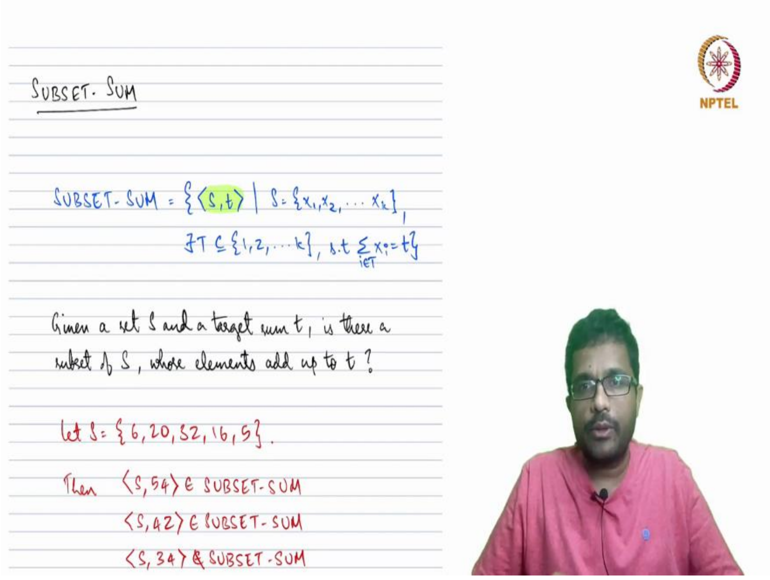
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
SUBSET-SUM is NP-Complete

(Refer Slide Time: 0:16)



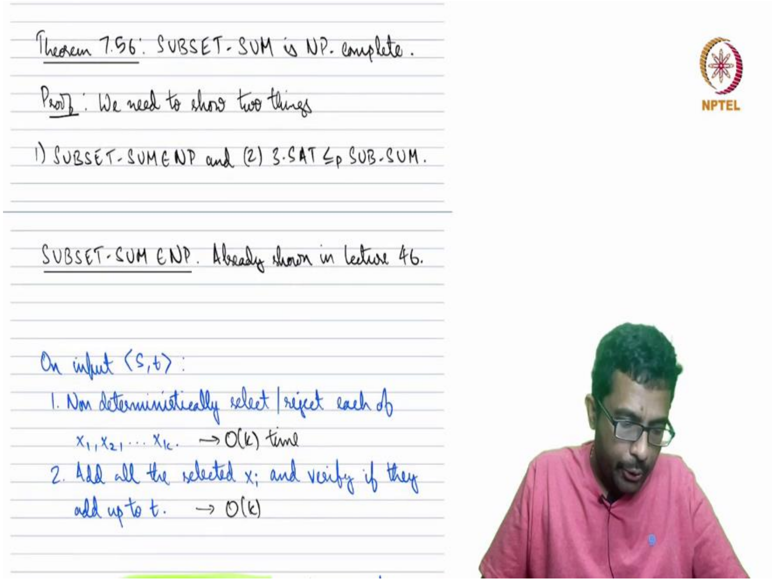
SUBSET-SUM

SUBSET-SUM = $\{\langle S,t\rangle \mid S=\{x_1,x_2,\dots x_k\},$
$\exists T \subseteq \{1,2,\dots k\}, \text{ s.t } \sum_{i\in T} x_i = t\}$

Given a set S and a target sum t, is there a subset of S, whose elements add up to t?

let $S=\{6,20,32,16,5\}$.

Then $\langle S,54\rangle \in$ SUBSET-SUM
$\langle S,42\rangle \in$ SUBSET-SUM
$\langle S,34\rangle \notin$ SUBSET-SUM



Theorem 7.56: SUBSET-SUM is NP-complete.

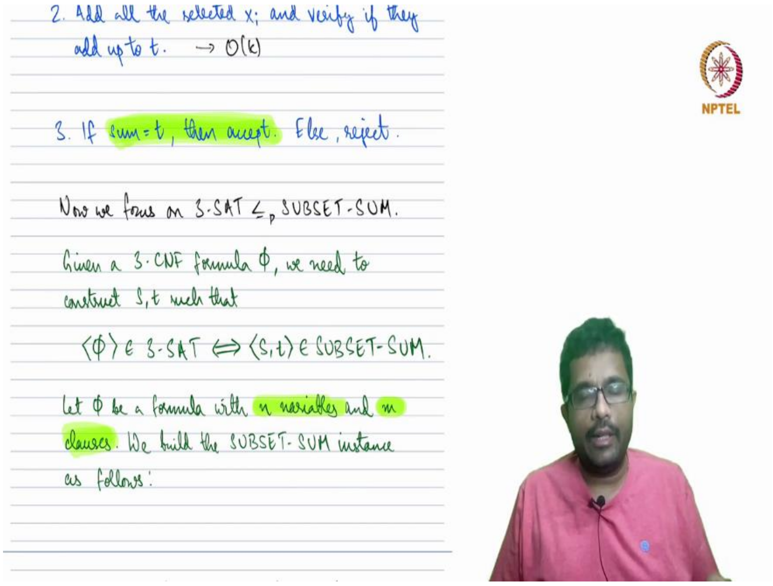
Proof: We need to show two things

1) SUBSET-SUM $\in$ NP and (2) 3-SAT $\leq_p$ SUB-SUM.

SUBSET-SUM $\in$ NP. Already shown in lecture 46.

On input $\langle S,t\rangle$:

1. Non deterministically select / reject each of $x_1,x_2,\dots x_k$. $\rightarrow O(k)$ time
2. Add all the selected x_i and verify if they add up to t. $\rightarrow O(k)$



2. Add all the selected x_i and verify if they add up to t. $\rightarrow O(k)$

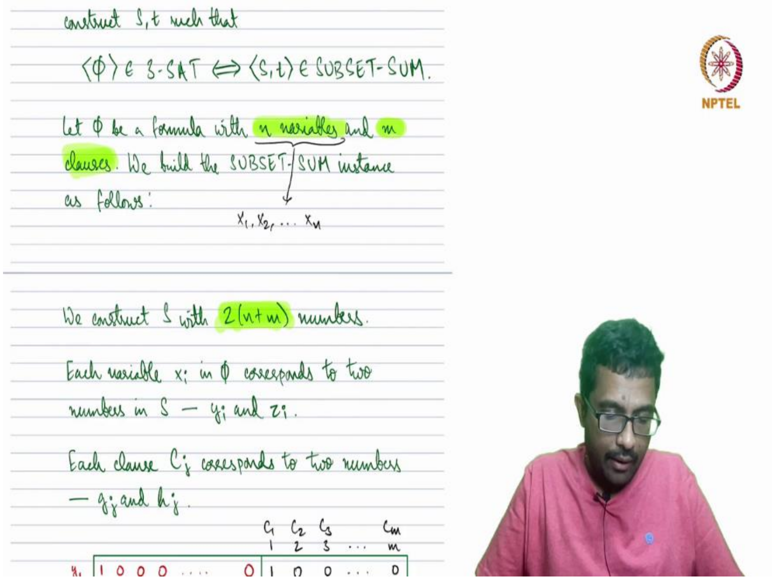
3. If sum = t, then accept. Else, reject.

Now we focus on 3-SAT $\leq_p$ SUBSET-SUM.

Given a 3-CNF formula ϕ, we need to construct S, t such that

$\langle \phi \rangle \in$ 3-SAT $\Leftrightarrow$ $\langle S, t \rangle \in$ SUBSET-SUM.

Let ϕ be a formula with n variables and m clauses. We build the SUBSET-SUM instance as follows:



construct S, t such that

$\langle \phi \rangle \in$ 3-SAT $\Leftrightarrow$ $\langle S, t \rangle \in$ SUBSET-SUM.

Let ϕ be a formula with $\underbrace{\text{n variables}}_{x_1, x_2, \dots x_n}$ and m clauses. We build the SUBSET-SUM instance as follows:

We construct S with 2(n+m) numbers.

Each variable x_i in ϕ corresponds to two numbers in S — y_i and z_i.

Each clause C_j corresponds to two numbers — g_j and h_j.

$$\begin{array}{c|cccc c|ccccc|}
 & & & & & & C_1 & C_2 & C_3 & & C_m \\
 & & & & & & 1 & 2 & 3 & \dots & m \\
\hline
y_1 & 1 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0
\end{array}$$

Hello and welcome to lecture 54 of the course Theory of Computation. In the previous lectures, we have seen a few NP-complete languages. We first saw that SAT, 3-SAT, and CNF SAT were NP-complete, which are based on Boolean formulas. Then we saw that clique was NP-complete, vertex cover was NP-complete, and Hamiltonian path was NP-complete; these problems were focused on certain types of structures in graphs.

In this lecture, we are going to see another problem that is NP-complete, which is subset sum. As you will see, this problem has a different flavor compared to the other two types. One was Boolean formula-based, and one was graph-based. This is a different type of problem. In fact, we have already seen this problem when we introduced NP, but let us go over it again.

Subset sum is the problem of, given a set S and a number t (a set S of numbers and a target number t), determining if the set S has a subset that sums to the target sum. For example, if the set S is {6, 20, 32, 16, and 5}, S and 54 is in subset sum because there is a subset that sums to 54. The subset is 32, 16, and 6. Another example is S and 42, which is also a member of subset sum because 6, 20, and 16 add up to 42. However, S and 34 is not a member of subset sum because there is no subset that adds up to 34.

The question is: Is there a subset that adds up to the given value? There is no subset that adds up to 1. So, given a set S and a target sum t, is there a subset of S that sums to t? As you see, it is a completely different type of problem. There are no graphs or Boolean formulas involved. All you have is a set S and a number t. Is there a subset of the set S that adds up to the number t? This problem is NP-complete.

By following the approach we used in previous lectures, we need to do two things: show that this problem is in NP and show that a known NP-complete language reduces to subset sum. Showing that it is in NP is something we have already done in lecture 46. We do the standard guess and verify method. We non-deterministically select a subset of S by picking or not picking each element. Let's say S has k elements. We pick or not pick each element, get a subset of S, and then verify whether the selected subset adds up to t. This is straightforward.

We use non-deterministic guessing followed by verification. If the selected subset adds up to t, we accept; otherwise, we reject. If there is a subset, it will accept, and if there is no subset, it will reject. This is a valid non-deterministic algorithm and runs in polynomial time. Hence, subset sum is in NP.

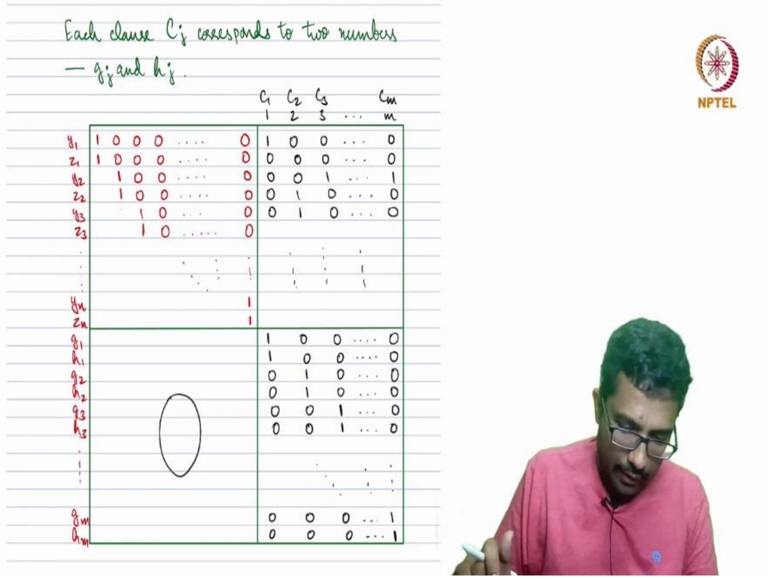
Now, let us focus on the main part, which is to show that an NP-complete language reduces to subset sum. We show that 3-SAT reduces to subset sum. Like in many other cases before, we show that 3-SAT reduces to subset sum. What do we have to do? Given a Boolean formula or a 3-CNF formula φ, we construct a set S and a number t such that φ is satisfiable if and only if

S contains a subset that sums to t. φ is in 3-SAT if and only if S, t is a "yes" instance of subset sum.

Let φ be a formula with n variables and m clauses. It is a 3-CNF formula, meaning it is an AND of m clauses, where each clause is an OR of 3 literals. There are n variables. Let the n variables be X1, X2, ..., Xn. We have an AND of clauses, where each clause is an OR of 3 literals. The literals could be of the form Xi or Xi complement. Now, we will construct the subset sum instance.

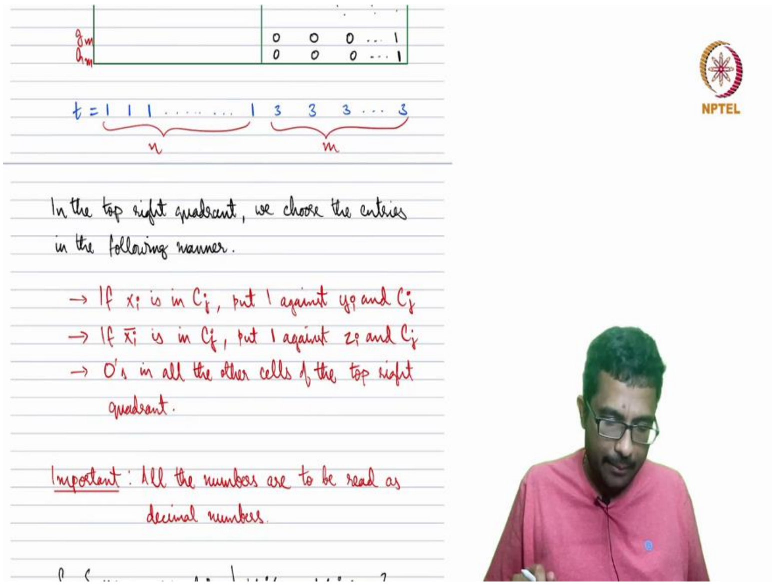
We should produce a set S and number t such that if φ is satisfiable, the set should have a subset that sums to t. If φ is not satisfiable, the set should not have any subset that sums to t. We construct S with 2 times (n + m) numbers, where n is the number of variables and m is the number of clauses.

(Refer Slide Time: 6:30)



Each clause C_j corresponds to two numbers — g_j and h_j.

		C_1 1	C_2 2	C_3 3	...	C_m m
y_1	1 0 0 0 0	1	0	0	...	0
z_1	1 0 0 0 0	0	0	0	...	0
y_2	1 0 0 0	0	0	1	...	1
z_2	1 0 0 0	0	1	0	...	0
y_3	1 0 ... 0	0	1	0	...	0
z_3	1 0 0					
⋮	⋱ ⋮	⋮	⋮	⋮		⋮
y_n	1					
z_n	1					
g_1		1	0	0	....	0
h_1		1	0	0	....	0
g_2		0	1	0	...	0
h_2		0	1	0	...	0
g_3	0	0	0	1	...	0
h_3		0	0	1	...	0
⋮				⋱	⋮	⋮
g_m		0	0	0	...	1
h_m		0	0	0	...	1



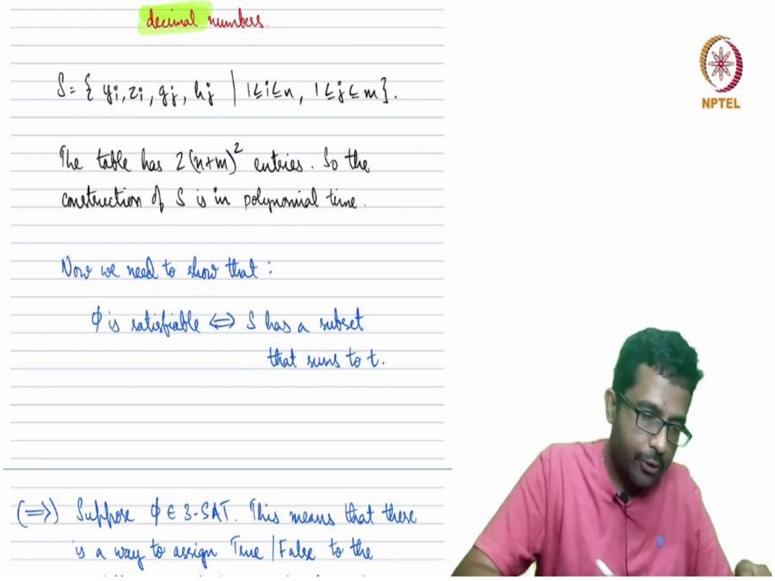
$$\begin{array}{c|c|cccc|}
g_m & & 0 & 0 & 0 \cdots & 1 \\
h_m & & 0 & 0 & 0 \cdots & 1 \\
\hline
\end{array}$$

$$t = \underbrace{1\ 1\ 1 \ \cdots\cdots \ 1}_{n}\ \underbrace{3\ 3\ 3 \cdots 3}_{m}$$

In the top right quadrant, we choose the entries in the following manner.

- → If x_i is in C_j, put 1 against y_i and C_j
- → If $\bar{x}_i$ is in C_j, put 1 against z_i and C_j
- → 0's in all the other cells of the top right quadrant.

Important: All the numbers are to be read as decimal numbers.



decimal numbers.

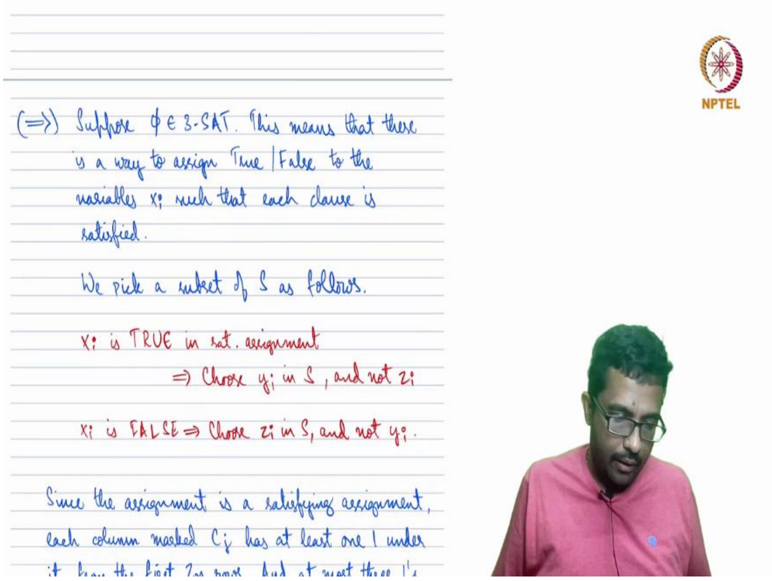
$S = \{ y_i, z_i, g_j, h_j \mid 1 \le i \le n,\ 1 \le j \le m \}$.

The table has $2(n+m)^2$ entries. So the construction of S is in polynomial time.

Now we need to show that:

ϕ is satisfiable $\Leftrightarrow$ S has a subset that sums to t.

($\Rightarrow$) Suppose $\phi \in$ 3-SAT. This means that there is a way to assign True / False to the



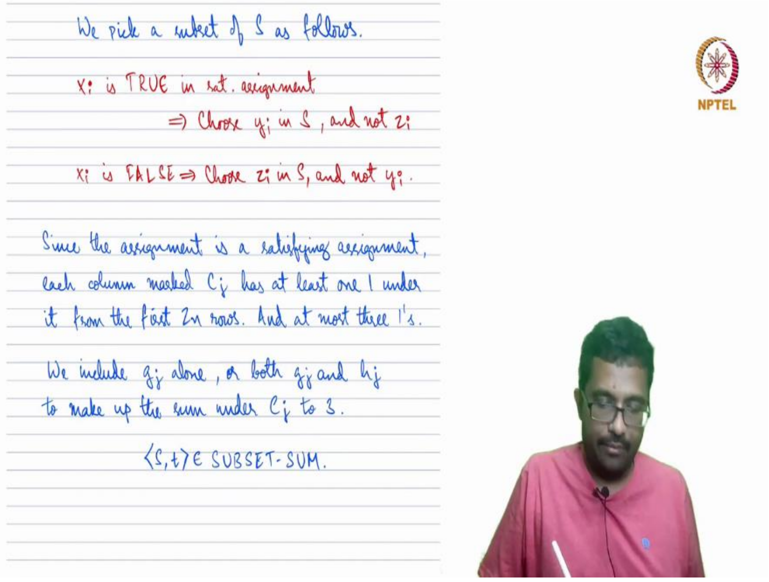
($\Rightarrow$) Suppose $\phi \in$ 3-SAT. This means that there is a way to assign True / False to the variables x_i such that each clause is satisfied.

We pick a subset of S as follows.

x_i is TRUE in sat. assignment $\Rightarrow$ Choose y_i in S, and not z_i

x_i is FALSE $\Rightarrow$ Choose z_i in S, and not y_i.

Since the assignment is a satisfying assignment, each column marked C_j has at least one 1 under it from the first $2n$ rows. And at most three 1's



We pick a subset of S as follows.

x_i is TRUE in sat. assignment
$\Rightarrow$ Choose y_i in S, and not z_i

x_i is FALSE $\Rightarrow$ Choose z_i in S, and not y_i.

Since the assignment is a satisfying assignment, each column marked C_j has at least one 1 under it from the first 2n rows. And at most three 1's.

We include g_j alone, or both g_j and h_j to make up the sum under C_j to 3.

$\langle S, t \rangle \in$ SUBSET-SUM.

So, the way we build the instance is like this. Look at this table that we have. What we are going to do is, for each row in this table, which has 2 times (m + n) rows, each row corresponds to one number in the set. Each entry is either 0 or 1, and we read the entries from left to right to get the number that this row corresponds to. Even though it consists of only 0s and 1s, we are going to read it as a decimal number, not a binary number. This is very important: we are interpreting these numbers as decimal numbers.

So, there are decimal numbers that consist of only the digits 0 and 1. These are going to be the numbers in the instance. Let us see how to build this table, and then I will again explain how to interpret it. This table has (n + m) columns. The first n columns are indexed by 1 to n. The next m columns are indexed by the clauses, C1 to Cm. So, we have n columns, followed by m columns.

The rows, 2(n + m) in total, each correspond to a number. The first row is labeled y1, the second row is labeled z1, then we have y2 and z2, y3 and z3, and so on, up to yn and zn. So that marks this boundary.

Next, let's fill in the table:

1. For the y1 row, set the 1st column to 1 and the rest to 0.
2. For the z1 row, set the 1st column to 0 and the rest to 0.
3. For the y2 row, set the 2nd column to 1 and the rest to 0.
4. For the z2 row, set the 2nd column to 0 and the rest to 0.
5. Continue this pattern for all y and z rows until yn and zn.

After the y and z rows, we have the clause rows:

6. For each clause row corresponding to Ci, set the corresponding clause column to 1 and the rest of the columns to 0. If a variable xj appears in the clause Ci, set the j-th column to 1 for that row. If the complement of xj appears in Ci, set the j-th column to 1 for that row.

Finally, the table will look like this:

- Columns: 1 to n, C1 to Cm
- Rows: y1, z1, y2, z2, ..., yn, zn, Clause1, Clause2, ..., Clausem

To build the instance, we need to look at this table. Each row in this table has 2 times (m + n) rows, and each row corresponds to one number in the set. Each entry in the row is either 0 or 1, and we read the entries from left to right to get the number that this row corresponds to. Even though these entries consist only of 0s and 1s, we interpret them as decimal numbers, not binary numbers. This is crucial: these numbers are read as decimal numbers.

The table has $(n + m)$ columns. The first n columns are indexed by 1 to n, and the next m columns are indexed by the clauses, C1 to Cm. Therefore, we have n columns followed by m columns. The rows, totaling $2(n + m)$, each correspond to a number. The first row is labeled y1, the second row is labeled z1, then we have y2 and z2, y3 and z3, and so on, up to yn and zn.

We then move on to the clause rows, labeled g1, h1, g2, h2, and so on, up to gm and hm. For convenience, we divide this table or matrix into sections, which will help us easily figure out the entries. Constructing the set involves finding the numbers, but these numbers are simply read from this table or matrix from left to right.

The numbers in the set are y1, z1, y2, z2, y3, z3, up to yn, zn, followed by g1, h1, g2, h2, g3, h3, up to gm, hm. This gives us 2n rows followed by 2m rows. Each number is derived by reading the row from left to right. For example, y1 is read as 1, 0, 0, 0, 0, 0 up to the nth place, followed by 1, 0, 0, 0.

Filling the table starts with focusing on the first quadrant, which has the first 2n rows and the first n columns. It is simple: y1 and z1 start with 1, followed by all 0s. The rows y1 and z1 are identical in this part, and this pattern continues for y2, z2, y3, z3, and so on. They will differ in the sections that follow.

So, for y1, the first column is 1, and the rest are 0s. For z1, the first column is 0, and the rest are 0s. For y2, the second column is 1, and the rest are 0s. For z2, the second column is 0, and the rest are 0s. This pattern continues for all y and z rows up to yn and zn.

For the clause rows, if a variable xj appears in clause Ci, the corresponding clause column is set to 1 for that row. If the complement of xj appears in Ci, the j-th column is set to 1 for that row. Interpret each row as a decimal number. This set of numbers, along with the target sum, forms the instance for the subset sum problem. The target sum t will be constructed based on the satisfiability of the formula φ. If φ is satisfiable, the set should have a subset that sums to t. If φ is not satisfiable, the set should not have any subset that sums to t.

So in this part, y1 and z1 have a 1 under the first column, and the rest are 0s. y2 and z2 have a 1 under the second column, and the rest are 0s. y3 and z3 have a 1 under the third column, and the rest are 0s, and so on, until yn and zn, which have all 0s except for a 1 at the nth position. Although I could have filled leading 0s here, as a leading 0 does not change the number, I have not done so. Thus, yn and zn have all 0s except for a 1 at the nth position. This is how we fill this part of the table.

Next, let's see how we fill the part corresponding to the clauses. The rows correspond to y1, z1, y2, z2, and so on, while the columns correspond to the clauses C1, C2, up to Cm. For example, if x1 appears in clause C1, we put a 1 in the entry corresponding to y1 and C1. Similarly, if x2 appears in clause C3, we put a 1 in the entry corresponding to y2 and C3, and if x2 appears in clause Cm, we put a 1 in the entry corresponding to y2 and Cm. For z2 and C2, if x2 complement appears in clause C2, we put a 1 in the entry corresponding to z2 and C2.

To summarize: if xi is in clause Cj, we put a 1 against yi and Cj. If xi complement is in clause Cj, we put a 1 against zi and Cj, and everything else is 0. This indicates that x1 appears in C1, x2 appears in C3, and x2 complement appears in C2. Each variable may appear in multiple clauses, so there could be many 1s in this part, but each clause contains only 3 literals. Therefore, under any clause in this part, we will find exactly 3 ones, because each clause contains exactly 3 literals.

For instance, if clause C3 contains x2, x5, and x6, then y2, y5, and y6 will have a 1. Alternatively, if C3 contains x2, x5 complement, and x6 complement, then y2, z5, and z6 will have a 1. This ensures that each clause column in this part of the table has exactly 3 ones, representing the literals present in the clause.

So, for whichever 3 literals are in the clause, the rows corresponding to those literals will have a 1 under the respective clause. Each row may have multiple 1s, but each column will have exactly 3 1s. When I mention "here," I am referring to the top right part of the table. This part is the only section that depends on the Boolean formula. The other parts depend on the Boolean formula only for the number of variables, but this top right part actually looks at the formula and decides the entries based on it. The rest is straightforward.

The bottom left part of the table is very easy. This entire section is filled with 0s, representing nothing significant. It is just leading zeros for the numbers. Now, let's move on to the bottom right part, which is similar to the top right part.

In the bottom right part, the rows are indexed as g1, h1, g2, h2, g3, h3, and so on up to gm, hm. The columns are indexed from C1 to Cm. There are 2m rows and n columns. The first two rows, g1 and h1, have a 1 under C1 and 0s following it. Similarly, g2 and h2 have a 1 under C2 and 0s following it. This pattern continues with g3 and h3 having a 1 under C3 and 0s following, and so on, until gm and hm have a 1 under Cm and 0s following it. This completes the bottom right part.

To summarize: in the top left part, y1 and z1 have a 1 under the first column, y2 and z2 have a 1 under the second column, y3 and z3 have a 1 under the third column, and yn and zn have a 1 under the nth column, with the rest being 0s. In the bottom right part, g1 and h1 have a 1 under C1, g2 and h2 have a 1 under C2, g3 and h3 have a 1 under C3, and so on up to gm and hm having a 1 under Cm, with the rest being 0s. The bottom left part is entirely 0s. Finally, in the top right part, if xi appears in clause Cj, we put a 1 against yi and Cj. If xi complement appears in clause Cj, we put a 1 against zi and Cj. This indicates, for example, that x2 appears in C3 and x2 complement appears in C2. This is how we build the table.

Now we can read the entries accordingly, interpreting each row as a decimal number to form the set for the subset sum problem.

So y1 is the number 1, 0, 0, 0, 0, 0, and then 1, 0, 0, 0. y2, for instance, is 0, 1, 0, 0, 0, 0, and 0, 0, 1, 0, 0, 0, and 1. For example, gm and hm are simply 1 because all the leading zeros at the end result in 1. Therefore, gm is 1 and hm is 1. g1 is 1 followed by m-1 zeros, h1 is 1 followed by n-1 zeros. Once again, all numbers are in decimal, not binary. This is very important to note. Finally, this gives you the set S. I have described the 2m + 2n numbers completely. This listing of numbers gives us the set S.

What is t? t is simply this number: it is a number with n 1s followed by m 3s. I have aligned it with the table so that it is clear. It is n 1s followed by m 3s. There is no confusion with t because it is certainly not binary, as we have 3s as digits. So, t is also in decimal. Everything here is in decimal, even though the elements of S have digits 0 and 1; they are to be read in decimal. We have completed the construction and described it completely. Again, all the numbers are to be read as decimal numbers.

The construction is clearly in polynomial time because it mainly involves building this table, which has 2(m + n) rows and (m + n) columns, resulting in (2m + 2n)^2 entries, which is polynomial in the size of the given formula. The construction is polynomial time.

What remains is to show the correspondence: the formula is satisfiable if and only if the set has a subset that sums to t. t is defined as the number with n 1s followed by m 3s. We need to show this correspondence in both directions. First, we will show the forward direction. We assume that the formula is satisfiable and then show that there is a subset that sums to t.

If the formula is satisfiable, there is a way to assign true or false to the variables such that each clause is satisfied. Now, how do we pick the subset of S? From S, we need to pick a subset such that its sum is t. Each row corresponds to a number, and we need to achieve the sum defined by t. The first n positions in t are all 1s. These 1s must come from the top left section of the table, as the bottom left section contains all 0s.

For each variable i, we must pick either yi or zi, but not both, because picking both would change the sum. We will pick exactly one of yi or zi for each variable. Here's how we will decide which one to pick: if the variable xi is assigned true, we pick yi; if xi is assigned false, we pick zi. This ensures that the sum of the selected subset matches the n 1s in the first n positions of t.

Next, we need to ensure that the sum also includes the m 3s. To do this, we use the rows corresponding to the clauses. For each clause, we add the numbers corresponding to the literals that satisfy the clause. If a literal appears positively in a clause and the corresponding variable is true, we include the number associated with that literal. If a literal appears negatively in a clause and the corresponding variable is false, we include the number associated with that literal's complement.

By selecting numbers in this way, we ensure that each 3 in t is accounted for by the numbers corresponding to the literals in the clauses. Since each clause is satisfied by the assignment, each clause column in the top right and bottom right sections of the table will have exactly 3 ones, ensuring the correct sum.

Thus, we have shown that if the formula is satisfiable, there exists a subset of S that sums to t.

So, we assume that the formula is satisfiable. If the satisfying assignment sets x1 to true, we pick y1 and not z1. If the satisfying assignment sets x1 to false, we pick z1 and not y1. Similarly, for y2 and z2, if x2 is true in the satisfying assignment, we pick y2 and not z2. If x2 is false in the satisfying assignment, we pick z2 and not y2. This rule tells us how to pick y1 or z1. For each i, we pick exactly one of yi or zi. This will handle the first part of t.

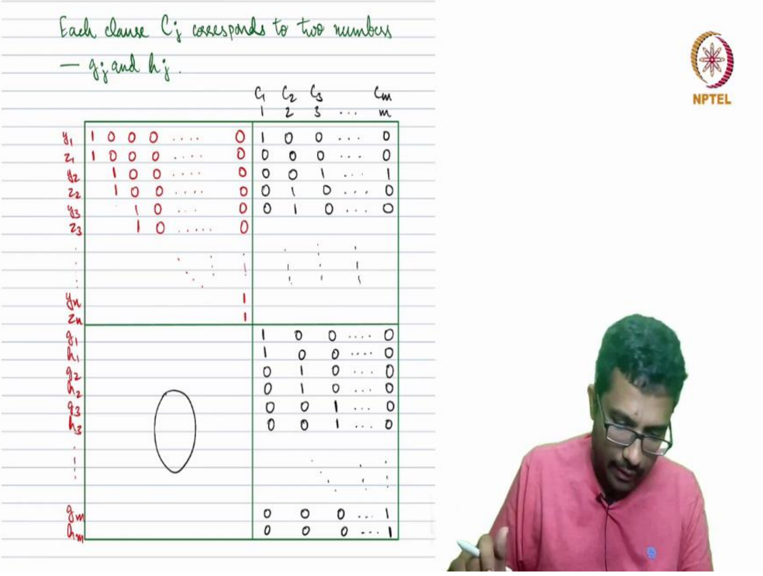
Now, to handle the part of t with the trailing 3s, we need to explain how to pick the remaining numbers from g1, h1, g2, h2, and so on. Notice that each clause has a satisfying assignment, meaning each clause has at least one true literal. For example, consider clause 1. If x1 is the true literal in the clause, and x1 is set to true, we would have picked y1 and not z1. Since x1 appears in the clause, there will be a 1 contributed by y1 in that column. Similarly, if x2 complement is in clause 2 and satisfies it, x2 is set to false, meaning we pick z2, and there will be a 1 under C2 contributed by z2.

The point is that by the choice of y1, z1, y2, z2, and so on, each clause already has at least one 1 under it. Because it is a satisfying assignment, under each clause there will be at least one 1, and at most three 1s, contributed by the rows in the top right part. If a clause has only one true literal, it will have one 1, and we need two more 1s to sum to 3. We can add g1 and h1 to contribute the additional 1s. If a clause has two true literals, we add g2 but not h2 to make the sum 3. If a clause already has three true literals, it will have three 1s, and we will not include either gm or hm.

By including or excluding each g and h appropriately, we ensure that the sum for each clause is exactly 3. This ensures that under each clause, the sum is 3. The choice of y and z rows ensures that the sum for the first part is 1. Since we are treating the numbers as decimals, there is no carryover or dependency between digits, and the sums are straightforward.

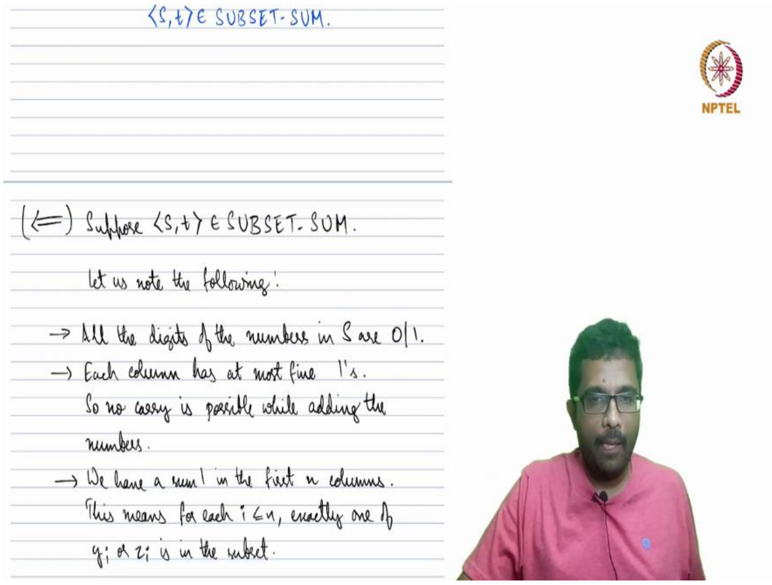
In summary, we choose yi if xi is true, and zi if xi is false. For each clause, depending on the number of 1s contributed by the chosen yi or zi, we include both gj and hj, only gj, or neither, to make the sum under the clause equal to 3. This shows that the set S has a subset that sums to t when the formula is satisfiable. This completes the proof that if the formula φ is satisfiable, then there is a subset in S that sums to t.

(Refer Slide Time: 28:28)



Each clause C_j corresponds to two numbers — g_j and h_j.

		C_1 1	C_2 2	C_3 3	...	C_m m
y_1	1 0 0 0 0	1	0	0	...	0
z_1	1 0 0 0 0	0	0	0	...	0
y_2	1 0 0 0	0	0	1	...	1
z_2	1 0 0 0	0	1	0	...	0
y_3	1 0 ... 0	0	1	0	...	0
z_3	1 0 0					
⋮	⋱ ⋮	⋮	⋮	⋮		⋮
y_n	1					
z_n	1					
g_1	0	1	0	0	....	0
h_1		1	0	0	....	0
g_2		0	1	0	...	0
h_2		0	1	0	...	0
g_3		0	0	1	...	0
h_3		0	0	1	...	0
⋮				⋱		⋮
g_m		0	0	0	...	1
h_m		0	0	0	...	1



$\langle S, t \rangle \in$ SUBSET-SUM.

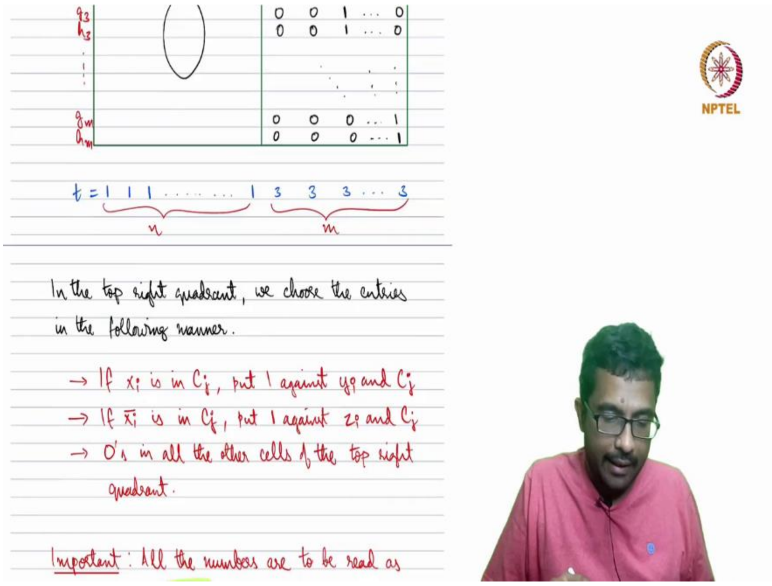
($\Leftarrow$) Suppose $\langle S, t \rangle \in$ SUBSET-SUM.

Let us note the following:

→ All the digits of the numbers in S are 0/1.

→ Each column has at most five 1's.
So no carry is possible while adding the numbers.

→ We have a sum 1 in the first n columns.
This means for each $i \leq n$, exactly one of y_i or z_i is in the subset.



$$\begin{array}{c|c|ccccc|}
g_3 & & 0 & 0 & 1 & \cdots & 0 \\
h_3 & & 0 & 0 & 1 & \cdots & 0 \\
\vdots & & & & \ddots & & \vdots \\
g_m & & 0 & 0 & 0 & \cdots & 1 \\
h_m & & 0 & 0 & 0 & \cdots & 1 \\
\hline
\end{array}$$

$$t = \underbrace{1\ 1\ 1 \cdots\cdots 1}_{n}\ \underbrace{3\ 3\ 3 \cdots 3}_{m}$$

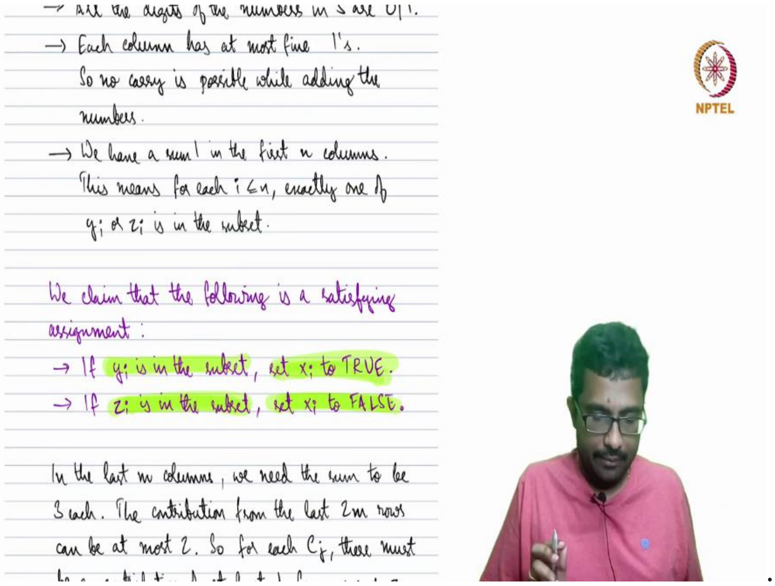
In the top right quadrant, we choose the entries in the following manner.

→ If x_i is in C_j, put 1 against y_i and C_j

→ If $\bar{x}_i$ is in C_j, put 1 against z_i and C_j

→ 0's in all the other cells of the top right quadrant.

Important: All the numbers are to be read as



→ All the digits of the numbers in S are 0/1.

→ Each column has at most fine 1's.
So no carry is possible while adding the numbers.

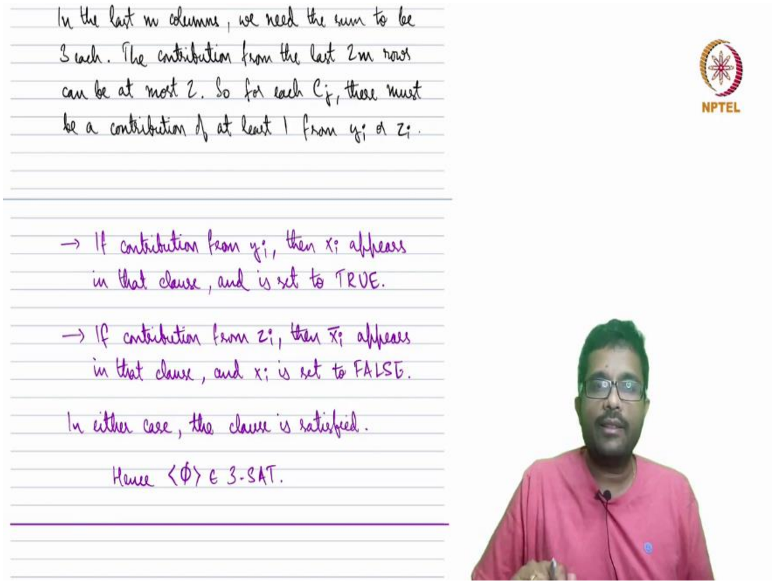
→ We have a sum 1 in the first n columns.
This means for each $i \leq n$, exactly one of y_i or z_i is in the subset.

We claim that the following is a satisfying assignment:

→ If y_i is in the subset, set x_i to TRUE.

→ If z_i is in the subset, set x_i to FALSE.

In the last m columns, we need the sum to be 3 each. The contribution from the last 2m rows can be at most 2. So for each C_j, there must



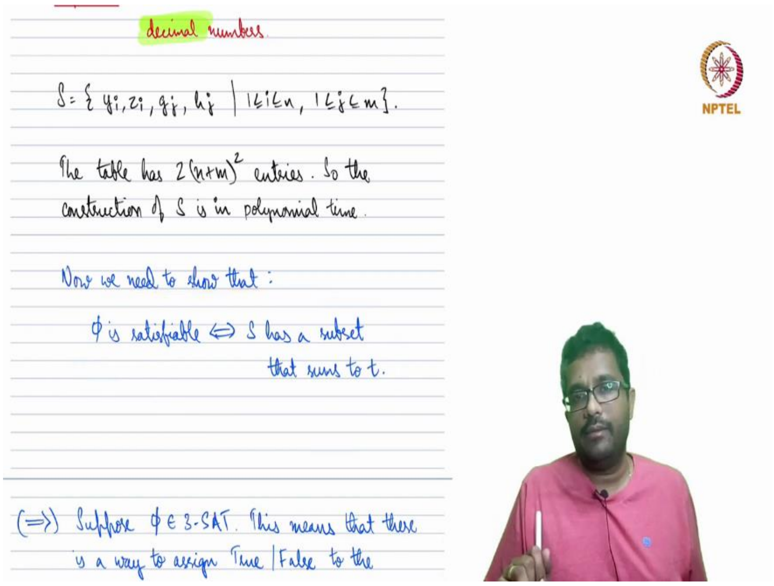
In the last m columns, we need the sum to be 3 each. The contribution from the last 2m rows can be at most 2. So for each C_j, there must be a contribution of at least 1 from y_i or z_i.

→ If contribution from y_i, then x_i appears in that clause, and is set to TRUE.

→ If contribution from z_i, then $\bar{x}_i$ appears in that clause, and x_i is set to FALSE.

In either case, the clause is satisfied.

Hence $\langle \phi \rangle \in$ 3-SAT.



decimal numbers.

$S = \{ y_i, z_i, g_j, h_j \mid 1 \le i \le n, 1 \le j \le m \}$.

The table has $2(n+m)^2$ entries. So the construction of S is in polynomial time.

Now we need to show that:

ϕ is satisfiable $\Leftrightarrow$ S has a subset that sums to t.

($\Rightarrow$) Suppose $\phi \in$ 3-SAT. This means that there is a way to assign True/False to the

Now, let's consider the other direction. Suppose there is a subset that sums to t. Let's understand some properties. For the leading n columns, there are only 2 ones in each column because the bottom part is all zeros, and the top part contributes exactly 2 ones. For the last m columns, the top part corresponds to the literals of a clause, contributing exactly 3 ones. The bottom part contributes 2 ones, making a total of 5 ones in each of these columns.

Since the sum in each column is no more than 5, even if all the numbers (ys

and zs, and the gs and hs) are included, there will be no carry-over in the addition. This means that each column can be considered separately without affecting other columns. This lack of carry-over ensures a clean and independent summation of digits in each column.

Given that there is a subset that sums to t, and knowing there is no carry, we understand that from the first 2n rows (ys and zs), we can only have one of each pair (yi or zi). If both yi and zi were picked, we would not achieve the one in the target sum t for that column. Similarly, if neither were picked, the result would be zero for that column. Thus, we need exactly one of yi or zi for each i.

This gives us the assignment: if yi is in the subset, set xi to true; if zi is in the subset, set xi to false. This ensures that for each i, either yi or zi, but not both, is in the subset, thereby matching the requirement of having exactly one one in the leading columns of t.

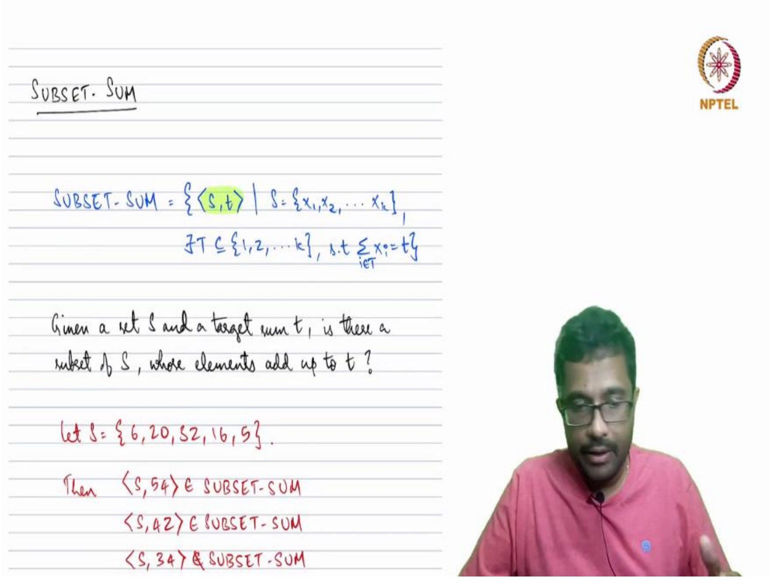
Next, we need to show that this assignment satisfies the formula. Each clause must be satisfied. For the trailing m columns (the clause columns), the sum is 3, with contributions from the top part (ys and zs) and the bottom part (gs and hs). The gs and hs can contribute at most 2 ones, meaning that the remaining one must come from the top part. If the contribution comes from yi, it means xi appears positively in the clause and is set to true. If the contribution comes from zi, it means xi complement appears in the clause and xi is set to false.

By this method, we ensure that each clause has at least one true literal, satisfying the clause. Since the target sum t has 3s in the clause columns and the gs and hs can contribute at most 2, there must be a contribution from the top part for each clause, ensuring at least one true literal per clause.

In summary, for each i, we pick exactly one of yi or zi, setting xi to true if yi is in the subset and false if zi is in the subset. This guarantees that the sum in the leading n columns matches the target. For the clause columns, the contribution from the gs and hs ensures that the sum is 3, with the remaining ones coming from the literals in the clauses, ensuring each clause is satisfied. Thus, if the set has a subset that sums to t, the formula is satisfiable.

This completes the proof that if there is a subset that sums to t, the formula is satisfiable. Hence, the set S has a subset that sums to t if and only if the formula is satisfiable, proving the NP-completeness of the subset sum problem.

(Refer Slide Time: 37:41)



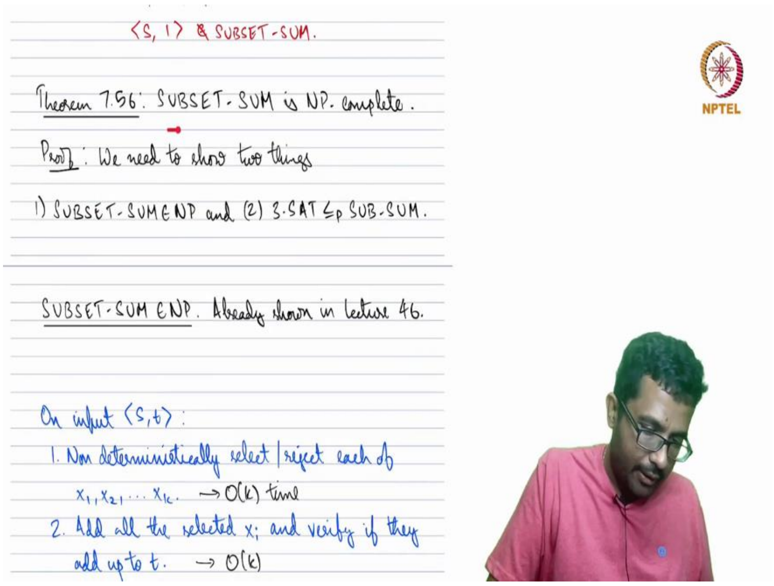
SUBSET-SUM

SUBSET-SUM $= \{ \langle S,t \rangle \mid S = \{x_1, x_2, \dots x_k\},$
$\exists T \subseteq \{1,2,\dots k\}, \text{ s.t } \sum_{i \in T} x_i = t \}$

Given a set S and a target sum t, is there a subset of S, whose elements add up to t?

Let $S = \{6, 20, 52, 16, 5\}$.

Then $\langle S, 54 \rangle \in$ SUBSET-SUM
$\langle S, 42 \rangle \in$ SUBSET-SUM
$\langle S, 34 \rangle \notin$ SUBSET-SUM



<S, 1> & SUBSET-SUM.

Theorem 7.56: SUBSET-SUM is NP-complete.

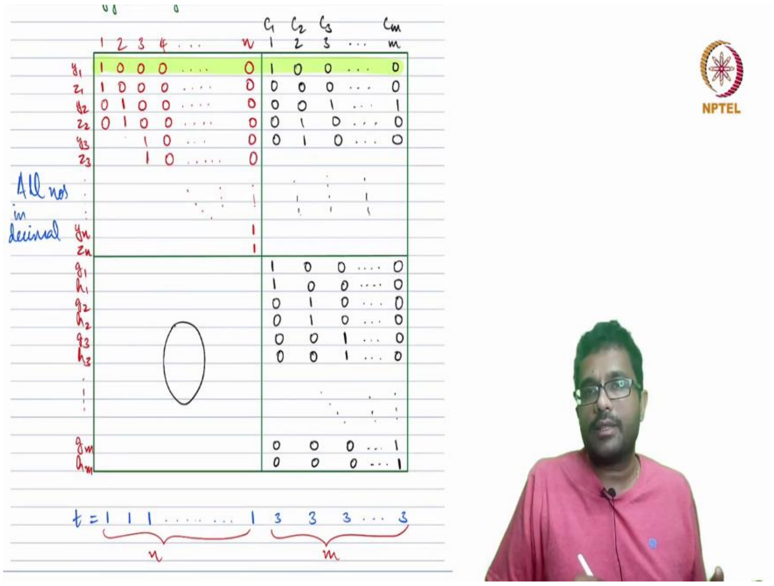
Proof: We need to show two things

1) SUBSET-SUM ∈ NP and (2) 3-SAT $\leq_p$ SUB-SUM.

SUBSET-SUM ∈ NP. Already shown in lecture 46.

On input <S, t>:

1. Non deterministically select | reject each of $x_1, x_2, \ldots x_k$. → O(k) time
2. Add all the selected x_i and verify if they add up to t. → O(k)



	1	2	3	4	…	n	C_1 1	C_2 2	C_3 3	…	C_m m
y_1	1	0	0	0	….	0	1	0	0	…	0
z_1	1	0	0	0	….	0	0	0	0	…	0
y_2	0	1	0	0	….	0	0	0	1	…	1
z_2	0	1	0	0	….	0	0	1	0	…	0
y_3			1	0	…	0	0	1	0	…	0
z_3			1	0	…..	0					
⋮					⋱	⋮	⋮	⋮	⋮		
y_n						1					
z_n						1					
g_1							1	0	0	….	0
h_1							1	0	0	….	0
g_2							0	1	0	…	0
h_2							0	1	0	…	0
g_3				0			0	0	1	…	0
h_3							0	0	1	…	0
⋮									⋮	⋱	⋮
g_m							0	0	0	…	1
h_m							0	0	0	…	1

All nos in decimal

$$t = \underbrace{1\ 1\ 1 \cdots\cdots 1}_{n}\ \underbrace{3\ 3\ 3 \cdots 3}_{m}$$

So there you go, you have yet another problem, which involves sets of numbers and finding a subset that achieves a target sum. It is entirely different from graphs with cliques or independent sets, or Boolean formulas. It is a different type of problem, but it is also NP-complete. To summarize, showing that it is in NP was easy using a standard guess and verify approach. To show that it is NP-complete, we reduced it from 3-SAT.

Given a 3-SAT instance, we built numbers whose digits are entirely 0s and 1s, but we view them as decimal numbers. There are 2n + 2m numbers, where m is the number of clauses in the formula, and n is the number of variables. The target sum is a number with n 1s followed by m 3s. We showed that if the formula is satisfiable, there exists a subset that sums to the target. Conversely, if there is a subset that sums to the target, the formula is satisfiable.

The construction is in polynomial time because the number of entries is polynomial in the size of the formula. This completes the proof.

That completes lecture number 54. In the next lecture, we will explore yet another NP-complete problem. Thank you.