the two clustering is.

So, we have those two we have the center selected by c 1 by the algorithm. And the center selected by the optimal by our algorithm and then by the optimal algorithm. So, let us focus on the centers selected by the optimal algorithm.

(Refer Slide Time: 42:23)

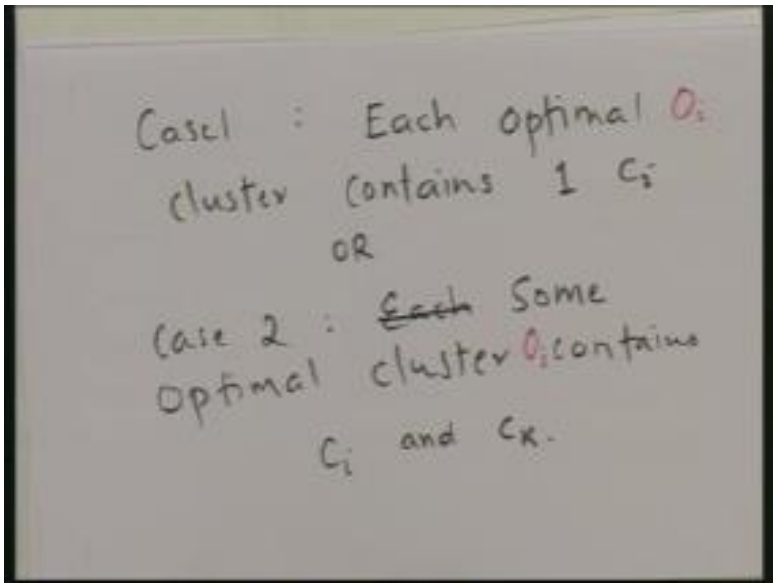


So, say this is o 1. This here is o 2. This here is o 3. This may be here is o 4. May be this is o 5. And let us also look at the corresponding clusters. So, say these are the corresponding clusters. I am not going to draw all the points inside but, I am just drawing the corresponding clusters. So, now let us ask how do the centers that we selected relate to these clusters. A natural question to ask is, is there exactly one center we select from each of the optimal clusters or do we perhaps select two centers from each cluster.

Natural guesses to say that perhaps, we just select our clusters our centers are also selected such that, that one comes from that each one comes from the other, from the

optimal cluster. Of course, it could be this or it could be the case that we select two centers. Two of our centers are selected from the same cluster, as the algorithm selects as the optimal algorithm selects. So, basically there are two cases.

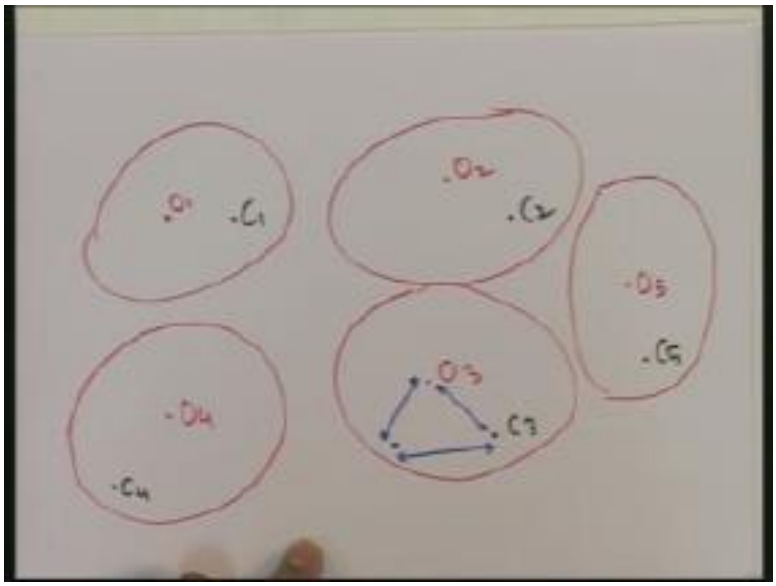
(Refer Slide Time: 43:46)



Case 1, each optimal cluster contains 1 c sub i. Or case 2, each optimal cluster or some optimal cluster contains say c sub i and c sub k. So, in fact let us call this optimal cluster o sub j. You might think that, we also need to consider a third case. Some optimal cluster contains no centers picked by our algorithm. But, note that in this case some other optimal cluster must contain at least two of our centers.

And so in fact, we will be having our case two in this possibility as well. So, let us now analyze case one in detail. In fact, each optimal cluster contains 1 c i. And we are free to give the optimal clusters whatever, names we want. So, we can say that each optimal cluster o sub i contains, exactly 1 c i. And in fact, we will name it correspondingly.

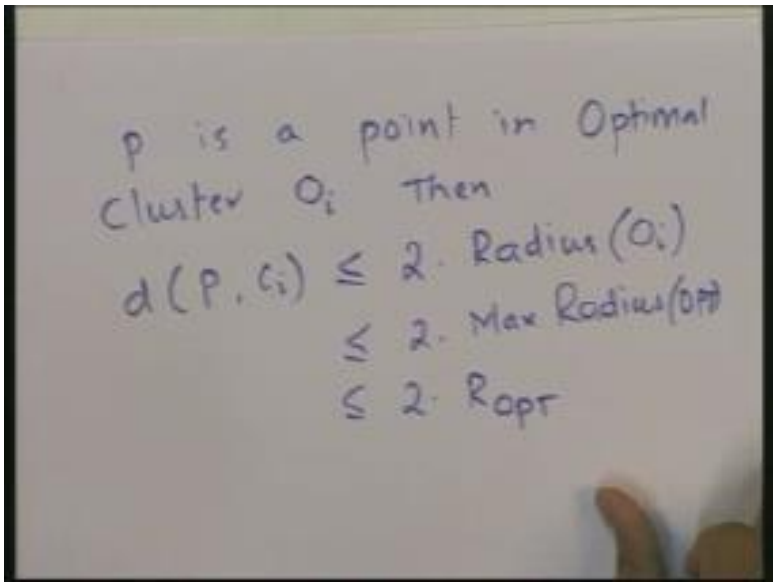
(Refer Slide Time: 45:36)



So, let us come to this picture over here. So, in this picture may be what we are looking at is, this is c 4 this is c 3 this is c 5 c 2 c 1. So, the algorithm happen to choose center such that, exactly one point is picked from the clusters that the optimal algorithm would have chosen. So, this is the first case that we are looking at. So, basically the idea now is that we will prove that in both cases. We will not do much worse than, what the optimal algorithm did.

So, now let us pick point p in one of these clusters. What do I know about the distance of this p from c 3. I know that, this distance is at most the sum of this distance plus the sum of this distance. But, what is this distance itself? This distance is at most, the radius of this cluster. This distance is also at most the radius of this cluster. So, that means that this distance is at most twice the radius of this cluster.

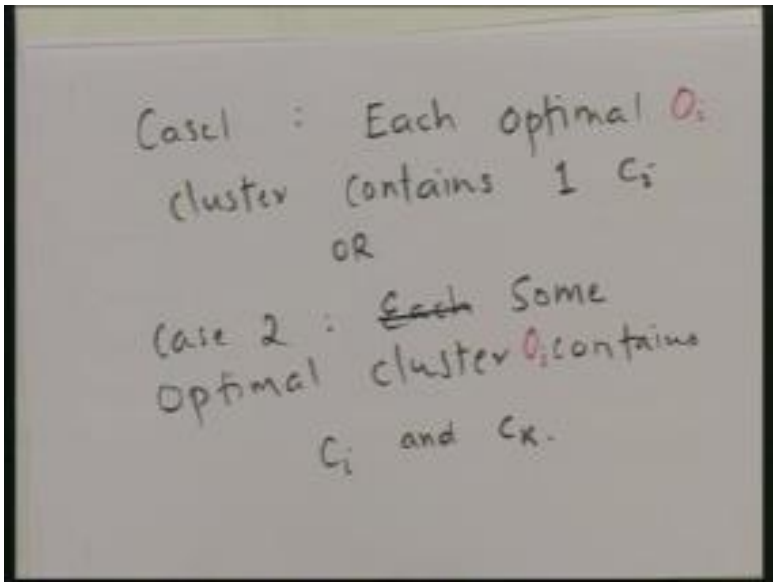
(Refer Slide Time: 47:17)



So, what is the important observation that we have made. We have established that, if P is a point in optimal cluster, c cluster O i capital O i. Then, the distance of P from c i is at most two times the radius of O i. But, this is nothing but, this is certainly at most 2 times the max radius of the optimal clustering. So, we are going to call this R sub OPT. So, this is 2 times R sub OPT. So, we have really proved what we wanted.

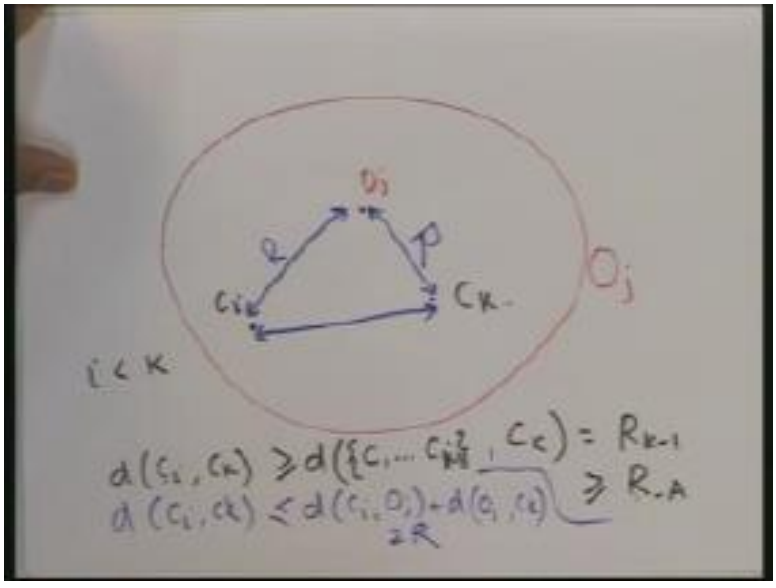
Surprising as it may seem, we have proved that this point is close to some center not necessarily the center with which it is associated, in the final clustering that our algorithm produces. It might be associated with some other center. But, its distance to this center c i is itself less than 2 times R OPT. And this is true for every point. And therefore, in this case we have in fact established the theorem which said that, which says that.

(Refer Slide Time: 48:49)



So, we have proved that the distance of the quality of the clustering, our quality of clustering produces clusters of radius at most twice the radius produced by the optimal cluster. So, now we come to case 2. So, case 2 says some optimal cluster O j contains c i and c k.

(Refer Slide Time: 49:16)



So, let us draw picture. So, here is my optimal cluster. Let me call it capital O j. Here is little o j, its center. And suppose that it contains cluster centers c i and c k which our algorithm picked. Well, what do we know about distance from c i to c k. So, I claim that

this is greater than or equal to the distance of c 1 to c i. And I am going to assume here without loss of generality, that i is less than k. So, the distance of this set c 1 to c i or rather c 1 to c k minus 1 with center c k.

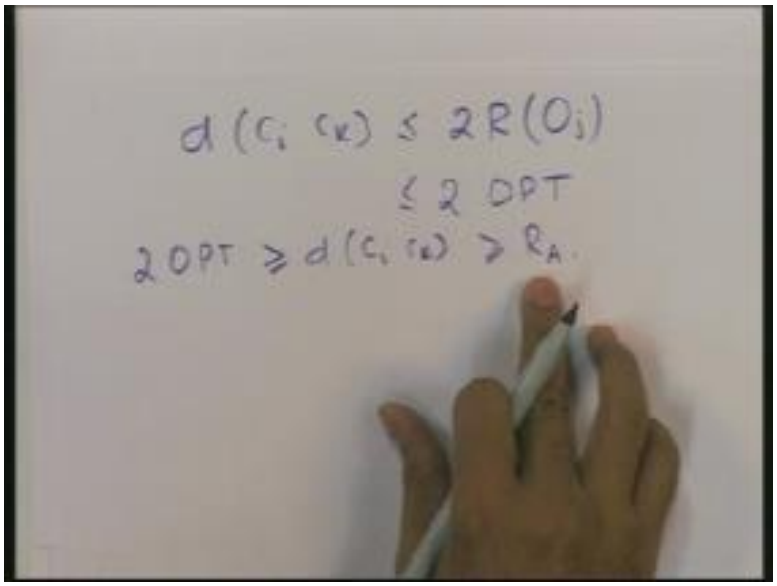
So, here I am only taking the distance from c i to c k. Here, I am allowing a large number of other points to come in. So, this distance can only be bigger than this distance. But, what is this distance. So, this distance from what we argued earlier is something quite is related to our clustering. So, we argued that R k plus 1 has to be less than R k, which is this. So, comparing these two we can argue that, this distance has to be bigger than.

So, this distance is nothing but, R k minus 1. But, this R k minus 1 is certainly bigger than or equal to the final radius, that the algorithm got. So, what we have argued is that the distance between these two clusters, if they happen to fall within the same optimal cluster. So, distance between any two clusters, any two cluster centers is going to be. The optimal is going to be bigger than the optimal radius produced by our algorithm.

Here we have not yet use the property that these two centers actually lie in this single optimal cluster. So, we will use that property now. So, what do we know about the two, these distances. So, we know that this distance, again by the triangle inequality is at most this distance plus this distance. So, what do we know? We know that, d c i c k is less than or equal to the distance from c i to o j and the distance from o j to c k.

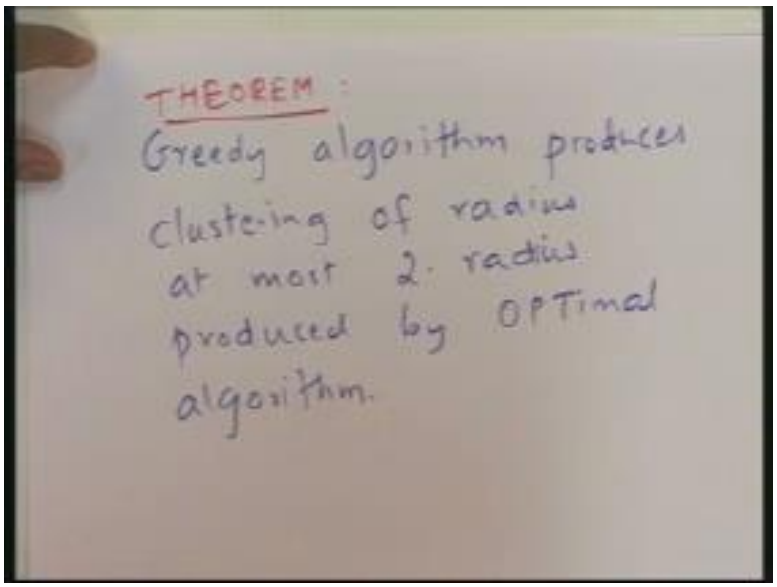
But, what is this tell us. So, this lies inside this cluster. And therefore, this is at most the radius of this cluster. This is at most the radius of this cluster. So, this is just 2 times the radius of this cluster. So, what have we proved?

(Refer Slide Time: 52:57)



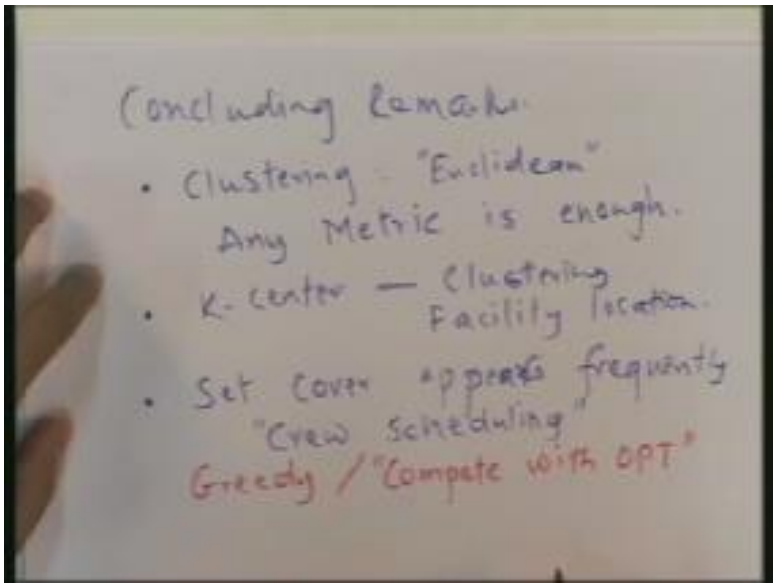
We have proved that, the distance from c i to c k is at most 2 times the radius of cluster o j. But, this is less than 2 times the optimal radius. So, together what have we argued? So, we have argued between these two things. That twice the optimal is greater than or equal to distance from c i to c k, which in turn is greater than or equal to the radius produced by the algorithm. So, even in this case we have argued that, the radius that is produced by the algorithm is at most twice the optimal radius. And that is what we argued in the first case as well.

(Refer Slide Time: 53:56)



So, in conclusion we can say that greedy algorithm produces clustering of radius at most 2 times radius produced by optimal algorithm. This is the main theorem of this entire exercise. So, let me now summarize and have some concluding remarks.

(Refer Slide Time: 54:56)



First, in the clustering example the distances I mentioned had to be Euclidean. But, we did not really use the Euclidean property. We just use the property that, the distances we just use the triangle inequality. So, essentially this means that any metric is enough. So, this clustering algorithm works not only for Euclidean distances but, anything which satisfies the metric properties. That is, it satisfies essentially the triangle inequality that will work.

So, instead of clustering the same t center problem or the k center, can be related not only to clustering but, it can also be related to some kind of a facility location problem. So, let you think about this. I would also like to observe that, set cover like problems also appear quite frequently, just as this clustering problem. Many scheduling problems and important problem amongst them is so called crew scheduling, is a variation on set cover.

And in general, the greedy algorithms and a style of analysis used over here which is which I would like to call, compete with the optimal also appears quite frequently. So, we have seen two styles of analysis, compete with the optimal and compete with and

compare to the lower bounds. So, between these two styles we should be able to design a fair number of approximation algorithms.

Thank you.