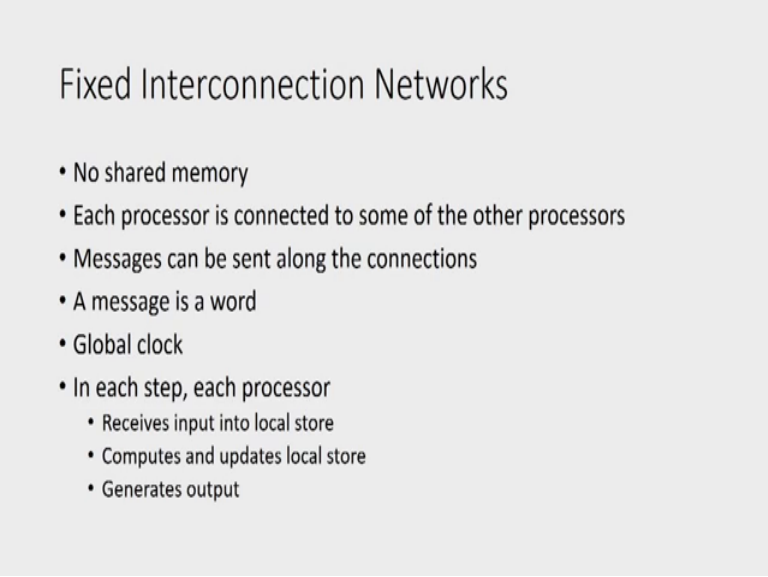
Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 03
Interconnection Networks

Welcome to the third lecture of the NPTEL MOOC on Parallel Algorithms. In the previous lecture you saw the shared memory model of computation which is called a parallel random access machine model in that the processors share up memory and the processors communicate with each other through the memory, but then there are some models of computation in which such a shared memory is not available. We shall see several such models today these are fixed interconnection network models.

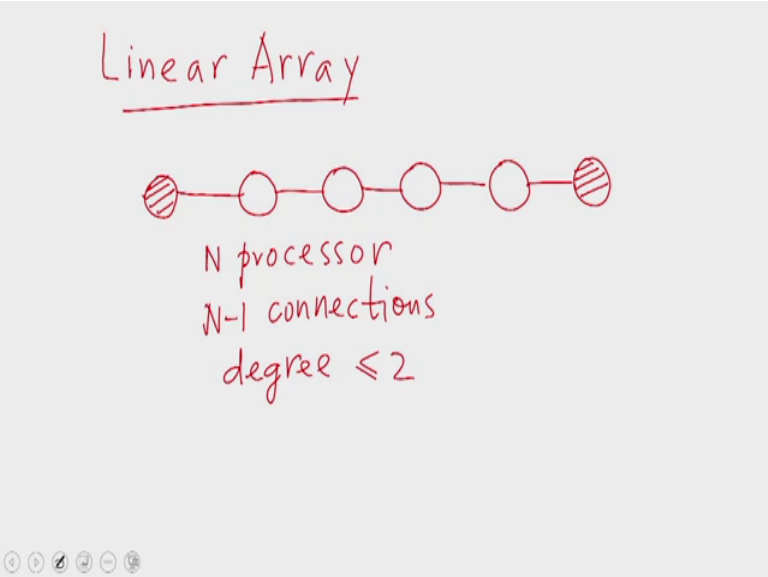
So, in these models as I mentioned there is no shared memory, each processor is similar to a random access machine and each processor is connected to some of the other processors. In each step the processor is capable of sending messages to its neighbors along the connections; we assume that the message is of a word size. And this is again a synchronous machine in the sense that every machine is fed the same clock. So, the machine the processors are all executing in lockstep in each step each processor performs these sub steps.

(Refer Slide Time: 01:38).



In the sub step the processor receives input into the local store, along the connections its neighbors would be sending messages to it these messages would be received by the processor in the sub step. And these received messages could be stored in the local store. And then based on the messages received and the previous contents of its local store it will perform some computations, this is what happens in the sub step and finally, in the third sub step it generates output that has to be sent to its neighbors along the connections. So, that is what a fixed index in network looks like.

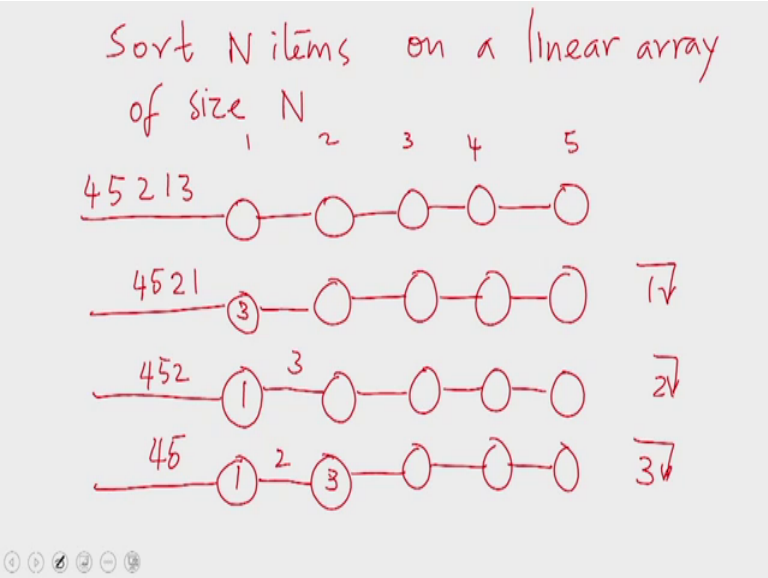
(Refer Slide Time: 02:24)



Now, we will begin with the simplest of the fixed interconnection networks, which is a linear array. In a linear array we have a number of processors, which are interconnected using at most two connections per processor. So, a processor has a left labor and the right neighbor, except possible further two extreme processors. The leftmost processor has no left neighbor and the rightmost processor has no right neighbor.

So, this is what a linear array? So, if you have N processors in a linear array there are N minus 1 connections, the degree of every vertex degree is the number of neighbors that vertex has degree of a vertex is at most 2. Now, what sort of problems could be solved on such interconnection networks a wide variety of algorithmic problems could be solved on such interconnection networks. So, let us begin with a common problem that we discussed in algorithm forces which is sorting.

(Refer Slide Time: 03:52)



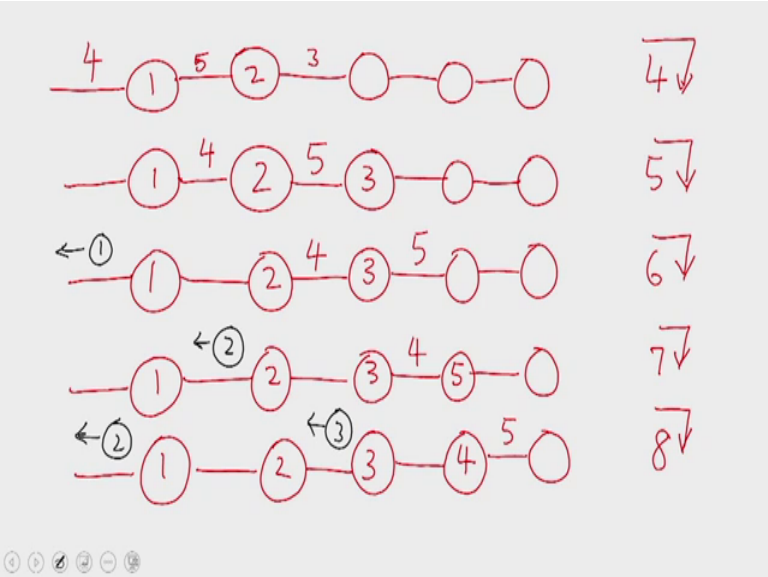
Let us say we have to sort N items on a linear array of size N. So, let us say we have a linear array of 5 notes and on the input line to the processor we have the input elements coming along. So, these are the future input elements 4 5 2 and 1 and 3. So, before the step input element 3 is waiting to enter the processor. So, this is the state of the processor before the step. So, in the step only process of one is active the process of one had a message waiting for it.

So, during this step the message will be delivered to the processor and that will be stored in the local store of the processor. So, now, the remaining inputs are 4 5 2 and 1 to be coming in the future clock cycles, this machine has no output to generate and the contents of the other processors are empty. Now, in the clock cycle the processor has one coming in it already has a value stored inside it which is 3, what it does is this it compares the incoming value with the already stored value it finds that 1 is less than 3 therefore, it will send 3 out as a message on to its right neighbor and the incoming value 1 will be stored instead. The other processors continue to be empty. So, this is what happens after the 2nd step at the end of the 2nd step, this is the contents of the processes at the end of the 1st step.

Now, in the next step processes 1 and 2 are active let me number the processes from here so, in the 3rd step processes 1 and 2 are active processor 2 has an incoming message 3 and it has no contents. So, let us assume that process of 2 receives message 3 and stores

it is in its local memory. The processor already contains one and it has an incoming message too so, it compares 2 with 1 finds that 2 is larger than 1 therefore, it does not store too, but it sends 2 out as a message to the 2nd processor its content continues to be the previous one the remaining input is 4 and 5. So, this is the content of the list at the end of the third step. Now going on to the 4th step.

(Refer Slide Time: 07:48)



In the 4th step the 2nd processor is active it has a local value of 3 and an incoming message of 2, the incoming message is 2 and that 2 is smaller than 3.

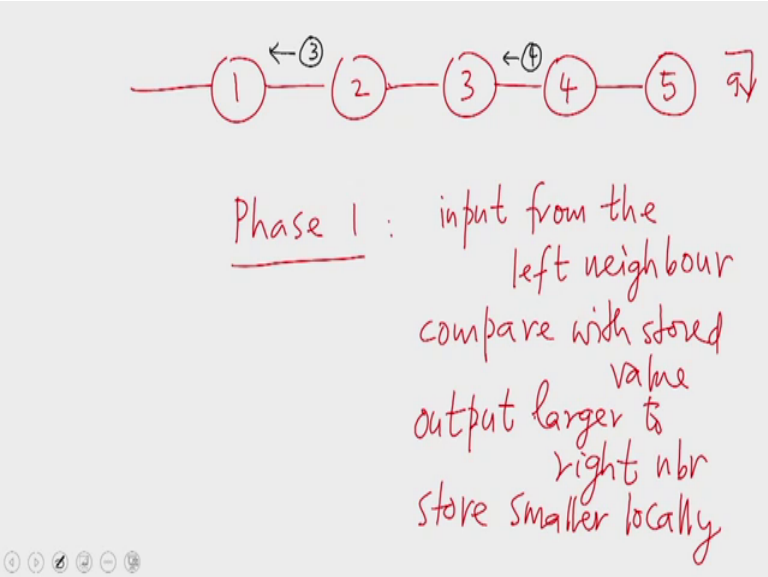
So, the 2nd processor will store 2 inside and send 3 along its connection to the 3rd processor as a message. The 1st processor has 5 coming in and it already has a value of 1, it finds that one is smaller than 5 therefore, 5 will be sent down as a message to processor 2. The other processors continue to have empty values 4 is the next input. So, this is what happens in step 4 so, at the end of the step 4 this is where how the list looks like. Now, coming to the 5th step the 3rd processor has a message 3 coming in and that will be stored there, the 2nd processor has a stored value of 2 it has a an incoming message of 5.

Since 5 is greater than 2 this processor does not change its contents and sends 5 on to the 3rd processor. Similarly process of one has an incoming message of 4, it finds that 4 is greater than 1 therefore, 4 will be sent along to processor 2 the other processors will contain nothing as before the list will look like this at the end of the 5th step.

So, now you can see that elements 1 2 and 3 have reached their final destinations. Considering in this fashion in the 6 step process of 1 has no incoming message processor 2 has 4 coming in, but 2 is smaller than 4 therefore, 2 is repaint and 4 is sent on to processor 3. Processor 3 has 5 coming in and 5 is larger than 3 therefore, 3 is retained and 5 is sent along to process of 4. So, this is how it looks like at the end of the 6th step in the 7th step.

The 3rd processor has 4 coming in 3 is what it contains it will send 4 on to processor 4 and retain its value 3. The 4th processor has 5 and 5 will be stored in its internal memory. And the 5th processor as it does not contain anything and then in the 7th step the 4th processor is active the 4th processor has for coming in and its internal value is 5 since 4 is smaller than 5 4 will be stored in its internal memory and 5 will be sent on to the last processor. So, this is how they will just look the array looks like at the end of the 8th step.

(Refer Slide Time: 11:36)



Now, going on to the 9th step here all these processors have stabilized, only the last processor had an incoming message at the end of the previous step therefore, in this step that value will be received and stored in its internal memory. So, now, the array looks sorted.

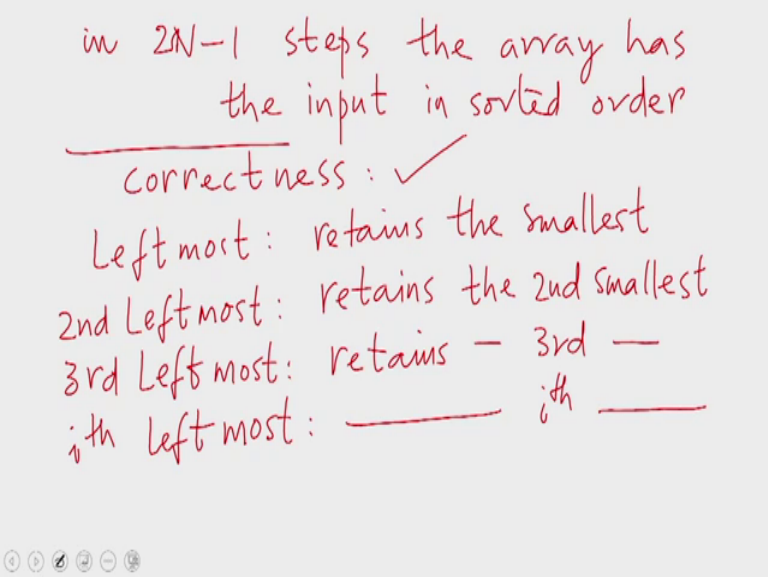
So, the algorithm involved can be summarized like this is the phase of the algorithm where the sorting is done. So, in the phase of the algorithm each processor does this it

receives an input from the left neighbor, it compares the input with stored the stored value output the larger to the right neighbor and store the smaller value locally so, this is what the processors do repeatedly.

Once again to take an example consider the 6th step and look at the post 3rd processor the 3rd processor has a stored value of 3 and that is an incoming value of 4 it compares the incoming value with its stored value. Since the incoming value is larger the incoming value will be going out as a message to the 4th processor. The smaller of the 2 values namely 3 will continue to be stored in its internal memory.

Now, look at process of 4 in step 7 the incoming value is 4 the stored value is 5, when you compare these 2 values you find that the incoming value is the smaller one therefore, the incoming value has to be stored in the local memory. So, 4 is stored in the local memory the larger of the 2 namely 5 will be sent on as a message to the final processor. So, this is the algorithm we have been using in the phase. So, now, we claim that if we run the phase at the end of 2 N minus 1 steps the array has the input in sorted order let us see how we can throw that let us prove the correctness.

(Refer Slide Time: 14:27)



Looking at this example we find that when the input streams in through the leftmost processor it will retain the smallest element. So, here the input is coming in the order 3 1

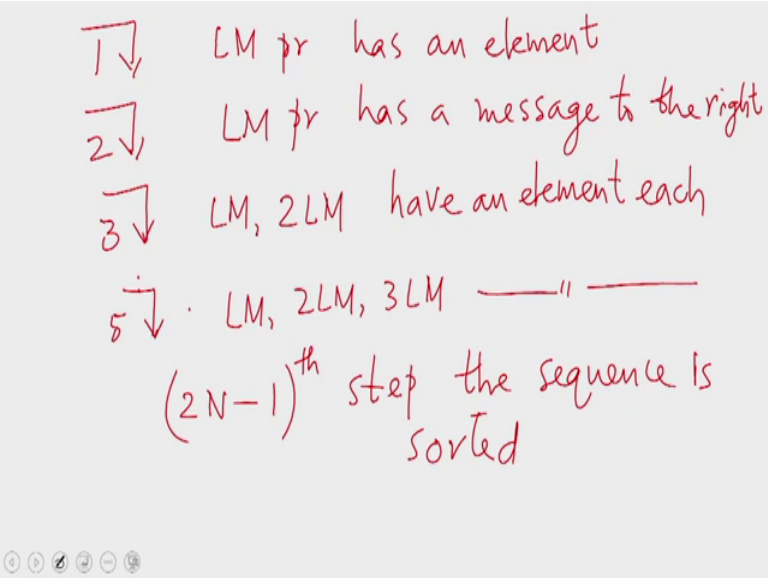
2 5 and 4 and these are the elements that the processor gets to see. So, it initially stores 3, but later on when one comes in it finds that 1 is smaller than 3 and retains 1 and sends 3 on to the next processor. The remaining elements are all larger than 1 therefore, they will never be stored in the processor. Therefore, we find that the leftmost processor looks at every single element, but retains the smallest element with it the leftmost processor 3 retains the smallest element, but passes on all the larger elements on to the next processor.

Now, if you look at the 2nd processor in this example we find that the 2nd processor gets to see all the remaining element that is it sees the elements 3 2 5 and 4 in that order. So, the 2nd processor will store 3 and then later on when 2 comes in 3 is replaced by 2 the remaining elements 5 and 4 when they come in that order will not be stored in processor 2, but passed on to the remaining processors to the right. Therefore, we can conclude that the leftmost processor retains the smallest of the elements that it gets to see which happens to be the smallest element.

So, the processor filters out the leftmost process of filters out the smallest element and sends the remaining elements onto the 2nd processor, the processor filters out the smallest element keeps it with itself and send the remaining to the 3rd processor. Then naturally the 3rd processor you can imagine retains the3rd smallest element.

Continuing like this we can argue that the ith leftmost element retains the ith smallest element. And in particular the end leftmost element which happens to be the rightmost processor the will receive the largest element and then store it with itself it will never get to see any of the other elements. Therefore, when the algorithm terminates the ith left most processor will contain the ith smallest element and therefore, the input will be in sorted order, which proves the correctness of the algorithm which establishes that the algorithm indeed sorts the input correctly.

(Refer Slide Time: 18:50)



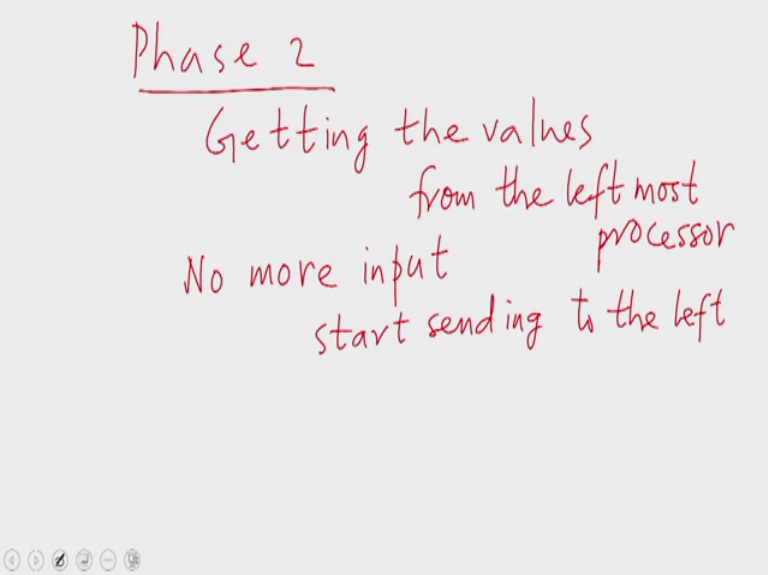
Now, let us see how long the algorithm runs at the end of the 1st step the leftmost processor has an element none of the other processors has an element in the 2nd step at the end of the 2nd step. The leftmost processor has a message to the right and then we find that at the end of the 3rd step the leftmost processor. And the 2nd leftmost processors both have an element each you can see this from this example at the end of the 1st step the leftmost process has an element.

At the end of the 2nd step the 1st processor has a message going out to its right neighbor. And then at the end of the 3rd step the 1st processor and the 2nd processor both have elements stored in them, but none of the other processors has anything stored in them. So, continuing like this we find that at the end of the 5th step the left most element the leftmost process of the 2nd leftmost processor and the 3rd leftmost processors all have an element each none of the other processors has an element and then in at the end of the 7th step the leftmost 4 processes will have elements none of the others will have an element. So, continuing like this by induction we can show that at the end of the 2 N minus 1st step.

The sequence is sorted so, this algorithm takes 2 N minus 1 steps to get the input sorted, but this is only the 1st phase of the algorithm when we put the elements in sorted order. In the 2nd phase we have to get the sorted elements out of the array. So, initially the array was empty the input stream in and the input is populated into the array and when

the populating of the array is done the array is in sorted order. This populating of the array will take 2 N minus 1 steps.

(Refer Slide Time: 21:47)



Now, how do we get the values out the 2nd phase is about getting the values out.

Let us assume that we are getting the values out from the leftmost processor. So, we are going to follow the following protocol each processor will start passing left, when no more input is received from the left neighbor. If there is no more input then the processor knows that all the input that had to come has come and now the processor contains the correct value that it should contain therefore, this value can now be output from the left end. So, if there is no more input received from the left neighbor start sending the values to the left.

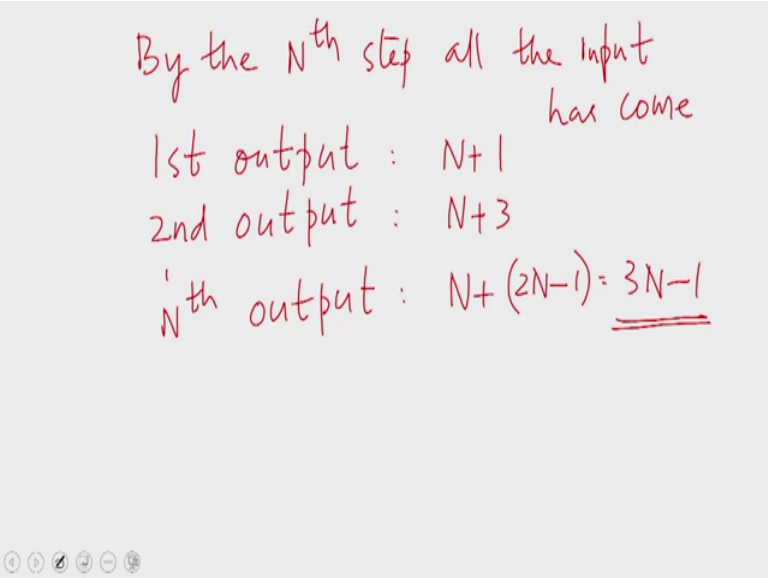
So, let us see when the 1st processor can start sending the values to the left. The processor stops getting messages after the 6th step there were 5 elements coming in this example and those 5 elements were received in the 5 steps. Therefore, in the 6th step it realizes that it has no input message coming in. So, after the 6th step in the 6th step since there is no incoming message it can send 1 out as a message to the left side. So, the value 1 is now being output which is the smallest of all the values. So, the smallest of all the values is now being output. Now, the 2nd processor realizes that it has no incoming message in the 7th step.

Therefore, in the 7th step the 2nd processor will send out value 2 to its left neighbor, this is the output that is generated by the 2nd processor. So, when the 1st processor receives this value from the right in the 8th step it will pass it on to the positive out pass it out along the left connection. So, in the 8th step this will be passed on as an output onto the left side of the 1st processor.

So, let us assume that this is where we receive the output. So, the 1st output will be received in the 6th step and the 2nd output will be received in the 8th step. Now, look at processor 3 processor 3 does not receive any message in the 7th step therefore, in the 8 step process the 3 will start sending out its value to the left.

So, now you can see the pattern in the 6th step the 1st processor sends out its value in the 7th step. The 2nd process ends out its value and in the 3rd step in the 8th step the 3rd process ends out its value all to the left side the output received at the leftmost end of the array would be in the 6th step and then in the 8th step and so on. In the 9th step the 4 will be sending out its content to the left side and processor 2 will be sending out its content to the message it had received to the left side which is the value 3. So, these 2 messages will still remain in the system at the end of the 9 step and this continues.

(Refer Slide Time: 25:46)



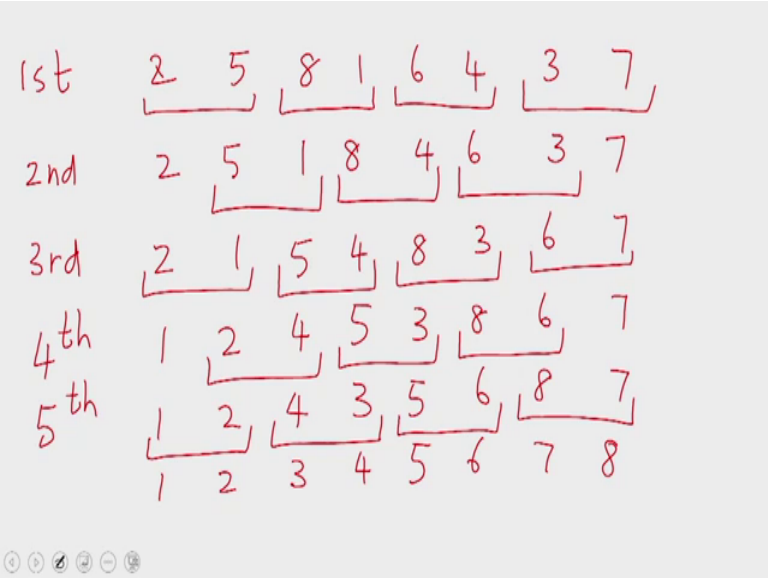
Therefore in general what we can argue with this about the 2nd phase by the Nth step all the input is received. So, the processor can start sending out messages from the next step. So, the output which is the smallest of all the elements this happens in the N plus step.

The output happens in the N plus third step you can see that the outputs are all spaced by 1 step.

That is between 2 consecutive outputs there are 2 steps, the 1st output happens in the 6th step whereas, the 2nd output happens in the 8th step. So, continuing like this the 3rd output happens in the N plus 5th step and so on therefore, the Nth output which is the last of the outputs happens in step number N plus 2 N minus 1 which is 3 N minus 1 so, by the 3 N minus 1st step you have received all the outputs.

So, here you should see that the elements will get sorted only by the 2 N minus 1st step, but many outputs would already be received by that step. The output will start coming out from the Nth N plus 1st step therefore, the time complexity of the algorithm is 3 N minus 1 the algorithm uses N processors a linear array of N processors and runs in 3 N minus 1 steps. So, this is the algorithm on the ion and interconnection model that we have seen, on a linear array on a streaming in input can be sorted in 3 N minus 1 steps when the size of the array is N. Now, in every situation this may not be what we want.

(Refer Slide Time: 28:23)



Here we consider an input that was streaming in suppose the input was already stored in an array in that case how would you sort the array for example, let us say we have a linear array that contains these elements in this order. So, let us say these are the contents of the consecutive processes of a linear array, to sort them using the previous algorithm

we would have to output these elements. And then input them back into the array and sort them as we did before.

So, to get them out of the array we will have to spend N steps and then the 3 N minus 1 steps algorithm I will have to be executed for a total of 4 and minus 1 steps, but as it happens there exists an algorithm which is much faster, this algorithm is called the odd-even transposition sort. So, in the odd even transposition sort we group the elements into groups of size 2 each and then sort each group. In the next step the grouping is different so, let me demonstrate.

In the 1st step I will group the elements in this fashion. The two elements will form a group the next 2 elements will form a group the next 2 elements will form a group and so on. And then each group is sorted by comparing and exchanging. So, within the 1st group I have 2 and 5 we compare 2 with 5 we find the 2 smaller than 5 so, there is nothing to do. So, the two elements will be retained as it is, when we look at the next pair of elements which is 8 and 1 we find that 8 is larger than 1. So, when we sort them one should come first.

Therefore we compare them and then exchange them. After the exchange 1 will come to the left side and 8 will come to the right side, mind you all these elements are on linear array therefore, there is a connection between adjacent elements. So, such a swap is possible by sending the elements as a message between 2 adjacent process, when we compare the next pair of elements we find that they also should be exchanged to be in sorted order. The last pair is already in sorted order. So, there is nothing to do so, in the 1st step we have done is to pair off the elements from the left hand within each pair we compare the elements and exchange them if necessary this exchange happens along the interconnection in the linear array.

Now, in the next step we will group the elements in this fashion, we leave out the 1st element and then we form groups of 2 each. And then again within each group we will compare and sort mind you all the groups are being sorted in parallel. So, this is only the 2nd parallel step whereas, this was the 1st parallel step. So, when we compare and exchange 2 is not being compared with any element so, 2 retains its original position 5 and 1 get is swapped 1 will come to the left and 5 will come to the right 4 and 8 also get swapped, 3 and 6 also gets swapped 7 is not being compared with any any element.

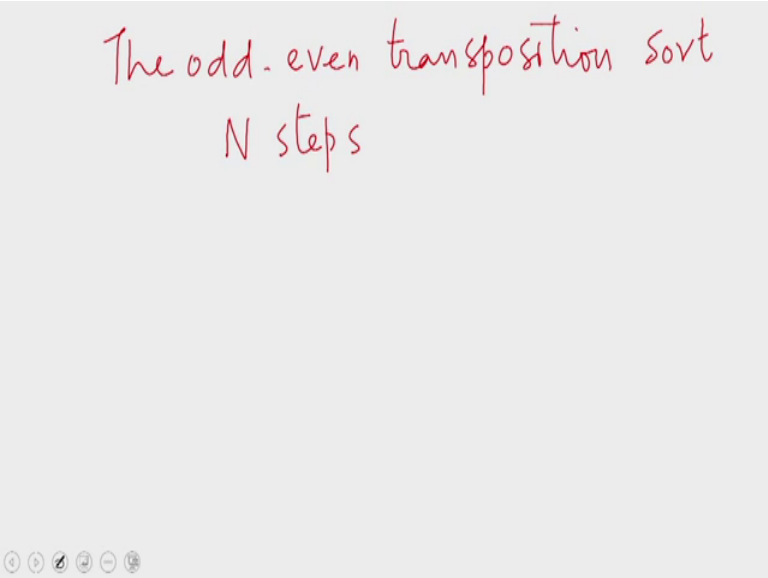
So, this is the content of the linear array after the 2nd step. So, this is how it looks like at the end of the third step at the beginning of the third step. In the third step we again go back to the original odd even pairing. So, in the step we had an odd even pairing and odd element is compared with the next even element whereas, in the 2nd step we have any even odd pairing and even element is paired with the next odd element.

In the 3rd step we again have an odd even paring so, 2 is paired with 1 5 is paired with 4 8 is paired with 3 and 6 is paired with 7 and we perform a comparison and exchange within each group. So, the 1st group is set right the 2nd 2nd group is set right the 3rd group is also set right the 4th group is already in sorted order. So, the content of the array will look like this at the beginning of the 4th step. The 4th step is again an even odd pairing step every even element is paired with the next odd element.

So, 2 is paired with 4 5 is paired with 3 8 is paired with 6 and then we perform a compare and exchange 1 is without a pair, therefore 1 remains as it is do end for our only in sorted order 5 and 3 are set right to 3 and 5 8 and 6 are set right to 6 and 8 7 is a loner. So, this is how the array looks like at the beginning of the 5th step.

Now, again we have an odd even pairing 1 and 2 are paired 4 and 3 are paired 5 and 6 are paired 8 and 7 are paired, when we compare and exchange in the 5th step we will have 1 2 3 4 5 6 and 7 8. So, the elements are now in sorted order. So, without having to take out all the elements and then stream them back in and use the 1st algorithm we have managed to sort the set of elements in place. The elements have always been within the linear array and in every single step we compare adjacent elements and exchange them if necessary, this algorithm is called the odd even transposition sort.

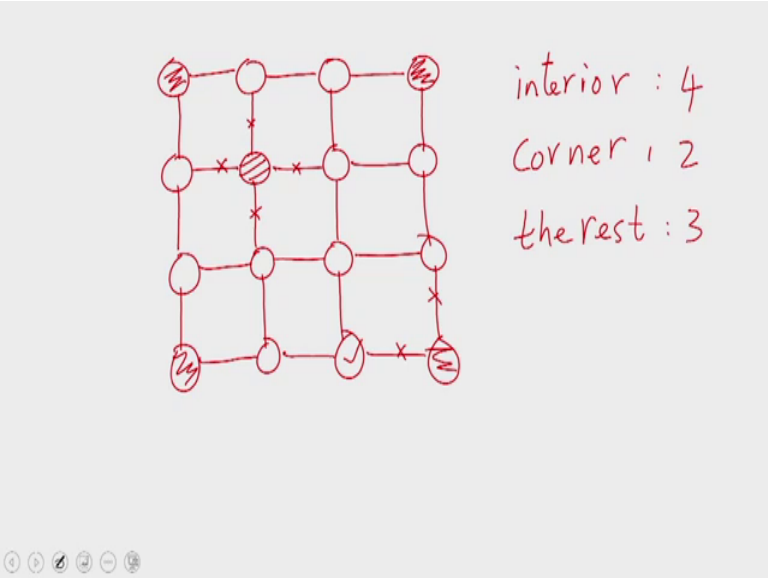
(Refer Slide Time: 34:07)



The odd even transposition sort algorithm as I explained proceeds like this. In the 1st step we pair odd elements with the next though the odd elements with the next even elements, in the 2nd step we pair even elements with the subsequent order elements and so on. And then within each pair we perform a comparison and exchange if necessary. So, that the elements will be in sorted order, if we continue this process finally, the elements will be sorted we shall see the correctness and the analysis of the algorithm later, but at the moment I will tell you that this algorithm terminates in N steps. Which is far better than the 4 N minus 1 we proposed earlier, that is if you stream out all the elements and use the 1st algorithm we saw which runs in 3 N minus 1 steps the total number of steps is 4 N minus 1. So, this is far better than that.

So, we have seen a couple of algorithms on the linear array. Now, now let us move on to an interconnection network that is slightly more complicated.

(Refer Slide Time: 35:27)

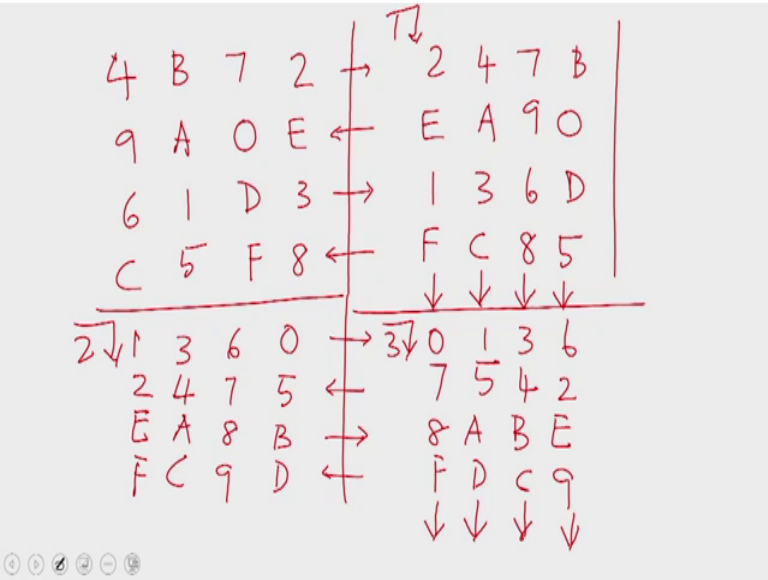


In this case the processes are arranged in a two dimensional array and the processors are interconnected using horizontal edges as well as vertical edges.

So, here you find that there are 3 kinds of nodes there are the interior nodes; these are the interior nodes the interior nodes have 4 edges 2 horizontal edges going out of it and 2 vertical edges going out of it. So, interior edges have a degree of 4, then there are the corner edges the corner wages have one horizontal and one vertical edge going out of it. So, the corner edges have a degree of 2 each, any node which is neither an interior edge nor a corner edge is a boundary edge of the sort. These edges have a degree of 3.

So, we can say that the maximum vertex degree of a node in a 2 dimensional mesh is for the boundary edges can have a degree of 2 or 3. The corner edges in particular have degree of 2 and the boundary edges that are not in the corner will have the degree of 3.

(Refer Slide Time: 37:50)



Now, once again let us see how we start on a two dimensional array. Let us say we have the elements distributed on a two dimensional array in this function, if B is the hexadecimal B, let us say we have a two dimensional array two dimensional mesh in which the elements are stored in this fashion and these elements have to be sorted. So, here A B C D E F are the hexadecimal values 10 11 12 13 14 and 15 so, to sort them what we do is this.

In the 1st step of the algorithm we sort the rows horizontally, the odd rows are sorted from left to right and the even those are sorted from right to left, the odd rows are sorted from left to right therefore, the 1st row will be sorted into 2 4 7 and B.

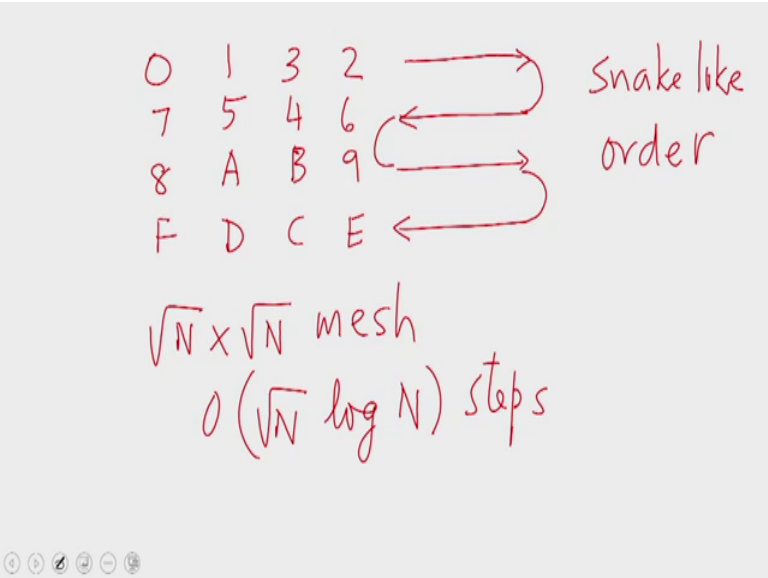
And the 2nd row is sorted from right to left therefore the 2nd row will be from right to left 0 9 A and E and the 3rd row again is sorted from left to right. So, we have 1 3 6 and D. And the 4th row is sorted from right to left we have 5 8 C and F. So, this is what happens in the 1st step in the 1st step every row is sorted the odd rows are sorted from left to right and even those are sorted from right to left for sorting the rows we use the odd even transposition sort which we have just seen. So, if we have an N by N mesh then a row has a size of N and a column has a size of N therefore, the row can be sorted in N steps using the odd even transposition sort. So, this step can be executed in N parallel steps.

Now, in the 2nd phase with in the 2nd step of the algorithm, we will sort every single column of the linear array the columns are all uniformly sorted from top to bottom. So, when you sort the columns from top to bottom the array that we get is this 1 2 E F is how the 1st column looks like. The 2nd column is 3 4 A and C and the 3rd column is 6 7 8 and 9 and the 4th column is 0 5 B and D. So, here every column is sorted from top to bottom using odd even transposition sort therefore, the time taken is the size of the array which is N if we consider an N by N mesh.

Now, in the 3rd phase of the algorithm which is similar to the 1st phase we sort the rows, but we change the direction of the alternate rows the 1st row sorted from left to right. Therefore, the 1st row will contain 0 1 3 and 6 the 2nd row sorted from right to left the 2nd row will contain 2 4 5 and 7 the 3rd row is sorted from left to right again. So, the 3rd row will contain 8 A B and E the 4th row is sorted from right to left. So, it will contain 9 C D and F.

So, after the 1st 4 phases the array will look like this. Now, in the next phase we once again sort the elements top to bottom, this is the initial array this is the array after the 1st step, this is the mesh after the 2nd step and this is the mesh after the 3rd step. Now, in the 4th step we will sort the columns vertically.

(Refer Slide Time: 42:35)

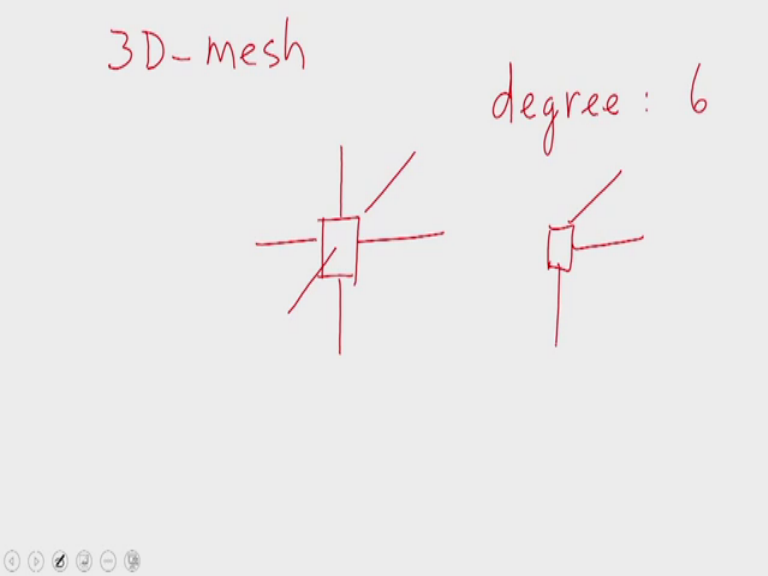


The 1st column therefore, will be 0 7 8 and F the 2nd column will be 1 5 A and D the 3rd column will be 3 4 B and C. And the 4th column will be 2 6 9 and E do they look sorted

now, they do except in that the alternate rows are in reverse order that is the 1st row is sorted like this, the 2nd row is sorted like this the 3rd row is sorted from left to right and the 4th row is sorted from right to left..

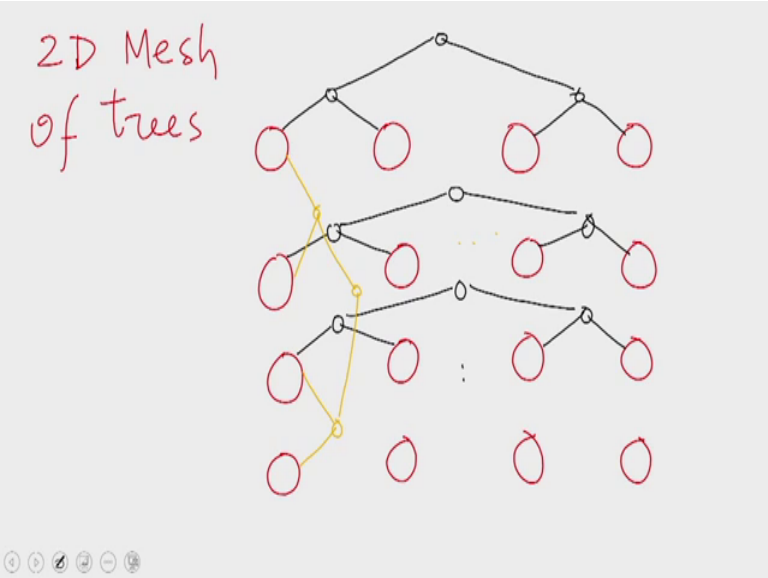
So, combining them we find that the elements are in sorted order. So, we can say the elements are sorted in a snake like order and you can see that the elements get sorted surprisingly fast in fact, we can show that on a root N by root N mesh the elements can be sorted in order of routine login steps. The analysis as well as the correctness proof we shall see later 2 dimensional measures can be further generalized into 3 dimensional measures.

(Refer Slide Time: 44:13)



In a 3 D mesh a processor has neighbors to the left to the right above below and also in the front and to the back. So, an interior processor in a 3 dimensional mesh has a degree of 3 degree of 6, 2 in each dimension of course, the boundary processors will have 5 more degrees in particular a corner processor will have only one neighbor along each dimension. So, the corner processors will have a degree of 3 each and then we can consider 4 dimensional measures 5 dimensional meshes and so on increasing the connectivity of the meshes.

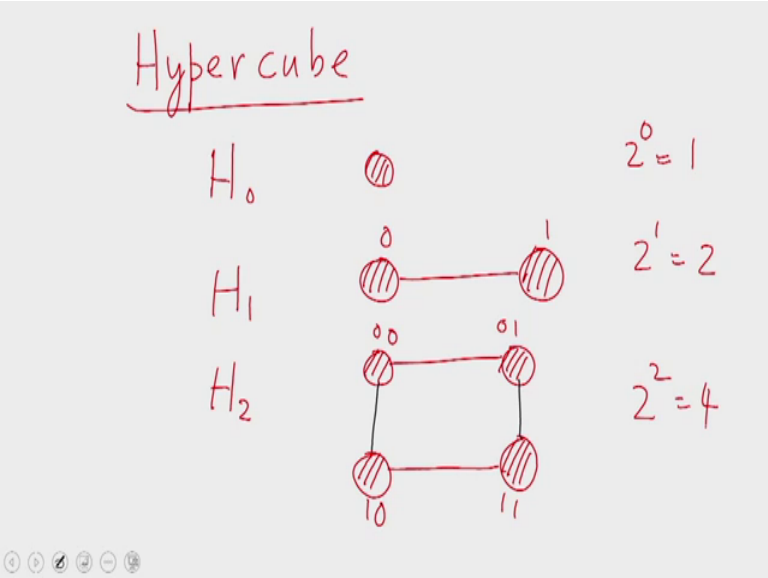
(Refer Slide Time: 45:06)



When we have a 2 dimensional mesh of processors instead of connecting them using horizontal and vertical edges, we could connect them using binary trees that hold auxiliary processors for example, you could construct a binary tree over the 1st row of the measure process mesh of processes in this fashion similarly over the 2nd row we construct a binary tree and so on..

Similarly we could construct processors we could use auxiliary processors to construct binary trees over the columns as well, we could construct the binary tree over the 1st column in this fashion. Similarly we construct a binary tree of every single column. So, what we get is 2 dimensional mesh of trees, we shall also see several endure connection networks that are closely related to hyper cubes.

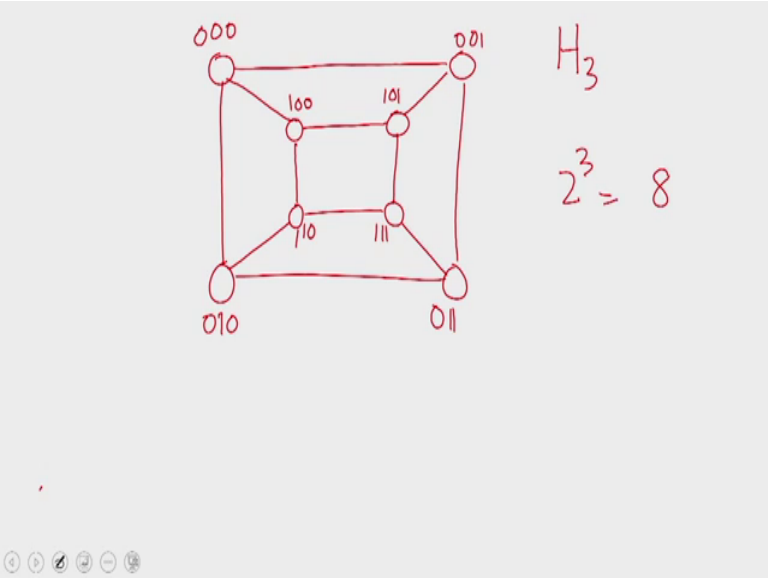
(Refer Slide Time: 46:52)



I will define what a hypercube is in this lecture and during the course of the program. We shall see several interconnection networks that are related to hyper cubes, a hypercube of dimension 0 is a single processor without any connection a hypercube of dimension 1 is constructed by taking 2 copies of H 0 and connecting the nodes. And then to construct a hypercube of dimension 2, we take 2 copies of H 1 and connect the corresponding nodes.

We can name the nodes of a hypercube using binary strings. So, let us say the notes of H 1 are named in this fashion the left nodes named 0 and the right notice named 1 to construct H 2 we took 2 copies of H 1. So, in them the left node is name 0 and the right node is labeled 1, but to distinguish the 2 copies we will add 1 extra bit. So, let us say the top copy is labeled 0 and the bottom copy is labeled 1. So, a hypercube of dimension 2 can be labeled in this fashion.

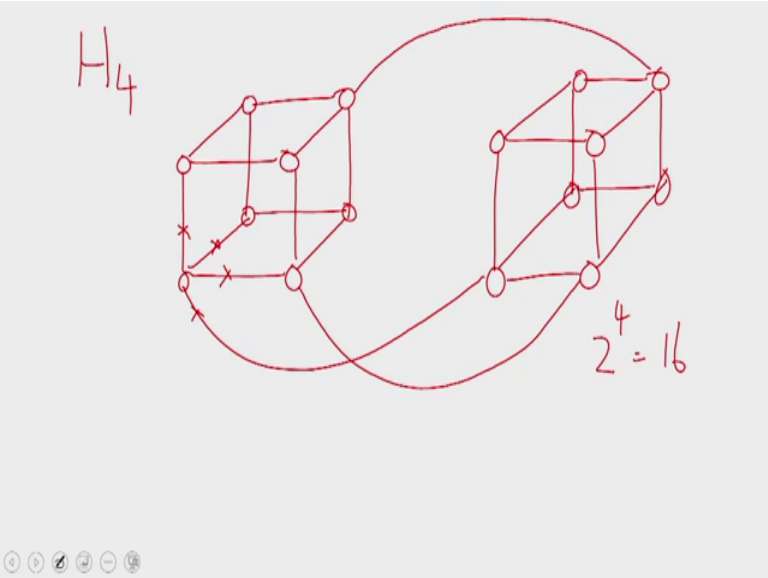
(Refer Slide Time: 48:52)



Then a hypercube of dimension 3 can be obtained by taking 2 copies of 2 dimensional hyper cubes.

So, this is one copy this is another copy and then we connect the corresponding nodes, nodes that occupy the same related positions are connected. So, these nodes are named in this fashion 0 0 and 0 1 in along the top page and 1 0 and 1 1 along the bottom nature and here also the same. So, you can see that nodes of the same name are connected between the 2 copies. Now to distinguish the 2 copies we will add 1 extra bit. The 4 nodes of the outer cube are given a label of 0 and the 4 notes of the inner cube are given label of 1. So, this is a hypercube of dimension 3.

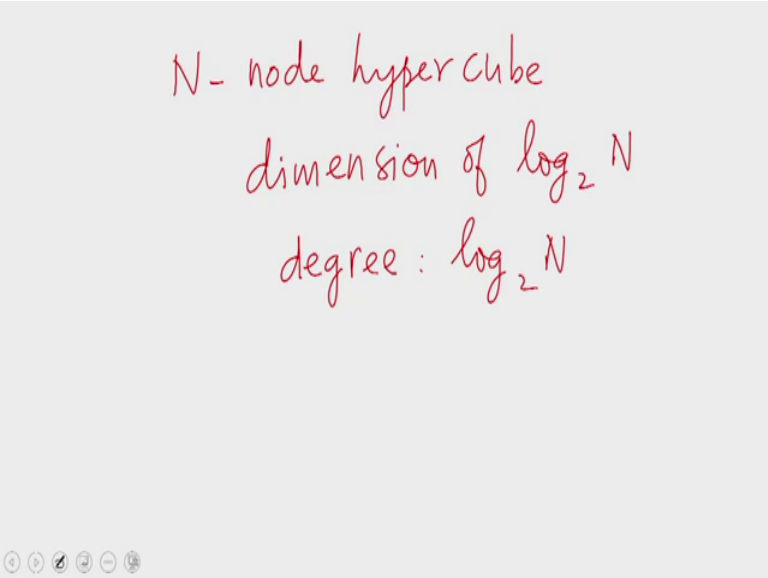
(Refer Slide Time: 50:07)



Now, coming on to the 4th dimension we take 2 hyper cubes of dimension 3 and inter connect the corresponding nodes. So, this is this node corresponds to this, these 2 are corresponding nodes they are connected and these 2 are corresponding nodes and they are connected, these 2 are corresponding nodes and they are connected and so on. So, this is how you construct?

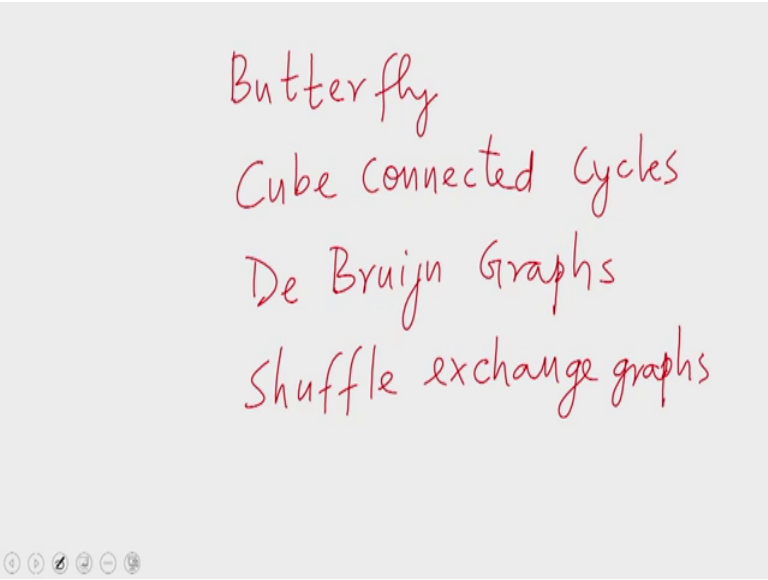
A hypercube of dimension 4 so, you can see that as you go from one dimension to the next the number of nodes in the hypercube doubles in H 0 there is only 1 node 2 power 0 equal to 1 in H 1, there are 2 nodes 2 power 1 equal to 2 in h 2 there are 4 nodes 2 part 2 equal to 4 in H 3 there are 8 nodes 2 power 3 equal to 8 and in H 4 there are 16 nodes 2 power 4 equal to 16 and the vertex degree is equal to the dimension of the cube for example, in H 4 a node has a degree of 4 example this corner node has 4 adjacent edges every vertex has exactly the same vertex degree here.

(Refer Slide Time: 52:02)



So, in general if you take an end node hypercube. This has a dimension of log N, when you have a hypercube of N nodes, then N would have to be a power of 2 and therefore, log N is an integer log into the base 2 is an integer. So, we consider an N node hypercube in this the degree of every node is log N which is equal to the dimension. So, in general in H k a hypercube of dimension k the degree of every single node is k, there are several inter connection networks that are closely related to hyper cubes namely.

(Refer Slide Time: 53:00)



The butterfly networks the cube connected cycles De Bruin graphs, Shuffle exchange graphs and many others we shall see some of them during this course and we shall design algorithms on them and analyze the algorithms that is it from the 3rd lecture hope to see you in the next lecture.

Thank you.