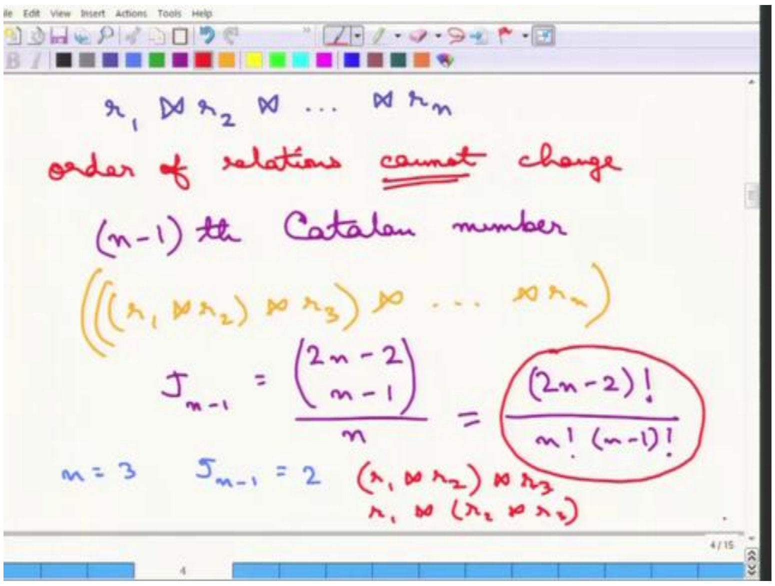
Fundamentals of Database Systems

Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 26
Query Optimization: Join Order

(Refer Slide Time: 00:18)


$r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$

order of relations cannot change

$(n-1)$ th Catalan number

$(((r_1 \bowtie r_2) \bowtie r_3) \bowtie \ldots \bowtie r_n)$

$$J_{n-1} = \frac{\binom{2n-2}{n-1}}{n} = \boxed{\frac{(2n-2)!}{n!\,(n-1)!}}$$

$n = 3 \quad J_{n-1} = 2 \quad (r_1 \bowtie r_2) \bowtie r_3$

$r_1 \bowtie (r_2 \bowtie r_3)$

Fine, so let us then move to an another important topic of Query Optimization, which is the join order down. What we mean by that join order is the following, suppose the following join, following natural join needs to be computed. So, there are n relations and the natural join of r 1 joined r with 2 and so on so forth with r n needs to be evaluated. The question is which is the best way of doing these joins, clearly r 1 can be first join with r 2, the result can be then join with r 3 and so on and so forth or r n can be first join with r n minus 1 and that can be join with r n minus 2 and so on and so forth, so there are many possible ways.

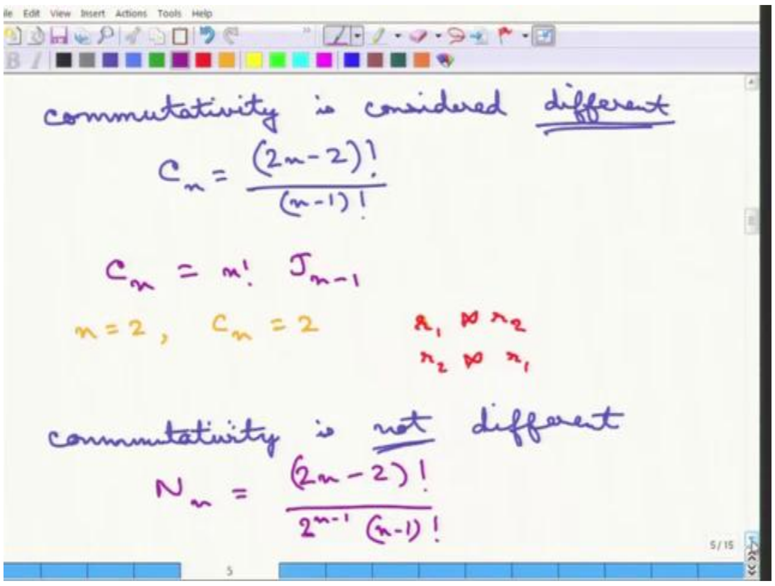
So, what is the best way of doing it? Now before that, the first important question is, what are the number of possible ways. First thing is, if the order of the relations cannot change, so that means, the r 1 and r 2; the order of these joins cannot change. So, r 1 cannot be join with r 3 and that cannot join with r 2, etcetera. So, r 1 must be join with r 2 or r 2 must be join with r 3 and that is need to be join with r 1 that is, so if these cannot change, then the total possible ways of competing this join is the Catalan number, this the n minus 1th Catalan number.

The definition of a Catalan number is precisely this, this is the number of ways of bracketing this expression up. So, one way of bracketing this expression up is r 1 join with r 2; that is then join with r 3; that is then joined and so on and so forth up to r n. And one can see that there are multiple ways of bracketing this up and this is precisely what we want to find out and that is precisely the definition of the Catalan number of the order of n minus 1th, because one of them is already done.

So, there is n minus ways or n minus 1 ways of doing it, the n minus 1th Catalan number can be simply written as, so this is j n minus 1 can be simply written as the following way is choosing n minus 1 out of 2 n minus 2 options and all n of them is equivalent. So, this is divided by n. So, this, the equivalent expression of this is the factorial of 2 n minus 2 divided by the factorial of n and the factorial of n minus 1.

So, that is the number of ways as one can see that, this is quite large for large n's, but for example, for small n, this is not very large. So, for example, if n is equal to 3, then your J n minus 1 is only 2. One can evaluate this expression, why is that, because there are only two ways of doing it, either it is r 1 join with r 2; that is join with r 3 or it is r 2 join with r 3 and that is join with r 1. But, in general this is the very, very large number with large n's, so it is not very easy to evaluate this expression.

(Refer Slide Time: 03:24)



commutativity is considered <u>different</u>

$$C_n = \frac{(2n-2)!}{(n-1)!}$$

$$C_n = n!\ J_{n-1}$$

$n = 2, \quad C_n = 2$ $\quad r_1 \bowtie r_2$, $\quad r_2 \bowtie r_1$

commutativity is <u>not</u> different

$$N_n = \frac{(2n-2)!}{2^{n-1}(n-1)!}$$

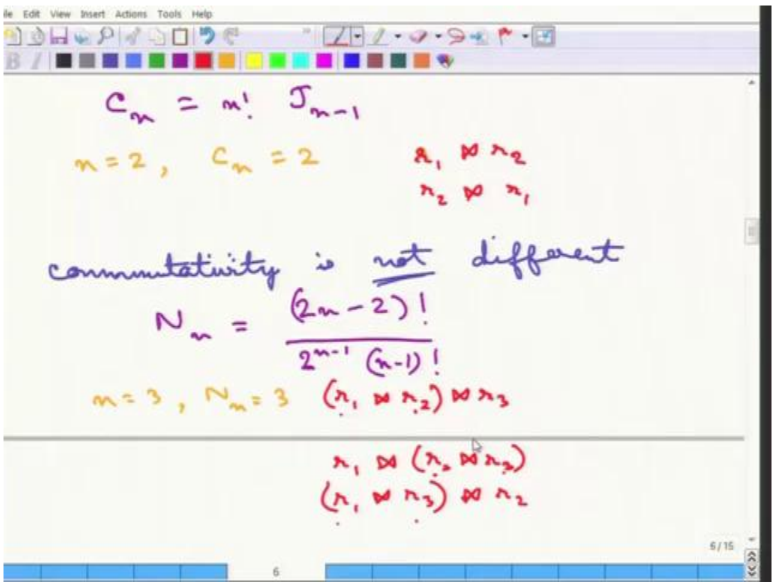
Now, this is when the order of relations cannot change, another way of doing it is that, when the commutativity is considered different, what does that mean is that, when r 1 joined r 2 is different, is considered different from r 2 joined r 1; that way. And r 1 joined r 2 joined r 3 is different from r 3 joined r 2 joined r 1 is considered different, then the total number of ways of evaluating this is C n, which is your 2 n minus 2 factorial divided by n minus 1 factorial.

Now, if you remember, what, that the connection between C n and J n minus 1 or the n minus 1th Catalan number is that, this is factorial n times J n minus 1. So, what does it mean? Because, it is considered different, so all these factorial n combinations are different, so where each of these there are factorial n combinations, which are all different, so the C n is essentially factorial times the Catalan number.

So, again we can see that the n equal to 2 here, C n is equal to only 2, because and the example is simply r 1 joined r 2 and r 2 is joined with r 1, but for large n, if for n's, this is even larger than the previous one, so this is again much more difficult, so that is the other way of thinking about these problem. When commutativity is not considered to be different, so when r 1 joined r 2 is the same as r 2 joined r 1 is not different, then this number comes down to be, we can write it down simply.

This can be written down as N n simply 2 n minus 2 factorial divided by 2 to the power of n minus 1 of n minus 1 factorial. So, again the connection between N n and C n is obvious that N n is C n divided by 2 to the power n minus 1.

(Refer Slide Time: 05:31)



$C_n = n!\, J_{n-1}$

$n = 2, \quad C_n = 2 \qquad r_1 \bowtie r_2$

$\qquad\qquad\qquad\qquad r_2 \bowtie r_1$

commutativity is <u>not</u> different

$$N_n = \frac{(2n-2)!}{2^{n-1}\,(n-1)!}$$

$n = 3, \quad N_n = 3 \qquad (r_1 \bowtie r_2) \bowtie r_3$

$r_1 \bowtie (r_2 \bowtie r_3)$

$(r_1 \bowtie r_3) \bowtie r_2$

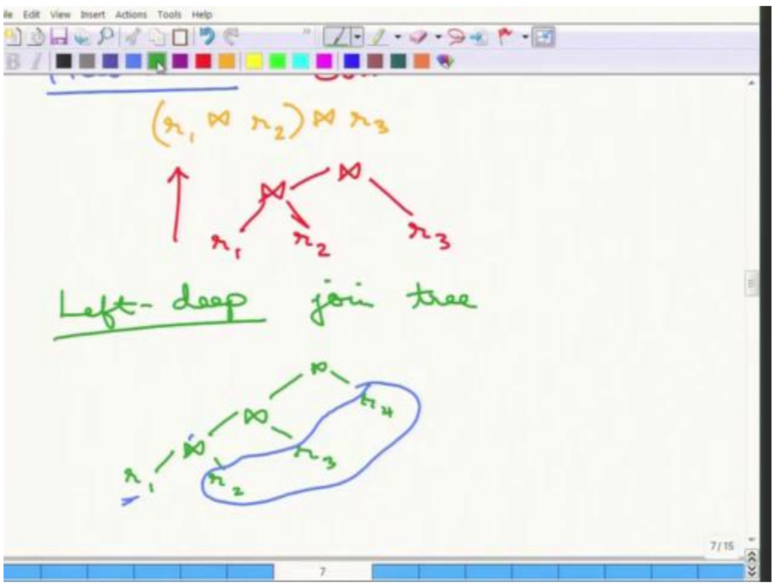
6/15

6

The reason is, once you fix the one thing, the other n minus 1 relation can go anywhere and for n equal to 3, N n is equal to 3 and examples, the three things are r 1, for example r 1 r 2 that is joined with r 3; that is one. The next one is r 1 can be joined with r 2 and r 3 or r 1 can be joined with r 3, which can be then joined with r 2. So, essentially first r 1 and r 3 can be joined or r 2 and r 3 can be joined or r 1 and r 3 can be joined and since connectivity is not different, it does not matter how you write r 1, r 2 or which order ((Refer Time: 06:10)).

So, that is fine; that is the number of join orders, now the question is, why each of these cases the number of join orders to evaluate this too many. And one, the algorithm or whatever system that is trying to evaluate which algorithm to use cannot really look at all of these possibilities. Because, I mean this is exponentially growing with n and for say n equal to 4 or 5, this is just the two larger number to evaluate and it does not make sense.

(Refer Slide Time: 06:46)



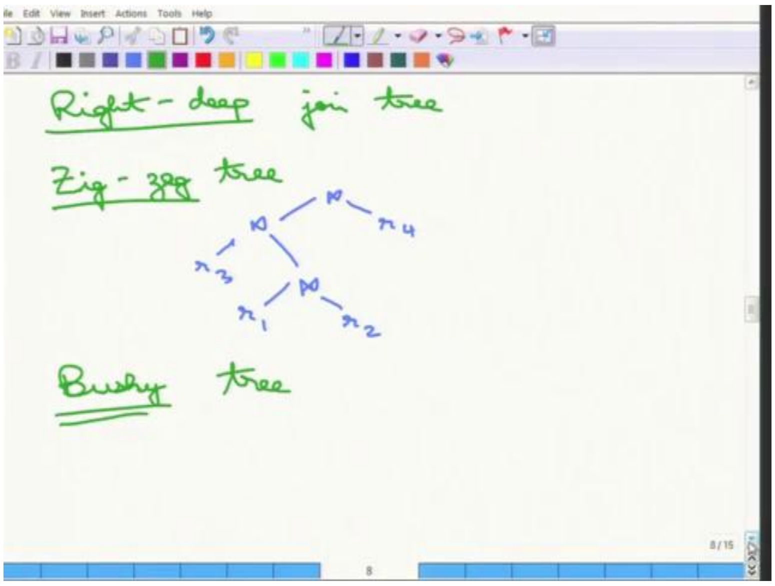
So, what the database engines actually do is that, they employs certain heuristics, so in case you are not familiar with the term heuristics, heuristics is a way of approximation of doing an algorithm in an approximate manner. In the sense that, we are not trying to get the correct answer always, it is not the optimal answer that one wants to looks for, but it is an efficient way of doing it; that is roughly correct.

The heuristics is the concept of join tree, now what exactly is the join tree is the way of writing it down, which we order the things are done. So, if considered this example of data three relations r 1, r 2, r 3 and suppose r 1 r 2 is done fast and then, that is joined with r 3. The corresponding join tree is the following, so r 1 is joined with r 2 and then, that is joined with r 3.

So, it is obvious on the join tree that how the, which joins are done in which order, so it gives me this one whatever written. So, the question of evaluating all the join orders is the question of enumerating all the join trees. Now, all the join trees as we have already argued can be too many, so the heuristics that tries to do is to do shorten join trees. It does not try to enumerate joins trees of every manner; but it tries to do only of shorten type join tree.

So, first one is called a left deep join tree. So, a left deep join tree essentially the right side for every internal node, the right side is the single relation and the left tree can be anything, can be a single relation or can be a join. So, essentially the tree looks in the following, it is just the tree looks the, takes the following set, this is r 1, then r 2, this is then joined with r 3, this is then joined with r 4 and so on and so forth. Note that, all these right sides are at the single relations and the left side is either a single relation or a intermediate join. So, this is a left deep join tree.

(Refer Slide Time: 08:50)

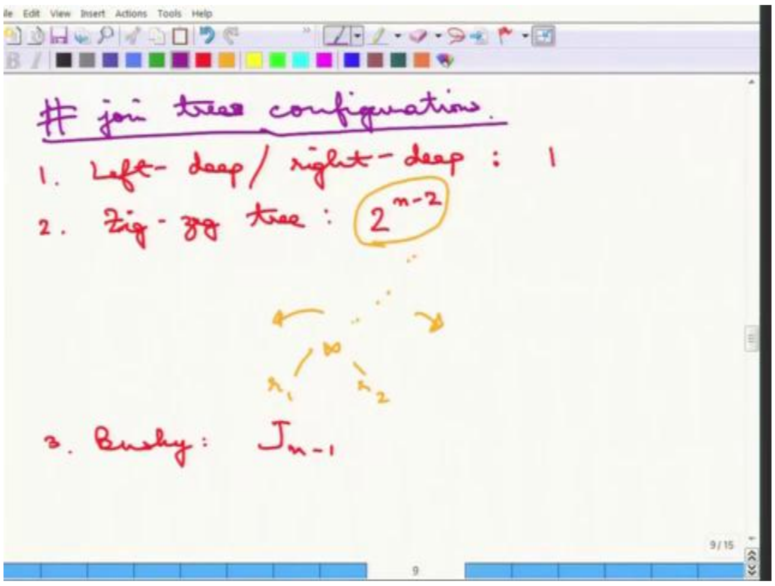


And similar to a left deep join tree, a right deep join tree can also be thought about, where the depth is only on the right side, the left side is only a single relation; that is why it is called a right deep join tree fine. Then, the third one in this space is called a zigzag tree, where it is not specified which side is a single relation, but at least one of the sides must be a single relation.

So, at least one child of an internal node is a single relation; that is called a zigzag tree and an example of a zigzag tree may be the following is that, this is fine and then, let us say, this is on the right side and then, let us say, this is on the left side. So, this is a zigzag tree, now note that, for every join at least one child is a single relation, so that is a zigzag tree.

And finally, there is a bushy tree, it is called a bushy tree, where there is no such restriction and that is essentially the generalized form of a join tree and the number of bushy tree is essentially the all the join orders, so that is the question. So, that is what we have been trying to have, so bushy tree is a most general form, but we will not be evaluating.

(Refer Slide Time: 10:03)



join trees configurations.

1. Left-deep / right-deep : 1

2. Zig-zag tree : $\boxed{2^{n-2}}$

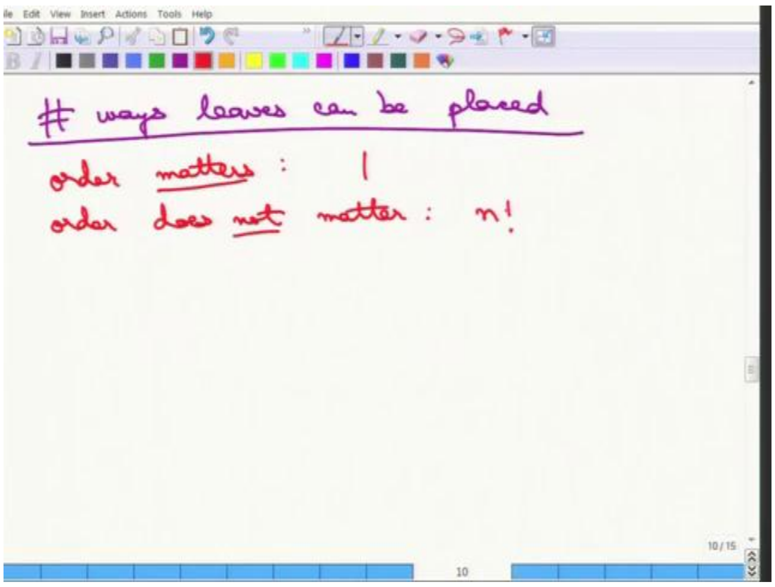
3. Bushy : J_{n-1}

Now, let us try to analyze the number of join trees for each of these kinds of four trees that we are seen. So, number of join trees for the left tree deep or the right deep, it is the same thing, it not any different. For the left deep and join the right deep, the number of possibilities only is 1, because for left deep, there is only one way, you can do it, only the right side is a single relation and for the right deep, there is only the left side is single relation.

For a zigzag tree, the number of configuration that is possible is 2 to the power n minus 2. Now, let us past for moment to see, why this is 2 to the power of N n minus 2, the reason why this is 2 to the power of n minus 2 is that, so the first two leaves r 1 and r 2 can be placed here and then, for any of the upper joins, the single relation can be placed either on the left side or on the right side.

So, for any of these n minus 2 single relations, it can have two choices left and right. So, that is why, these comes out to be 2 to the power of n minus 2 and for bushy trees, this way we've already seen and this is essentially the n minus 1th Catalan number that we have earlier see. This, the number of join tree configurations, join tree configurations that we can see and the question, then is that, this is just a number of shapes that the join tree can take.

(Refer Slide Time: 11:43)

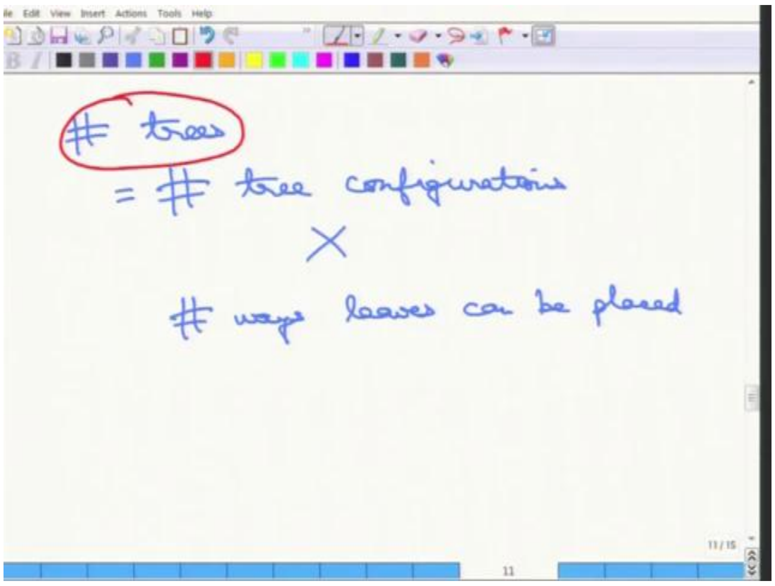

# ways leaves can be placed

order matters : 1

order does not matter : n!

And the other question, related question is the number of ways leaves can be placed. So, if the order matters, so r 1, then r 2, then r 3, there is suppose the number of ways is only 1, because order matters. If the order does not matter, so if order matter, this is 1, if order does not matter, then need does not matter, vary r 1 and r 2 is placed, etcetera. So, the r 1 to r n can be placed in any permutation; that is why the total number of ways leaves can be placed is factorial n.

(Refer Slide Time: 12:22)

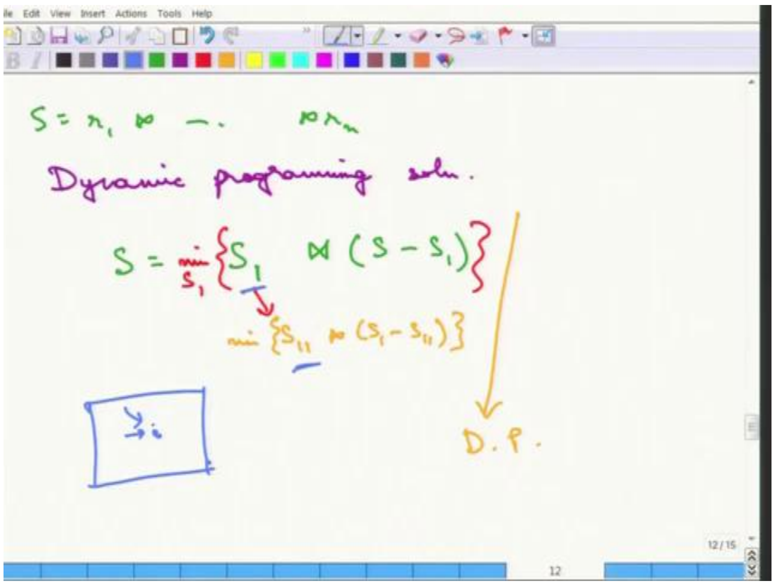


trees
= # tree configurations
×
# ways leaves can be placed

So, if we solve these two, the total number of trees is essentially total number of tree configurations, these are the ways the trees can be generated, times the number of ways, leaves can be placed. So, now, if we say it is a bushy tree with order matters, we know what is the way, if is a left deep tree with the order does not matter, then we know how many trees can be done etcetera, etcetera. So, that is the way of evaluating the join order.

So, now, this is all fine. So, this is just the way of finding the total number of trees and in general, what is being done is that only left deep tree, the practical systems only evaluate left deep trees or right deep trees and may sometimes evaluates zigzag tree is if n is not very large.

(Refer Slide Time: 13:21)



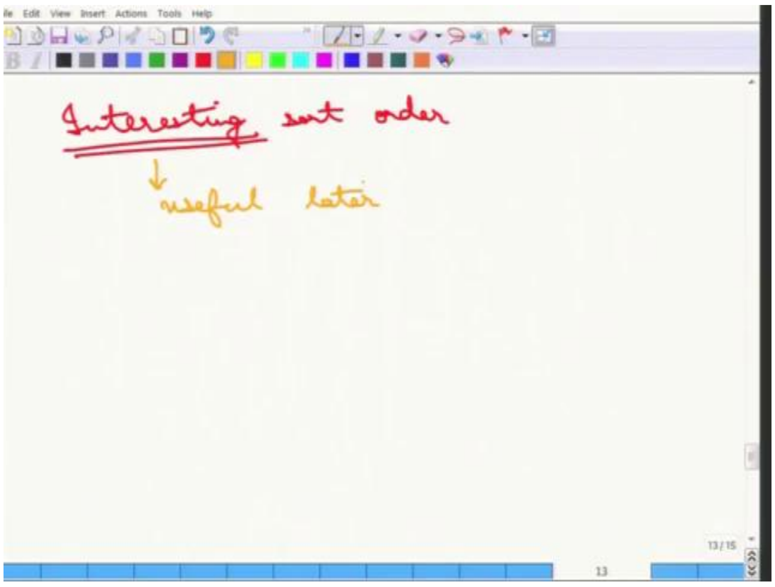
$S = r_1 \bowtie \ldots \bowtie r_n$
Dynamic programming soln.
$S = \min_{S_1} \{ S_1 \bowtie (S - S_1) \}$
$\min \{ S_{11} \bowtie (S_1 - S_{11}) \}$
D.P.

And what is an algorithm for doing it, is that, consider the join of n things to be done, so r 1 up to r n. So, how to actually do it, it has the dynamic programming solution, there is the dynamic programming solution; that is being employed. The dynamic programming solution does the following thing. Consider this is your S now, S can be broken up into the following manner, S can be written down as S 1 join with S minus S 1.

So, this is the set of all relation is that needs to be join, which is S, S 1 is a set of all relation some partition of S. So, the partition is S 1 and S minus S 1. So, these can be written in best way. Now, the question is, what is the best way of evaluating S, the best way of evaluating S is to minimized is over the S 1. So, choose that S 1 that minimize is this configuration.

How is that being done, so how do we find out, what is the best way of doing S 1, S 1 can be again broke down into S 1 1 and S 1 minus S 1 1. Now, what is the best way of finding out S 11 is the minimum way of doing S 1 1 and so on and so forth. So, that is this lend itself to a dynamic programming solution and that is how, this is being done and solutions of each of these sub sets essentially S 1 and S 1 1 etcetera are stored in a table like manner, here is the dynamic programming solution. And then, compute any one of them, it uses everything will surrounded it. So, it uses all the previous computations, fine. So, that is the algorithm for these join order and these as one can see, this is exponential and so on and so forth.

(Refer Slide Time: 15:13)



So, the dynamic programming solution, even though, it is faster, but it still there are many, many ways one can do it. There is a related concept called then interesting sort order. So, interesting sort order is that, one can do the certain in any way, but the interesting sort order is that the records are sorted in particular manner; that is going to be useful later.

So, suppose there is an operation after which there is a selection on the particular attribute a. Now, in that old operation, after which the solution is going to be applied, in the old operation, if the output is produced in a manner, where the records are sorted according to a, then the subsequent selection becomes much faster. So, that is an interesting sort order, because it is useful later. And note that the previous operation can be outputting it in any order, but it makes it useful later, if that interesting sort order, where it is sorted by a is produced.