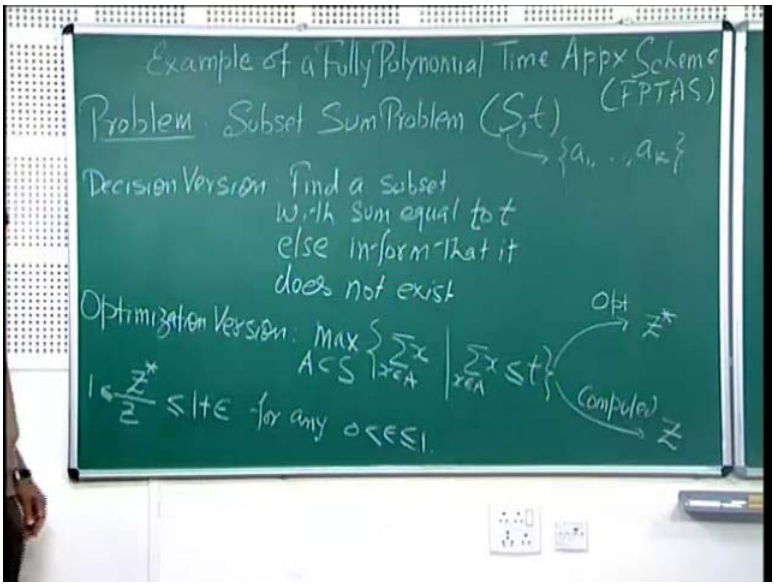
Computer Algorithms - 2
Prof. Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 34
Approximation Algo III

(Refer Slide Time: 00:20)



In today's lecture I will describe an example of a fully polynomial time approximation scheme or fptas. And today's problem is subset sum problem, this problem has two inputs a set of integers, which we could assume to be a 1 to a k. And another integer t, a subset sum refers to the sum of the integers of some subset of S. So let me describe the problem the decision version of this problem, the decision version says that find a subset of S, whose sum is equal to t and if no such subset exists then in from that it does not exist.

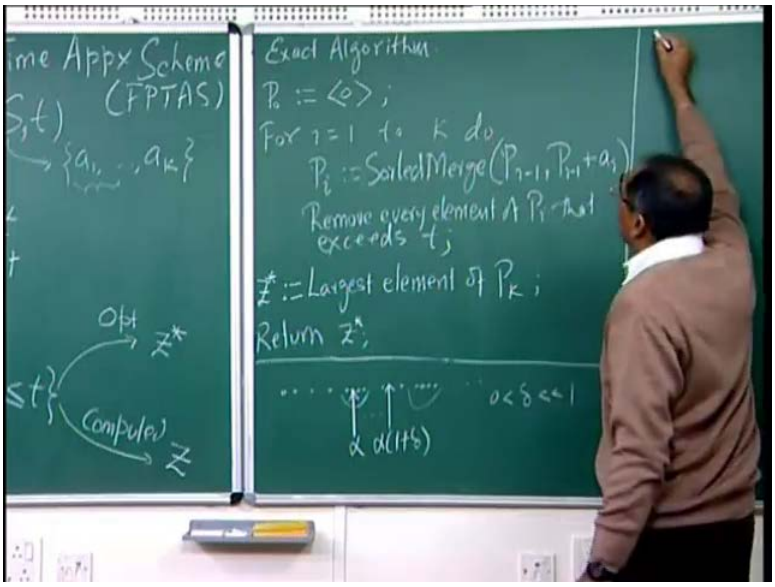
So, here is find a subset with sum equal to t else inform that it does not exist, but of course, we are doing approximation and we want an optimization version. So, in this case, we want to compute the largest possible subset sum not exceeding t. So, such a number will always exist of course, in most cases here you would not find a subset which is adding up to t, but here we are asking for the largest sum. So, we want to maximise a subset a over the subsets of S such that the sum x, x in a subject to the fact that this sum is less than or equal to t. So, this is the objective clearly if you find that the answer is

equal to t then the corresponding decision version is solved because you know that here is a subset that adds up to t. Otherwise this value will be always strictly less than t.

So, suppose the optimum value of this, optimum value is say denoted by z star that is the largest subset sum less than equal to t is z and computed value, value is z the computed value through the approximation scheme that we will discuss. Then we are going to show that z star divided by z, which we know clearly has to be greater than equal to 1. We will show that it can be bounded above by 1 plus epsilon 1 plus epsilon for any choice of epsilon, less than equal to 1, for any. This is what our goal is that is why we call it a scheme that we can compute the result arbitrarily close to the optimum.

So, now let us see how do we proceed to solve this problem what we will do is first we will describe an exact algorithm, which obviously will take exponential times the reason being there are 2 to the power k subsets, if we generate each of them we compute each subset sum and then pick the largest we are going to have an exponentially large amount of time, but then we will modify it and device an approximation scheme out of it. So, let us begin with an exact algorithm.

(Refer Slide Time: 05:45)



We begin with defining a sequence containing just the 0, so we will view these P i s as sequences or lists and in the beginning it contains just 0 in it. One remark any of these numbers is 0 then we can ignore it because it does not really add any value any number

is exceeds t in that case we can ignore it, because that will definitely give you a subset exceeding t. So, every number has to be strictly between 0 and t.

Now, we are going to generate for i equal to 1 to k all the subset sum in the following fashion, what we will do is we will generate this incrementally that is to say P i will have the list P i will have subsets subset sums of all the integers from a 1 to a i. So, P i will contain everything that P i minus 1 contains, and then we will also add to them a i. So, this is since we are maintaining a list for the purpose of keeping this as a list is because we will essentially in implementation this is fairly straight forward. We will or rather sorted merge, so we will keep this as a sorted list so to emphasize that sorted merge, we have l or rather P i minus 1, P i minus 1 we are going to merge these two lists P i minus 1 and P i minus 1 plus a i what this refers to is that we add a i to each member of P i minus 1 and generate another list of the same size as P i minus 1 and merge these two lists.

Now, clearly what you notice is that if this contains the sum of every subset of first i minus 1 integers, this will contain then the sums of every subset formed by a 1 through a i. Now, at this point the only thing we have to do is eliminate those numbers, which exceed t. So, we will simply put that remark remove every element of P i that exceeds t. So, we clean up and now ready to go for the next iteration.

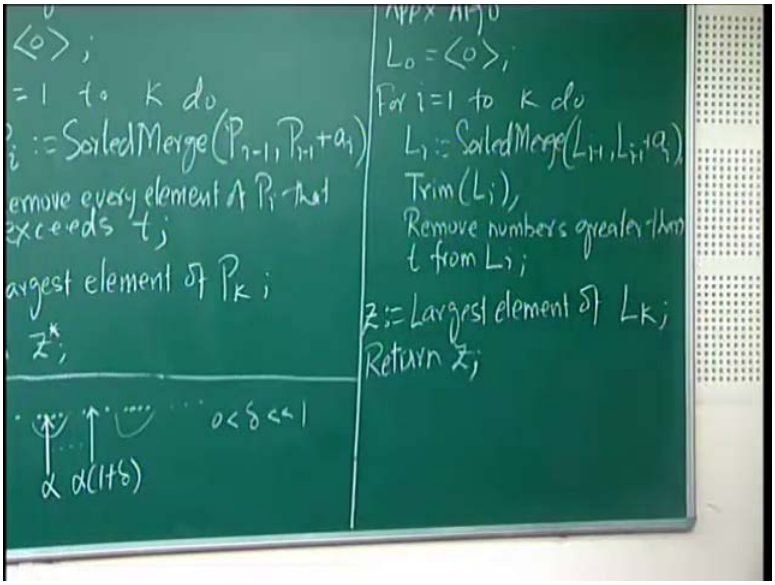
At this point when we have P k which means we have got a list containing every subset of these integers, which do not exceed t. We are going to take the largest among them, the largest element of P k it is fairly straightforward and returns z star. So, this is simple algorithm to exactly compute the answered to the problem. So what do we do to make it an approximation algorithm, which runs in polynomial time. Clearly note that here we have generated all the truth for k actually not explicitly generated the subsets, but we generated the sums of every subset all the 2 power k. So, this obviously takes exponential time.

So, the way we can improve this in terms of time is that we maintain smaller lists, so in order to reduce the size of the list, this is the way we will proceed suppose the numbers in the list are over the real line, spread like this. In other words there are clusters of subset sums in list P i somewhere, we will try to reduce or replace these by fewer numbers. So, you know there are too many numbers here over here and here. So, what we will do is that suppose you have a number alpha at any in the list at some position

you have a subset sum equal to alpha. Then we will not allow any number in between alpha and alpha times one plus delta.

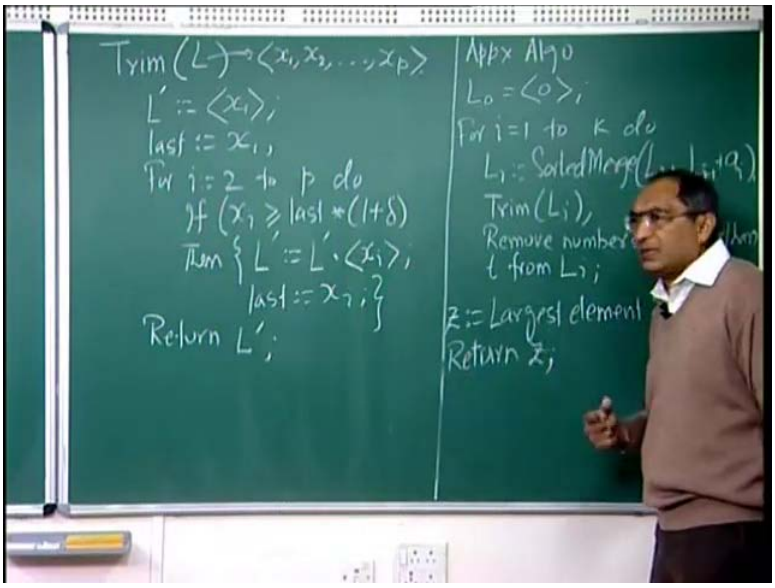
Now, here delta will be chosen suitably later on we will see how to choose it, but it is a number which is positive and much smaller than one. So, we will not keep anything in between here as a result what will happen is that the ratio of jth number and j minus first number will be greater than or equal to one plus delta, that we will control the population size of this list. So, we will essentially copy this, but on top of it we will add one more condition here that trim this list, so that we do not have numbers which are very close by.

(Refer Slide Time: 12:35)



So, let me now repeat this approximation algorithm is L 0 again starts with a 0 for i equal to 1 to k, L i is sorted merge of L i minus 1 and L i minus 1 plus a i. Trim L i now this is the step that will basically do this and then remove the larger numbers, everything greater than t. Remove numbers greater than t from L i, so the only difference is this step, the rest is the same and now we will determine the largest element of L k and return it. So, this is it this is the only modification we have made in the algorithm.

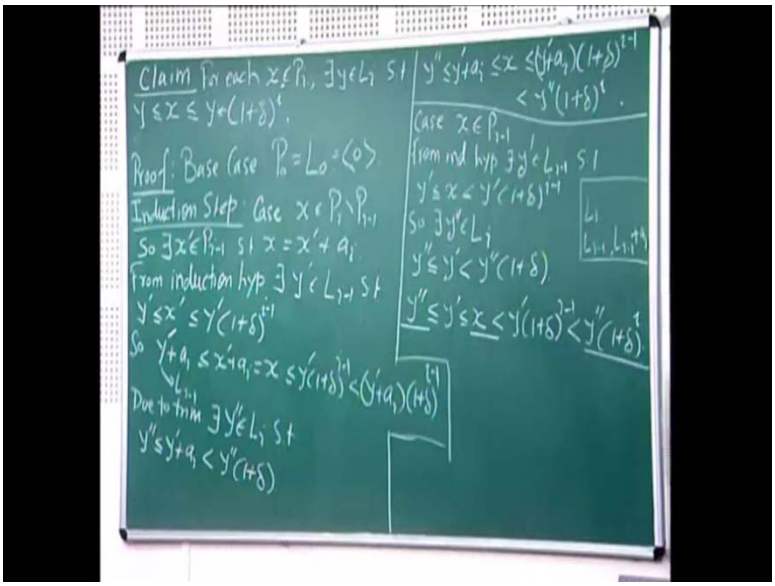
(Refer Slide Time: 14:24)



So, now let us write down what trimmed as trim of a list L let us say this list x 1, x 2 x p and then we start with an L prime equal to the list containing only the first integer of L. Then for i equal to 2 to p, there is one more thing I want to say that is the last integer we have added to the list. So, we begin with last equal to x 1 this is the last integer added to the list. And then for i equal to 2 to p, then we see if x i is greater than or equal to maybe last times one plus delta. So, then it is far enough then we are going to add it then L prime is uptended with we add this x i to the list and set last equal to x i that is all.

So, we have removed numbers we have very close and now we can just return L prime, return L prime this is what trimmed value. So, the question now is finally, how good is the number finally, we have computed namely z, so let us start the analysis. And first of all we will try to prove that by suitable choice of delta, we will find z star divided by z bounded by 1 plus L.

(Refer Slide Time: 16:59)



So, our first claim and the main result in the analysis is this which says for each x in P i there exists a y in L i such that y, y is less than or equal to x and that is less than equal to y times 1 plus delta power i, this is trying to show that we do have a number. For every number of the original list there is a number which is close by in the computed list L i. So, let us try to prove this we will prove using induction, so the base case is trivial base case is that the actually two lists are same is that P 0 is equal L 0. So, there is nothing to prove both of them are equal to the list containing only 0.

Now, induction step so let us assume that the claim holds for i minus 1 and we want to prove for i. Let us take two cases sub let me say sub case or case rather case that x that we have chosen from P i belongs to p i, but was new number p i was not present in p i minus 1. That means this is a number which is a result of a number in p i minus 1 plus a i. So there exists in x prime in P i minus 1 such that x is x prime plus a i, so we have this now since x prime is in P i minus one we will exploit the induction hypothesis.

So, from induction hypothesis there exists a y prime in L i minus 1 such that L sorry y prime is less than equal to x prime, less than equal to y prime 1 plus delta power i minus one because you are working with list i minus 1 and p i minus 1. So, we have this, so y prime plus a i is less than equal to x prime plus a i which is same as x i, x rather and that in turn is less than equal to y prime 1 plus delta power i minus 1. That is less than y

prime plus a i times one plus delta power i minus 1. So, this is simply adding a i and this is a trivial inequality, now notice that this number y prime belonged to L i minus 1.

So, when we were building L i this must have been this number y prime plus a must have been created and maybe it was eliminated for being too close to some already existing number. So, due to trim then exists a y double prime which finally, stays in L i such that y double prime is less than or equal to y prime plus a i and that in turn is less than y double prime 1 plus delta. So, this is guaranteed the way trim works, we must have it is possible that y double prime is y prime plus a i itself.

So, let us relate y double prime with x using this inequality I have y double prime, which is less than equal to y prime plus a i that this is the inequality that we have used. That in turn is less than equal to x from this inequality, which is less than or equal to y prime plus a i into 1 plus delta power i minus 1. And now I used this part of the inequality that y prime plus a i is bounded above by y prime y double prime 1 plus delta and 1 plus delta power i minus 1 makes it 1 plus delta power i. So, this is exactly what we wanted y double prime is less than equal to x, which is less than equal to y double prime 1 plus delta power i this is the case.

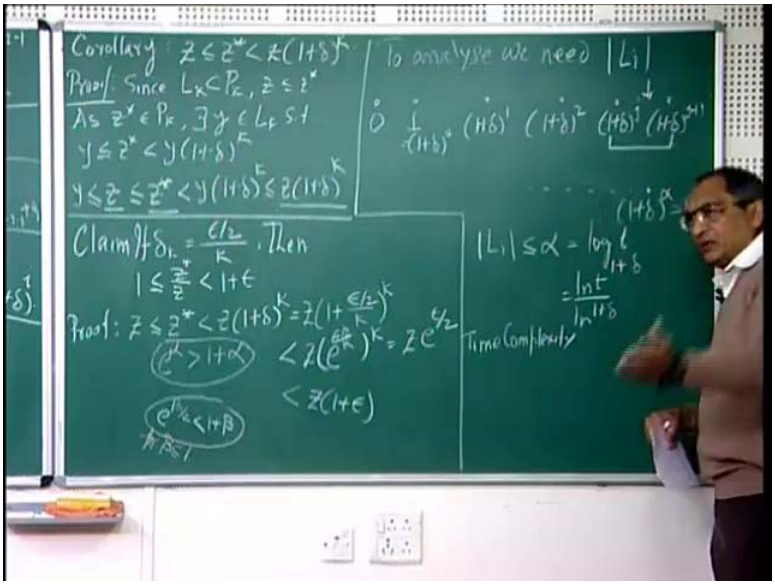
Now, let me take the second case, the case of x already existing in P i minus 1. So, x belongs to P i minus 1, this is the second part of this. So, once again from hypothesis, from induction hypothesis there exists a y prime in L i minus 1, such that y prime is less than or equal to x which is less than or equal to y prime 1 plus delta power i minus 1, this we get from the given fact, which holds for i minus 1, one at least.

Now, what do we do from y prime, now note that y prime was in L i minus 1. So, in the formation of L i, so recall L i which was found by L i minus 1 and L i minus 1 plus a i both and then we trimmed it. So, y prime was present here, so it might have gone into L i or there was a number too close to y prime. So, there exists a y double prime in L i such that it is less than or equal to y prime, which is strictly less than equal to y double prime one plus delta. So, we have this guarantee from the trimmed algorithm.

So, let us combine this and this inequality, we have y double prime less than or equal to y prime, which is less than equal to x and that in turn is less than y prime 1 plus delta power i minus 1. This inequality further bounds it as y double prime 1 plus delta power i.

Once again we have this inequality that we wanted to prove over here, so that establishes the claim. So now once we have the main result let us see what are the consequences.

(Refer Slide Time: 26:58)



The corollary of this result one of the corollary is rather is that z is less than or equal to z star which is less than equal to 1 plus delta power k, this is easy to prove. The first claim is trivial because this is the largest element of L k, this is the largest element of P k and L k is a subset of P k. So, clearly this could be larger than this, now since L k is contained in P k sorry rather L k contained in P k, z has to be less than equal to z star this is trivial. Now, from our result that we have derived as z star belongs to P k there exists a y in L k such that y is less than equal to z star less than equal to y 1 plus delta k, and z is the largest element of L k.

So, we have y less than or equal to z we already have argued z is less than equal to z star that is less than equal to y 1 plus delta power k. And again y is less than equal to z this inequality says that it is z 1 plus delta power k. So, this is this establishes what we wanted to prove, well it indicates that z star is not too far from z the gap can be at most one plus delta power k.

So, now it is a matter of choosing right delta so that we can establish our objective. Namely z star by z is bounded by 1 plus epsilon, so after this let us choose delta k equal

to epsilon by 2 by k epsilon by 2 k. And then we are going to use some inequalities, so our claim is or I would say let so we have a claim which says if this is this, then we have what we wanted one less than equal to z by z star by z, which is less than one plus epsilon. So, let us prove this.

Well what we have from the above result z is less than or equal to z star that is bounded by z 1 plus delta power k yes power k, that we will now claim is less than let me first of all substitute this, which is z 1 plus epsilon by 2 by k power k this is epsilon. Now, let us use this inequality e to the power alpha is greater than 1 plus alpha that the expansion of e power alpha is one plus alpha plus alpha square by 2 factorial alpha cube, three factorial and so on. So, this is there let us use this and take this as alpha so based on that we are getting let me just this is z, e to the power epsilon by 2 by k whole thing power k. So, I have replaced one plus epsilon by 2 k by e to the power epsilon by 2 by k which is same as z e to the power epsilon over 2.

Now, there is another inequality for number smaller than 1, e to the power beta by 2 is less than 1 plus beta for beta less than or equal to 1. So, this is another inequality which allows me to further bound this by z 1 plus epsilon, so that is what we wanted to establish. So, by choosing delta as epsilon by 2 k we have found a number z close enough to close enough to z star, so that establishes the correctness of the approximation. So, the question now is what is the time complexity, what is the space complexity? Well what you noticed in the algorithm is that we are actually scanning through the list, and we merge the list and its variant where we add i th integer. So, the entire time it takes is proportional to the length of the list in each iteration. So, what is the goal so to analyse we need the length of ith list this denotes the number of elements in the list alright.

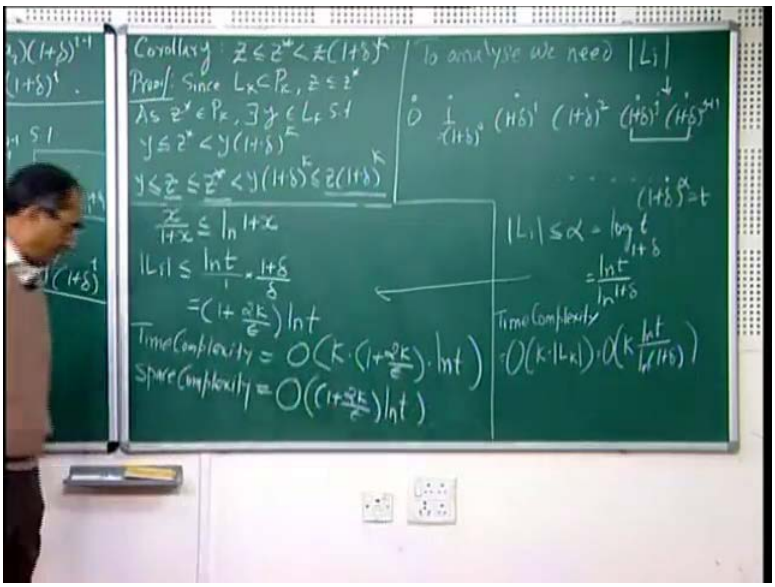
Now, let us try to look at this real line say this is 0, 1 which can be written as 1 plus delta power 0, then we have 1 plus delta power 1 and so on 1 plus delta power j say we go upto 1 plus delta power alpha, which is equal to t the largest number that we are interested in so the entire range from 0 to t is split into these intervals current. Now, suppose there is an integer in let us say in this interval, in the list L i then we know that the next integer has to be this times 1 plus delta, or larger.

Hence this cannot fall inside this interval, so the next number has to be outside this interval must be beyond this point clearly because that number times 1 plus delta has to

be greater than this. What we conclude is that inside the interval, we cannot have more than one number in the list. Hence the total number of elements in the list cannot be more than the number of such intervals, which is alpha. So, we can say that the size of list is less than or equal to alpha and that is nothing but you just take the log. So, let us just log of t base one plus delta that is the alpha which we will write as log natural t divided by log natural 1 plus delta.

This is important to notice that although this is log natural log base two of t is the size of t, which is one of the inputs. So, we are now dealing with a number of the order of the input size, which is important this is a constant. So, the time complexity must be of the order of k times this remember this, this number is independent of i. So, k times this number the reason is we are scanning through these lists in each iteration, there are k iterations is of the order k times mod of L i mod of L k rather which is the largest of them. And that is of the order k times log natural this log natural one plus delta.

(Refer Slide Time: 38:03)



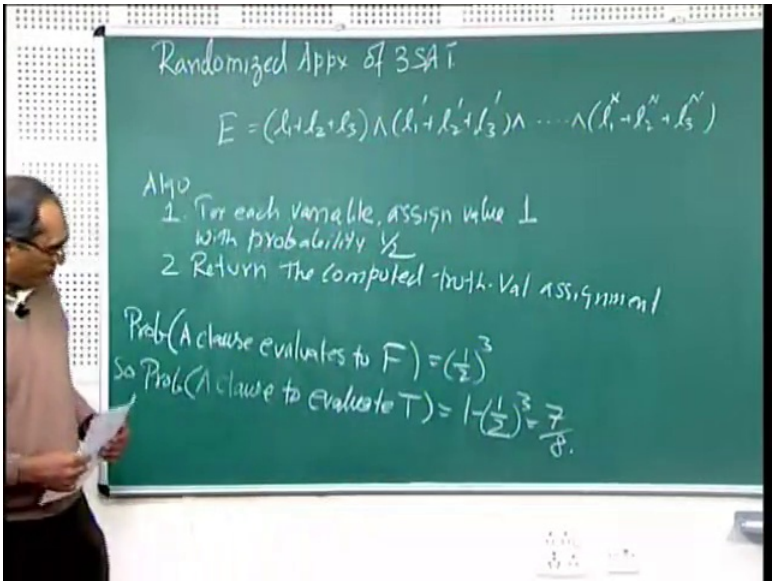
Now if we want to further simplify this we will use a another inequality which says that x divided by 1 plus x is less than equal to natural log of 1 plus x, this is easy to prove. If you set x equal to 1 sorry x equal to 0, this is 0 and this is 0 because log one is 0 and then you differentiate both sides. And then you notice that the derivative for both are positive, but this is less than this so this grows slower than this let me now use this inequality. So, we get L i to be less than equal to natural log t log 1 plus delta so that comes on the

lower side in the denominator, we get 1 plus delta divided by delta that is equal to 1 plus one over delta, delta was epsilon by 2 k 2 k by epsilon, natural t let me 2 k by epsilon.

So, time complexity is now equal to k times 1 plus 2 k by epsilon times natural log of t. Now, let us take a look at this k is one of the input parameters, log base two of t is the size of the largest integer all other integers are of course, less than equal to t. Other parameter here is one over epsilon, notice that one over epsilon is occurring by itself not as an exponent of some number. Hence this is a fully polynomial time approximation scheme fptas.

The space complexity well at any point in time we keep one list, and by constructing the next list we might need space which is about at most four times that. So, this is only of the order of the size of a line which is as we have seen nothing but 1 plus 2 k by epsilon log natural of t. So, that completes the argument that it is indeed a fptas complete. Next we want to give you a very small example of a randomised algorithm and its analysis.

(Refer Slide Time: 41:19)



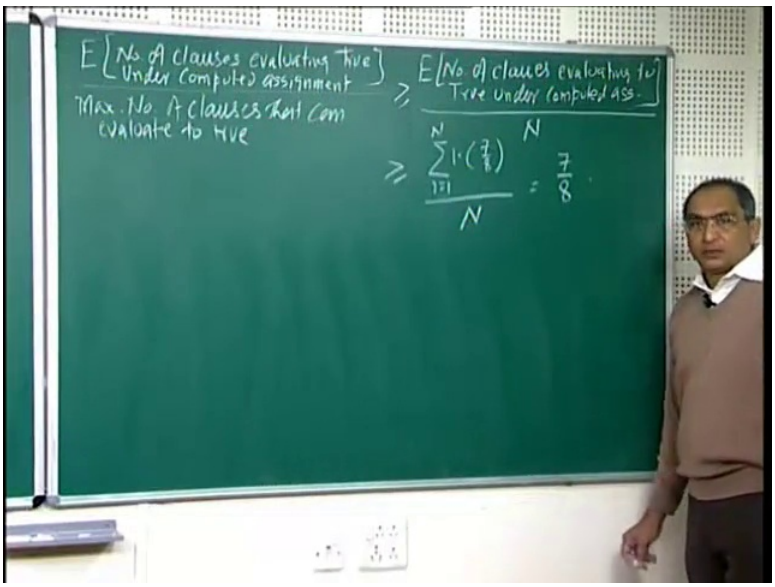
So, our next problem is a randomised approximation of three set, three set you are aware of the problem of satisfiability that you are given a Boolean expression in conjunctive normal form, and you want to check whether it is satisfiable or not. Three set means that that expression, let us say expression E is say it has got n clauses every clause has exactly three literals in it. So, given this the expression restricted in this fashion also it is hard to determine whether it is satisfiable or not in polynomial time.

So, our goal is to solve this problem, but we are going to work on its optimisation version. We want to compute an assignment of the variables, which satisfies maximum number of clauses, obviously that will solve three set that we are going to approximate that I am going to show that a very trivial random assignment will satisfy a very large number of clauses. So, let us do the this assignment so our algorithm is the first step is for each variable assign value one with probability half one half, and with one half probability this will be assigned 0. Please note that all these assignments are independent all variables are assigned independently.

Now in the first step we have generated a value assignment and of course, return this return the computed truth value assignment. So, the question is how good is this? Well let us take a look at any clause the clause has three literals, so the probability that first remember a literal is a variable or its negation. So, with probability one half this can be false, with probability one half this can be false and with one half L 3 can be false.

So, the probability that a clause evaluates to evaluates to false or 0 has to be one half power 3 because every literal must evaluate to 0, for this clause to be false. So, probability of a clause to evaluate true is 1 minus 1 over 2 cube that is 7 over 8. So, this clause evaluates to true with very high probability 7 by 8. So, now we want to compute the following remember this is a randomised algorithm, so we cannot have exact ratio of the evaluated number to optimum number, but we will have an expectation of this.

(Refer Slide Time: 46:15)



So, expected number of clauses evaluating true, true divided by maximum number of clauses that can evaluate to true, evaluate to true. This is the number we want to show is greater than equal to some number, note that this is less than or equal to no, this is greater than equal to expected number of clauses, number of clauses evaluating to t. This is under given under computed a truth value assignment to true under computed assignment. Note that the total number of clauses are N so that has to be an upper bound to this number.

And since the truth value can be computed because these are all separate clauses. So, each evaluates to one with probability 7 by 8, so this itself is greater than equal to sum the probability of it given clause evaluating to 1, we multiply that by 1. So, 1 times 7 by 8, i going from 1 to capital N divided by N which is nothing but 7 by 8. So, what we notice is that even with a simple truth value assignment computed completely randomly ensures that at least 7 by 8 clauses will be true on an average. So, that completes our discussion of the approximation algorithms in the series complete.