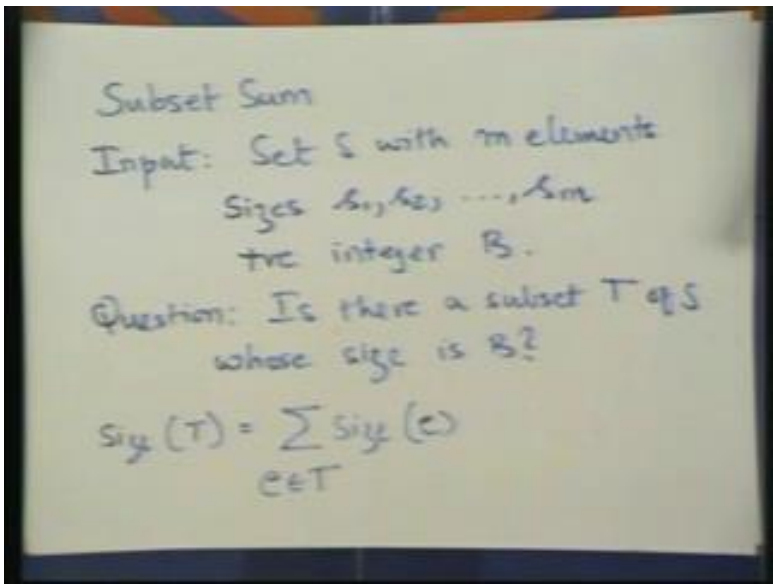
Design & Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 31
NP-Completeness – VI

(Refer Slide Time: 01:07)



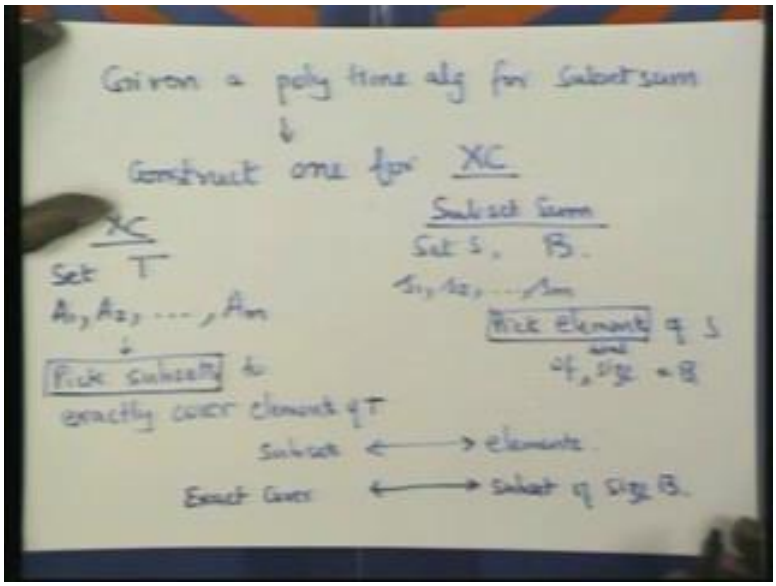
The next problem we consider is called subset sum. In this problem the input is a set S with m elements each element has a size. So, you have sizes s 1 s 2 and so on to s m each element has a size. Also you are given a positive integer B we will assume that these sizes are also positive integer. You have a set S with m elements each element has the size m and also part of the input is a positive integer B. The question that we want to answer is this is there a subset subset T of s whose size is B. So, this is a question that we want to answer. Now, what is the size of a subset? The size of a subset is just the sum of the sizes of the elements in the subset.

Let me define this, the size of T is nothing but the sum of the sizes of the elements e measure in T. So, take all the elements in in the subset T and sum of their sizes this will give you the size of a subset. Now, if you look at size if you look at subsets in s each of them will have some size right. Now, we would like to pick up the subset of s whose size is exactly B this is the problem this is the algorithmic problem that you want to solve and this is the next problem we show is this NP complete. As before let us first prove that

this problem is in NP this should not take as too much time what put the prover give that verifier.

So, that the verifier can look at an input to to the subset sum and conclude and convince himself that. In fact, it is a s input. Now, for a s input there there must exist a subset whose set whose size is exactly B right. So, the prover just gives this subset to the verifier the verifier takes this subset sums up the sizes verifies that this. In fact, is B and this verification happens very fast he just has to sum up the sizes of the subset. So, subsets are measured NP our next job is to show that subset sum is NP hard. And to do this we will use the exact cover, what does it mean that we going to use exact cover what we are going to do is this. So, we will assume that there is a polynomial time algorithm for subset sum if there is 1 such we will construct polynomial time algorithm for exact cover. So, this polynomial time algorithm for exact cover will take input for exact cover. Then if we convert it your input for subset sum somehow feed it into the subroutine for subset sum get an output back and then you know given answer for exact cover.

(Refer Slide Time: 04:55)



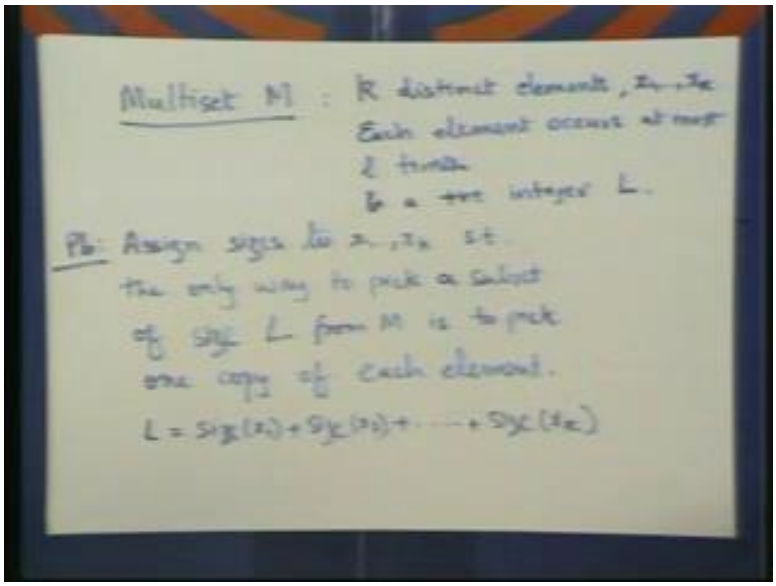
So, let us do this. So, we are given a polynomial time algorithm for subset sum what you want to do is construct a polynomial time algorithm construct 1 for exact cover frame model construct a polynomial time algorithm for exact cover. So, since we are constructing 1 for exact cover somehow we should convert input for exact cover into input for subset sum. So, what are the inputs for exact cover? So, let us see we have

exact cover on this side the input is the set T and subsets A 1 A 2 so on up to Am. What a subset sum? Subset sum has a set S that is this positive integer B. And then we have sizes s 1, s 2 up to sm that these 2 m's are the same is not a coincidence mean I have chosen this anyway let us just proceed.

Now, what do we want to do here in XC? We want to pick subsets to exactly cover elements of the set T you want to pick some of these. So, that this set P is covered exactly what do we want to do here? We want to pick elements of s of size of total size equal to B. Now, here we have to pick subsets it says picks up sets here it says pick elements. So, there must be a correspondence there must be a correspondence between subsets in the exact cover input and elements in the subset sum input. So, somehow subsets on 1 side and elements on the other side must correspond. So, that is the reason why m is I have chosen m to be the same because they will be the same. So, for each subset in the exact cover input there will be an element in the subsets sum input right. So, you want to pick subsets here and you want to pick elements here. So, subsets will correspond to elements.

So, let us write these subsets, it correspond to elements. What is the other thing? The other thing is here I want to exactly cover elements of T here I want to pick 1 of size B somehow exact cover on this side must translate to the subset of size B. This side I want to pick exact cover this side this side I want to pick a subset of size B the crucial thing of course, is what is B, what is B and what are the sizes s 1 s 2 s 3 up to sm? I know that with each element of s what it is a subset of exact cover that corresponds? How do I pick these sizes to make sure that an exact cover on 1 side corresponds to a subset of size B? So, this is what this is a problem that we would like to solve now. So, to solve this let us do a small switch. So, we will solve a different we look at a problem which is very related. And then we show at a solution to this problem can be transformed I mean is exactly what we are looking for a problem is this.

(Refer Slide Time: 09:36)

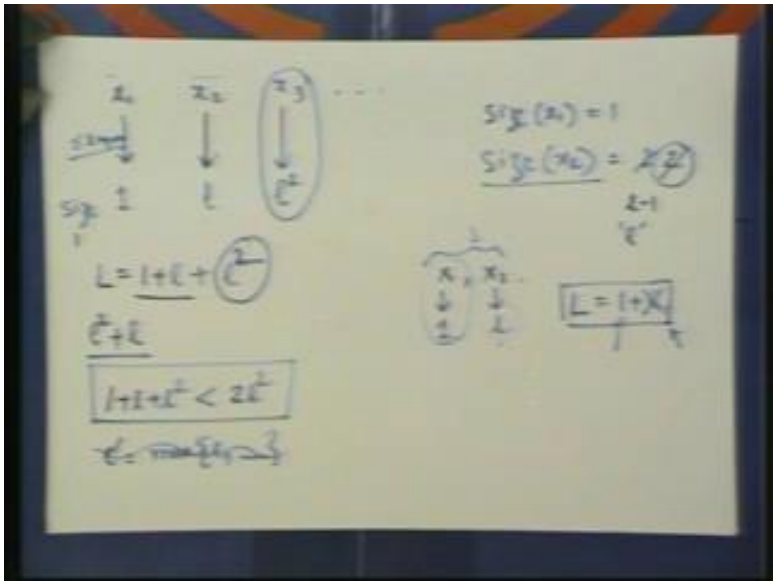


So, you have given a multi set M what is a multi set multi set is a is like set except that elements can be repeated you have multi set M the size k distinct elements let us say x 1 and so on up to xk each element occurs at most l times. Each element each of this can have at most l other copies of itself and we are given a positive integer l. Now, what do we do with this? What we want to do is assign. So, the problem is this. So, assign sizes to each distinct elements to x 1 up to xk such that the only way to pick a subset of size L from M is to pick 1 copy of each element. Let us see what this means. So, I you have a multi set there are k elements and there are l copies there could be l copies or perhaps less there is at least 1 copy and at most l copies of each element and there is a positive integer l this positive integer not given to you. So, let me clarify this is a positive integer you will have to fix also.

Now, what you want to do is give sizes to these elements x 1 to xk I want to give some sizes I also want to fix this positive integer l. So that when from this multi set if I pick any subset this subset can also be a multi set. So, if I remove any subset. So, that the size is exactly l then it must be the case that I have picked exactly 1 copy of each element 1 copy of x 1 1 copy of x 2 1 copy of x 3 and so on up to 1 copy of xk. Then we at least know that what l should be right l then should be size of x 1 plus size of x 2 dot dot dot up to size of xk this we know, because when I pick a subset of size l I must get 1 copy of each element nothing else. Now, these sizes are to be picked in such a way that any other subset of size l if I pick any other subset that must also contain 1 copy of each element if

I put 2 copies of any element the size of such subset cannot be l. So, this is the goal. So, how do we do this?

(Refer Slide Time: 13:34)



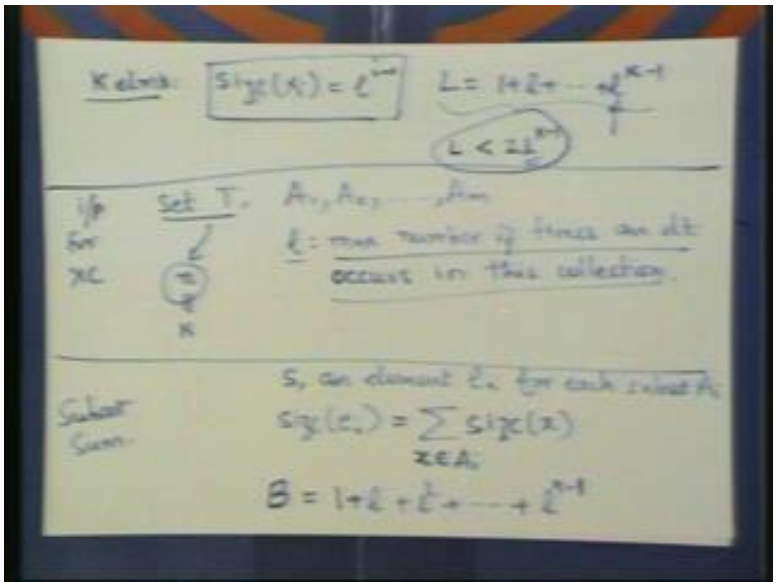
So, let us see. So, we have elements x 1 x 2 let us say x 3 and so on now let us just focus on the first 2 elements let us look at x 1 and x 2. So, without loss of general generality I can say that x 1 has size 1. So, size 1 we have at most l copies of each let us say this is the size 1. In fact, what I want is the sizes to let us say increasing x 1 is the smallest size x 2 next 1 x three. So, I am going to assign them in increasing order x 1 is of size 1 what can be the size of x 2. So, let us look at x 2. So, size of x 1 is 1 what is size of x 2 can it be 1 really not because if its size is 1 I can always replace a copy of x 2 with the copy of x 1 can it be 2 well not really, because if I could replace x 2 with 2 copies of x 1 then I am again failing you know. So, somehow the size should be such that I should not be able to replace x 2 with some copies of x 1 right. So, I guess the smallest we can take is let us say l minus 1 or l if I take l the way to do it the way I could replace x 2 is with all of x 1. Let just look at the 2 element case supposing I just had 2 elements. So, I just had x 1 and x 2 supposing x 1 I said had size 1 x 2 has size l and my capital L has to be 1 plus l.

Now, what can we do? Well, if I pick a subset of this size I cannot do it only with copies of x 1 right there are l of them. So, the total size is is l. So, I cannot pick something of this this size this is my set S using only x one. So, I will have to pick at least 1 copy of x

2. So, if I r pick 1 copy of x 2 then this l goes out of the picture and now well, I have 1 and the only way to fill this is is to pick 1 copy of x 1. So, if my sizes are 1 and l and capital L is 1 plus small l then the only way to pick a subset from this set. So, that the size is 1 plus l is to pick 1 copy of x 1 and 1 copy of x 2 there is no other way we can do this. So, let us see if we can generalize this right. So, supposing we have 3 elements x 1 x 2 and x 3 this well we will still retain it as l what do we think? What do you think we should put here well, there are l elements here each of size l that gives l squared. So, may be l squared is a good, good, good bound here. Of course, anything larger than l also will work here you can check that again anything larger than l will also work similarly something larger than l squared may also work. So, let us check this.

So, l is nothing but 1 plus l plus l squared these are this is when the set has s 3 elements. Now, if I just took x x 1 and x 2 the total size I will get is l times l which is l squared plus 1 times l which is l right. I can only get l plus l squared which means there must exist at least 1 copy of x 3 in in capital L. So, there must be exit exist 1 copy of x 3. There cannot exist 2 copies of x 3 that I can make sure by stating that 1 plus l plus l squared would be less less than 2 l squared and for reasonable l's this is this is satisfied. So, I can just pick I I can do this otherwise you pick l prime to be maximum of this l and some other constants. So, that this is satisfied we will just assume that this is satisfied. So, I have to pick 1 copy of x 3 now we are down to 2 elements 1 of size 1 the other of size l and it is the same argument. The only way I can I can do this is to pick 1 element of size l and the other element of size one. So, to pick an to pick a subset of size capital L the only the only way to do it is to pick 1 element of size l squared 1 of size l and 1 of size 1 which is 1 copy of each element and. In fact, a straightforward generalization of this actually works.

(Refer Slide Time: 19:36)



So, I have I have k elements I pick the size of xi to be l to the i and capital L to be 1 plus l and so on up to l to the k minus 1. Actually this is l to the i minus 1 x 1 was 1 x 2 was l and so on. So, size of xi is l to the i minus 1 and capital L is this. Then the only way to pick something of this size is to pick 1 element of each kind either you cannot do anyway. So, what is this got to do? What is this got to do with our set cover? So, before that it is just easier to prove that exactly 1 copy of the last element has to be for that we will assume that l is less than twice l to the k minus 1. So, for large enough l this is satisfied. So, this l we can take to be the maximum of the initial l that was given and an l which satisfies this. So, what is this got to do with our with exact cover? So, let us let us come back and do this. So, exact cover the input is a set T and I have subsets A 1 A 2 so on up to An these are the subsets. Now, I will choose l to be the maximum number of times an element occurs in this collection.

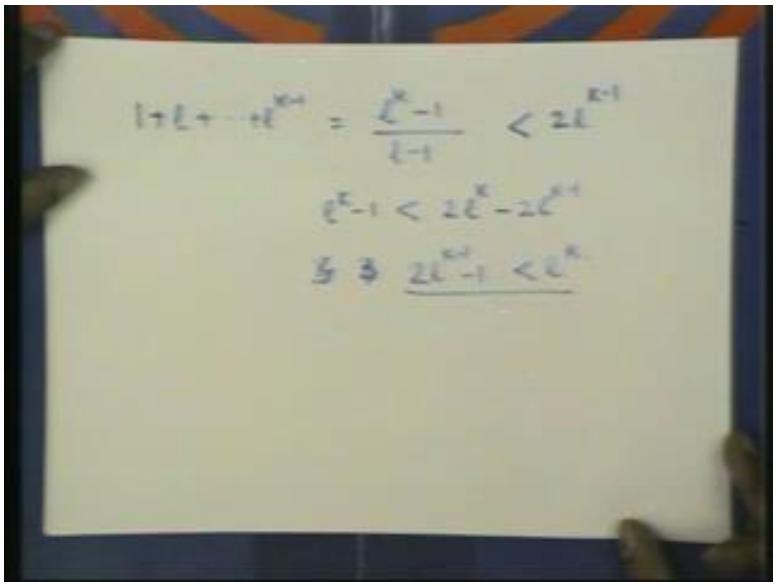
So, is the maximum number of times that an element occurs in this collection the number of the number of ele the number of elements is nothing but size of size of T which is n which was k for us there were k distinct elements here there are n distinct elements right. And with each collection there are a few elements and l is just the number of maximum number of times an element occurs in this collection good. So, then how do we fix the sizes? So, the question is how do we fix sizes of each of these these each subsets. So, what we do is we fix the size of these elements first I mean fix the size of elements like this size of each element is l to the i minus 1. So, we think of each element of T and for

each element of T for the ith element I pick the size as this what is the size of a subset this is what we really want which is going to be our input. The size of the subset is just the sum of the sizes of these elements right.

So, subset sum so we want to transform this to subset sum. So, we need a set S and an element ei or each subset ki for each of these subsets there is an element ki. Now, the size of ei is nothing but the size sum of the sizes. So, we will think of first fixing sizes of elements of T that is fixed like this. The size of the ith element is just l to the i minus 1. So, now, I can fix size of ei for each x in Ai remember that x is an element of T I just add the size of x. So, if it is the let us say jth element of T its size is l to the j minus 1 right if it is jth element in T in T it is nothing the size is l to the j minus 1 L.

Then is the other thing or B is nothing but our l which is which is 1 plus l plus l squared and so on up to l to the n minus 1. Now, if this set p has an exact cover supposing this set T has an exact cover then if I look at those subsets corresponding to those subsets. I will have elements in the subset sum if I sum sum up these sizes it will be nothing but 1 plus l plus l squared plus so on up to l to the n minus 1. And for the reverse direction pick any subset of size 1 plus l plus and so on up to l to n minus 1 the only way to get something of this size is to pick 1 copy of each element this this ends a discussion of of sub set sum.

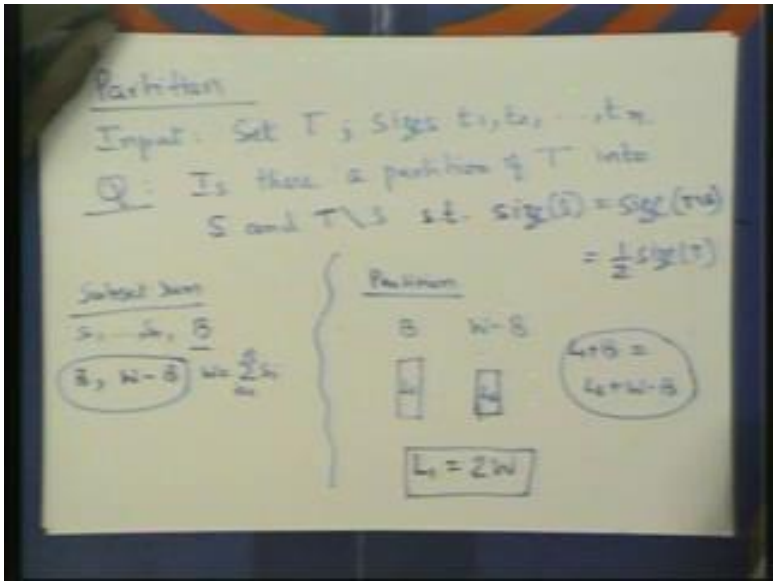
(Refer Slide Time: 26:24)



So, let me just do 1 small calculation just in case there was a confusion. So, what is 1 plus l and so on up to l to the k minus 1 this is nothing l to the k minus 1 up on l minus 1.

So, instead of summing this up like this you can you can sort of you can calculate it fairly easily. So, this proves that. So, calculating the input is easy and so this proves that subset sum is NP complete I think I mention that we need to choose l. So, that. So, that this is is less than twice l to the k minus 1. So, that can also be done. In fact, you can check that for more most ones this will be true. So, just check that this is true because we just multiply it out you get that l to the k minus 1 is less than twice I just multiply these 2 right. So, twice l to the k minus twice l to the k minus 1 so I will need l thrice. So, I have 3 sorry I have 2 l to the k minus 1 minus 1 should be less than l to the k. So, you can check that for reasonable values of k this is true for l 3 or 4 it is true I can just take some l which is larger than that. So, that was 1 thing which was which I am not completely cover it whose that subset sum is is NP complete.

(Refer Slide Time: 28:55)



And the last problem we have in this series is called partition it somewhat similar to to subset sum. So, the input so it is called partition the input is a set T sizes T 1 T 2 up to tn. The question we would like to ask is is there a partition of T into subsets let us say s and T minus s. So, partition of T into 2 parts which is s and T minus s such that size of s equal size of a T minus s. So, this should be exactly half size of T. So, can you partition it in 2 parts both of which have exactly equal size again the size of the subset is nothing but sum of the sizes of the elements in this subset. So, can you partition these into 2 parts so that size of both are equal. Firstly, the total size must be even if let us say all sizes are

integers positive integers then the sizes for instance it must be. So, let us assume that these things are true.

Now, this problem is also in NP clearly because if the answer is yes then the proof that the prover gives the verifier is just this partition for each element he tells which part it belongs right. Now, a verifier takes this partition he verifies that the sum of the sizes of elements in the first partition exactly equals sum of the sizes of the elements in the second partition. So, this problem is in NP we would like to prove that this is this problem is NP hard and for most problems where sizes are involved we would like to use subset sum. So, what we show is if there is a polynomial time algorithm for partition there is 1 for subset sum that is what we do. So, let us see. So, there is an algorithm for partition for subset sum the input is are some sizes s 1 to sn and I have B I want to know if there is a subset of size B. So, here is partition here I want to know if there is a partition into 2 equal parts what is this partition say this means that there must be one of size B and 1 of size.

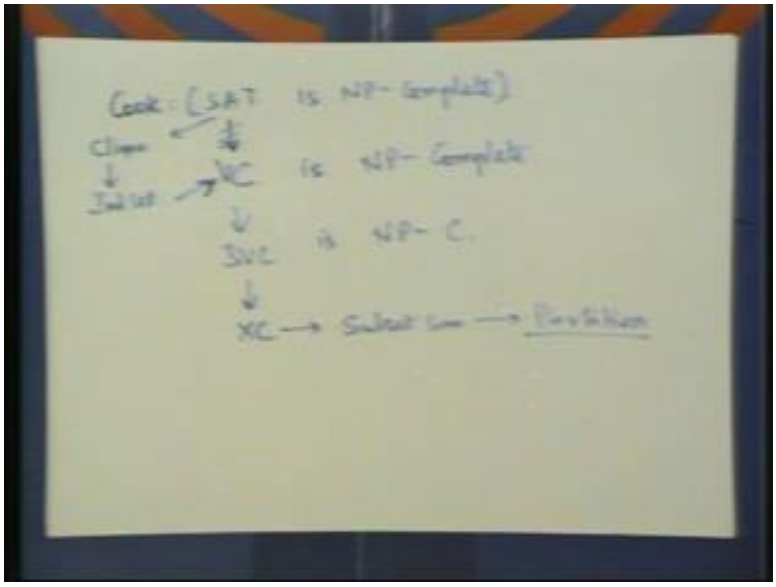
Let us say w minus B where w is nothing but sum of the sizes. So, this asks if there is a partition of this form this asks if there is a partition where they both equal. So, how do we sort of use this to solve subset sum? Well, here is B what I am looking for is B and let us say w minus B what I do is I add say 2 extra elements 1 of size l 1 1 of size l 2. So, that l 1 plus B equals l 2 plus w minus B. Supposing I do this. So, I take this input, input for subset sum I add 2 more elements 1 of size l 1 and 1 of size l 2 right. Now, I ask is there a part and l 1 and l 2 satisfy this l 1 plus B exactly equals l 2 plus w minus B. Now, I ask is there a partition into 2 equal parts. So, I look at this supposing there is a partition into 2 equal parts supposing. So, supposing l 1 and l 2 land up in different partitions you partition into 2 eq 2 equal parts if l 1 and l 2 are in different partitions.

Then there is a yes there is an answer to the subset sum problem right there is a yes answer to the subset sum problem. So, if there is a yes answer to the subset sum problem then there is clearly an answer to the partition right. I take an answer to the subset sum problem put l 1 along with B and w minus l 2 in the sorry an l 2 along with w minus B then I know there is an answer. Now, if I look at an answer for partition can I construct an answer for subset sum. Well I can if l 1 and l 2 land up in different partitions can I make can I force l 1 and l 2 to land up in different partitions the answer is yes just take l

1 and l 2 large enough. For instance if I take l 1 let us say equal to twice w if I take l 1 to be twice w then you can check that l 1 and l 2 will land up in different partitions.

Because if l 1 and l 2 both of them land up in the same partition on the other side the maximum size that can happen is w right which is the sum of all sizes the sum of the sizes of all elements in the set set S. So, if I take l 1 to be twice w then l 1 and l 2 this is elements which have sizes l 1 and l 2 must land land up in 2 different 2 different partitions. And the rest of the elements when I look at the rest of the elements the ones with l 1 must sum to be B the ones in l 2 must sum to w minus B and this is the answer that we are looking for. So, this is then a reduction that shows that partition is is NP complete. So, this let me quickly revise what we have done. So, far I have not written a formal proof that partition is NP complete, but I hope that you can fill in the detail. So, let us revise what we have done so far.

(Refer Slide Time: 37:05)



So, Cook told us that SAT is NP complete and then we use this to prove that VC is NP complete then we showed that 3 VC is NP complete. Getting from SAT to VC we first showed that clique is NP complete and then independent set and then VC. So, it it is not a direct sort of reduction we first showed that clique was NP complete. Then independence set then vertex cover then 3 three VC then we showed that XC is NP complete then we show subset sum finally, partition. So, let us now look at a problem that we sorted out with. So, remember that you are joined a company and your boss had

given a problem where you are given jobs with with time execution time for each job and he wanted to schedule them on let us say 2 processes. So, that the time that last job finishes was smallest. Supposing you could have supposing just was possible mean you actually had an efficient algorithm. Let us say polynomial time algorithm for this problem then you know observe that you can solve partition. The reason is you just take take the input for partition.

The same input you feed into your your algorithm which did the split in to 2 processors the sizes are just given as running time. Now, you look at the partition that your algorithm proof is gives you back if they are both of the same size then there exist a partition of the original set into 2 equal halves right where the sizes are equal if not there is not. So, the problem that you solve is more general than partition. So, if you could have solve that all that problem that your boss gave you then you could have solve partition and by this sequence of reductions that we that we did you could have you would actually have constructed an algorithm for all of NP that seems highly unlikely. So, hopefully when you give this the, this set of arguments or this argument to your boss will be both reasonable and intelligent with intelligent enough to understand and may be give you a raise instead of firing.

Thank you. This finishes this module on NP complete.