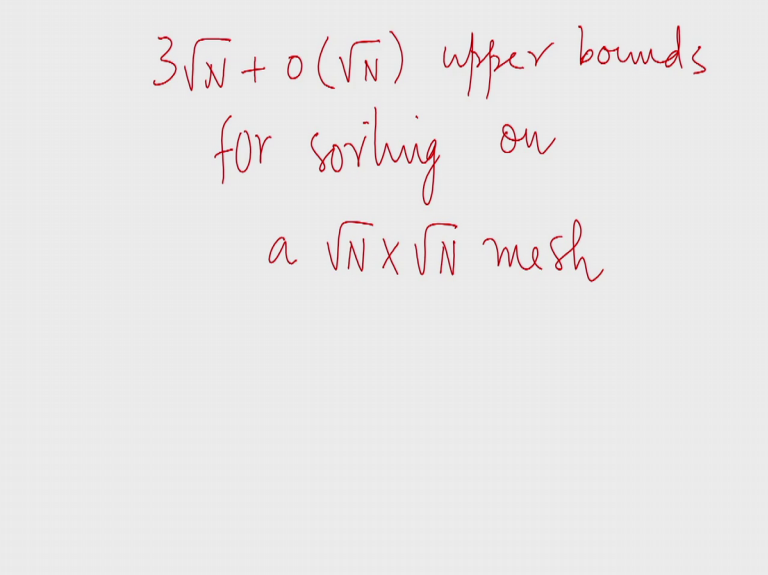
Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

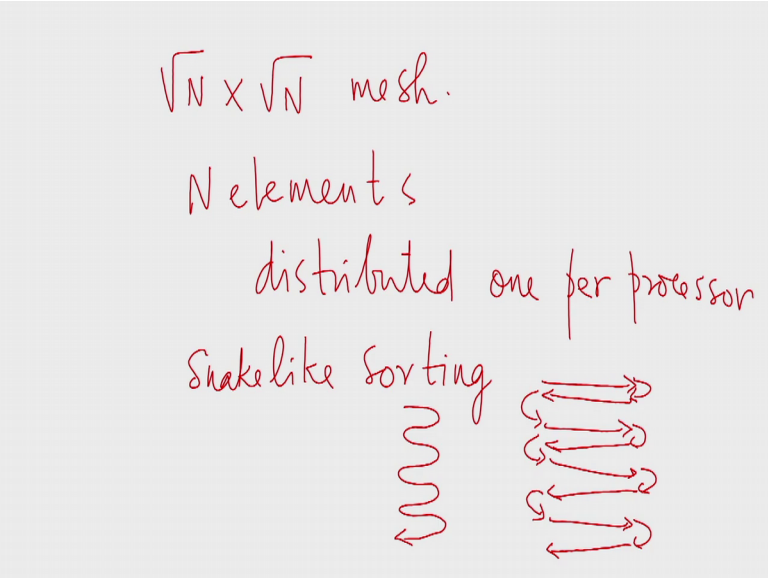
Lecture – 26
Sorting, Offline routing on a 2D mesh

Welcome to the 26th lecture of the MOOC on Parallel Algorithms, continuing with the algorithm that we were discussing in the previous class, we wanted to find a 3 root N plus small o root N time algorithm for sorting N items on a root N by root N mesh.

(Refer Slide Time: 00:42)

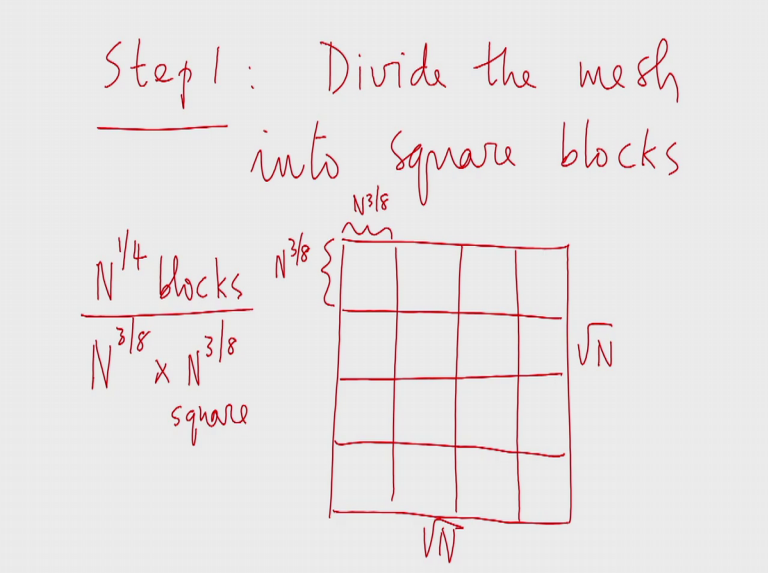


(Refer Slide Time: 00:44)



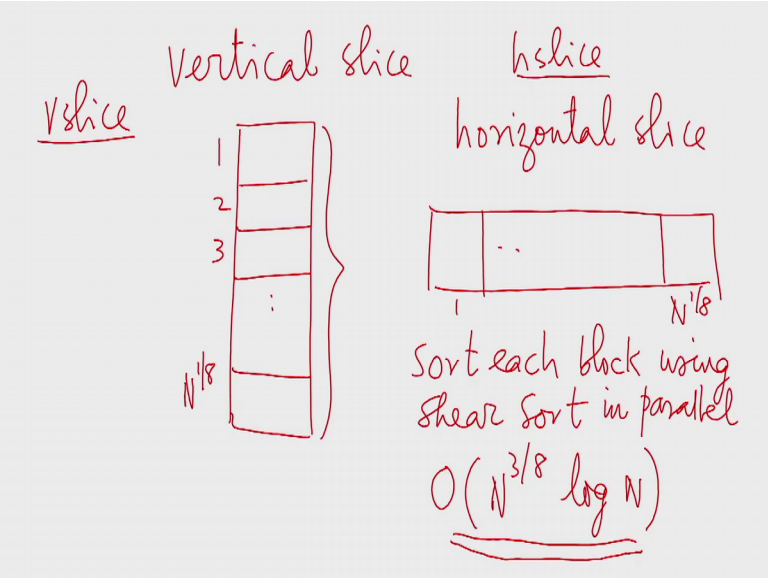
The items had to be sorted in a snake like order, we use the shear sort algorithm that we saw in a previous class as a subroutine here.

(Refer Slide Time: 00:52)



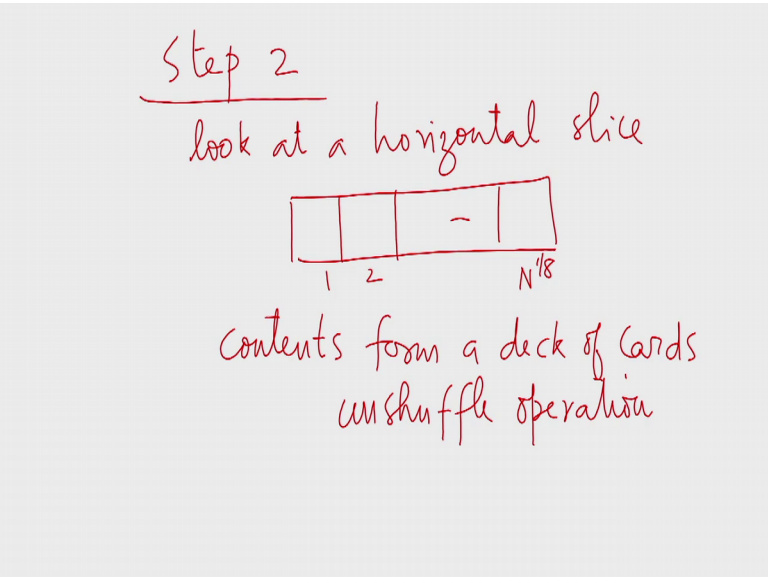
The algorithm proceeds in this fashion when we are given a root N by root N mesh in which every processor holds an item that is to be sorted. We divide the mesh into square blocks. Thus, the squares are of size N by N power 3 by 8 by N power 3 by 8. So, you have N power 1 by 4 blocks on the whole.

(Refer Slide Time: 01:14)



And then we sort each of these blocks in snake like order using shear sort. This sorting will take order of N power 3 by 8 times log N time. Vertical slice is a sequence of blocks stacked one over above, a top to bottom and a horizontal slice is a similar set of blocks that are arranged horizontally from the left to the right.

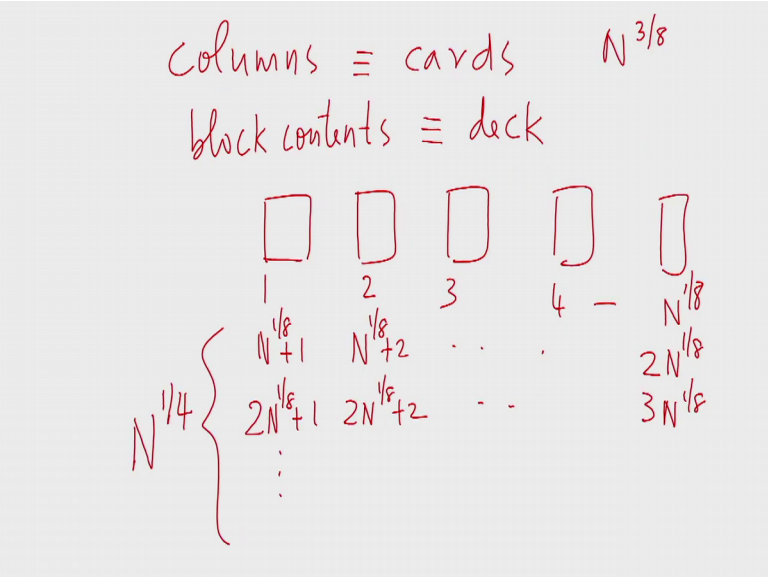
(Refer Slide Time: 01:36)



Then in the second step, we do an unshuffle operation. That is we consider the vertical slices, we take the contents of a vertical slice imagine that this is a deck of cards and then

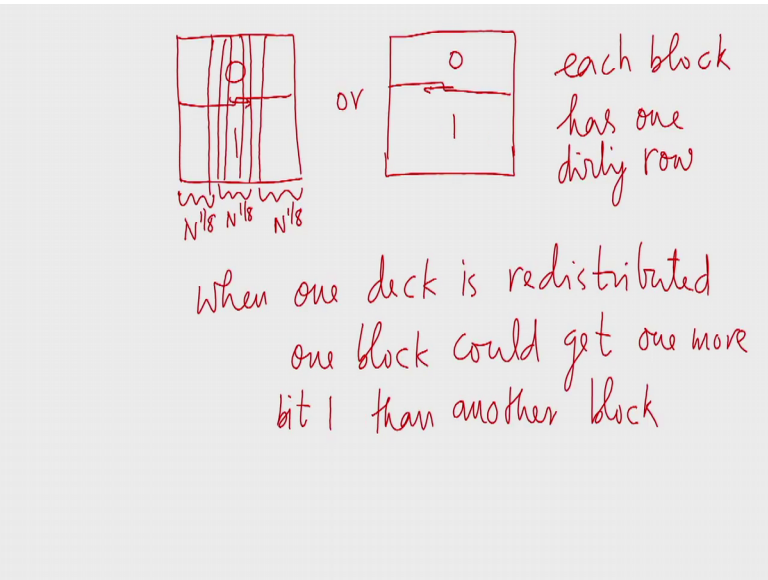
distribute these cards namely the columns over the vertical slices that is; we assign each card to a vertical slice. This we do with every single vertical slice.

(Refer Slide Time: 02:00)



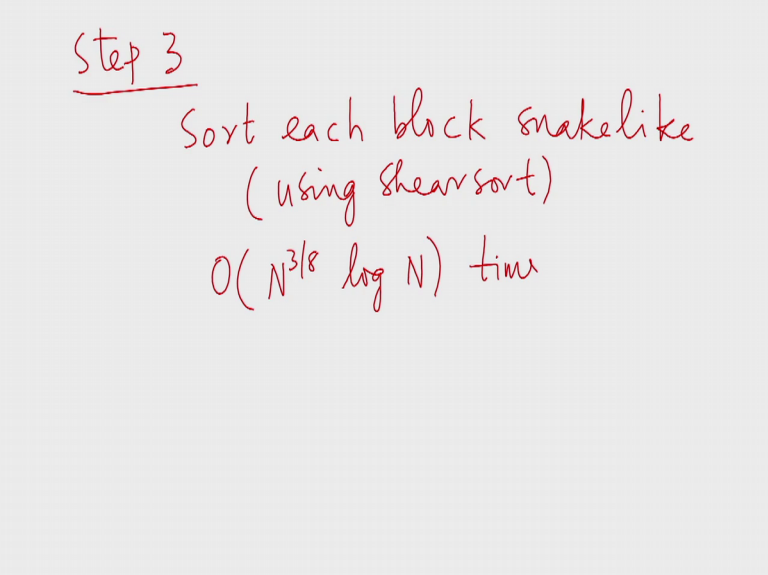
So, this unshuffle operation essentially involves routing the data items. So, every data item or the processes sitting on the data item statically calculates the destination at which the data item has to be sent. And, once every data item knows the destination where it should be going then all the data items can walk in lockstep until they reach the destination and fall in place.

(Refer Slide Time: 02:22)



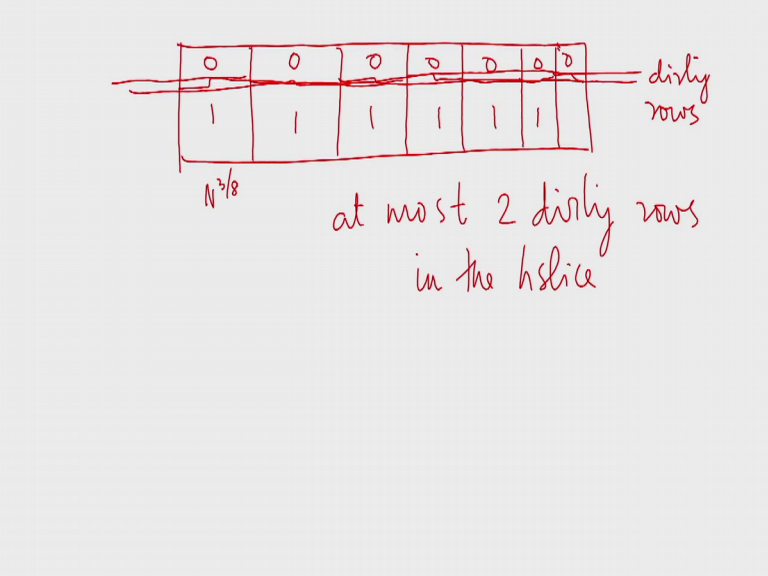
So, this will require root N plus small o root N time.

(Refer Slide Time: 02:27)



And then in the third step we sort each block in a snake like fashion, again using shear sort this takes order of N power 3 by 8 times log N time. So, so far the second step is the dominating step.

(Refer Slide Time: 02:39)

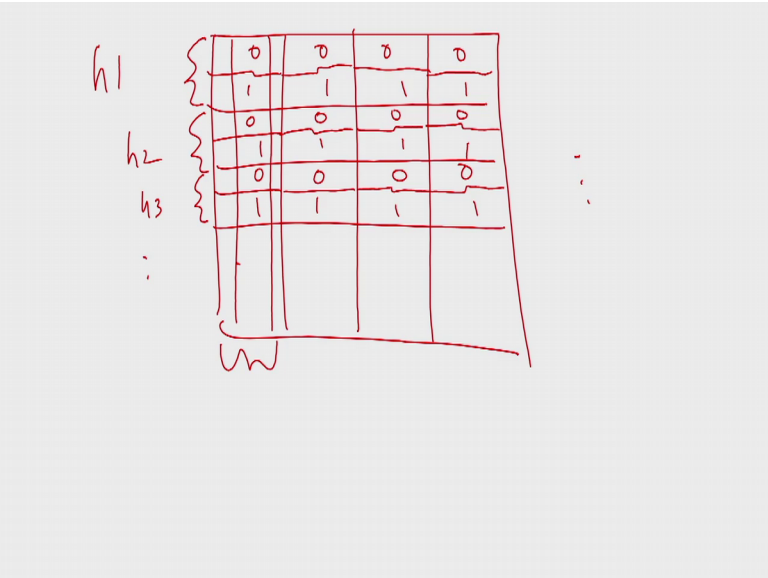


So, at this point we would have established that there are at most 2 dirty rows in every horizontal slice. This is because when we unshuffle the vertical slices in every horizontal slice when we considered two blocks what we understand is that when one, but one

block of a horizontal slice is unshuffled over the various blocks of the horizontal slice. The maximum gain that one block can get over another in the number of 1's is 1 and when this is summed over all the blocks of the horizontal slice we get N power 1 by 8.

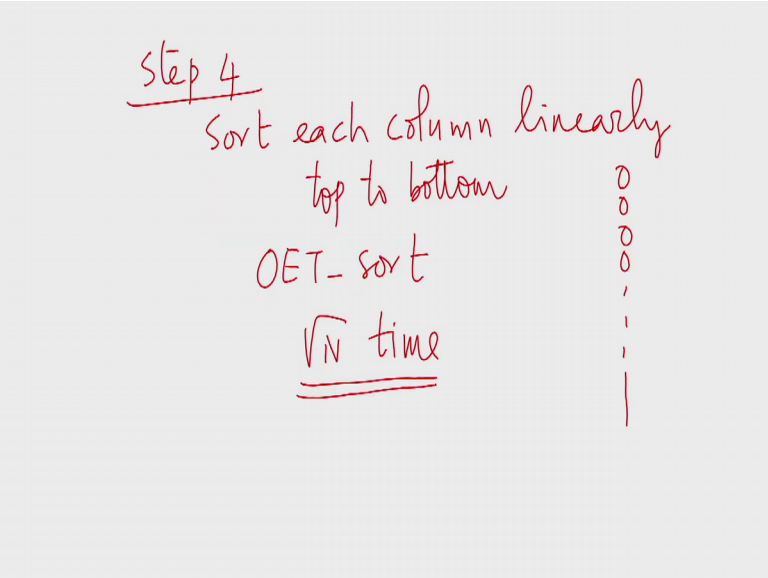
So, there are at most N power 1 by 8 elements N power 1 by 8 extra 1's in any block of the horizontal slice over another block of the same horizontal slice. This ensures that there are at most 2 dirty rows in every horizontal slice because, the width of a block is N power 3 by 8 which is far more than the N power 1 by 8 that we were talking about.

(Refer Slide Time: 03:36)



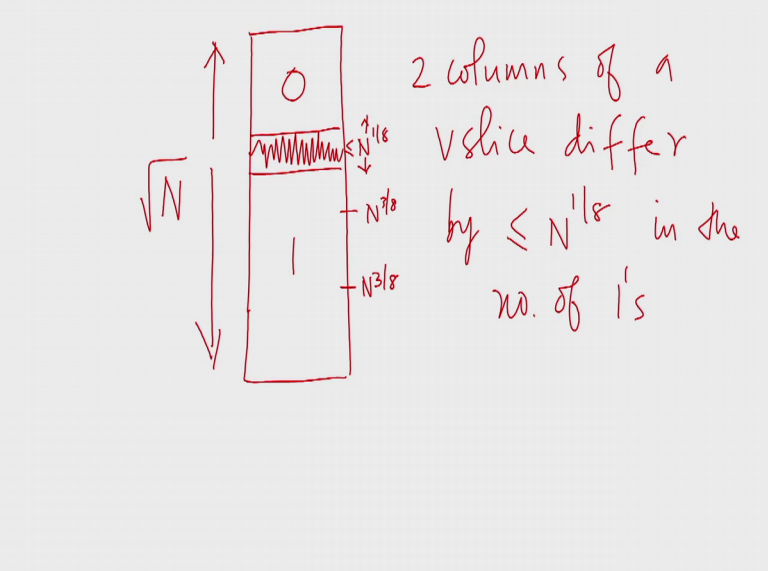
So, the horizontal slice is now look like they look almost clean with 1 kink then each block of each horizontal slice and these kinks are happening in 2 consecutive rows. So, every horizontal slice has 2 dirty rows.

(Refer Slide Time: 03:51)



And then in step 4, we sort every column this is akin to letting the 1's fall down in every column. So, now, the 1's settle at the bottom and the 0's are at the top.

(Refer Slide Time: 04:04)

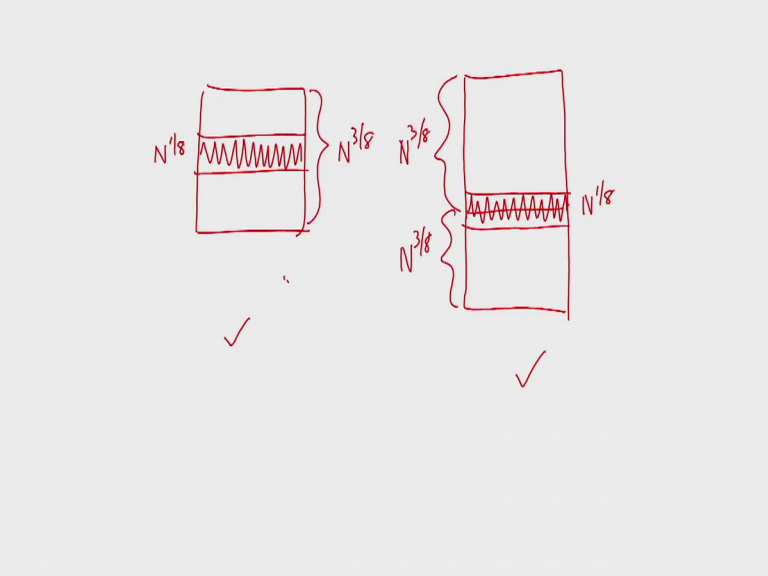


So, if you imagine that the 1's are buildings and the 0's from the sky then, the skyline of the dirty rows will occupy a very small band. They occupy a small band of size at most N power and by 8; that is because 2 columns of a vertical slice differ by at most N power 1 by 8 in the number of 1's. This is clear from the previous picture at the end of the third phase when you take 2 columns belonging to the same vertical slice they have almost the

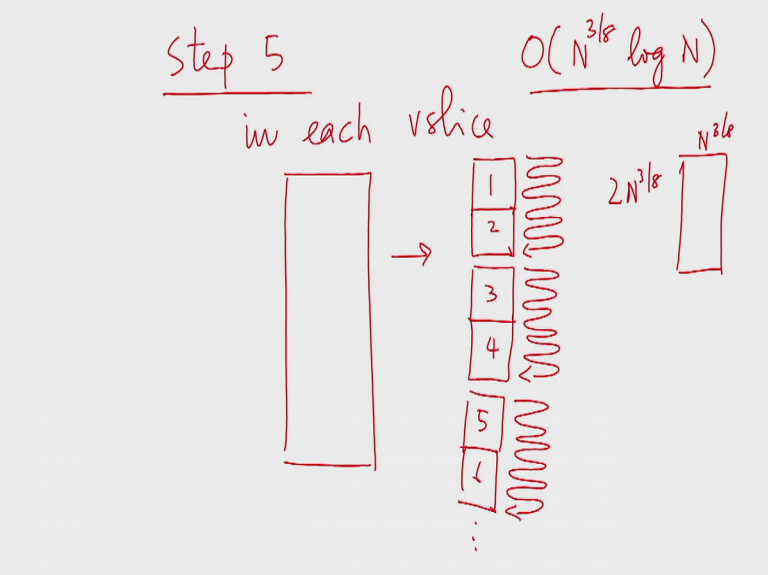
same number of 1's, the 2 columns will differ by at most one in each block of this vertical slice. Therefore, now we have a skyline which is very narrow compared to the width of the sky as well as the width of the buildings that is the number of 1's and the number of 0's.

The dirty band is of size at most N power 1 by 8, in a vertical slice each block has a height of N power 3 by 8. Therefore, this dirty band should be entirely within a block or it should span at most 2 blocks. So, it will be one of these two cases.

(Refer Slide Time: 05:05)

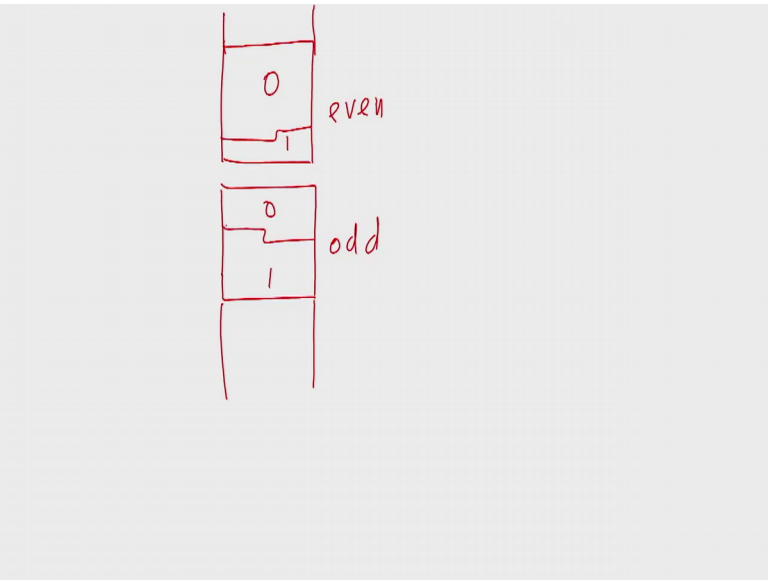


(Refer Slide Time: 05:09)

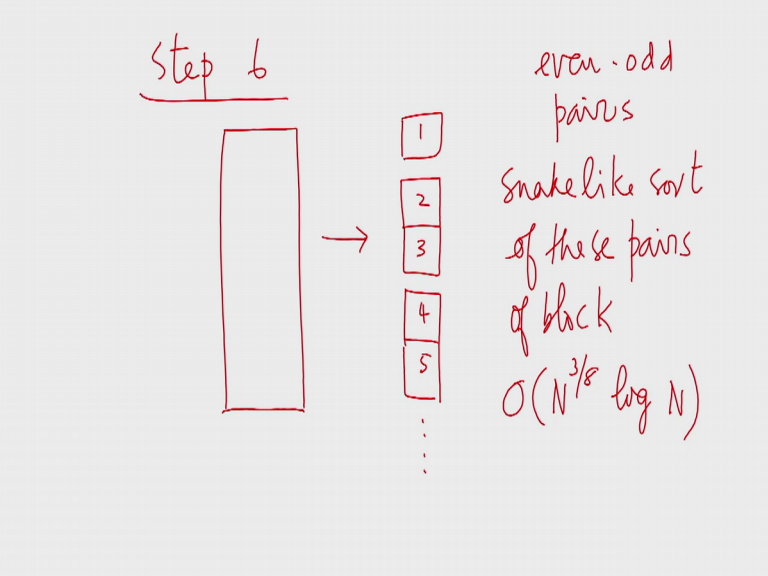


To handle these two cases what we do is to pair off the blocks belonging to a vertical slice and then sort each pair in a snake like fashion using our previous algorithm; the shear sort algorithm which runs in order of N power 3 by 8 times log N time. Since we do not know which pair of blocks the dirty band would overlap. It could overlap an even odd pair or an odd even pair. We consider both the pairs, first we consider odd even pairs that is 1 is paired with 2, 3 is paired with 4 and so on and within each pair we perform a sort. Now every pair is sorted, but then the dirty band could be spanning an even odd pair. To take care of that eventuality we consider the even odd pairs as well.

(Refer Slide Time: 05:49)

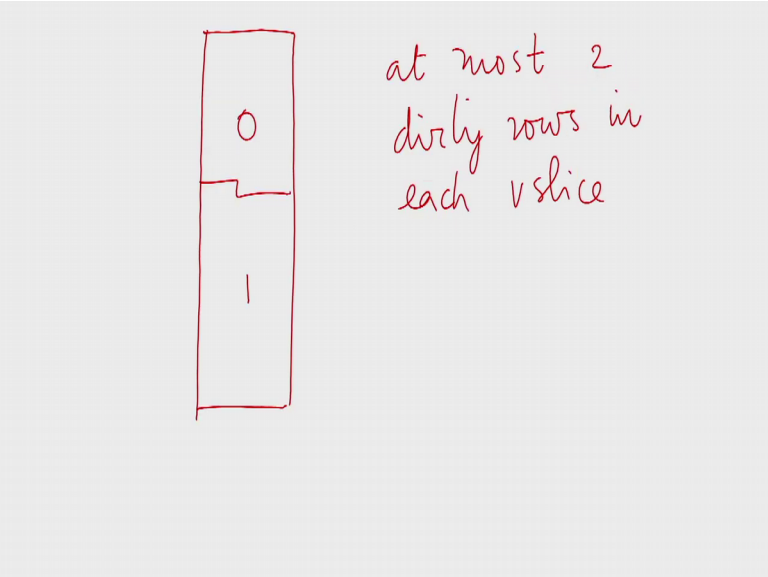


(Refer Slide Time: 05:52)



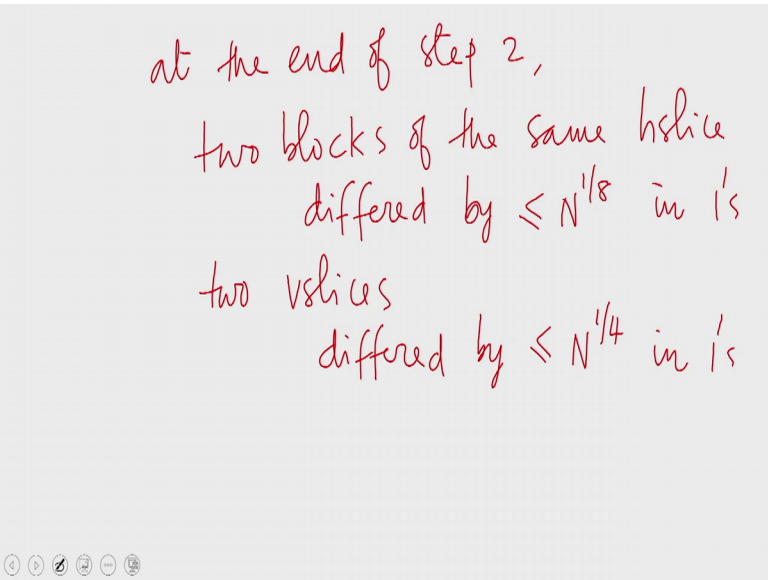
In step 6, we consider pairings of the sort and sort every pair. This will ensure that every vertical slices clean.

(Refer Slide Time: 06:01)



Now, there are; there is at most 1 dirty row in each vertical slice. This is wrong I should say there is at most 1 dirty row in each vertical slice, the vertical slice has just been cleaned up. So, every vertical slice has exactly 1 dirty row now.

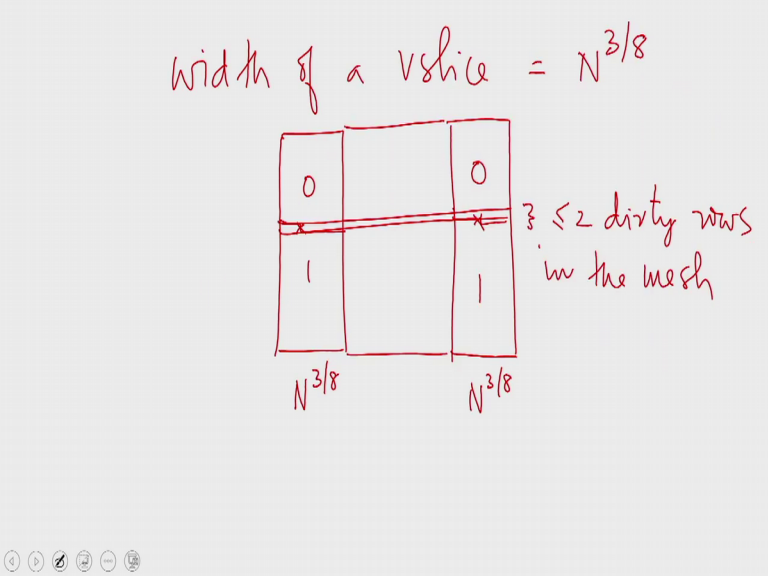
(Refer Slide Time: 06:20)



So, at the end of step 2, two blocks of the same horizontal slice differed by at most N power 1 by 8 in the number of 1's. So, if you aggregate over vertical slices, this is

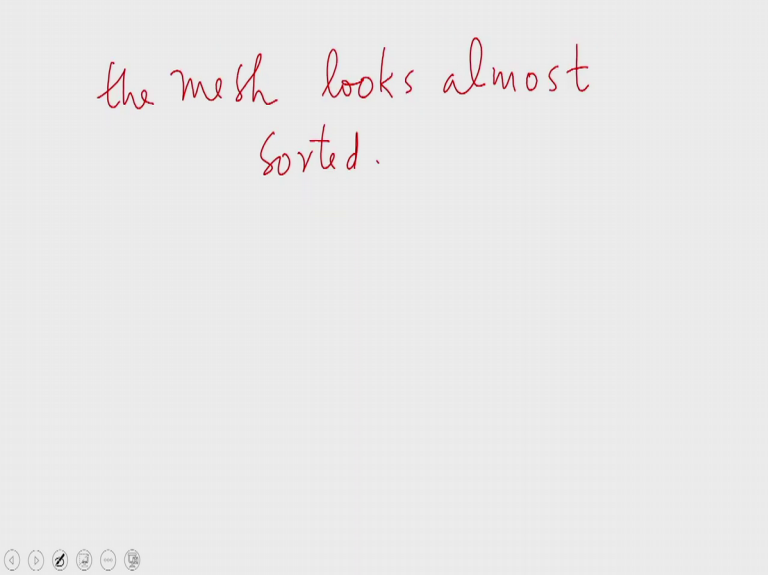
tantamount to saying that two horizontal slices; two vertical slices differ by at most N power 1 by 4 in the number of 1's present in them. This condition holds even now because after step 2, in steps 3, 4, 5 and 6 we did not have any data moving between the columns between vertical slices.

(Refer Slide Time: 06:52)



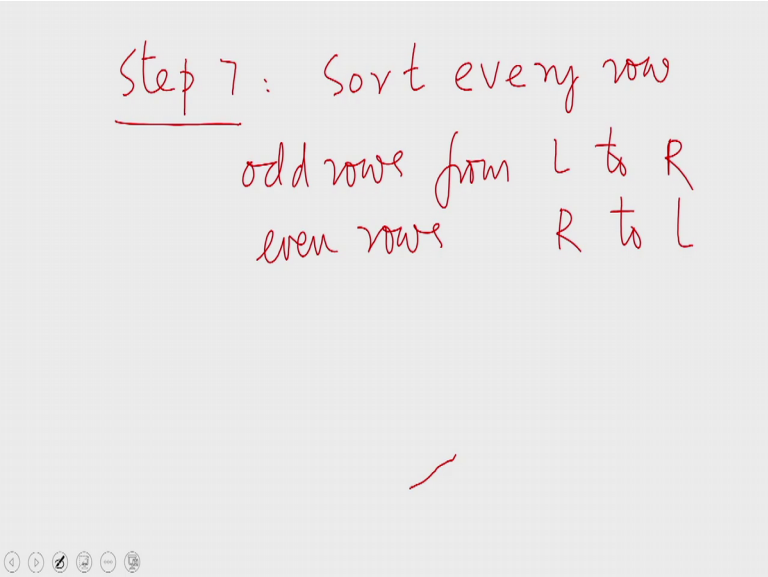
Therefore, this condition holds good even now and the width of a vertical slice is N power 3 by 8, since 2 vertical slices differ by at most N power 1 by 4 in the number of 1's they have; there could be at most 2 dirty rows in the whole mesh right now.

(Refer Slide Time: 07:10)



So, this is where we had stopped in the previous lecture. So now, the mesh looks almost sorted; all it all that is necessary is to clean up the remaining 2 dirty rows.

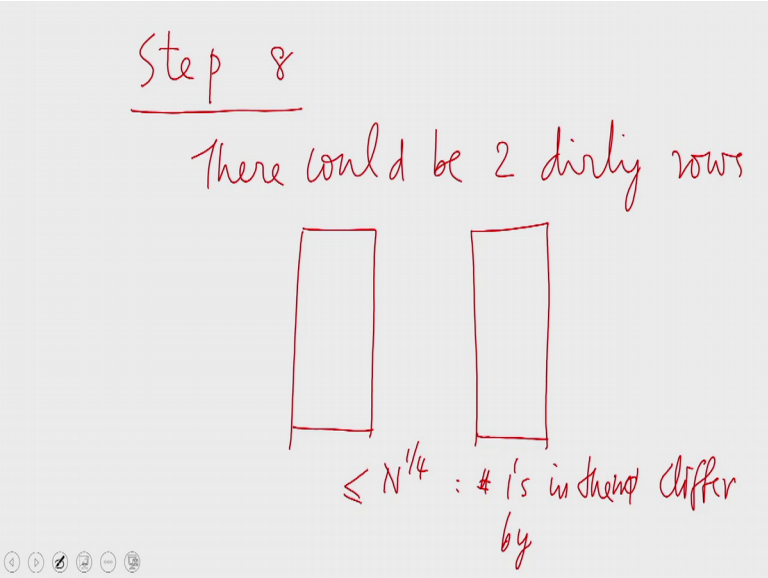
(Refer Slide Time: 07:20)



In step 7, we sort every row, the odd rows are sorted from the left to the right and the even rows are sorted from right to the left. This is the standard order that we have.

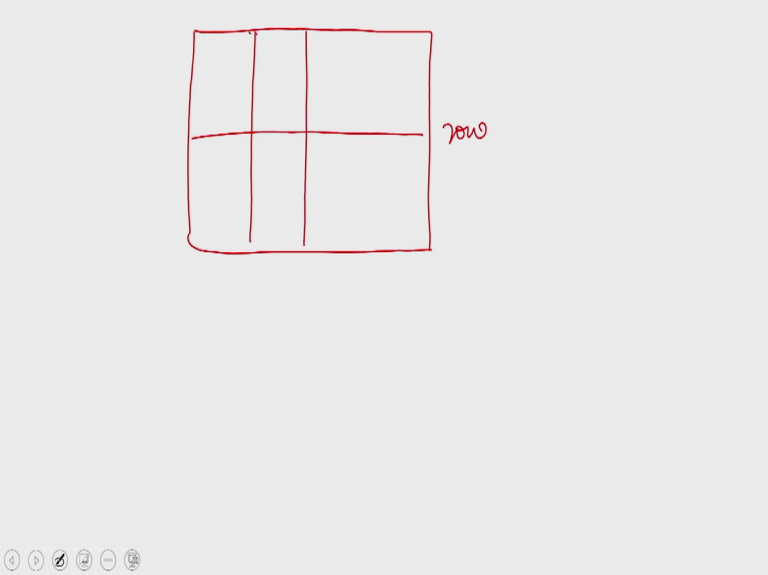
So, in step 7 we perform the sorting. Once this is done, we would have solved the case where there is exactly 1 dirty row. Had it been the case that there is exactly 1 dirty row at the end of step 6, then that would be fixed by now, but then there is also the possibility that there could be 2 dirty rows that is the case to be handled now.

(Refer Slide Time: 08:09)



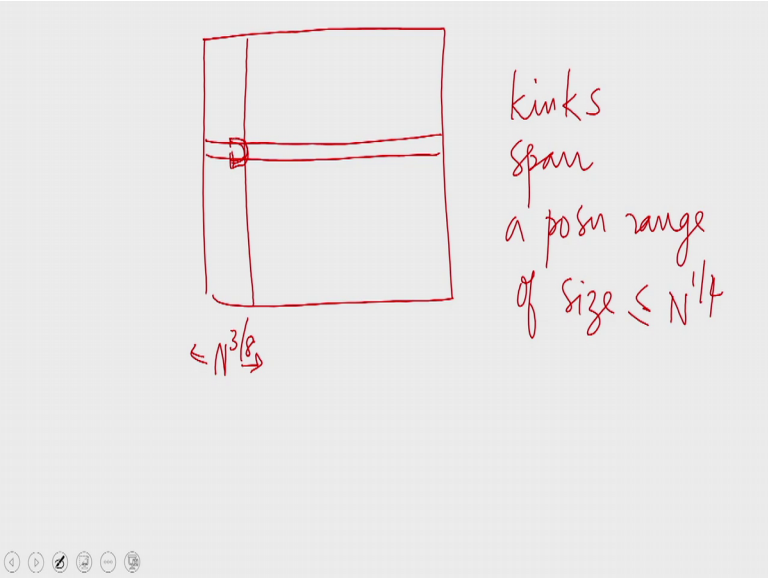
But we know that when we consider two vertical slices. The number of 1's in them differ by at most N power 1 by 4; the number of 1's in them differ by at most N power 1 by 4.

(Refer Slide Time: 08:58)



Therefore, in the whole of the array in the entire mesh. Now, when I have sorted the rows consider a dirty row that has been sorted now. Suppose the in a vertical slice the between two vertical slices we have at most N power 1 by 4 1's in other words when you take two vertical slices the number of 1's in one could be at most N power 1 by 4 more than the number of 1's in the other.

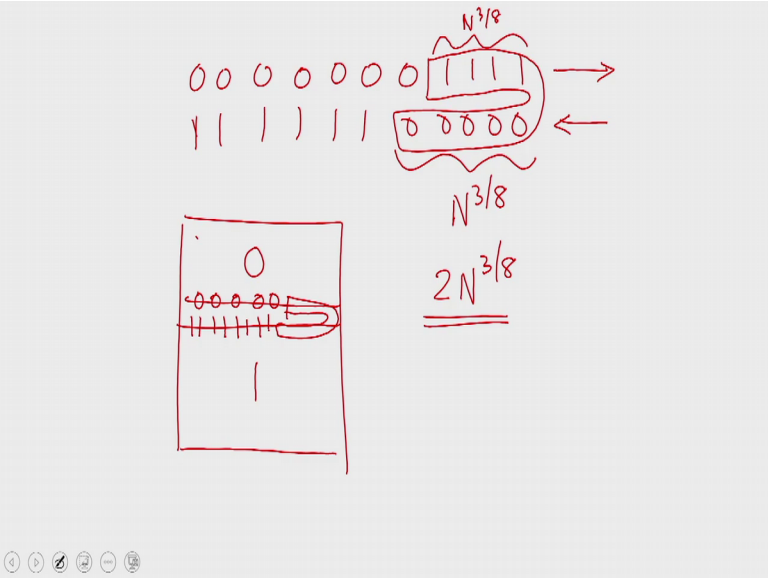
(Refer Slide Time: 09:30)



Therefore, if you consider the entire mesh now we know that there are 2 dirty rows, there are 2 adjacent rows that are both dirty. Therefore if you take a vertical slice, it should be that the kink in that is happening rather late that is why it is spilling over into the next row. This is because the kinks can span a position range of size at most N power 1 by 4 whereas, the width of a vertical slices N power 3 by 8 much larger than N power 1 by 4.

Therefore, if the there are 2 dirty rows, then the kink in the vertical slice is happening rather late in all the vertical slices. What this means is that the number of dirty, the size of the dirty window is quite small with respect to the size of the vertical slice. Or, in other words when you get the entire row sorted when the entire dirty row is sorted assuming that we are sorting them in this order. Assume arbitrarily that the top row is sorted from the left to right and that the bottom row is odd from the right to the left, it could just be the other way around.

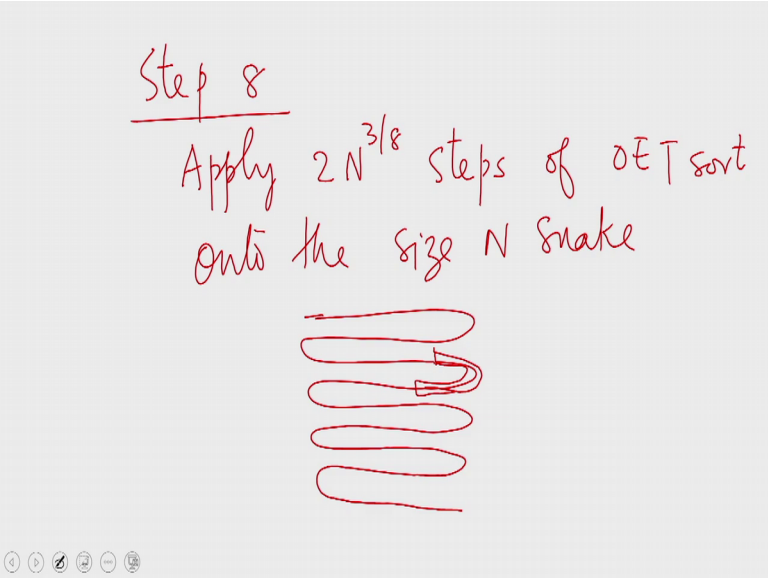
(Refer Slide Time: 10:49)



If this is the case then this range is at most N power 3 by 8 in size aggregating over all the vertical slices. Similarly, this is also at most N power 3 by 8 in size which means the dirty window overall is of size twice N power 3 by 8, this is a graph of a bound. The maximum size of the dirty window is twice N power 3 by 8.

So, the entire mesh is now clean except for a small dirty window that appears over 2 adjacent rows and the size of this dirty window is twice N power 3 by 8; everything above is 0 and everything below is 1. And, even within these rows prior to the dirty window all the elements are clean and after the dirty window all the elements are clean. So, now, all that is necessary is to get these dirty elements cleaned up. How do we get them cleaned up? That is where the next step comes in.

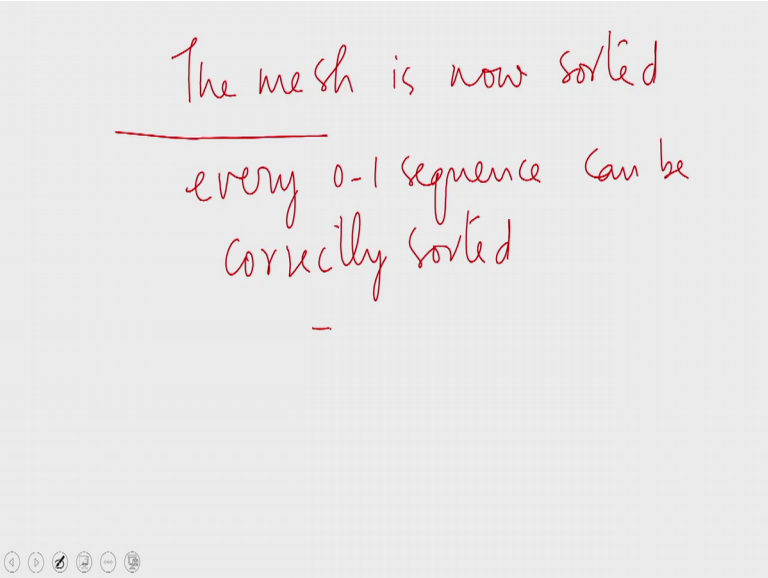
(Refer Slide Time: 12:24)



In step 8, we apply twice N power 3 by 8 steps of odd even transposition sort onto the snake, that is the mesh is now thought of as a linear array ignoring the remaining connections that is we use the vertical connections only when we turn at the right and left N's, otherwise we use only the horizontal connections. This snake like order can be thought of as a linear array this is a sub graph of the two dimensional mesh that we are dealing with.

So, in this we find that the dirty window happens to be somewhere, this dirty window has a size of at most twice N power 3 by 8. So, if you run the odd even transposition sort for twice N power 3 by 8 steps we end up cleaning those window, this window is all that that is to be cleaned up the rest of the meshes already clean. Therefore, now the sorting is accomplished, the mesh is now sorted.

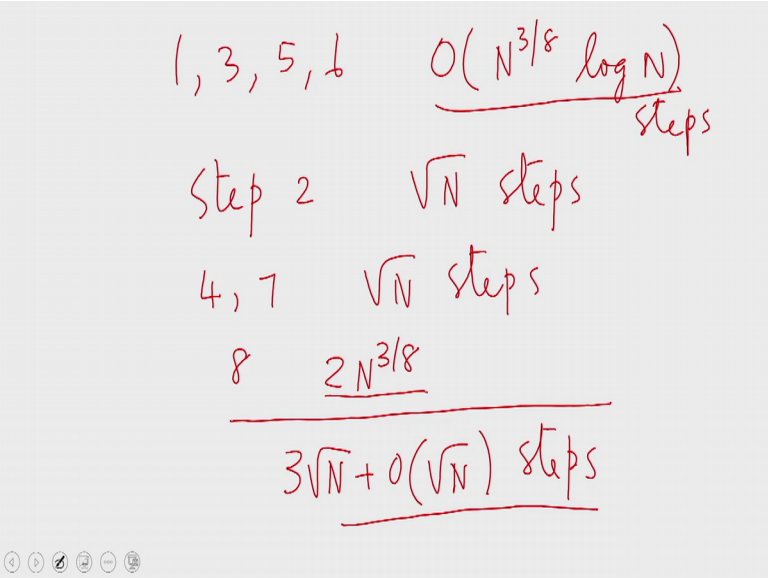
(Refer Slide Time: 13:44)



So, this establishes that every 0 1 sequence can be correctly sorted which means that, on any sequence run from a linearly ordered set this algorithm will work correctly. So, to do a recap what this algorithm does is to divide the mesh into several squares, sort each square in a snake like fashion and then unshuffle the vertical slices over the vertical slices. After that sort this twice once again and then sort every column linearly top to bottom and then pair of the blocks of every vertical slice initially 1, 2, 3, 4, 5, 6 etcetera.

And, then leaving out 1, 2, 3, 4, 5, 6, 7 etcetera even odd; in each of these pairings we sort the pairs in snake like order using shear sort, after this we sort each row linearly in the appropriate order. And finally, all that we have to do is to execute 2 N power 3 by 8 steps of odd even transposition sort. That would ensure that the original sequences in sorted order.

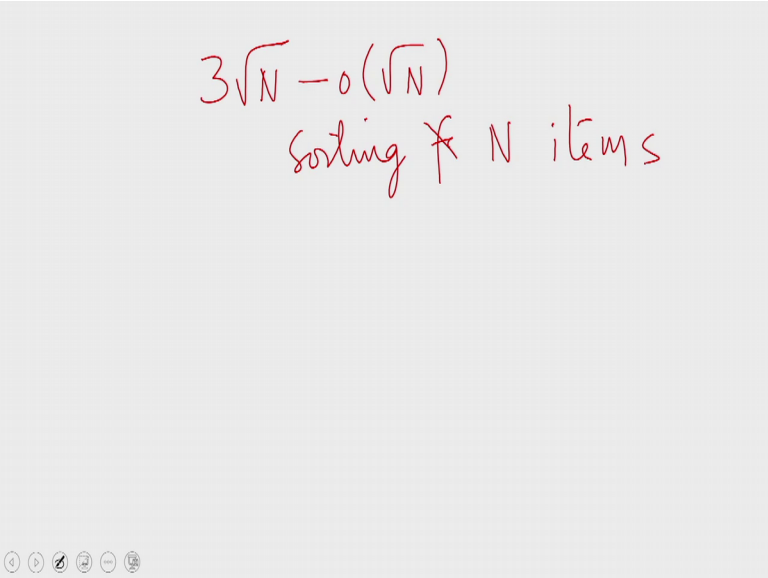
(Refer Slide Time: 15:12)



Now, what is the time complexity of this? Steps 1, 3, 5 and 6 as we have seen run in order of N power 3 by 8 times log N steps, step 2 runs in root N steps, steps 4 and 7 also run in root N steps, step 8 runs in twice N power 3 by 8 steps. So, summing all this together we find that the total time taken is 3 root N plus small o root N steps.

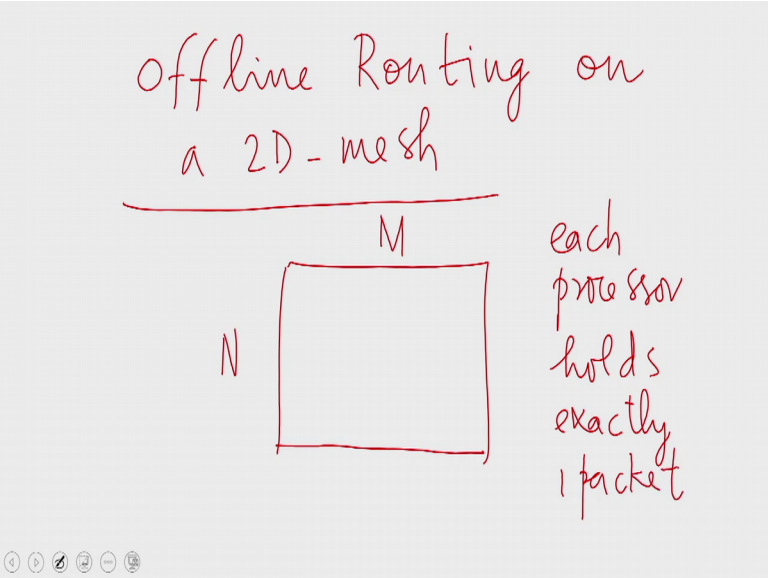
The remaining terms these are all small o root N; the time taken by steps 1, 3, 5, 6 and 8 are all small o root N. The dominating steps are 2, 4 and 7 these steps take root N time each. So, the total time taken is 3 root N plus small o root N. So, compare this with the lower bound that we saw in the last class.

(Refer Slide Time: 16:22)



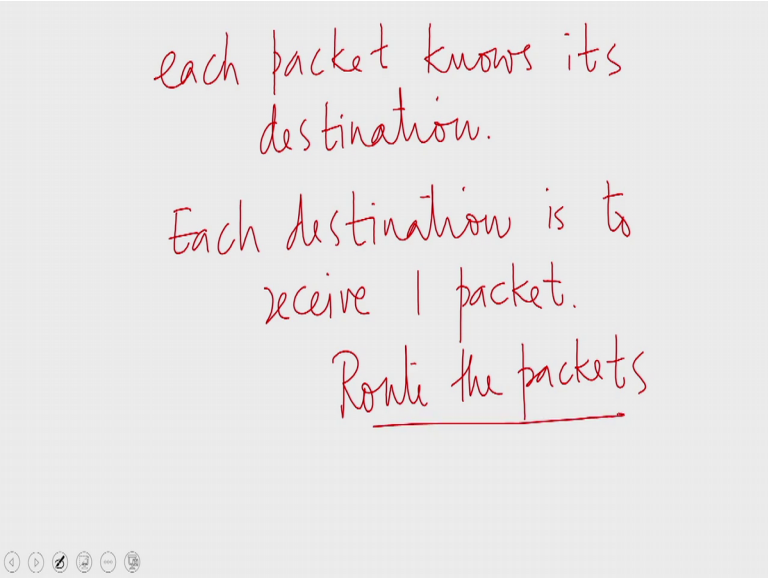
We established lower bound of 3 root N minus small o root N for sorting N items on a root N by root N mesh and we have found a 3 root N plus small o root N algorithm which very closely matches this lower bound ok. Now, we will see another algorithm on a 2 dimensional mesh.

(Refer Slide Time: 16:55)



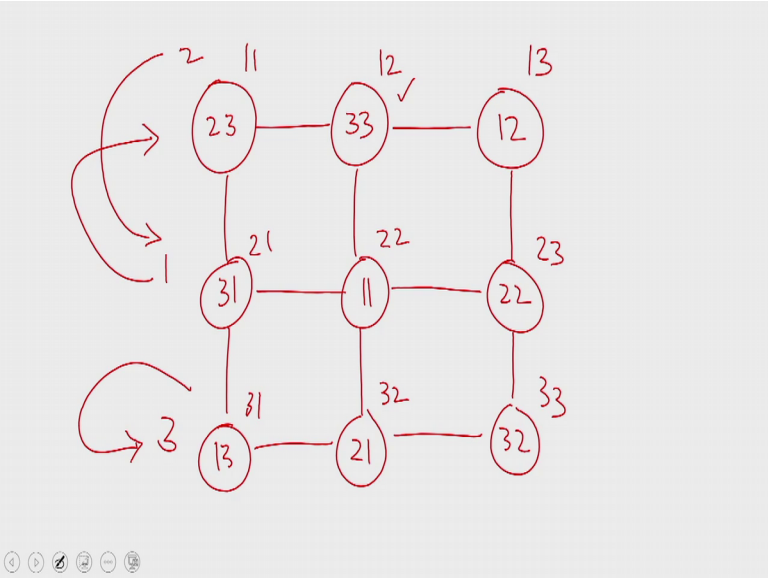
This is an offline routing problem, in this we assume that we are given an N by s mesh you given let us say an N by M mesh. There are N rows and M columns, we are given an N by M mesh in which each processor holds a packet, holds exactly 1 packet.

(Refer Slide Time: 17:54)



And along with the packet is the destination of the packet. So, we assume that each packet knows its destination. And let us also assume that, each destination is to receive exactly one packet. We want to route the packets to their destinations as fast as possible. This is the problem we want to solve. Let us take an example first.

(Refer Slide Time: 18:48)

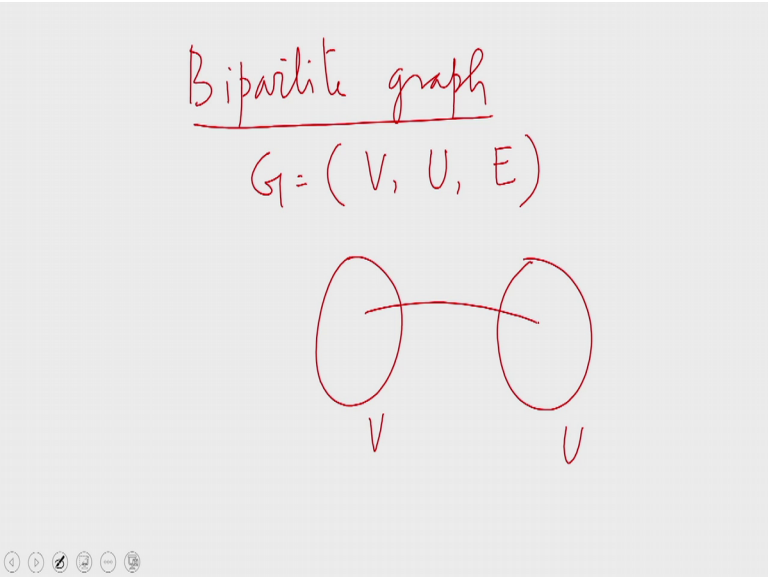


So, we have a mesh, we have a 3 by 3 mesh. So, the processes of the mesh are numbered like this 1 1, 1 2, 1 3 this is the first row so 1 2 belongs to the first row and the second column then 2 1, 2 2 and 2 3, then 3 1, 3 2 and 3 3 and let us say the packets are held by

the processors. Let us assume that processor 1 1 holds a packet that must be sent to 2 3, 1 2 holds a packet that must be sent to 3 3, 1 3 holds a packet that must be sent to 1 2 and so on.

So, the value written outside is the address of the processor, every processor is addressed using two numbers the row number and the column number and the value inside the node is the destination of the packet that is processor 2 2 is holding a packet which must be sent to processor 1 1. So, we want to perform these routings simultaneously and as fast as possible.

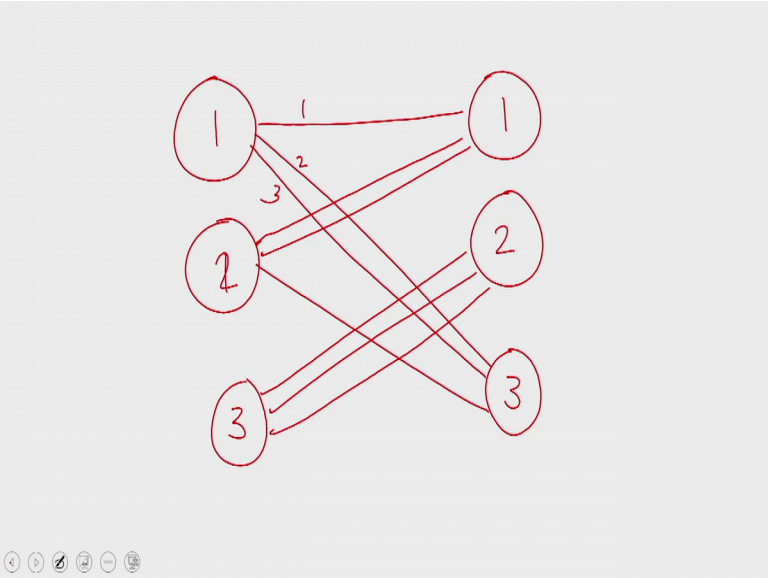
(Refer Slide Time: 20:14)



To this end what we do is to construct a bipartite graph. A bipartite graph is a graph that has 2 vertex sets V and U. So, that every edge in the graph is between these two vertices these two vertex sets. So, we have from the vertex set V and the vertex set U and every edge in the graph is between these two vertex sets.

The graph cannot have an edge between two vertices belonging to the same set that is two vertices belonging to V cannot have an edge between them and two vertices belonging to U cannot have an edge between them. So, such a graph is a bipartite graph.

(Refer Slide Time: 21:07)



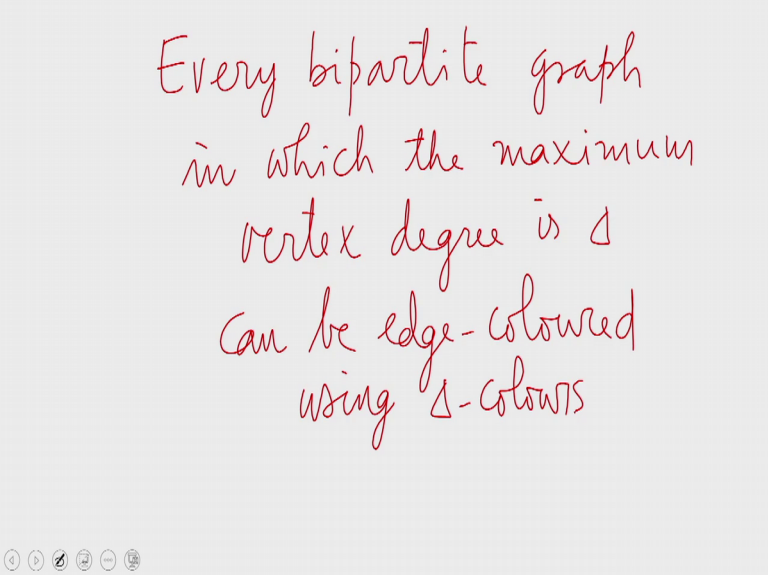
So, let us construct a bipartite graph. In the bipartite graph I take 1 vertex for each column of the mesh. So, these represent the columns, on the other side also I take vertices corresponding to the columns. Now when we look at the mesh we find that we have a packet going from 1 1 to 2 3. So, this is the packet that is traveling from column 1 to column 3. 1 1 is a node in the first column and 2 3 is a node in this third column. So, this is a packet which is traveling from 1 1 to first column to the third column. Therefore, we will draw an edge from 1 to 3 and then we have a packet which is going from 1 2 to 3 3 this is from the second column to the third column.

So, we draw an edge from node 2 on the left side to node 3 on the right side. So, likewise when we complete the graph, we find that we have edges of the sort. So, there could be multiple edges between a pair of nodes if there are multiple packets going between the corresponding columns here you find that there are 3 packets going from column 3 to column 2 which can be seen here, packet 1 3 is going to 1 2. So, that is a packet going from column 3 to column 2. Similarly, packet there is a packet going from 2 3 to 2 2 that is again from 3 to 2 and then 3 3 to 3 2 that is again another packet from 3 to 2. So, there are 3 packets going from column 3 to column 2.

So, accordingly we construct a graph this is a bipartite graph there is no edge between two vertices belonging to the same side and then a well known graph theoretic fact is

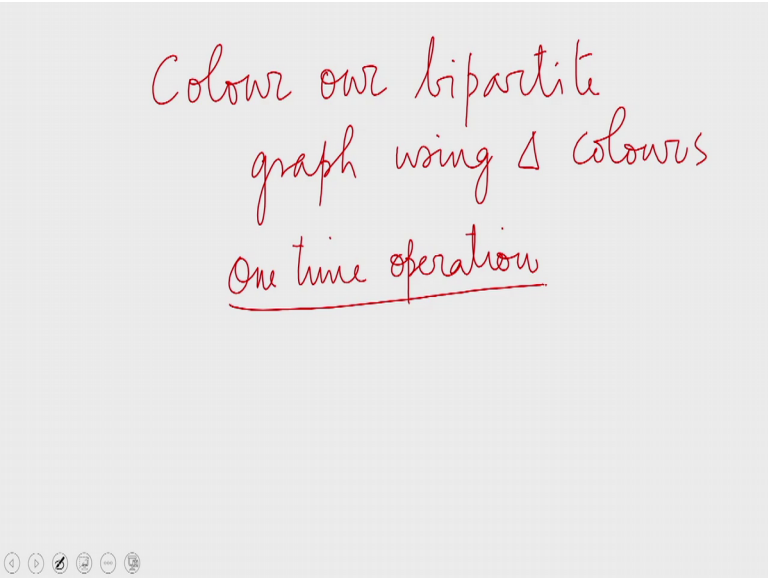
that, every bipartite graph in which the maximum vertex degree is delta can be edged colored using delta colors.

(Refer Slide Time: 23:09)



Edge coloring of a graph is the process of assigning colors to the edges of the graph so that no to adjacent with the same vertex get the same color. So, what this graph theoretic result says is that in every bipartite graph in which the maximum vertex degree is delta we can color the edges of the graph using at most delta colors. We will prove this result in a moment, but for now you assume the result.

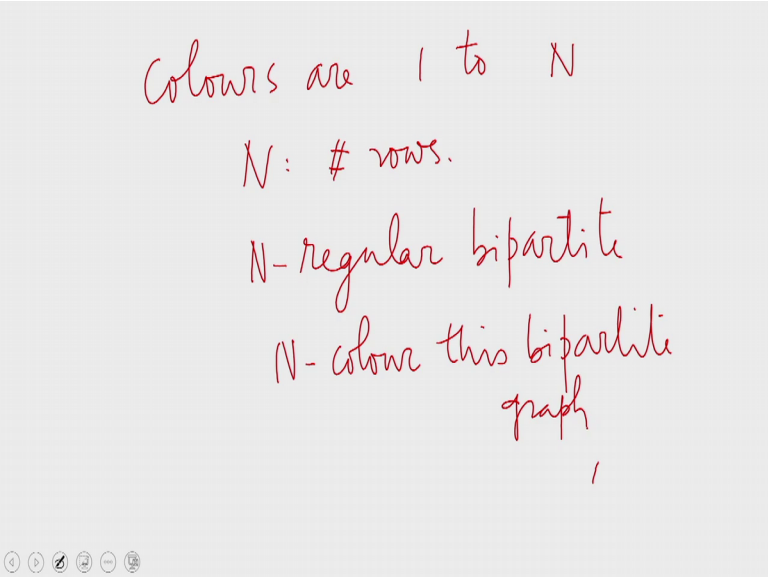
(Refer Slide Time: 24:26)



So, if this result is true, then we could color our bipartite graph using delta colors. Now we are doing this offline. Offline in the sense that, suppose we have to solve the same routing problem again and again that is we have the same mesh and we have the same source destination pairs which means, we have let us say the same mapping. And, we have to send packet after packet from these same sources to same destinations in other words there is a stream of packets that are going from packet 1 2 to 3 3 for example.

So, this source destination mapping is to be repeated time and again, if that is the case then we can do some offline calculations on the basis of this source destination mapping. That calculation in this case involves forming a bipartite graph and coloring it with delta colors. So, we do not go into the complexity of this delta coloring because this is a onetime operation. But once this delta coloring is available then we can use it to quickly route any number of packets. So, assume that this delta coloring is available and let us see how fast we can route one set of packets.

(Refer Slide Time: 26:06)



So, let us say that colors are 1 to N, where N is the number of rows. This is because the degree of the bipartite graph that we have formed is N we are given an N by M mesh. So, we have m columns, the columns can be numbered from 1 to m the rows can be numbered from 1 to N and then when we form the bipartite graph we take one vertex for each column on either side and then we draw edges for the packets out of every column we have N packets going out in every column we have N processors so every processor

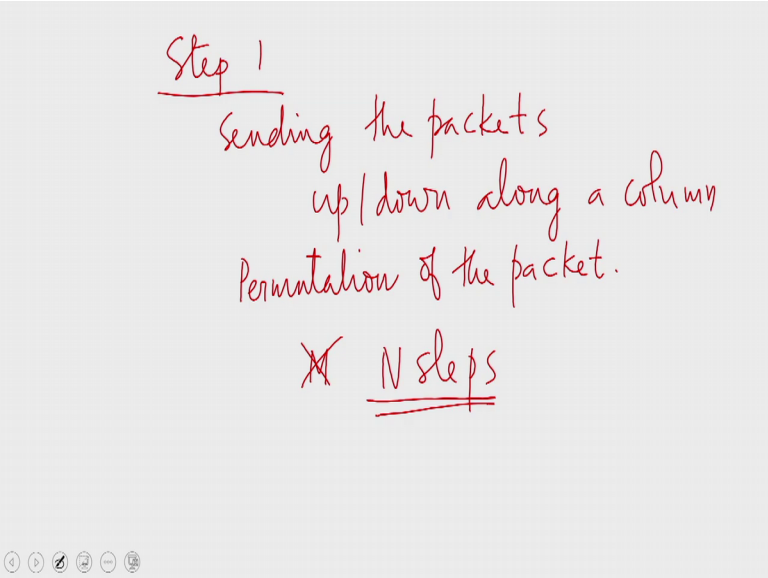
has 1 packet going out. So, there are N packets going out of every single processor, which means the bipartite graph that we have formed is of degree N, it is an N regular bipartite graph. A regular graph is one in which the vertex degree of every node is the same. So, here the degree of every single node is the same N. So, what we have is an N regular bipartite graph what we do is to edge color this bipartite graph using N colors.

Now, after N coloring them what we know is that, at every vertex no to incident edges have the same color the edge coloring is valid. Therefore, going back to our example we find that the 3 edges going out of vertex 1 have got different colors. For example, let us say this is 1, this is 2 and this is 3 or in other words going back to the mesh the first columns packets have got colors let us say 1, 2 and 3. The 3 packets of the first column have got three different colors and then what we are planning to do is to send these packets along the rows that are numbered with its color.

So, the packet which is originating at 2 1 has got a color 1 therefore, this packet should be traveling in the first row. The packet which is originating at 1 1 has got a color of 2 therefore, this should be traveling in the second row and the packet which is originating at 3 1 has got a color of 3, therefore, this should be traveling at the third row. So, now after the offline coloring is made available, the first step of the algorithm that we performance this send all the packets to their appropriate row, the appropriate row is the 1 which is numbered the same as the color of the packet. Now corresponding to each packet we have an edge in the bipartite graph, the color that is given to the edge is the same as the color that is given to the packet, this color is used to route the packet to the correct row.

In other words the packet which is originating at 1 1 is an edge in the bipartite graph this edge has got a color of 2 and therefore, this packet will be routed to the second row of the; second row of the mesh. So, this packet is going to travel along the second row. Now how can this routing happen this involves sending the packets along packets up or down along the column.

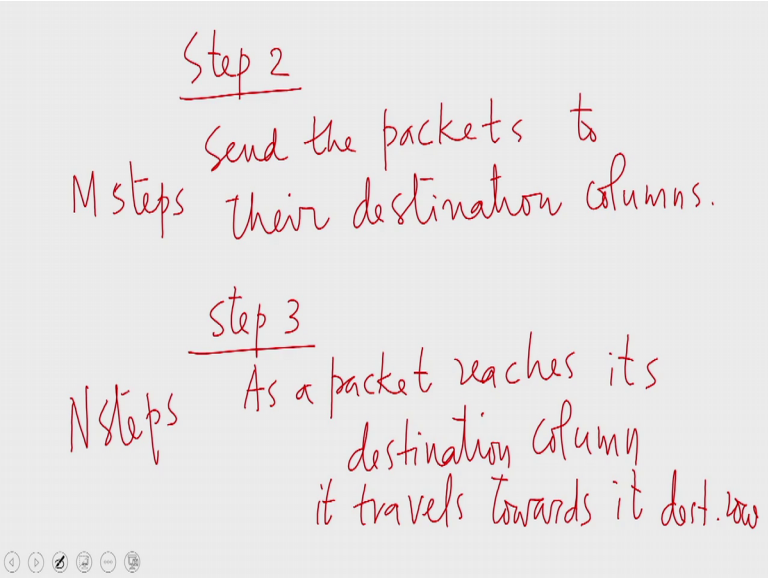
(Refer Slide Time: 29:42)



In other words this is realizing of a permutation of the packets. Every packet has got the unique color, the color is the same as the number of some row. So, all that the packet has to do is to go to the correct row within that column within its own column, but then how can this be achieved this is exactly similar to the second step of the previous algorithm we saw. In that we had the unshuffle operations where the packets travel to their destinations in lockstep the same thing will be done here. We have a number of packets all of them have already calculated their destinations within their columns and then they only have to go up or down along their columns.

So, at every processor we assume that there is one incoming message from each of its neighbors. So, there is one message coming down from the top and one message going up from the bottom the processor has to accept these two messages decide; if one of them is designed to itself if it is, then it is taken out of the queue otherwise the messages have to be sent along the way the message which came from the bottom has to be send up and the message which came from above has to be send down. So, this can be done in N steps. This is moving data within a column the column has the size of N so the moving of the packets can be achieved in N steps.

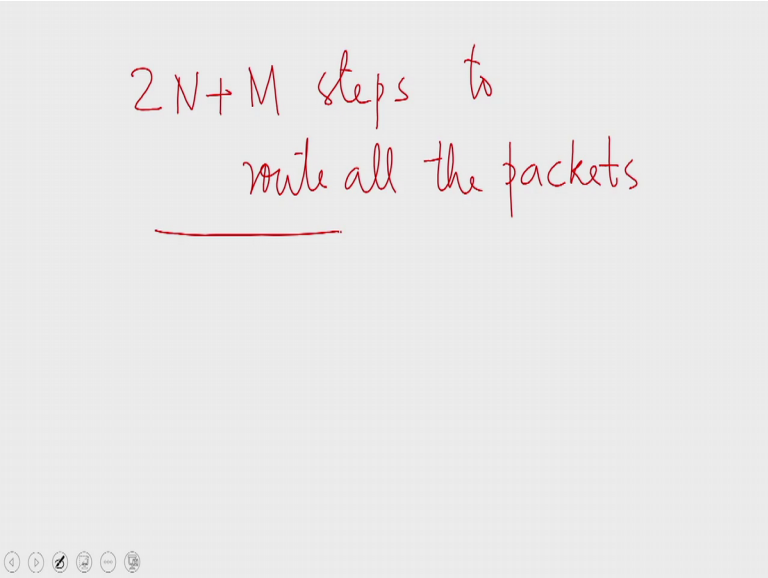
(Refer Slide Time: 31:45)



Now the packets have all reached their correct rows, each packet has identified along which row it wants to travel and then in step 2, send the packets to their destination columns. Every packet knows its column, then all the packets have to do is to go and join their destination columns. There could be multiple packets reaching the same column, but that does not matter, when at any point in time only one packet will be reaching a column at one particular node. And once they reach the column, they have to travel towards their destination node and then in step 3, this is done in continuation to step 2 as a packet reaches its destination column, it travels towards its destination row.

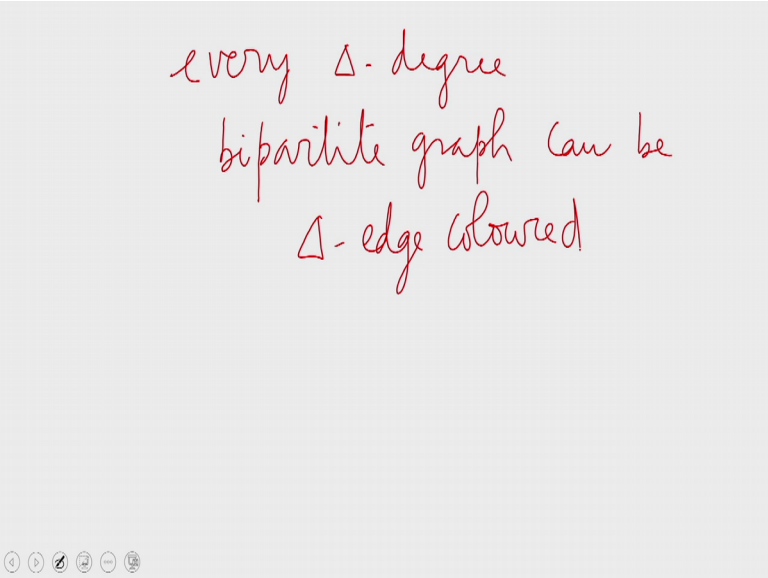
The destination row need not be the same as the one along which it travels. The row that it has got to travel along is the color that has been given to the packet. So, once it reaches the destination column, it has to then travel towards the destination row. Now at the receiving column also we have at most one packet of a color coming to it. So, there can be at most one packet coming to a column along each row. Therefore, once the packets are collected at the columns at the end of step 2, they can be send along to their destination row in step 3. So, step 2 we can see involves moving the packets along their rows so this will take M steps and this will take N steps again.

(Refer Slide Time: 34:08)



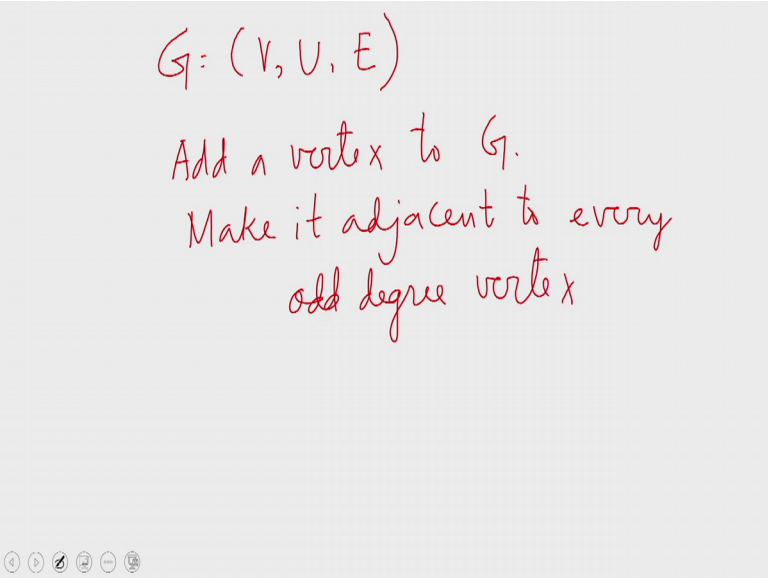
So, the total time taken is 2 N plus M steps to route all the packets. This is optimal because it could be that a packet has to travel a distance of N plus M in any case. Now it remains to show that, every bipartite graph of a maximum degree delta can be indeed delta colored row when we deal with edge colors.

(Refer Slide Time: 34:41)



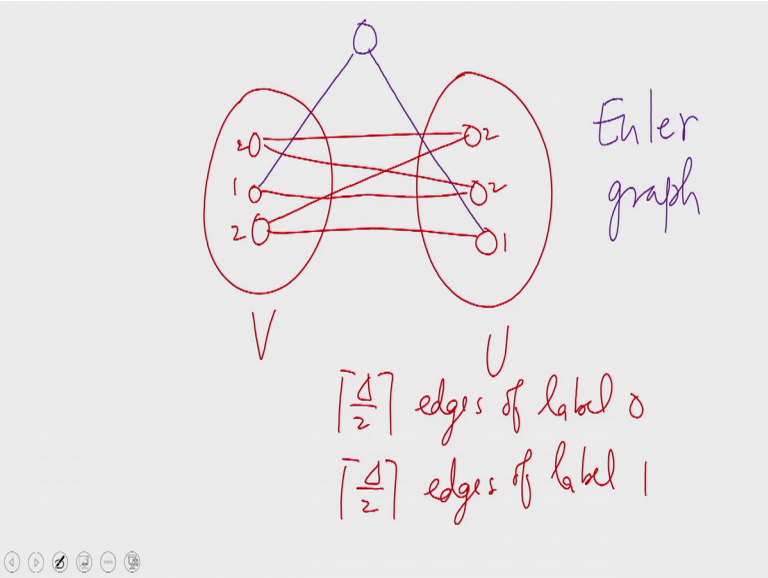
Every delta degree bipartite graph can be delta edge colored.

(Refer Slide Time: 35:07)



To prove this, consider a graph G equal to V, E which is a bipartite graph so let me call it G equal to V, U, E there are two vertex sets V and U and E is the edge set. So, the maximum vertex degree of the graph is let us say delta. The first thing that we do is to add a vertex to G and make it adjacent to every odd degree vertex.

(Refer Slide Time: 35:57)



For example when we have a graph of the sort; when we have a graph of the sort we find that the degree of the vertices is when we calculate the vertex has a degree of 2, this has a degree of 2, this has a degree of 1, this has a degree of 2, this has a degree of 1 and this

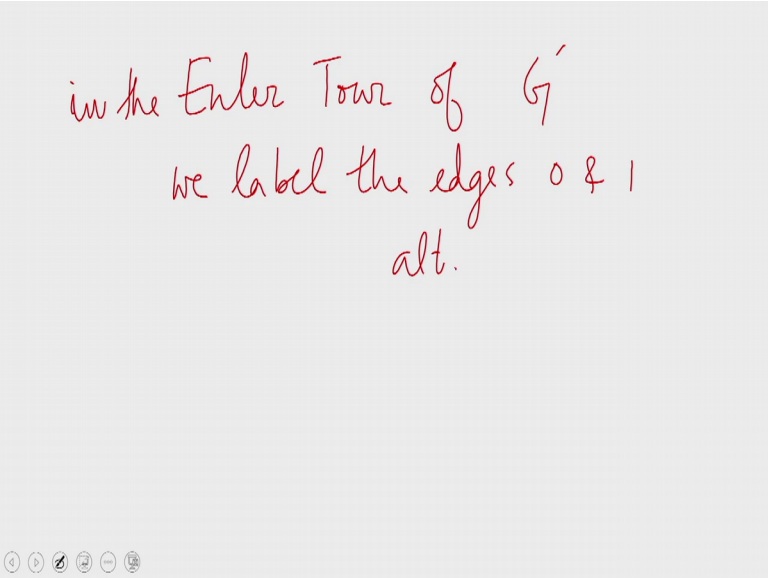
has degree of 2. So, there are exactly 2 vertices of odd degree. We take one extra vertex which does not belong to either V or U and then make it adjacent to the odd degree vertex on either side. The resultant graph will not be bipartite anymore, but that does not matter.

So, we enhance the graph in this manner. Once we do this, what we are ensuring is that the vertex degree of every node in the resultant graph is even that is because in every graph, the sum of vertex degrees is twice the number of edges. Therefore, the sum of vertex degrees is even because that is twice the number of edges. So, if the degrees of all the vertices put together has to be even, then the number of the odd degree vertices in the graph also has to be even because they should sum to an even value.

So, an even number of odd numbers added together will give us an even value. Therefore, this graph has an even number of odd degree vertices. When we add this new vertex and make it adjacent to all the odd degree vertices, then all the odd degree vertices will become of even degree now because they are getting one extra edge, the blue edge is adding to the red edges that it already had. So, the vertex degree of all the odd degree vertices will now be even the degree of the blue vertexes also going to be even that is because there is an even number of odd degree vertices in the original graph.

So, this graph now is an Euler graph. Therefore, we can find an Euler tour of it that is we can start at the vertex visit every single edge and come back to the same vertex without visiting any edge twice.

(Refer Slide Time: 38:27)



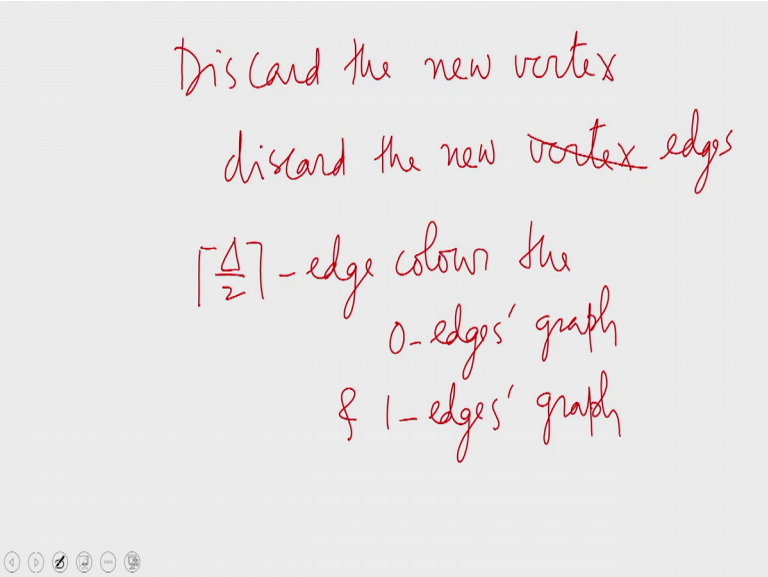
So, let us say when we do an Euler tour G prime is the enhanced graph. Let us say in the Euler tour of G prime we label the edges 0 and 1 alternately, that is when we come into a vertex through an edge labeled 0. We go out through an edge labeled 1 that is as we go through the Euler tour we label the edges 0 and 1 alternately. Therefore, when I look at the vertex I find that the number of 0 labeled edges at that node is exactly half, the vertex is come in to several times and is gone out several times whenever you come into a vertex you trace an edge of a particular label, when you go out you will be tracing an edge of a different label. One of them is of label 0.

So, these two cancel each other let us say then every edge that is incident at the vertex has a pair of the complement label. Therefore, the number of edges of label 0 at every vertex has to be exactly half. So, in this case that will be ceiling of delta by 2, at every single vertex in the graph in the enhanced graph G prime there are at most a, there are exactly ceiling of delta by 2 edges of label 0 and ceiling of delta by 2 edges of label 1 at every single vertex. That is assuming that the original graph was delta regular, then let us discard the new vertex which is the blue vertex in the figure and discard the new edges.

So, we are going back to the original graph and then let us tare the graph into 2 which means we are making two copies of the graph we will have the same set of vertices in both the copies, but then edges of label 0 will be in one copy and edges of label one will

be in the other copy. So, the graph is split vertically the set of vertices are maintained in both the copies, but an edge will belong either to this copy or to this copy.

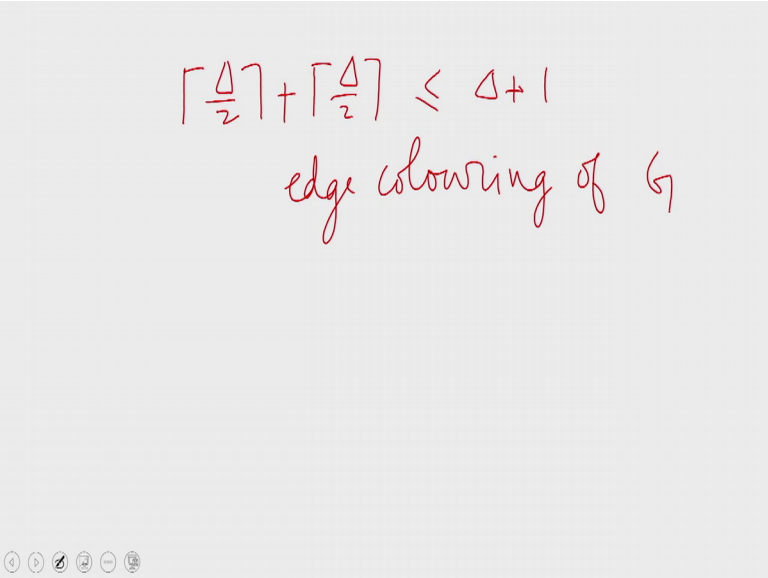
(Refer Slide Time: 40:20)



So, imagine that these two graphs are one above the other, at the upper level let us say we have edges that are labeled 0, at the lower level we have edges that are labeled 1. So, when you consider these two graphs, we find that both the graphs are of degree sealing delta by 2 then, let us recursively assume that they can be ceiling delta by 2 edge colored.

Let us we consider the 2 graphs induced by the edges of label 0 and the edges of label 1 both our ceiling delta by 2 degree graphs so they can be ceiling delta by 2 edge colored. And, then when we put the colorings together assuming that these two colorings are exclusive in the sense that a color which is used at the lower level is not present in the top level and vice versa.

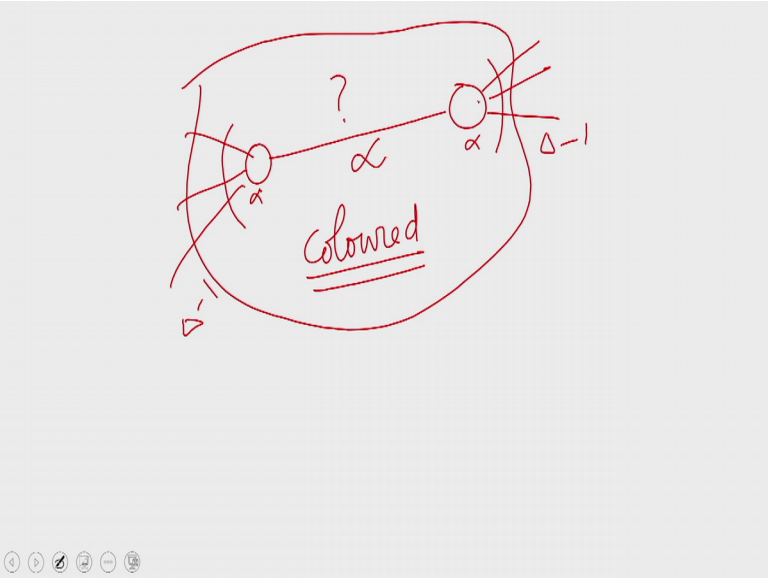
(Refer Slide Time: 42:27)



Therefore when we put the edges together to reconstruct the original graph, we would have obtained a ceiling delta by 2 plus ceiling delta by 2 which is less than or equal to delta plus 1 edge coloring of G. This will be fine if ceiling of delta by 2, if delta is even; if delta is even then ceiling of delta by 2 is the same as delta by 2. Therefore the sum of the 2 colors would be delta by delta. Therefore, now we would have delta edge colored the original graph, but if delta is odd then we have one extra color, we have now delta plus one edge colored the graph for induction to hold good we should get rid of this one extra color.

So, what we do is to remove all vertices all edges of this extra color that is every edge of color delta plus 1 is removed from the graph and then we add these edges one by one to the remaining graph. So, at any point when we add an edge of color delta plus one into the remaining graph this edges deemed uncolored. So, we have the situation where the graph has exactly one uncolored edge, but the remaining edges are all colored.

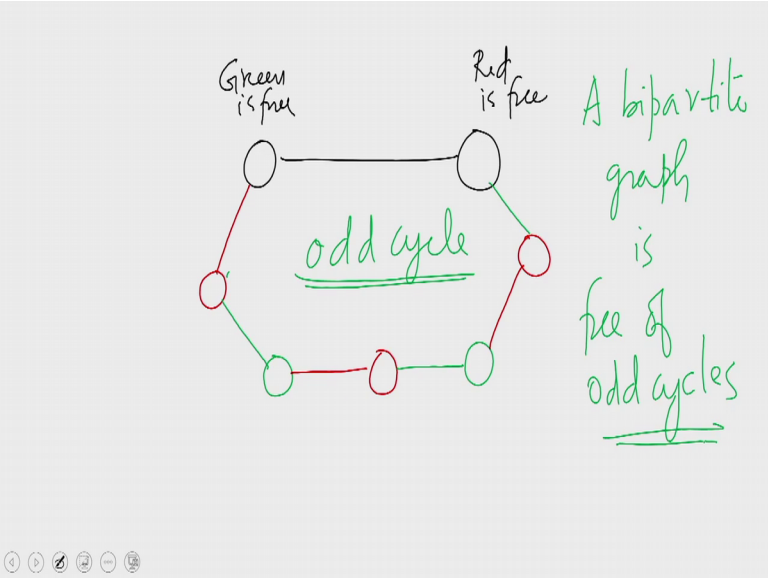
(Refer Slide Time: 43:42)



So, we have one uncolored edge, the rest of the graph are all colored. What is needed is to find a color for this edge. We are seeking to do a delta coloring if there are delta colors at either end if there is at least one free color at either end and this free color happens to be the same there should be one free color at either end. Because, the maximum vertex degree of a node is delta and the other neighbors all put together can be at most delta minus 1 in number.

Therefore, there is one free color at these two vertices at each of these vertices, if that free color happens to be the same. In other words if there is no edge of color alpha here and there is no edge of color alpha here then we can safely give color alpha to this edge. It will not conflict with any of these edges of any of these edges then the problem is solved, but if there is no edge of no color is free, no same color is free at both the vertices then we have an issue. So, let us see what we will do in that case.

(Refer Slide Time: 44:55)



So, let us consider an uncolored edge. So, that at its two different end points the same color is not free. So, let us say color green is free here and color red is free here. Then let me start with the edge of color red here that will take it take us to some node from that node let me take the edge of color green that will take me to some other node. So, what I do is this at this end point green is free, but red is present, if red is absent since red is here I would be able to color this red and the problem would be solved.

So, let me assume that red is not free, if red is not free then there is an edge of color red here. So, let me move along that edge I come to some other vertex, at this vertex if green is free, well and good. If green is not free, then there is an edge of color green there I will travel along that vertex the travel along that edge to come to some vertex. At this vertex if there is an edge of color red I will take it and then from there if there is an edge of color green I will take it and I keep going like this through edges of colored red and green alternately until I come to a dead end.

So, I come to a vertex from which I cannot extend this red, green play any further that is this vertex does not have an edge of color red then, when I look at this path of colors red and green alternating in it I find that this is an island of red and green in a sea of different colors every vertex can have at most one edge of a particular color in the edge coloring problem because, no two edges adjacent at incident at the vertex can have the same color. Therefore, this path is surrounded by a sea of different colors, those colors do not mind if

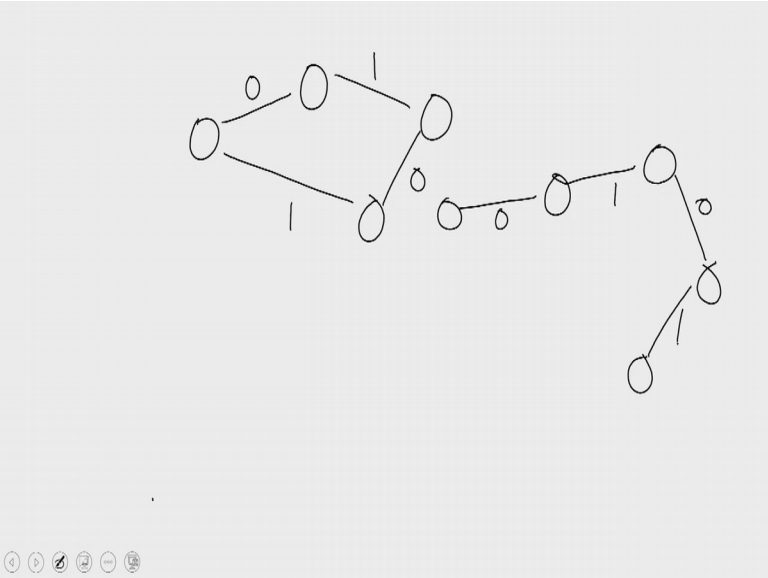
colors red and green are swapped within this path. Therefore I can now safely swap the colors change this to green, this to red, this to green and this to red. If I do this then I would have ensured that green is not free here, but red is free instead then, red is free here as well I would be able to give red to this edge.

This would be the case if only we do not end up getting to this vertex I would be in trouble if this process of taking red and green vertices had taken me to the other vertex. If this were the case then when I swap red and green along this path I would only be exchanging one trouble for the other then red would be free here and green would be free here. And this edge would still be uncolored, but I claim that this will not happen because if this were to happen then, these edges together will form an odd cycle, but then a bipartite graph cannot have an odd cycle.

Therefore, I would certainly not get this condition, I would come to a dead end before we reach here. Therefore, I will be able to swap red and green in the resultant path without affecting the surroundings which means at this node then red would be free and the uncolored edge could be given the red color. So, that would resolve one uncolored edge. Likewise, if I put all the uncolored edges back into the graph and assimilate them into the coloring that, already exist the entire graph would be colored.

So, this concludes the recursion. The basis of the recursion is where you finally, end up with chains and cycles that is you keep on splitting the graph into graphs of smaller degrees. This recursion would end when we have degree two graphs in which case we have chains and cycles but, since a bipartite graph cannot have odd cycles we have only even cycles we would have chains and even cycles. A graph that comprises only of chains and even cycles is a degree two graph and it can be two colored, two edge colored. That is when I have the graph of the sort this can be two edge colored where 0 and 1 are the colors. So, this would be the basis of the recursion.

(Refer Slide Time: 50:02)



So, that establishes that every bipartite graph of a maximum vertex degree of delta can indeed be edge colored with delta colors. So, that shows that the routing can indeed be done in 2 N plus M steps in the N by M mesh as we intended.

We shall see an application of this routing later in the course. So, that is it from this lecture hope to see you in the next.

Thank you.