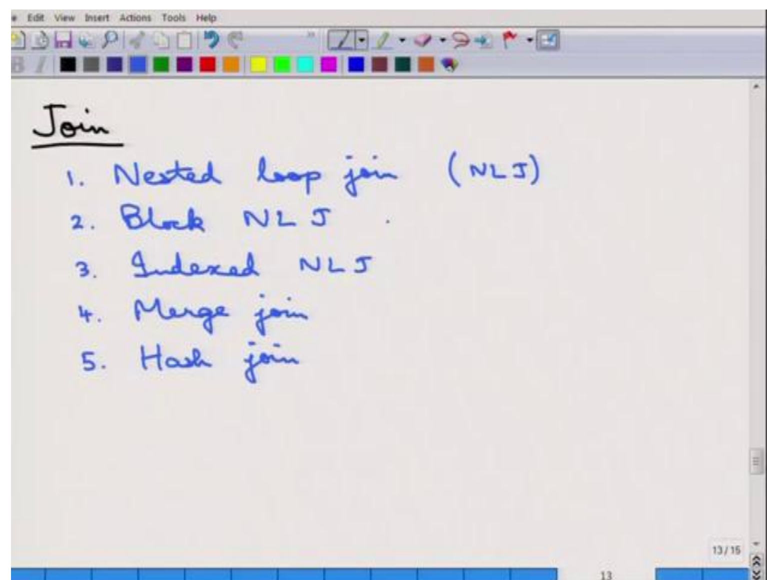
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 22
Query Processing: Nested - Loop Joins and Merge Join

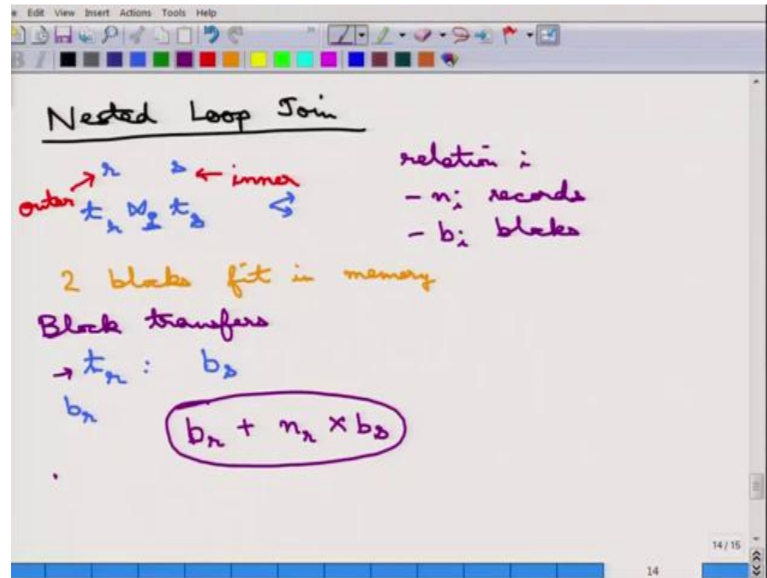
Let us continue with some other kind of algorithms, so we have covered selection we have covered the sorting.

(Refer Slide Time: 00:22)



So, next let us continue with join. Join is one of the very important algorithms in a database and there are different join algorithms, that we will analyze. The first is called the nested loop join, the second one is a version of the nested loop join called the block, so if this is called NLJ, this is the block nested loop join, the third one is called an indexed nested loop join, the fourth one is a merge join and the fifth one is a hash join. So, the second and third are the variations of the first, so we will start off with the first algorithm, which is the nested loop join.

(Refer Slide Time: 01:15)



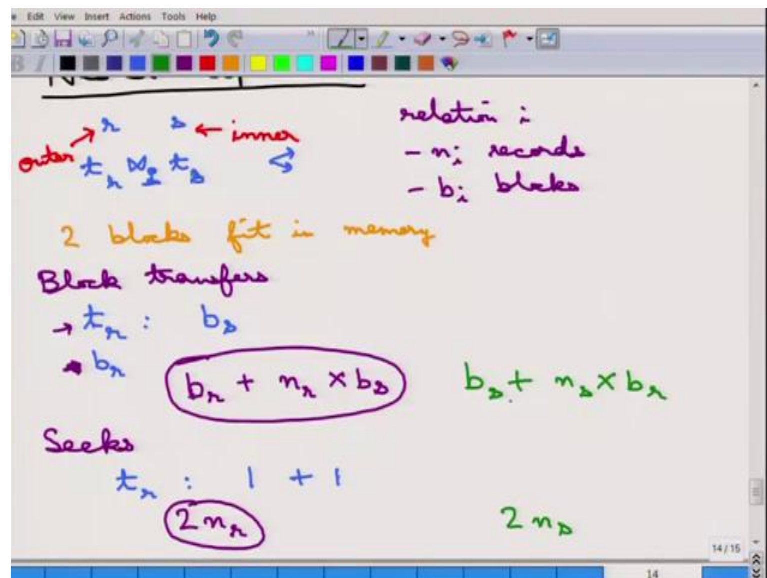
Now, nested loop join is the simplest algorithm to understand and to apply. What depends, what is a nested loop join does is that it picks up. So, remember that a join is between two relations r and s, so what the nested loop join picks up is picks up tuple from r, then picks up another tuple from t 2, performs a join if it satisfies the condition it is the output; otherwise it is not. So, this is a very simple algorithm to understand and very simple algorithm to execute.

And just like the linear search, this is applicable for any kind of predicate in the join, it does not matter, what the join predicate is. So, what it does is the following, so there is a outer relation, so there is a concept of an outer relation and an inner relation. So, suppose this is the outer relation and suppose this is an inner relation, because this is a nested loop, so this is the outer loop and then this is the inner loop. So, r and s and now, suppose let us assume that there are two relations, the relation i contains n i records in b i blocks, fine. So, the relation i has b i blocks, which totally contains n i records.

And assume this is an assumption, that no buffering is being done. So, essentially assume that only two blocks fit in memory, but there is of course, other the memory has enough space for doing other temporary computational perspectives. So, if two blocks is being done, then what it does is the following, is the number of block transfers, how do we analyze the number of block transfers. The block transfers is done in the following manner, is that every time at record t r, so suppose this record is t r and this is t s.

So, every time a record t r is read from r, all the records from s are brought. So, that records every time t r is being done, there are b s block transfers are being done, because all the blocks from s are being read. Now, how many transfers are done for t r, there are b r transfers done, because the records are read one by one. So, the total number of transfers simply is the following, it is b r plus this is being done n r times. How many times this is being done? This is being done n r times. So, this is n r times b s; that is the total number of block transfers, so b r plus n r times b s.

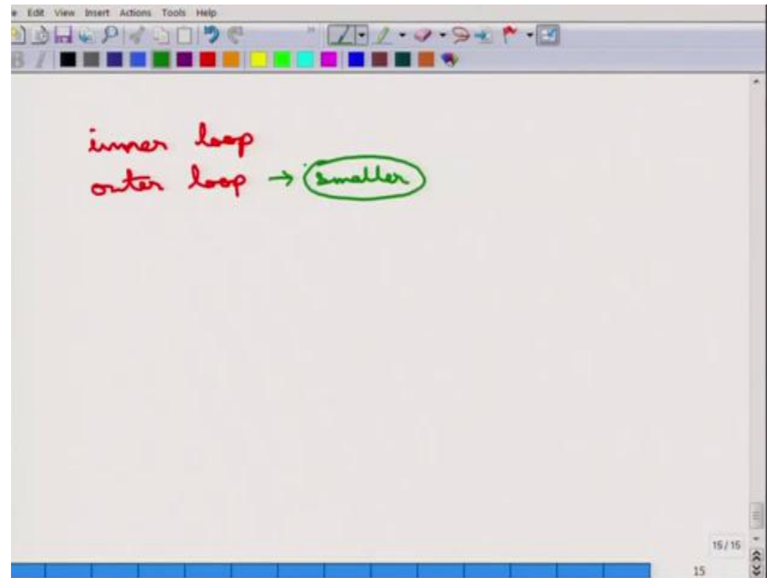
(Refer Slide Time: 04:10)



What is the total number of seeks? How do we analyze the total number of seeks? Again the same principle is applied. So, how do we analyze the number of seeks? So far every time record t r is being read, a seek is being done to read the entire relation of s. But, once that is being read, a seek is again done, such that the next record of r can be read and this is being done n r times, so the total number of seeks is simply twice of n r.

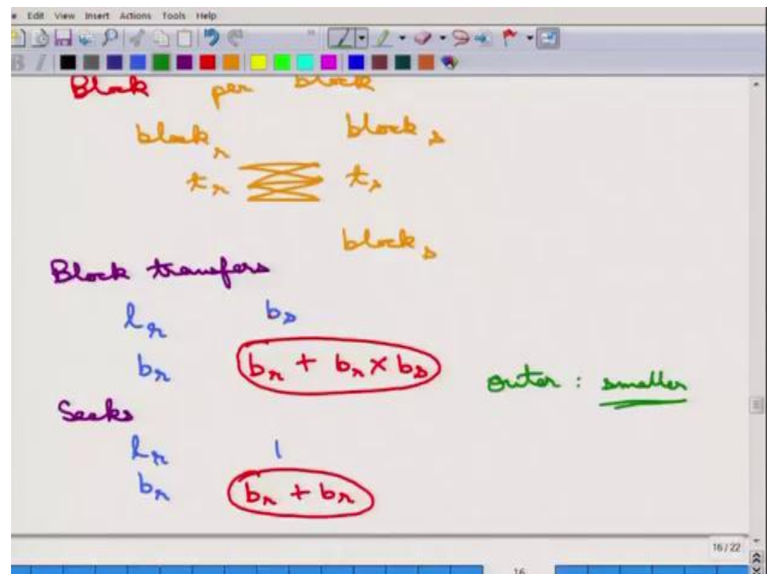
So, this is the nested loop join, let us now move on to the analysis of… There is an inner relation and an outer relation, there is an inner loop and an outer loop. The question is, which one should be the inner loop and which one should be the outer loop. So, if we go back to this analysis ((Refer Time: 04:57)), let us see. So, if r is being used as a outer loop, then this is the cost, on the other hand if s is being used as a loop, then the cost is b s plus n s times b r versus this is 2 of n s, so this is the cost if we change the outer loop and the inner loop.

(Refer Slide Time: 04:48)



So, which one is a better cost? Cost wise it is better if the outer relation is smaller. So, the outer loop the outer relation should be smaller, that produces a lesser cost. So, this is the interesting question that which one should be the outer relation, the outer relation should be always smaller, fine.

(Refer Slide Time: 05:40)



So, let us continue with the next algorithm, which is the block nested loop join. Now, one analysis that all of us should be doing is that the nested loop join is extremely slow and it is very much wasteful. Why is it wasteful? Because, even when a block is read

from r, it processes only one tuple and then, throws away the rest of the tuples and it reads s again for that same tuple the next time.

So, that does not make any sense, because once as we have been saying the number of disk seeks is very important and once a disk block is read for all, all the tuples in that block should be utilized, so that is what exactly the block nested loop join does. So, what it does is the following algorithm, is for… So, this is essentially the block version of the nested loop join. So, it does not do it, process it tuple by tuple, it process it per block.

So, it reads a block from r, it reads a block from r, let us say this is block r that is being read, then it reads a block from s. Then, computes every tuple that is part of these two blocks and computes all the joins there and outputs it and then, it is done for the block, then it reads the next block from s and so on, so forth, so that is what the block nested loop join does. So, it is essentially for each block in r and each block in s, for each tuple in r, and then each tuple in s, it does the join predicate and then, it shows away the block r, so it has done completely.

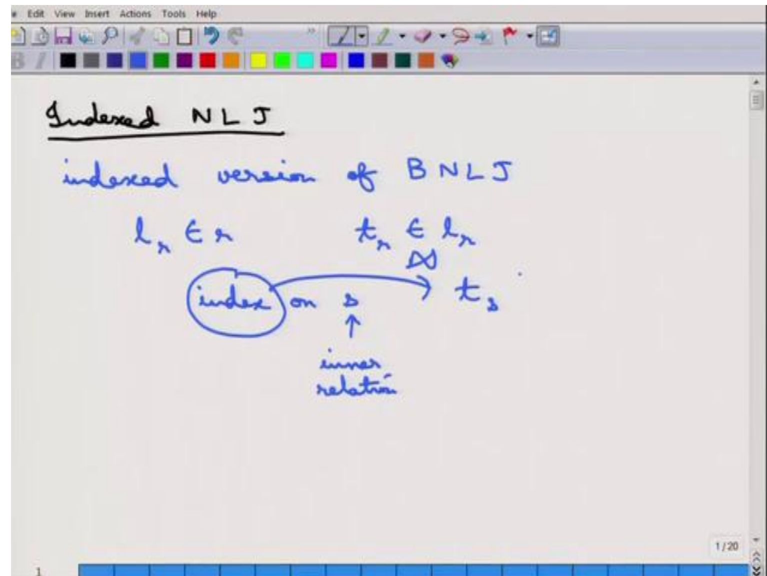
Once more this is applicable for any kind of join and it is just a version of nested loop join, which is optimized for block. Now, what is the cost for this kind of an algorithm? So, number of block transfers, let us try to analyze the number of block transfers that it does. So, every time a block l r is being read, so a block l r from r is being read, it transfers all the blocks from s, so that it requires b s transfers. And, how many times this l r is being done? It is being done only b r times, not n r times as in the last phase you can, it does it block by block.

So, the total number of seeks block transfers, so the total number of block transfers is simply b r plus b r times b s, that it is being done. Because, this b s is being done for each block only and then, the total number of block transfers that is done is b r. So, it is simply b r plus b r times b s and if we analyze the seeks in the following in the same manner, we will see that every time a block l r is being read, there are two seeks that is being done. So, every time block is being read there is one seek that is done for s and this is being done for all the blocks in r, so the total number of seeks is essentially b r plus b r, so that is just 2 b r.

So, this is a much better algorithm cost wise and once more, which what should be the outer relation. The outer relation, should it be smaller or should it be larger, again outer

relation should be smaller. So, the smaller relation should be met the outer, this is just like the nested loop join, so this is the block nested loop join.

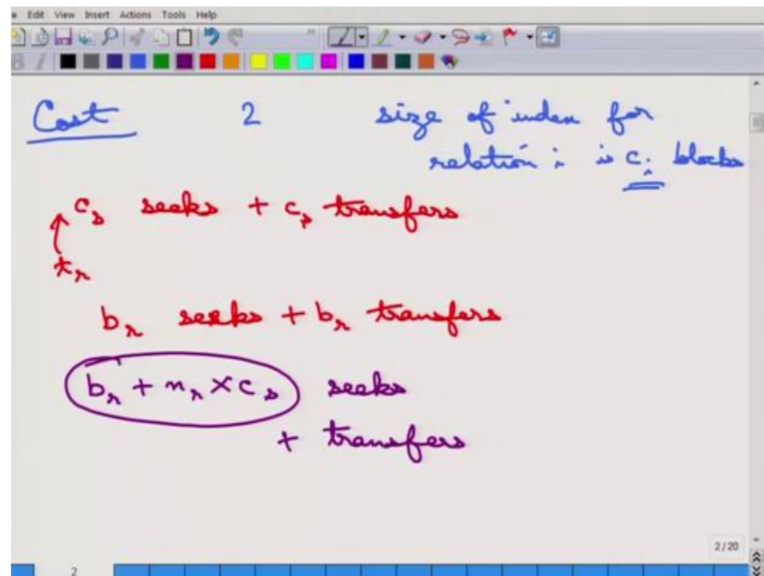
(Refer Slide Time: 09:18)



The next algorithm in this phase is the indexed nested loop join. So, this is just the indexed version of the block nested loop join, indexed version. So, what do we mean by that is block nested loop join, so what do we mean by that is that, for each block l r from r and then, for each record t r in that block l r there is an index. So, there is an index built on, there is an index on the other relation s; that is the inner relation, so this is the inner relation, there must be an index built on r, on s.

The corresponding, using this index the corresponding tuples are found out from s, that can join with t r. So, using this index the tuples t s are found out, that can be joined with t r. So, that is the indexed nested loop join algorithm.

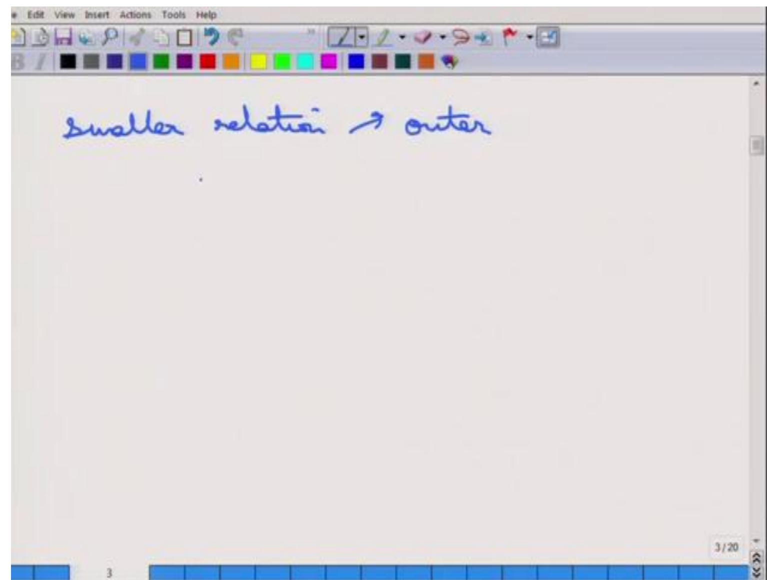
(Refer Slide Time: 10:26)



It is not very difficult to understand, what the algorithm is when let us do the cost analysis of that indexed nested loop join. So, once more the assumption is that only two blocks fit in memory and size of the index for relation i, let us say is c i blocks, so this is what a parameter that is. Now, the cost is c s seeks and transfers c s seeks plus c s transfers for selection on an index is for every record in r. This is for every record t r in r, this is the cost c s seeks and c s transfers, because we need to go over all these c i blocks etcetera.

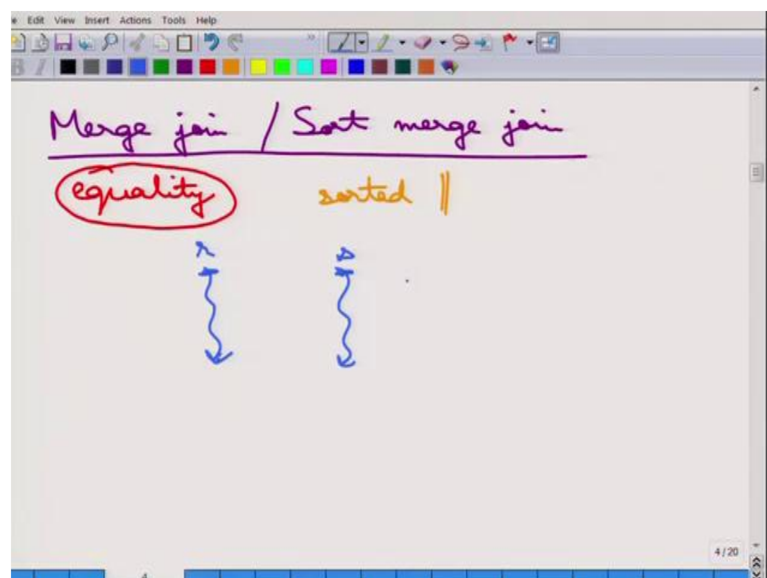
And then, there are b r seeks plus b r transfers for the blocks in r. So, essentially the total cost therefore, is this is b r plus n r times c s, the number of seeks and the same number of transfers as well. So, b r plus n r times c s, so that is the cost of the indexed nested loop join.

(Refer Slide Time: 11:55)



Now, if the index is available for both the relations, this smaller relation should be the outer relation. So, r should be the outer relation if r is smaller than s, because the cost is b r plus n r times c s. So, that is about the indexed nested loop join.

(Refer Slide Time: 12:13)

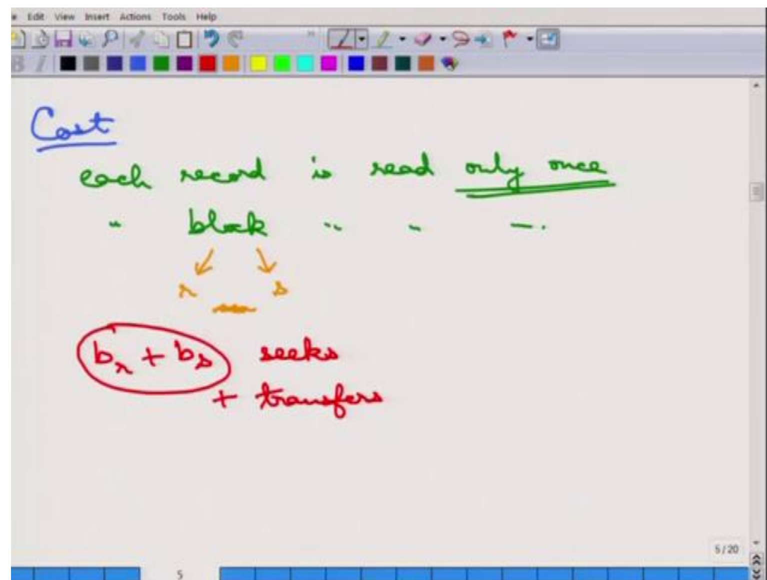


Next we will cover the merge join or the sort merge join this is also known as the sort merge join. So, what does this algorithm do is that this is applicable first of all only when the join condition is equality, so only when this is an equality join, this is very important to understand, otherwise this cannot be applicable. And this is assumed that the relations

are sorted, so if not this sorting cost must be paid. So, this must be first sorted and then, the merge join algorithm needs to be applied.

So, what it does is that there are these two relations, let us say r sorted and s is sorted and just like the merge join the merge proceed here in sorting goes it proceeds one block at a time. And then, it tries to see whether it can be joined, if yes, they join it otherwise, they proceed to the next one and so on, so forth, that is all.

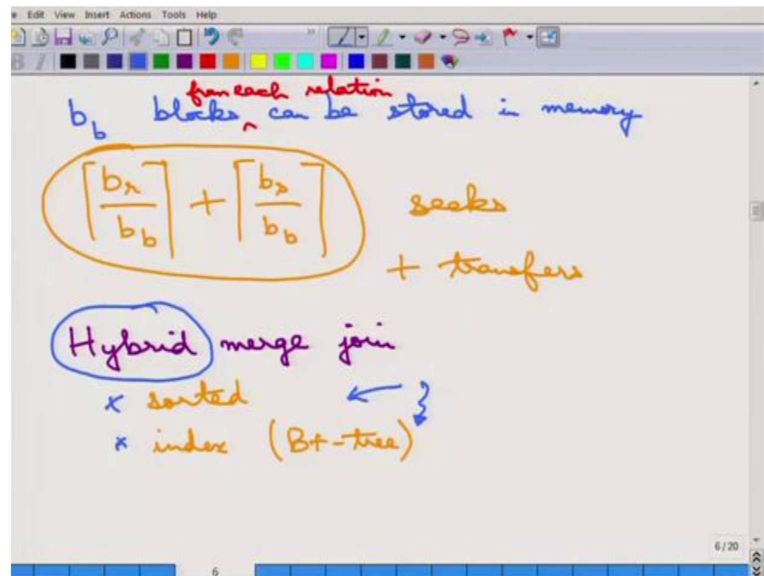
(Refer Slide Time: 13:17)



So, this is essentially just the merge stepping the merge sort that we do. So, what is the cost of this algorithm, the cost of this merge sort algorithm can be analyzed in the following manner is that each record is read only once, because the algorithm proceeds in a sorted manner over the two relations, so that is fine. Therefore, each block is also read only once there is the same thing.

However, the blocks between r and s these blocks can be in r and in s they can be used in any interleaf manner. So, essentially there may be that many seeks as the total number of blocks, so the total number of blocks is b r plus b s, so that many seeks may be needed. And similarly, that many transfers may also be needed, so b r plus b s seeks plus b r plus b s transfers may be needed. Because, the blocks can be interleaved and each whenever a block of r is read a block of s it needs to be read and so on, so forth it can be happened, so that is the cost of this merge join.

(Refer Slide Time: 14:28)



Now, one thing that needs to be handled is suppose the memory has space for bb blocks, so memory, so bb blocks bb number of blocks can be stored in memory order once. So, what does that mean is that instead of only two blocks bb blocks from each relation I should say from each relation can be stored in memory. So, then instead of reading one block at a time, what can be done is that bb blocks can be read each time.

So, the number of passes it requires is b r by bb for r and b s by bb for s and of course, this is the ceiling function. So, this many number of seeks and transfers are needed if bb blocks can be stored in the memory. So, this is fine and if the relations are not sorted, then the secondary index can be used or the relations may be need to be sorted. Then, there is another algorithm called the hybrid merge join, where what it does is that the there is one of the relations is sorted, but the other relation has a index built into it the index; that means, is the b plus tree index that is built into it.

So, what is being done is that, this sorted relation the records the blocks and the records in the blocks are read one by one and then, query is made into the index to retrieve the matching tuples in the other relation. So, that can be done and this algorithm is, then called the hybrid merge join this is called an hybrid, because not all of them are sorted the one is sorted the other is indexed. So, that is why it is called an hybrid merge join.